Annexes au

Mémoire

présenté par

Frédéric Desprez

en vue de l'obtention du diplôme

d'habilitation à diriger des recherches

de l'Université Claude Bernard de LYON

(Numéro d'ordre 48-2001)

Spécialité : **Informatique**

CONTRIBUTION A L'ALGORITHMIQUE PARALLELE

Calcul numérique : des bibliothèques aux environnements de metacomputing

Curriculum vitae et articles

Date de soutenance :	20 juillet 2001	
Composition du jury :		
Rapporteurs	Pierre	MANNEBACK
	Thierry	PRIOL
	Jean	ROMAN
Examinateurs	Michel	COSNARD
	Jack	DONGARRA
	Thomas	LUDWIG
	Yves	ROBERT

Laboratoire de l'Informatique du Parallélisme
UMR 5668 CNRS – ENS Lyon – INRIA Rhône-Alpes.

CONTRIBUTION A L'ALGORITHMIQUE PARALLELE

Calcul numérique : des bibliothèques aux environnements de metacomputing

Annexes : curriculum vitae et articles

Frédéric Desprez

20 juillet 2001

Ce document présente quelques unes de mes publications les plus significatives de mon travail depuis la soutenance de ma thèse. L'ordre n'est pas chronologique mais il respecte l'ordre des chapitres du document principal d'habilitation. Les trois premiers articles correspondent au chapitre 2, les deux suivants au chapitre 3 et enfin les deux derniers correspondent au chapitre 4.

Chapitre A Curriculum Vitae
Je présente mon travail de recherche, d'enseignement et de tâches administratives depuis ma thèse ainsi que toutes mes publications.

Chapitre B Optimization of a *LU* Factorization Routine Using Communication/Computation Overlap
Ce premier article présente mes travaux avec S. Domas et B. Tourancheau sur l'optimisation de la factorisation *LU* grâce aux recouvrements calculs/communication. Même si les gains ne sont pas très important (20% au maximum), nous avons réussi à les modéliser de manière précise et à tirer un modèle général de cette routine. Ces résultats ont été ensuite réutilisés dans ALASCA pour calculer le coût de calcul des routines afin de trouver les meilleures distributions.

Chapitre C Optimal Grain Size Computation for Pipelined Algorithms (1996)
Ces travaux ont été initié durant mon année au LaBRI en collaboration avec P. Ramet et J. Roman. Après avoir étudié durant ma thèse les recouvrements dans des cas linéaires simples, nous avons étendu ces résultats à des polynômes quelconques. Une bibliothèque a été développée qui permet de fournir aux LOCCS la taille de paquets optimale.

Chapitre D Communications Optimizations and Efficient Load-balancing for a Volume Rendering Algorithm on a Cluster of PCs (2000)
Cet article présente mes travaux avec F. Chaussumier sur l'équilibrage des charges et les recouvrements calculs/communications dans un algorithme de rendu volumique. C'est une extension à un problème irrégulier de mes travaux précédents qui concernaient le traitement de structures de données régulières.

Chapitre E Scheduling Block-Cyclic Array Redistribution
Ces travaux concerne l'optimisation de l'ordonnancement des communications pour le problème de la redistribution de données distribuées de manière cyclique par blocs sur un réseau de processeurs. Les applications de ce type de redistribution sont nombreuses (bibliothèques numériques parallèles, compilation d'HPF, etc.).

Chapitre F HPFIT : A Set of Integrated Tools for the Parallelization of Applications Using High Performance Fortran
Cet article présente une partie du projet HPFIT avec notamment les travaux autour de la parallélisation automatique et la recherche de schémas macro-pipelines.

Chapitre G Mixed Parallel Implementations of the Top Level Steps of Strassen and Winograd Matrix Multiplication Algorithms (2001)
Ces travaux présentent mes derniers travaux autour de l'algorithmique parallèle mixte, c'est-à-dire qui utilise à la fois le parallélisme de données et le parallélisme de tâches. Nous avons optimisé les

communications et le placement de données pour deux algorithmes classiques de l'algèbre linéaire, Strassen et Winograd.

Chapitre H Scilab to Scilab$_{//}$ – The OURAGAN Project (2001)
Cet article représente un bilan du projet OURAGAN dont le but est de paralléliser l'outil SCILAB avec différentes approches (bibliothèques de communication, interfaces pour des bibliothèques d'algèbre linéaire parallèle, serveurs de calcul).

Chapitre I A Scalable Approach to Network Enabled Servers (2001)
Cet article présente mes travaux les plus récents sur le développement d'un environnement extensible pour la mise en place d'applications de type ASP (Application Service Provider) dans un environnement de metacomputing.

CONTRIBUTION TO PARALLEL ALGORITHMIC

Numerical Computation: from Libraries to Metacomputing Environments

Appendix: vita and publications

Frédéric Desprez

July 20th, 2001

This document presents some of my main publications since my PhD defense.

Chapter A Vita (french)
I describe my research, teaching and administrative work since my PhD and it gives a list of all my publications.

Chapter B Optimization of a *LU* Factorization Routine Using Communication/Computation Overlap
This first paper presents my work in collaboration with S. Domas and B. Tourancheau around the optimization of the *LU* factorization using overlap of computation by communication. Even if the gains obtained are not that important (maximum of 20%), we managed to give an accurate model of this optimization that leads to a general model of this routine. Then the results were used in ALASCA to compute the computation cost of some numerical routines to be able to guess the most efficient distributions.

Chapter C Optimal Grain Size Computation for Pipelined Algorithms (1996)
This work started during my year at LaBRI in collaboration with P. Ramet and J. Roman. The results presented in this paper extend the results presented in my PhD thesis to general complexity functions for computations and communications. A library was developed that gives the optimal packet size to the LOCCS library routines.

Chapter D Communications Optimizations and Efficient Load-balancing for a Volume Rendering Algorithm on a Cluster of PCs (2000)
This paper presents my work in collaboration with F. Chaussumier around the load-balancing and the overlap of communications in a volume rendering application. This is an extension of my previous work to irregular computations.

Chapter E Scheduling Block-Cyclic Array Redistribution
This paper describes the optimization of the scheduling of communications for the problem of the redistribution of a matrix distributed in a block-cyclic way on a network of processors. There are many applications of such redistribution (parallel numerical libraries, HPF compilation, ...).

Chapter F HPFIT: A Set of Integrated Tools for the Parallelization of Applications Using High Performance Fortran
This paper presents a part of the HPFIT project and more precisely our work around automatic parallelization and the generation of macro-pipeline communications.

Chapter G Mixed Parallel Implementations of the Top Level Steps of Strassen and Winograd Matrix Multiplication Algorithms (2001)
This papers describes my latest work around mixed-parallelism, i.e. the simultaneous use of data and task parallelism. We optimized the communications and data distribution for two basic linear algebra algorithms, Strassen and Winograd.

Chapter H Scilab to Scilab$_{//}$ – The OURAGAN Project (2001)

This paper presents the OURAGAN project whose goal is to parallelize the SCILAB tool using different approaches (message-passing libraries, interfaces for parallel numerical libraries, computational servers).

Chapter I A Scalable Approach to Network Enabled Servers (2001)

This papers describes my latest work around the development of a scalable environment for ASP (Application Service Provider) applications in a grid environment.

Table des matières

A Curriculum vitae **3**
A.1 Curriculum–Vitae . . . 4
A.2 Développement de logiciels . . . 5
A.3 Encadrement de jeunes chercheurs . . . 6
A.4 Activités d'évaluation . . . 8
A.5 Participation à des projets de recherche avec l'industrie . . . 9
A.6 Collaboration nationales et internationales . . . 10
A.7 Tâches collectives . . . 12
A.8 Enseignement . . . 14
A.9 Publications . . . 16

B Optimization of a LU Factorization Routine **21**
B.1 Introduction . . . 22
B.2 Parallel Block *LU* Decomposition . . . 22
B.3 Related Work . . . 24
B.4 Analysis and Optimization of the ScaLAPACK *LU* Factorization . . . 25
B.5 Conclusion and future work . . . 34
B.6 References . . . 35

C Optimal Grain Size Computation for Pipelined Algorithms **37**
C.1 Introduction . . . 38
C.2 Computation/Communication Overlap . . . 39
C.3 Experiments . . . 42
C.4 Conclusion and future works . . . 43
C.5 References . . . 44

D Communications Optimizations and Efficient Load-balancing for a Volume Rendering Algorithm **45**
D.1 Introduction and motivations . . . 46
D.2 Shear-Warp algorithm . . . 46
D.3 Parallel algorithm . . . 47
D.4 Optimizations . . . 50
D.5 Experimental results . . . 51
D.6 Conclusion . . . 54
D.7 References . . . 55

E Scheduling Block-Cyclic Array Redistribution **57**
E.1 Introduction . . . 58
E.2 Motivating Examples . . . 59
E.3 Literature overview . . . 63
E.4 Main Results . . . 65
E.5 MPI Experiments . . . 75
E.6 Conclusion . . . 77
E.7 References . . . 78

F HPFIT : A Set of Integrated Tools for the Parallelization of Applications Using HPF 81
F.1 Introduction . . . 82
F.2 Previous Work . . . 82
F.3 The HPFIT Project . . . 83
F.4 TransTOOL . . . 85
F.5 Optimization Kernel . . . 87
F.6 Conclusion and Future Work . . . 91
F.7 References . . . 92

G Mixed Parallel Implementations of Strassen and Winograd Algorithms 95
G.1 Introduction . . . 96
G.2 Strassen and Winograd algorithms . . . 97
G.3 Mixed parallel algorithms . . . 98
G.4 Time cost models . . . 100
G.5 Experimental results . . . 104
G.6 Conclusion and future work . . . 104
G.7 References . . . 105

H Scilab to Scilab$_{//}$ – The OURAGAN Project 107
H.1 Introduction . . . 108
H.2 SCILAB Processes Duplication . . . 109
H.3 Network-Enabled Servers . . . 113
H.4 Conclusion and future work . . . 120
H.5 References . . . 121

I A Scalable Approach to Network Enabled Servers 125
I.1 Introduction . . . 126
I.2 DIET architecture and related tools . . . 126
I.3 CORBA based communication layer . . . 128
I.4 DIET initialization and operation . . . 129
I.5 Conclusion and future work . . . 133
I.6 References . . . 133

Chapitre A

Curriculum vitae

Ce document suit les indications pour la constitution du dossier de candidature à l'inscription à l'Habilitation à Diriger des Recherches de l'Université Claude Bernard – Lyon I, précisées par la DAED (document jaune). Les éléments suivants y sont développés :

– mon Curriculum–Vitae (section A.1),

– mes développements logiciels (section A.2),

– mes travaux d'encadrement de jeunes chercheurs (section A.3) ,

– mes activités d'évaluation (section A.4) ,

– mes participations à des projets de recherche avec l'industrie (section A.5),

– mes collaborations nationales et internationales (section A.6),

– mes tâches collectives (section A.7),

– mes activités en enseignement (section A.8).

– la liste de mes publications (section A.9).

A.1 Curriculum–Vitae

Frédéric DESPREZ

CR1 - INRIA Rhône-Alpes
Responsable scientifique du projet CNRS–ENS Lyon–INRIA ReMaP

Né le 2 novembre 1964 à Beauvais (Oise)
Marié, 3 enfants
Dégagé des obligations militaires
Nationalité : française

Adresse personnelle :	**Adresses professionnelles :**	
	Laboratoire de l'Informatique du Parallélisme	INRIA Rhône-Alpes
	Ecole Normale Supérieure de Lyon	ZIRST, 655 avenue de l'Europe
40 rue du Stade	46 Allée d'Italie	Montbonnot
38550 SABLONS	69364 Lyon Cedex 07	38334 Saint Ismier Cedex
Tel : 04 74 84 24 53	Tel : 04 72 72 85 69	Tel : 04 76 61 53 48
Mob : 06 12 92 60 03	Fax : 04 72 72 80 80	Fax : 04 76 61 52 52
	E-mail : Frederic.Desprez@inria.fr	
	URL : http://www.ens-lyon.fr/~desprez/	

Responsabilité au sein de l'INRIA

2000- Responsable scientifique du projet ReMaP.

1997- Suivi des thèses passées dans l'Unité de Recherche Rhône-Alpes.

Formation

1994 (6 Janvier) Thèse de Doctorat INP Grenoble, LIP (Laboratoire de l'Informatique du Parallélisme) : "Procédures de base pour le calcul scientifique sur machines parallèles à mémoire distribuée" sous la direction de B. Tourancheau (C.R. CNRS) et la responsabilité administrative de M. Cosnard (Prof. ENS Lyon). Jury : Denis Trystram (Président), Gaétan Libert et Yves Robert (rapporteurs), Gérard Authié, Michel Cosnard et Bernard Tourancheau (examinateurs) et Jack Dongarra et Marc Garbey (invités).

1990 D.E.A. d'Informatique fondamentale (ENS Lyon - 06/90) "Algèbre Linéaire sur Tnode" DEA préparé au Laboratoire LIP de l'ENS Lyon sous la direction de B. Tourancheau.

1989 Maîtrise d'informatique (Grenoble - 06/89 - Mention AB).

1987 Licence d'informatique (Grenoble - 06/87 - Mention AB).

Séjours à l'étranger de plus de 3 semaines

1996	1 mois dans le Computer Science Dept. à l'Université du Tennessee (Knoxville - USA) dirigé par le Professeur Dongarra.
1994	8 mois de postdoc dans le Computer Science Dept. de l'Université du Tennessee.
1993	1 mois dans le Computer Science Dept. à l'Université du Tennessee.

Stages, emplois et service militaire

1998-	CR 1 INRIA Rhône-Alpes, projet ReMaP.
1995-98	CR 2 INRIA Rhône-Alpes, projet ReMaP.
1994-95	Maître de Conférences à l'ENSERB (Université Bordeaux I) et recherche au LaBRI (équipe ALIENOR dirigée par Jean Roman).
1990-93	Thèse de doctorat au LIP.
1990	Chef de projet chez SMART (Valence) pendant 6 mois. Protection d'un réseau Unix du CNET (Issy-les-Mlx) avec des cartes à puces. Conduite de l'ensemble du projet (de la spécification à la réalisation) et encadrement d'un ingénieur.
1990	Stage de DEA au LIP (4 mois). Etude d'algorithmes parallèles fondamentaux et implémentation sur Tnode (TELMAT) à base de Transputers.
1989	Stage de 2 mois au LIP. Etude et réalisation d'algorithmes de produit de matrices parallèles sur machine Tnode.
1988	Service militaire au 27^{e} RCS de Grenoble (création d'une base de données sur un PC et formation des appelés et des secrétaires).

Langues étrangères

Anglais	parlé, lu et écrit couramment.

Participations à des projets nationaux, industriels et internationaux

Entre parenthèse la période pendant laquelle j'ai travaillé sur un projet.

• **Eureka EuroTOPS** (6/93-10/98)

Responsable scientifique (95-97)
Responsable tâche bibliothèques de calcul parallèles (09/93-12/95)
Responsable tâche outil de parallélisation de codes Fortran 77 (10/95-10/98)

• **INCO-DC Paralin** (06/96-09/99)

Responsable tâche parallélisation logiciel Scilab (06/96-06/98)

• **TTN ProHPC** (03/97-09/99)

Responsable action ACRA/Solid Dynamics (03/97-02/98)
Responsable action ACRA/CADOE (03/97-02/98)
Responsable action DYNA (03/98-10/98)
Responsable action METAL (09/98-01/99)

• **ARC INRIA OURAGAN** (01/98-12/00)

Responsable de l'ARC (90-00)

• **RNRT VTHD** (01/99-12/01)

Responsable du sous-projet 5 pour l'UR Rhône-Alpes
Responsable de l'action Scilab$_{//}$

A.2 Développement de logiciels

Mes travaux ont donné lieu à des développements logiciels importants. Certains de ces développements ont été réalisés dans le cadre des projets Eureka EuroTOPS, TTN ProHPC, PARALIN et l'ARC INRIA OURAGAN (décrits dans la section A.5).

- **TransTool** Intégration des différents outils dans un seul environnement, développement des interfaces C, Lisp et TCL, développement du noyau d'optimisation, d'HPFize (outil d'insertion de directives), etc. (*en collaboration avec A. Darte, J.-C. Mignot, C. Randriamaro, G. Silber*).
- **ScaLAPACK** Optimisation de la factorisation *LU* de ScaLAPACK et développement d'une routine de calcul de taille de bloc optimale (fonction de la taille de la matrice et du nombre de processeurs), étude de la redistribution de matrices (*en collaboration avec J. Dongarra, S. Domas, A. Petitet, C. Randriamaro et Y. Robert*). ScaLAPACK est diffusé par ftp.
- **LOCCS** Développement d'une version améliorée des LOCCS (avec interface portable MPI[1] avec nouvelles routines (*en collaboration avec P. Ramet et F. Chaussumier*).
- **Adaptor** Amélioration de l'intégration des LOCCS dans Adaptor, amélioration du driver, nouveaux types de schémas traités, etc. (*en collaboration avec T. Brandes et J. Zory*). Adaptor est diffusé par ftp.
- **Scilab** Intégration de PVM comme couche de communication d'une version parallèle de Scilab, développement d'une interface ScaLAPACK et définition des serveurs de calcul (*en collaboration avec les membres de l'ARC INRIA OURAGAN*). Scilab est un logiciel diffusé mondialement par ftp, notamment grâce à sa version Linux.

A.3 Encadrement de jeunes chercheurs

Je soutiendrais mon Habilitation à Diriger des Recherches début Juin 2001.

A.3.1 Etudiants en thèse

A.3.1.1 Thèse de Stéphane Domas

J'ai co-encadré à 75 %S. Domas (MENRT au LIP) avec Bernard Tourancheau sur des problèmes d'algèbre linéaire en parallèle. Son but était d'améliorer une bibliothèque existante (ScaLAPACK) en proposant de nouveaux algorithmes plus adaptés à une parallélisation, soit en proposant des nouveaux schémas de distribution de données ou en utilisant des pipelines de calculs et de communications.

Stéphane Domas a soutenu sa thèse le 23 Octobre 98. Il a été recruté en 2000 après son service militaire et un an d'ATER comme Maître de Conférences à l'IUT de Belfort et effectue sa recherche au LIFC dans l'équipe de Jean-Marc Nicod.

A.3.1.2 Thèse de Pierre Ramet

J'ai co-encadré P. Ramet avec Jean Roman (LaBRI) (25/75) sur des problèmes de recouvrements calculs/communications dans les codes scientifiques. Son but était de développer des techniques de calcul de grain pour des algorithmes pipelines lorsque les polynômes caractérisant les calculs (ou les communications) ne sont pas forcément des fonctions linéaires de la taille des données. Il a travaillé également sur un solveur parallèle creux de type Cholesky Crout.

Pierre Ramet a soutenu sa thèse le 12 Janvier 2000 et il a été recruté comme Maître de Conférences à l'IUT de Bordeaux I et effectue sa recherche au LaBRI dans l'équipe de Jean Roman.

A.3.1.3 Thèse de Cyrille Randriamaro

J'ai co-encadré Cyrille avec Yves Robert (50/50) sur des problèmes de distribution et redistribution semi-automatiques de matrices avec HPF dans le cadre du projet TransTool. Il a également travaillé sur l'optimisation de l'ordonnancement des communications pour la redistribution de matrices distribuées avec des schémas cycliques par blocs.

Cyril a soutenu sa thèse le 24 Janvier 2000 et a été recruté comme Maître de Conférence à l'Université d'Amiens et effectue sa recherche au LaRIA dans l'équipe de J.-F. Myoupo.

[1]Message Passing Interface

A.3.1.4 Thèse de Frédérique Chaussumier (soutenance en Juin 2001)

Frédérique (3ème année de CIFRE avec MS&I[2], 50% avec M. Loi) a travaillé sur l'optimisation des communications dans des applications irrégulières et sur environnement hétérogène. Dans un premier temps, elle a travaillé sur l'optimisation d'un algorithme de rendu volumique 3D en utilisant des techniques de recouvrement calcul/communication et d'équilibrage des charges élastique. La machine cible était une grappe de PC connectés par un réseau Myrinet et les contraintes industrielles étaient d'obtenir un rendu en temps réel lorsque l'on effectuait une rotation arbitraire du volume. Ensuite, elle a travaillé sur une application benchmark du programme américain ASCI, le Sweep3D en utilisant encore des techniques de pipelines.

A.3.1.5 Thèse de Frédéric Suter (en cours)

Frédéric (2ème année de MENRT au LIP, 100%) travaille sur des problèmes d'optimisation de codes numériques utilisant à la fois le parallélisme de tâches et le parallélisme de données. Les résultats de ses travaux seront utilisés par l'environnement DIET pour répartir la charge entre les divers serveurs.

A.3.1.6 Thèse de Martin Quinson (en cours)

Martin (1ère année de MENRT au LIP, 100%) travaille sur la mise en place de serveurs de calcul dans un environnement de metacomputing. Ses premiers travaux consistent à développer un outil d'évaluation de performances statiques et dynamiques de réseaux et de bibliothèques de calcul. Cet outil permettra à l'ordonnanceur d'un environnement de type Network Enabled Servers d'évaluer le coût des migrations de données entre serveurs et le coût de calcul des problèmes à résoudre sur ces mêmes serveurs.

A.3.2 Autres étudiants

- **Etudiants de DEA**
  - **Martin Quinson**, a effectué son DEA en 2000 à l'ENS. Martin a travaillé sur l'étude d'un environnement d'évaluation de performances dans un environnement de metacomputing et son intégration dans Scilab// (co-encadrement 50/50 avec F. Suter).
  - **Nathalie Viollet**, a effectué son DEA en 2000 à l'ENS. Nathalie a étudié la mise en place d'une base LDAP pour la recherche de ressources logicielles dans un environnement de metacomputing et son intégration dans Scilab// (co-encadrement 25/75 avec J.-F. Méhaut).
  - **Laurent Bobelin**, a effectué son DEA en 99 à l'ENS. Laurent a travaillé sur des heuristiques de placement de données pour des algorithmes à parallélisme mixte (co-encadrement 50/50 avec C. Randriamaro).
  - **Jacques-Alexandre Gerber**, a effectué son DEA en 98 à l'ENS. Jacques-Alexandre a travaillé sur des outils de transformation automatique de programmes Fortran 77 contenant des appels de bibliothèques BLAS et LAPACK vers HPF (encadrement à 100%).
  - **Fabrice Rastello**, a effectué son DEA en 97 à l'ENS. J'ai co-encadré Fabrice avec Yves Robert sur des optimisations de communications et du grain de calcul dans les codes numériques (technique du tiling) (co-encadrement 50/50 avec Y. Robert).
  - **Julien Zory**, a effectué son DEA en 96 à l'ENS. Julien a étudié la recherche automatique de schémas macro-pipelines et a continué l'intégration de la bibliothèque LOCCS dans le compilateur HPF Adaptor (encadrement à 100%).
  - **Pierre Ramet**, a effectué son DEA en 1995 au LaBRI. Pierre a travaillé sur des méthodes de calcul de taille de paquets optimales pour les pipelines de communication (co-encadrement 50/50 avec J. Roman).
  - **Pierre Garnier** , a effectué son DEA en 1995 au LaBRI. Pierre a travaillé sur la parallélisation d'un code du CEA à l'aide de High Performance Fortran (co-encadrement 50/50 avec J. Roman).
  - **Bruno Jargot**, a effectué son DEA en 1993 à l'ENS. Bruno a travaillé sur une interface orientée objet pour la bibliothèque LOCCS (encadrement à 100%).

[2]Matra Système & Information

- **Etudiants de DESS**
  - **Gilles Lebourgeois**, du DESS de Strasbourg (option parallélisme). Gilles a travaillé sur l'analyse syntaxique et sur l'algorithme d'Allen et Kennedy dans le cadre du projet TransTool. A la suite du stage, Gilles a été recruté 2 ans comme ingénieur d'étude INRIA dans le cadre du projet TTN ProHPC.
  - **Lionel Tricon** du DESS de Strasbourg (option parallélisme). Lionel a travaillé sur l'analyse de dépendances et sur la recherche de boucles *Cross-Processors* dans le cadre du projet TransTool. A la suite du stage, Lionel a été recruté 1 an comme ingénieur d'étude INRIA dans le projet ReMaP.
  - **Frédéric Naquin**, du DESS de Lyon 1 (prom. 97) (option réseaux). Frédéric a comparé les différentes couches de communications sur les réseaux de type Myrinet.
- **Etudiants de Maîtrise**
  - **Stéphane Vernat**, a effectué sa Maîtrise à Lyon 1 en 97. J'ai co-encadré S. Vernat avec Stéphane Ubéda sur la parallélisation du logiciel GRASS de systèmes d'information géographiques.
  - **Dominique Ponsard**, a travaillé en 98 sur le portage d'une application de modèle numérique de terrain sous MPI. Il s'agit d'une collaboration avec G. Vidal du laboratoire des Sciences de la Terre de l'ENS Lyon. A la suite du stage, Dominique a été recruté comme ingénieur d'étude au CNRS.
  - **Martin Quinson**, a effectué sa Maîtrise à St Etienne en 99. J'ai encadré M. Quinson sur l'optimisation de la routine de distribution et de récupération de données de la bibliothèque ScaLAPACK.
  - **Thierry Murgue**, a effectué sa Maîtrise à St Etienne en 99. J'ai co-encadré T. Murgue avec S. Ubéda sur le développement d'une interface graphique de visualisation de matrices creuses distribuées dans Scilab.
- **Autres encadrements**
  - **Olivier Reymann**, a travaillé en 96 sur la première version de TransTool, notamment sur l'étude de l'intégration de l'éditeur XEmacs dans une fenêtre Tk.
  - **Nicolas Bert**, a travaillé en 96 sur le nettoyage de la bibliothèque LOCCS et sur sa version MPI.

A.4 Activités d'évaluation

A.4.1 Jurys (thèses, diplôme d'ingénieur)

J'ai été membre de plusieurs jurys de thèses
- **Asier Ugarte**, soutenue le 04 janvier 2001 à Bordeaux (rapporteur),
- **Eddy Caron**, soutenue le 14 décembre 2000 à Amiens (examinateur),
- **Fabrice Rastello**, soutenue le 6 septembre 00 à Lyon (examinateur),
- **Cyril Randriamaro**, soutenue le 24 janvier 00 à Lyon (directeur),
- **Pierre Ramet**, soutenue le 12 janvier 00 à Bordeaux (directeur),
- **Julien Zory**, soutenue le 17 décembre 99 à Paris (examinateur),
- **Stéphane Domas**, soutenue le 23 octobre 98 à Lyon (directeur),
- **David Laimani**, soutenue le 6 janvier 97 à Besançon (examinateur),
- **Makan Pourzandi**, soutenue le 20 janvier 95 à Lyon (examinateur).

et d'un jury de diplôme d'ingénieur CNAM :
- **Véronique Chabanis**, soutenu le 31 janvier 00 à Grenoble (examinateur).

A.4.2 Autres tâches d'évaluation

J'ai du évaluer en 1997 le dossier déposé à la région Aquitaine par Serge Chaumette (LaBRI, Université de Bordeaux).

Cette année, j'ai dû évaluer une proposition de BQR pour l'INSA de Lyon ainsi qu'un projet autour de l'utilisation d'une plate-forme de type grille de calcul pour le gouvernement belge.

A.5 Participation à des projets de recherche avec l'industrie

A.5.1 Projet Eureka EuroTOPS

Le projet Eureka EuroTOPS avait pour but le développement d'outils pour la parallélisation d'applications sur machines parallèles à mémoire distribuée. C'était un projet européen dont les partenaires étaient Matra Systèmes et Information (MS&I), l'ENS Lyon, l'INRIA, le CNRS, Simulog, ESI, NA Software, et quelques autres. Les machines cibles étaient la CAPITAN et le cluster de PC avec réseau Myrinet Peak-Server installées au LIP.

Dans ce projet, j'ai participé dans un premier temps à la tâche portant sur les bibliothèques de calcul numérique parallèles et leur portage sur la machine CAPITAN. Mes travaux sur l'algèbre linéaire parallèle cités précédemment entrent directement dans ce projet.

Ma seconde tâche dans EuroTOPS consistait dans le développement du logiciel TransTool et de son utilisation avec les outils de Simulog et NaSoftware.

De 1996 à la fin du projet (décembre 98), j'ai été en outre le responsable scientifique du projet pour le LIP.

A.5.2 Projet INCO-DC Paralin

J'ai participé à la mise en place du projet PARALIN. Ce projet était un projet INCO-DC de la communauté européenne avec des partenaires universitaires et industriels de France, du Chili, d'Espagne et d'Uruguay. Le but était le transfert industriel du calcul parallèle dans l'industrie minière et de l'énergie au Chili et en Uruguay. Ce projet est en relation avec le projet PARANDES[3] dont le but était l'installation d'une machine CAPITAN au Chili.

L'INRIA était impliquée dans ce projet avec les projets ReMaP et Promath (INRIA Rocquencourt). Notre travail a consisté à porter une application d'optimisation de réseau électrique sur la version parallèle de Scilab à l'aide de PVM en collaboration avec Frédéric Bonnans de Promath.

Le projet s'est terminé en septembre 1999.

A.5.3 Projet TTN-ProHPC

J'ai participé pour le LIP et ReMaP à l'écriture du projet HPCN TTN ProHPC dont le but était de promouvoir l'utilisation du calcul parallèle dans l'industrie [BBD+98]. Le projet était constitué de quatre partenaires : l'ENS Lyon (coordinateur), l'INRIA, MS&I et Simulog.

J'étais responsable pour le LIP de deux tâches du TTN dans l'action ACRA pour la dissémination du calcul à haute performances dans les PME de la région. Les deux sociétés, CADOE et Solid Dynamics, ont développé des logiciels dans le domaine de la mécanique (simulation). J'ai participé à la parallélisation de ces logiciels en collaboration avec les ingénieurs du TTN en poste au LIP, C. Barberet et G. Lebourgeois.

L'action ACRA avec CADOE a donné lieu à une suite : le projet DYNA. Dans ce projet, j'ai étudié avec G. Lebourgeois la parallélisation du noyau de calcul du logiciel, en l'occurence un solveur creux itératif.

Un autre projet du même type a démarré avec la société SYSTUS : le projet METAL avec un code d'éléments finis. J'ai étudié la parallélisation du logiciel de la société en collaboration avec un ingénieur de recherche INRIA, R. Choquet.

Le projet s'est terminé en septembre 1999.

A.5.4 Projet RNRT VTHD

Le but du projet RNRT VTHD [4] est de connecter une certains nombre de centres de recherche de France Telecom et toutes les Unités de Recherche INRIA par un réseau à très haut débit (2.5 Gigabits par seconde). De nombreuses recherches sont effectuées autour des aspects réseaux et un sous-projet concerne les applications qui permettront d'exhiber les limites d'un tel réseau et les problèmes dus à sa mise en place. Je suis

[3]du programme ITDC'94.
[4]réseau à Vraiment Très Haut Débit.

responsable de ce sous-projet au niveau de l'UR Rhône–Alpes et je l'anime au niveau national avec Thierry Priol.

Nous avons également proposé une action de ce sous-projet dont le but est de porter nos serveurs de calcul sur le réseau VTHD. Une telle bande–passante va nous permettre de déporter des calculs avec un grain plus fin. Nous recrutons actuellement un postdoctorant (Eddy Caron) sur le sujet.

Le projet se terminera en décembre 2001.

A.5.5 Proposition de projet RNTL GASP

Je suis le responsable du projet RNTL GASP (Grid Application Service Provider) qui vient d'être déposé au ministère de l'industrie en février. Ce projet a pour but de développer une infrastructure logicielle permettant la mise en place simple et performante d'applications de type ASP sur une plate-forme de metacomputing. Les partenaires sont le projet ReMaP, le projet Résédas de l'INRIA Lorraine, l'équipe SDRP du LIFC, le laboratoire IRCOM, le laboratoire des Sciences de la terre de l'ENS Lyon et Sun Labs.

A.6 Collaboration nationales et internationales

A.6.1 Relations académiques

A.6.1.1 Computer Science Dept, The University of Tennessee, Knoxville, USA

J'ai collaboré avec l'équipe du Professeur Dongarra, autour de ScaLAPACK et en particulier de la redistribution de matrices dans le cadre d'un PICS, d'un projet CNRS-NSF puis d'un projet INRIA-NSF. Nous sommes en train de mettre en place une nouvelle collaboration avec l'équipe de Jack Dongarra autour du logiciel NetSolve et avec l'équipe de Rich Wolsky autour du logiciel NWS.

Par ailleurs, j'ai effectué un postdoctorat de huit mois dans ce même laboratoire à partir de février 1994. En accord avec Jack Dongarra, j'ai continué mes recherches sur les bibliothèques de calcul et de communication en les intégrant dans les outils développés dans son laboratoire. J'ai pu ainsi valider certaines méthodes sur différentes nouvelles machines et aider les chercheurs à obtenir des codes performants et lisibles grâce à mes bibliothèques. J'ai également travaillé sur les routines de distribution et de récupération de données dans ScaLAPACK.

J'ai ensuite effectué un séjour de trois semaines durant lequel nous avons travaillé sur la redistribution de données [DDP$^+$98].

A.6.1.2 GMD/SCAI, Bonn, Allemagne

J'ai collaboré pendant trois ans avec T. Brandes. Thomas est le concepteur du compilateur HPF Adaptor. Nous avons travaillé sur l'intégration de la bibliothèque LOCCS dans le compilateur et sur la définition des fonctionnalités de TransTool.

A.6.1.3 LaBRI, Bordeaux

Collaborations avec J. Roman, S. Chaumette, M.-C. Counilh, F. Pellegrini et P. Ramet de l'équipe ALIENOR dans le cadre du projet HPFIT autour de la visualisation de données distribuées et de la parallélisation d'applications manipulant des structures de données creuses, sur les bibliothèques LOCCS et OPIUM et dans le cadre de l'ARC INRIA OURAGAN autour de l'utilisation de solveurs creux parallèles dans Scilab.

A.6.1.4 Résédas, INRIA-LORIA, Nancy

Collaboration avec E. Fleury et E. Jeannot dans le cadre de l'ARC INRIA OURAGAN autour de l'optimisation du logiciel Netsolve (ajout de persistence de données sur les serveurs et ordonnancement des tâches) et interfacage de Scilab avec la bibliothèque ScaLAPACK.

A.6.1.5 LIFC, Université de Franche-Comté, Besançon

Collaboration avec J.-M. Nicod et son équipe dans le cadre de l'ARC INRIA OURAGAN autour de la gestion de serveurs sous CORBA et pour le développement d'un environnement hiérarchique d'agents.

A.6.1.6 LARIA, Université de Picardie, Amiens

Collaboration avec G. Utard et son équipe dans le cadre de l'ARC INRIA OURAGAN autour de l'interfacage de Scilab avec des bibliothèques out-of-core et l'ajout de types distribués dans Scilab.

A.6.1.7 Métalau, INRIA Rocquencourt

Collaboration avec C. Gomez, S. Steer et M. Goursat dans le cadre de l'ARC INRIA OURAGAN autour de la parallélisation de Scilab.

A.6.2 Autres relations académiques

Projet Promath, INRIA Rocquencourt : Collaboration avec F. Bonnans autour de la parallélisation d'algorithmes d'optimisation avec Scilab (projet PARALIN).

J'ai participé aux diverses actions nationales de collaborations inter-laboratoires comme CAPA, RUMEUR, iHPERF et Grappes.

A.6.2.1 Projets académiques

ARC INRIA OURAGAN J'ai été responsable de l'ARC OURAGAN (Janvier 99–Décembre 2000) qui a fait collaborer trois projets (Métalau à Rocquencourt, Résédas à Nancy et ReMaP à Lyon) et trois équipes (ALIENOR au LaBRI, l'équipe de J.-M. Nicod au LIFC et l'équipe de G. Utard au LaRIA).

Les objectifs sont d'offrir aux utilisateurs l'accès à diverses ressources de calcul (qu'elles soient matérielles ou logicielles) et ce, même si ces dernières sont distribuées au sein un réseau. Le but est d'avoir une plate-forme performante, facile d'utilisation (i.e., pas réservée uniquement aux seuls experts en programmation parallèle) intégrant les mécanismes nécessaires pour pouvoir effectuer et utiliser des ressources de calcul de façon distante. La facilité d'utilisation est primordiale et l'aspect interactif que l'on retrouve dans Scilab doit être conservé. La performance est garantie par l'intégration de bibliothèques parallèles de calcul dense et creux au sein du logiciel Scilab et par la mise en œuvre de politiques de redistribution, de répartition de charge permettant d'utiliser au mieux les ressources de calcul disponibles et ce de la façon la plus transparent possible pour l'utilisateur qui peut continuer à développer et prototyper ses applications en ligne.

Nous avons obtenu un financement de l'INRIA qui nous a permis de recruter un postdoctorant pour un an (E. Jeannot). Le travail d'Emmanuel a consisté à étudier NetSolve et à le modifier pour mettre en place l'architecture logicielle de nos serveurs de calcul. Il a par ailleurs étudié l'interfacage de bibliothèques parallèles de calcul creux au sein d'un outil comme Scilab (définition de nouveaux types et de nouveaux opérateurs).

L'ARC OURAGAN s'est terminée en décembre 2000.

Projet NSF–INRIA autour du creux J'ai participé à la mise en place d'une proposition de contrat NSF–INRIA sur des préconditionnements robustes et parallèles. Côté français, je suis coordinateur avec B. Philippe de l'IRISA. Y. Saad de l'Université du Minnesota est le coordinateur côté américain. Les laboratoires impliqués sont en France le CERFACS, le projet ReMaP, le projet ALADIN, le LaBRI et aux Etats–Unis l'Université du Minnesota, l'Université de l'Indiana et le Lawrence Berkeley Lab. Le projet concerne le développement de procédures de résolution de très grands systèmes linéaires à l'aide de méthodes itératives, directes et hybrides. J'interviens pour l'optimisation l'algorithmes numériques parallèles.

Montage d'un projet NSF–INRIA autour de NetSolve Je suis en train de mettre en place un programme NSF–INRIA entre l'Université du Tennessee, le projet Résédas de l'INRIA Lorraine et le projet ReMaP autour de nos travaux sur NetSolve. Cette collaboration a démarrée suite à la visite de Martin Quinson et Emmanuel Jeannot dans le laboratoire de Jack Dongarra. Ayant optimisé le logiciel NetSolve de manière importante grâce à nos recherches autour de Scilab (amélioration des transferts de données, augmentation de la précision des prédictions des coûts de calcul et de communication dans la grille, etc.), les chercheurs travaillant sur NetSolve nous ont proposé de mettre en place cette collaboration et de la financer en partie grâce à un programme NSF–INRIA.

Fédération de calcul lyonnaise J'ai participé à la réponse à l'appel d'offre de la région Rhône–Alpes. Le but de cette action est la mise en place d'une fédération de laboratoires et donc de rassembler les forces de calculs des divers centres de calcul et laboratoires lyonnais afin de favoriser les échanges entre les chercheurs de diverses disciplines. Nous avons proposé d'apporter notre expertise en matière d'algorithmique parallèle et de metacomputing ainsi que le portage d'une application sur la plate-forme obtenue grâce au financement de la région.

A.6.3 Relations industrielles

A.6.3.1 Sun Labs, Meylan

Je suis en train de mettre en place une collaboration avec la société Sun Microsystems et son entité de recherche Sun Labs autour de serveurs de calculs accessibles depuis Internet. Nous avons proposé un projet RNTL (Grid Application Service Provider ou GASP) avec eux et d'autres partenaires académiques.

A.6.3.2 Autres relations industrielles

En plus de mes collaborations avec des sociétés privées lors de projets industriels (voir chapitre A.5), j'ai travaillé plus précisement avec la société Simulog pour interfacer nos développements autour de TransTool avec l'outil Forsys-Partita.

A.7 Tâches collectives

A.7.1 Direction du projet ReMaP

Depuis Septembre 2000, j'ai pris la suite d'Yves Robert à la direction du projet CNRS–INRIA–ENS Lyon ReMaP [5]. Ce projet regroupe 10 permanents et 10 doctorants.

A.7.2 Organisation de conférences

En 1994, j'ai organisé avec S. Ubéda les Journées Industrielles du Parallélisme (JIP) à l'ENS. Ces journées avaient pour but de présenter aux industriels l'utilisation du parallélisme grâce à une série de cours, de présentations de "success stories" d'autres industriels et d'une journée de travaux pratiques sur machines (réseau de stations de travail). Nous avions également invité des vendeurs de machines pour qu'ils présentent leurs produits. Notre but était également de rencontrer les industriels et d'essayer d'évaluer leurs problèmes de performances afin d'orienter certaines de nos recherches pour leur résolution. En 1995, j'ai organisé la deuxième série de journées sur le même thème avec Jean Roman à l'ENSERB à Bordeaux.

J'ai participé avec Jack Dongarra à la mise en place de la série de conférences "European PVM Users Group Meeting" (appelée maintenant EuroPVM–MPI) dont la première édition a eu lieu en 1994 à Rome.

J'ai fait partie du comité d'organisation de RenPar'96, Rencontres Francophones du GDR PRS, qui ont eu lieu à Bordeaux en mai 96. J'étais entre autre responsable d'une journée applications parallèles.

[5] http://www.ens-lyon.fr/~desprez/ReMaP/

Je suis co-organisateur avec Jean-François Méhaut, Yves Robert et Eric Fleury du Workshop MSA [6] (*Metacomputing Systems and Applications*) qui a eu lieu en 2000 à Toronto en marge de la conférence ICPP'2000 (*International Conference on Parallel Processing*) et qui aura lieu cette année à Valence (Espagne) en septembre (durant ICPP'2001). L'objectif de ce workshop est de confronter les recherches sur les environnements exécutifs et les applications pour le metacomputing.

A.7.3 Comités d'édition

J'ai été co-éditeur (avec B. Tourancheau et L. Prylli) d'un numéro spécial de la revue Calculateurs Parallèles sur PVM (volume 8, numéro 2, date 1996).

A.7.4 Comités de programmes

J'ai fait partie du comité de programme d'EuroPar'96 et j'étais local-chair de la session *High-Performance Computing and Applications* qui a eu lieu à Lyon en août 1996. J'ai fait partie du comité de programme de la Conférence EuroPVM'96 qui a eu lieu début octobre à Munich. J'ai fait partie du comité de programme d'EuroPar'97 et j'étais co-chair de la session "Applications Industrielles". J'ai été invité à être local-chair de la conférence EuroPAR'99 qui a eu lieu à Toulouse en 1999 (workshop *Support Tools and Environments*). Je suis également chair dans la conférence EuroPAR'00 qui a eu lieu à Munich en 2000 (workshop *High-Performance Computing and Applications*) .

Je fais partie du comité de programme du journal "Parallel and Distributed Computing Practices" [7] dont l'éditeur en chef est Marcin Paprzycki.

A.7.5 Relecture pour des journaux et des conférences

J'ai été relecteur pour de nombreuses conférences (EuroPar, IPPS, ICPP, EuroPVM-MPI, PACT, HPCA, STACS, ...) et revues internationales (Parallel Computing, Parallel Processing Letters, Journal of Parallel and Distributed Computing, IEEE Transactions on Parallel and Distributed Computing, Theoretical Computer Science, Journal on Supercomputing, etc.).

A.7.6 Groupes de travail ParaMAp

En 1995, j'ai mis en place à Bordeaux le groupe de travail ParaMAp[8] dont le but était, comme pour PARAPPLI à Grenoble, de faire se rencontrer les experts du parallélisme et les chercheurs d'autres disciplines ayant besoin de puissance de calcul.

A mon retour en 96, j'ai mis en place le même groupe de travail à l'ENS. Durant l'année 96, nous avons eu 8 exposés autour de thèmes aussi divers que le placement d'antennes radios, les systèmes d'informations géographiques ou l'imagerie médicale. Toutefois, nous avons constaté la difficulté à faire collaborer des chercheurs de disciplines différentes. Nous avons cependant eu quelques succès comme par exemple le stage de Dominique Ponsard que j'ai co-encadré avec G. Vidal du laboratoire des Sciences de la Terre pour la parallélisation avec MPI d'un gros code de Modèle Numérique de Terrain. Ce logiciel a été déposé.

A.7.6.1 Suivi des thèses à l'UR Rhône-Alpes

A l'INRIA, j'assure le suivi des thèses effectuées au sein de l'UR. Mon travail consiste à contrôler les dossiers des thèses financées par l'UR, de donner un avis sur certains dossiers, à faire passer les thésards en comité des projets pour des rapports d'avancement, et enfin à effectuer un suivi régulier de toutes les thèses en cours dans l'UR.

[6] http://www.ens-lyon.fr/~desprez/FILES/RESEARCH/CONF/MSA
[7] http://orca.st.usm.edu/pdcp/
[8] Parallélisme Massif et Applications

A.7.7 Participation à diverses commissions

Depuis 1999, je fait partie des Commissions de Spécialistes de l'ENS Lyon et de Bordeaux.

Je fais partie du comité de sélection pour la mise en place de la grappe de 200 PCs qui sera installée cette année à l'UR Rhône-Alpes.

A.8 Enseignement

J'ai participé à des enseignements depuis le début de ma thèse. Disposant d'une bourse de l'INPG mais effectuant ma thèse à l'ENS Lyon, je n'ai pas pu être moniteur et j'ai donc été vacataire à l'ENS Lyon puis à l'IUT de Valence. Ensuite j'ai été recruté comme ATER à l'ENS Lyon à la fin de ma thèse. Ensuite, au retour de mon postdoctorat, j'ai été recruté comme Maître de Conférences à l'ENSEIRB de Bordeaux où j'ai enseigné dans la filière informatique. Suite à mon recrutement en tant que Chargé de Recherche à l'INRIA, j'ai souhaité continuer cette activité dans divers endroits (comme vacataire). J'ai pu ainsi enseigner en DEA à Lyon, de nouveau à l'ENSEIRB où j'ai été invité à donner un cours en 3ème année et en DESS réseau à Lyon I. J'ai également assuré des formations à Supelec, dans des tutoriaux en France et à l'étranger et dans diverses écoles.

Durant tous ces enseignements, j'ai été amené à suivre des stages d'étudiants de tous niveaux, à rédiger des polycopiés et à participer à l'activité pédagogique à travers des réunions et la mise en place d'options d'enseignement. J'ai également mis en place divers cours comme à l'ENSEIRB (système d'exploitation en 2ème année, parallélisme en 3ème année, équilibrage et régulation de charge en 3ème année) ou en DEA d'informatique fondamentale à Lyon.

A.8.1 Cours à l'Université, à l'ENS Lyon et en Ecoles d'ingénieur

2000-2001 CR1 INRIA

- J'ai renoncé cette année à mes cours pour m'occuper à plein temps du projet ReMaP.

1999-2000 CR1 INRIA

- Cours de Recherche Algorithmique Numérique Parallèle au DEA D'Informatique Fondamentale de Lyon (*24h de cours, 12 étudiants*)
- Cours d'équilibrage et régulation de charge en 3ème année ENSERB à Bordeaux (*20h de cours, 12 étudiants*)
- Cours et travaux pratiques de MPI en DESS Réseau à Lyon 1 (*2h de cours + 9h de TP pour 2 gpes, 40 étudiants*)

1998-1999 CR1 INRIA

- Cours et travaux pratique de MPI en DESS Réseau à Lyon 1 (*2h de cours + 9h de TPs pour 2 gpes, 40 étudiants*)
- Cours d'équilibrage et régulation de charge en 3ème année ENSERB à Bordeaux (*20h de cours, 8 étudiants*)

1997-1998 CR2-1 INRIA

- Projet d'algorithmique parallèle en Magistère 2ème année à l'ENS de Lyon.
- Cours de Recherche Algorithmique Numérique Parallèle au DEA D'Informatique Fondamentale de Lyon (*20h de cours, 8 étudiants*)

1996-1997 CR2 INRIA

- Projet d'algorithmique parallèle en Magistère 2ème année à l'ENS de Lyon.
- Cours sur les architectures parallèles dans le cadre du Cours Postgrade en Informatique Numérique de l'Ecole Polytechnique Fédérale de Lausanne (EPFL) (*8h de cours, 15 étudiants*).

• Cours de Recherche Algorithmique Numérique Parallèle au DEA D'Informatique Fondamentale de Lyon (*20h de cours, 10 étudiants*)

1995-1996 CR2 INRIA

• Projet d'algorithmique parallèle en Magistère 2ème année à l'ENS de Lyon.

1994-1995 MdC ENSERB : Co-responsable de la 3ème année option parallélisme,

• Parallélisme en 3ème année et DEA : étude et implémentation des algorithmes fondamentaux vus en cours (tris, recherches, algèbre linéaire, ...) avec PVM et HPF (*60h de TD/TP, 7 étudiants*),

• Systèmes d'exploitation en 2ème année : (Mise en place d'un nouveau cours, des TD et du projet) (*30 h de cours, 20h de TD, 48 étudiants*),

• Programmation en 1ère année autour du langage C (*30h de TD, 25 étudiants*),

• Organisation de diverses soutenances de projets pour les trois années,

• Cours de 3h à Besancon (DESS Parallélisme) sur les environnements pour le parallélisme

• Cours de 3h à Nice (3ème année ESSI) sur les bibliothèques de calcul parallèle.

1993-1994 ATER à L'ENS Lyon :

• Système et Logiciel de base en 1ère année de Magistère d'Informatique et Modélisation (Responsabilité du cours, des projets et examens) (*32h de cours, 15 étudiants*),

• Algorithmique et architectures parallèles en 2ème année de Magistère (Mise en place de nouveaux TD) (*32h de TD, 15 étudiants*).

1992-1993 Vacataire IUT Valence :

• Algorithmique et langage ADA au département ISI de l'IUT de Valence (*96 heures de TP, 70 étudiants*),

• DBASE IV et Teamwork (*10h de TP, 70 étudiants*),

1991-1992 • Permanence programmation pour les magistères 1ère année (ENS Lyon).

1990-1991 • Permanence programmation pour les magistères 1ère année (ENS Lyon).

• Système d'exploitation parallèle (Trollius) présenté aux étudiants de DEA d'informatique de l'ENS Lyon et à des chercheurs de l'école Centrale de Lyon (*12h de cours + TP, 20 auditeurs*).

• C avancé pour les moniteurs du CIES (*16h de cours + TP, 30 étudiants*).

A.8.2 Cours dispensés à des chercheurs et ingénieurs

2000-2001 • Cours à SUPELEC dans le cadre de la formation continue sur les environnements pour la parallélisation d'applications numériques (*3h de cours, 4 étudiants*).

• Tutorial sur HPF, MPI et OpenMP avec Franck Cappello et Fabien Coehlo à l'Ecole d'hiver iHPerf qui s'est tenue à Aussois en Décembre 2000.

1999-2000 • Cours à SUPELEC dans le cadre de la formation continue sur les environnements pour la parallélisation d'applications numériques (*3h de cours, 6 étudiants*).

1998-1999 • Cours et travaux pratiques sur MPI dans le cadre du Pôle Scientifique de Modélisation Numérique (PSMN) de Lyon (*20h de cours/TPs, 10 auditeurs*).

• Tutorial MPI–2–OpenMP avec Luc Giraud (CERFACS) lors de la conférence Euro-PAR'99 (Toulouse, Août 99) (*3h, 40 auditeurs*).

1997-1998 • Cours à SUPELEC dans le cadre de la formation continue sur les environnements pour la parallélisation d'applications numériques (*3h de cours, 8 étudiants*).

1996-1997 • Formation TTN ProHPC sur le parallélisme à l'ENS de Lyon

• Tutorial MPI lors d'EuroPAR'97 (Passau, Août 97) (*3h de cours, 50 participants*)

1995-1996	• Ecole du CNRS sur les outils du parallélisme à l'ENS de Lyon
1994-1995	• Formation MPI lors des Rencontres Francophones du Parallélisme RenPar'7 à Mons (*4h de cours*) (juin 1995),
	• Cours GRECO Informatique avec Bernard Tourancheau "La programmation d'applications scientifiques sur les ordinateurs parallèles à mémoire distribuée" (*6h de cours, 20 auditeurs*),
	• Cours sur les recouvrements calcul/communication dans le cadre d'une journée "Parallélisation automatique et les supports run-time" organisée par Marc Gengler à l'Ecole Polytechnique Fédérale de Lausanne,
	• Cours sur les environnements pour la parallélisation de code en milieu industriel lors des Journées Industrielles du Parallélisme (LIP ENS Lyon) (*1h30 de cours, 55 auditeurs*).
1993-1994	• Cours sur les architectures parallèles avec Michel Cosnard à l'Ecole d'Automne CAPA "Conception et Analyse des Algorithmes Parallèles" (*1h30 de cours, 50 auditeurs*),
	• Formation sur les environnements de programmation pour le parallélisme lors des Rencontres RenPar'6 à Lyon (*2h30 de cours, 30 auditeurs)*,
	• Cours et travaux pratiques sur les bibliothèques pour le "Cours Avancé de Calcul Scientifique Parallèle" du Centre Pour le Développement du Calcul Scientifique Parallèle de Lyon I (*2h de cours, 5h de TP, 30 auditeurs*).
1992-1993	• Cours sur les bibliothèques pour supercalculateurs dans le cadre de la formation des ingénieurs de recherche (ENS Lyon) (*2h de cours, 40 auditeurs*).
1991-1992	• Formations UNIX sur SUN aux chercheurs du laboratoire de mathématiques de l'ENS Lyon (*6h de cours + TP, 20 auditeurs*).

A.8.3 Responsabilités liées à l'enseignement

J'ai été co-responsable (avec Jean Roman) de la troisième année d'informatique à l'ENSEIRB pour l'année universitaire 94-95.

A.9 Publications

Livre et chapitres de livres

[CD94] M. Cosnard and F. Desprez. *Ecole d'Automne CAPA - Port d'Albret*, chapter Quelques Architectures de Nouvelles Machines. Hermes, April 1994.

[DF97] F. Desprez and P. Fraigniaud. *Ordinateurs et Calcul Parallèles*, chapter Les Bibliothèques de Communications, pages 293–312. Number 19 in Arago. OFTA, April 1997.

[GUD95] M. Gengler, S. Ubéda, and F. Desprez. *Initiation au Parallélisme : Concepts, Architectures et Algorithmes*. Manuels informatiques. Masson, 1995. 2-225-85014-3.

Revues internationales avec comité de lecture

[BBD+99] C. Barberet, L. Brunie, F. Desprez, G. Lebourgeois, R. Namyst, Y. Robert, S. Ubéda, and K. Van Heumen. Technology Transfer within the ProHPC TTN at ENS Lyon. *Future Generation Computer Systems*, 15 :309–321, 1999.

[BCC+97a] T. Brandes, S. Chaumette, M.-C. Counilh, A. Darte, J.C. Mignot, F. Desprez, and J. Roman. HPFIT : A Set of Integrated Tools for the Parallelization of Applications Using High Performance Fortran : Part I : HPFIT and the TransTOOL Environment. *Parallel Computing*, 23(1-2) :71–87, 1997.

[BCC+97b] T. Brandes, S. Chaumette, M.-C. Counilh, J.C. Mignot, F. Desprez, and J. Roman. HPFIT : A Set of Integrated Tools for the Parallelization of Applications Using High Performance Fortran : Part II : Data Structures Visualization and HPF Support for Irregular Data Structures with Hierarchical Scheme. *Parallel Computing*, 23(1-2) :89–105, 1997.

[CCCV$^+$01] E. Caron, S. Chaumette, S. Contassot-Vivier, F. Desprez, E. Fleury, C. Gomez, M. Goursat, E. Jeannot, D. Lazure, F. Lombard, J.M. Nicod, L. Philippe, M. Quinson, P. Ramet, J. Roman, F. Rubi, S. Steer, F. Suter, and G. Utard. Scilab to Scilab//, the OURAGAN Project. *Parallel Computing*, 2001. to appear.

[CD01] Frédérique Chaussumier and Frédéric Desprez. Communications Optimizations and Efficient Load-Balancing for a Volume Rendering Algorithm on a Cluster of PCs. *Parallel and Distributed Computing Practices*, 2001. To appear.

[DDMR96] A. Darte, F. Desprez, J.C. Mignot, and Y. Robert. TransTool : A Restructuring Tool for the Parallelization of Applications Using High Performance Fortran. *Journal of the Brazilian Computer Society*, 3(2) :5–15, November 1996.

[DDP$^+$98] F. Desprez, J.J. Dongarra, A. Petitet, C. Randriamaro, and Y. Robert. Scheduling Block-cyclic Array Redistribution. *IEEE Transaction on Parallel and Distributed Systems*, 9(2) :192–205, 1998.

[DDRR97] F. Desprez, J.J. Dongarra, F. Rastello, and Y. Robert. Determing the Idle Time of a Tiling : New Results. *Journal of Information Science and Engineering (Special Issue on Compiler Techniques for High-Performance Computing)*, 14(1) :167–190, March 1997.

[DDT95] F. Desprez, J.J. Dongarra, and B. Tourancheau. Performance Study of LU Factorization with Low Communication Overhead on Multiprocessors. *Parallel Processing Letters*, 5(2) :157–169, 1995.

[DG95] F. Desprez and M. Garbey. Numerical Simulation of a Combustion Problem on a Paragon Machine. *Parallel Computing*, 21 :495–508, 1995.

[DT94] F. Desprez and B. Tourancheau. LOCCS : Low Overhead Communication and Computation Subroutines. *Future Generation Computer Systems*, 10(2&3) :279–284, June 1994.

[DT95] F. Desprez and B. Tourancheau. Basic Routines for the Rank-2k Update : 2D Torus vs Reconfigurable Network. *Parallel Computing*, 21 :353–372, 1995.

Revues nationales avec comité de lecture

[CD94] M. Cosnard and F. Desprez. Quelques Architectures de Nouvelles Machines. *Calculateurs Parallèles*, 21 :29–58, March 1994.

[DGJP93] F. Desprez, C. Gavoille, B. Jargot, and M. Pourzandi. Tests des Performances des Communications de la Machine VOLVOX IS-860. *La Lettre du Transputer et des Calculateurs Parallèles*, 19 :11–35, October 1993.

[DT90] F. Desprez and B. Tourancheau. Modélisation des Performances des Communications sur le Tnode avec le Logical System Transputer Toolset. *La Lettre du Transputer et des Calculateurs Distribués*, 7 :65–72, September 1990.

[DU96] F. Desprez and S. Ubéda. Le Parallélisme dans l'Industrie : Rève ou Réalité ? *Technique et Science Informatique*, 15(5) :643–647, 1996.

Conférences internationales avec comité de lecture et publication des actes

[BBD$^+$98] C. Barberet, L. Brunie, F. Desprez, G. Lebourgeois, Y. Robert, S. Ubéda, and K. Van Heumen. Technology Transfer within the ProHPC TTN at ENS Lyon. In M. Bubak P. Sloot and B. Hertzberger, editors, *High-Performance Computing and Networking*, LNCS 1401, pages 3–12. Springer Verlag, 1998.

[BD96a] T. Brandes and F. Desprez. Implementing Pipelined Computation and Communication in an HPF Compiler. In *Europar'96 Parallel Processing*, volume 1123 of *Lecture Notes in Computer Science*, pages 459–462. Springer Verlag, August 1996.

[BD96b] T. Brandes and F. Desprez. Pipelining Data Parallel Computations in an HPF Compiler. In M. Gerndt, editor, *Sixth Workshop on Compilers for Parallel Computers*, volume 21, pages 73–89, Aachen, Germany, December 1996. Forschungszentrum Jülich.

[BDT93] C. Bonello, F. Desprez, and B. Tourancheau. Parallel BLAS and BLACS for Numerical Algorithms on a Reconfigurable Network. In S. Atkins and A.S. Wagner, editors, *NATUG6 - Transputer Research and Applications 6*, pages 21–38. IOS Press, 1993.

[CCD+94] C. Calvin, L. Colombet, F. Desprez, B. Jargot, P. Michallon, B. Tourancheau, and D. Trystram. Towards Mixed Computation - Communication in Scientific Libraries. In *CONPAR'94 - VAPP VI*, volume 854 of *Lecture Notes in Computer Science*, pages 605–615, Lintz, 1994. Springer Verlag.

[CD93] C. Calvin and F. Desprez. Minimizing Communication Overhead Using Pipelining for Multi-Dimensional FFT on Distributed Memory Machines. In D.J. Evans, H. Liddell J.R. Joubert, and D. Trystram, editors, *Parallel Computing'93*, pages 65–72. Elsevier Science Publishers B.V. (North-Holland), 1993.

[CDL99] Frédérique Chaussumier, Frédéric Desprez, and Michel Loi. Efficient Load-Balancing and Communication Overlap in Parallel Shear-Warp Algorithm on a Cluster of PCs. In P. Amestoy, P. Berger, M. Daydé, I. Duff, V. Frayssé, L. Giraud, and D. Ruiz, editors, *Proceedings of EuroPAR'99*, number 1685 in Lecture Notes in Computer Science, pages 570–577, Toulouse, 1999. Springer Verlag.

[CDP99] Frédérique Chaussumier, Frédéric Desprez, and Loic Prylli. Asynchronous Communications in MPI – the BIP/Myrinet Approach. In J. Dongarra, E. Luque, and Tomas Margalef., editors, *Proceedings of the EuroPVM/MPI'99 conference*, number 1697 in Lecture Notes in Computer Science, pages 485–492, Barcelona, Spain, September 1999. Springer Verlag.

[DD00] F. Desprez and S. Domas. Efficient Pipelining of Level 3 BLAS Routines. In *4th international meeting VECPAR 2000*, volume 3, pages 675–688, Porto, June 2000.

[DDD+98] F. Desprez, S. Domas, J. Dongarra, A. Petitet, C. Randriamaro, and Y. Robert. More on Scheduling Block-Cyclic Array Redistribution. In *Proc. of 4th Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR98)*, volume 1511 of *Lecture Notes in Computer Science*, pages 275–287. Springer-Verlag, Pittsburgh, PA, 1998.

[DDP+97] Frédéric Desprez, Jack Dongarra, Antoine Petitet, Cyril Randriamaro, and Yves Robert. Block-cyclic Array Redistribution on Networks of Workstations. In M. Bubak, J. Dongarra, and J. Wasniewski, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, LNCS 1332, pages 343–350. Springer Verlag, 1997.

[DDP+98] F. Desprez, J. Dongarra, A. Petitet, C. Randriamaro, and Y. Robert. Scheduling Block-Cyclic Array Redistribution. In E.H. D'Hollander, G.R. Joubert, F.J. Peters, and U. Trottenberg, editors, *Parallel Computing : Fundamentals, Applications and New Directions*, pages 227–234. North Holland, 1998.

[DDPMP96] V. Demian, F. Desprez, H. Paugam-Moisy, and M. Pourzandi. Parallel Implementation of RBF Neural Networks. In *Europar'96 Parallel Processing*, volume 1124 of *Lecture Notes in Computer Science*, pages 243–250. Springer Verlag, August 1996.

[DDRR97] F. Desprez, J. Dongarra, F. Rastello, and Y. Robert. Determining the Idle Time of a Tiling : New Results. In *Parallel Architectures and Compilation Techniques PACT'97*, pages 307–317, San Francisco, November 1997. IEEE Computer Society Press.

[DDT96] S. Domas, F. Desprez, and B. Tourancheau. Optimization of the ScaLAPACK LU Factorization Routine Using Communication/Computation Overlap. In *Europar'96 Parallel Processing*, volume 1124 of *Lecture Notes in Computer Science*, pages 3–10. Springer Verlag, August 1996.

[Des94] F. Desprez. A Library for Coarse Grain Macro-Pipelining in Distributed Memory Architectures. In *IFIP 10.3 Conference on Programming Environments for Massively Parallel Distributed Systems*, pages 365–371. Birkhaeuser Verlag AG, Basel, Switzerland, 1994.

[DFG+99a] F. Desprez, E. Fleury, C. Gomez, S. Steer, and S. Ubéda. Bringing Metacomputing to Scilab. In *Computer Aided Control System Design (CACSD 99)*, Hawaï, USA, August 1999. IEEE.

[DFG99b] F. Desprez, E. Fleury, and L. Grigori. Scilab// : User interactive application and high performances. In *Third World Multiconference on Systemics, Cybernetics and Informatics (SCI'99) and*

Fifth International Conference on Information Systems Analysis and Synthesis (ISAS'99), Orlando, USA, August 1999. IIIS.

[DFJ+00] F. Desprez, E. Fleury, E. Jeannot, J.-M. Nicod, and F. Suter. Computational servers in a metacomputing environment. In *Parallel Matrix Algorithms and Applications*, Neuchâtel, Switzerland, August 2000. SIAM. to appear.

[DFT94] F. Desprez, A. Ferreira, and B. Tourancheau. Efficient Communication Operations on Passive Optical Star Networks. In N. J. Davis, editor, *Massively Parallel Processing Using Optical Interconnections*, pages 52–58, Cancun, Mexico, April 1994. IEEE Computer Society Press.

[DG93] F. Desprez and M. Garbey. Parallel Computing of a Combustion Front. In *Parallel CFD'93 - Implementations and Results Using Parallel Computers*. Elsevier Science Publishers B.V., 1993.

[DP94] F. Desprez and M. Pourzandi. A Comparison of Three Matrix Product Algorithms on the Intel Paragon and Archipel Volvox Machines. In Bl. Sendov I.T.Dimov and P.S.Vassilevski, editors, *Advances in Numerical Methods and Applications NM&A - O(h^3)'94*, pages 234–244. World Scientific, 1994.

[DQS01] F. Desprez, M. Quinson, and F. Suter. Dynamic Performance Modeling for Network Enabled Solvers in a Metacomputing Environment. In *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2001)*, 2001. To appear.

[DRR96] F. Desprez, P. Ramet, and J. Roman. Optimal Grain Size Computation for Pipelined Algorithms. In *Europar'96 Parallel Processing*, volume 1123 of *Lecture Notes in Computer Science*, pages 165–172. Springer Verlag, August 1996.

[DS01] F. Desprez and F. Suter. Mixed Parallel Implementations of the Top Level of Strassen and Winograd Matrix Multiplication Algorithms. In *In proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS'01)*, San Francisco, April 2001. To appear.

[DT91] F. Desprez and B. Tourancheau. Reconfiguration versus Static Network in basic scientific routines. In K. Boianov, editor, *Workshop on Parallel and Distributed Processing WP&DP 91*, Sophia, Bulgarie, 1991. Elsevier Science Publishers B.V. (North-Holland).

[DT92] F. Desprez and B. Tourancheau. A Theoretical Study of Reconfigurability for Basic Communication Algorithms. In L. Bougé, M. Cosnard, Y. Robert, and D. Trystram, editors, *CONPAR 92 - VAPP V*, volume 634 of *Lecture Notes in Computer Science*, pages 817–818. Springer Verlag, 1992.

Conférences internationales avec comité de lecture sans publication des actes

[DT91a] F. Desprez and B. Tourancheau. Matrix Multiplication and Matrix Transpose : Fixed Topology vs. Switching Network (poster). In *EDMCC2 Munich*, 1991.

[DT91b] F. Desprez and B. Tourancheau. Reconfiguration versus Fixed Topology. In *European Parallel Computing Action - Bonn Workshop*. ESPRIT-PCA Workshop, 1991.

Conférences nationales avec comité de lecture sans publication des actes

[DDP94] F. Desprez, S. Domas, and M. Pourzandi. Comparaison entre 3 Routines de Calcul de Produit de Matrice sur Paragon et Volvox IS-860. In L. Bougé, editor, *Actes des 6es Rencontres Francophones du Parallélisme RenPar'6*, pages 109–118, 1994.

[Des94] F. Desprez. Recouvrements Calculs/Communications dans les Programmes Scientifiques. In L. Bougé, editor, *Actes des 6es Rencontres Francophones du Parallélisme RenPar'6*, pages 275–279, 1994.

[DF01] F. Desprez and F.Suter. Produit de matrices, Strassen et parallélisme mixte. In *Treizièmes Rencontres Francophones du Parallélisme*, Paris, La Villette, April 2001.

[DR00] F. Desprez and C. Randriamaro. Redistribution entre appels de routines ScaLAPACK. In *Actes des 12èmes Rencontres Francophones du Parallélisme RenPar'00*, pages 113–118, Besançon, France, June 2000.

[DZ97] F. Desprez and J. Zory. Performance des Macro-Pipelines dans les Programmes Data-Parallèles. In A. Schiper and D. Trystram, editors, *Actes des 9es Rencontres Francophones du Parallélisme Ren-Par'9*, pages 17–20, Lausanne, Switzerland, May 1997.

Rapports de recherches et techniques

[BCC+96] T. Brandes, S. Chaumette, M.-C. Counilh, A. Darte, J.C. Mignot, F. Desprez, and J. Roman. HP-FIT : A Set of Integrated Tools for the Parallelization of Applications Using High Performance Fortran. Technical Report RR-3059, INRIA, December 1996. also LIP ENS Lyon Tech. Rep. 96-28.

[CDL99] Frédérique Chaussumier, Frédéric Desprez, and Michel Loi. Efficient Load-balancing and Communication Overlap in Parallel Shear-Warp Algorithm on a Cluster of PCs. Technical Report RR1999-28, LIP ENS Lyon, 1999.

[CDP00] Frédérique Chaussumier, Frédéric Desprez, and Loic Prylli. Asynchronous Communications in MPI - the BIP/Myrinet Approach. Technical Report RR-3960, INRIA, 2000.

[DDD+98] Frédéric Desprez, Stéphane Domas, Jack Dongarra, Antoine Petitet, Cyril Randriamaro, and Yves Robert. More on Scheduling Block-Cyclic Array Redistribution. Technical Report 35-24, INRIA, October 1998. Proceedings of LCR'98, LNCS, Springer Verlag and also LIP ENS Lyon Tech. Rep. 98-17.

[DDP+97] F. Desprez, J.J. Dongarra, A. Petitet, C. Randriamaro, and Y. Robert. Scheduling Block-Cyclic Array Redistribution. Technical Report RR-3117, INRIA, February 1997. also UT CS Dept. Tech. Rep. CS-97-349, LAPACK Working note # 120 and CRPC-TR97714-S.

[DDPMP96] V. Demian, F. Desprez, H. Paugam-Moisy, and M. Pourzandi. Parallel Implementation of RBF Neural Networks. Technical Report 96-11, LIP ENS Lyon, 1996.

[DDRR97] F. Desprez, J.J. Dongarra, F. Rastello, and Y. Robert. Determining the Idle Time of a Tiling : New Results. Technical Report 32-72, INRIA, 1997. also LIP - ENS Lyon Tech. Report 97-35 and UT CS Dept. Tech. Rep. UT-CS-360.

[DDT94] F. Desprez, J.J. Dongarra, and B. Tourancheau. Performance Complexity of LU Factorization with Efficient Pipelining and Overlap on a Multiprocessor. Technical Report LAPACK Working Note 67, The University of Tennessee - Knoxville USA, 1994.

[DDT97] F. Desprez, S. Domas, and B. Tourancheau. Optimization of an LU Factorization Routine Using Communication/Computation Overlap. Technical Report RR-3094, INRIA, February 1997. also LIP - ENS Lyon Tech. Report 96-17.

[DFL93] F. Desprez, E. Fleury, and M. Loi. T9000 et C104, La Nouvelle Génération de Transputers. Technical Report 93-01, LIP - ENS Lyon, 1993.

[DFT93a] F. Desprez, A. Ferreira, and B. Tourancheau. Efficient Communication Operations in Reconfigurable Parallel Computers. Technical Report CS-93-209, The University of Tennessee - Knoxville USA, 1993.

[DFT93b] F. Desprez, P. Fraigniaud, and B. Tourancheau. Successive Broadcast on Hypercube. Technical Report CS-93-210, The University of Tennessee - Knoxville USA, 1993.

[DT92] F. Desprez and B. Tourancheau. LOCCS : Low Overhead Communication and Computation Subroutines. Technical Report 92-44, LIP - ENS Lyon, December 1992.

Mémoires

[Des90] F. Desprez. Algèbre Linéaire sur TNode. Master's thesis, Ecole Normale Supérieure de Lyon, June 1990. Laboratoire LIP.

[Des94] F. Desprez. *Procédures de Base pour le Calcul Scientifique sur Machines Parallèles à Mémoire Distribuée*. PhD thesis, Institut National Polytechnique de Grenoble, January 1994. LIP ENS-Lyon.

Chapitre **B**

Optimization of a LU Factorization Routine Using Communication/Computation Overlap

Conference : EuroPAR

Date : 1996

Authors :

F. Desprez, S. Domas and B. Tourancheau
LIP, CNRS URA 1398, INRIA Rhône-Alpes
Ecole Normale Supérieure de Lyon
46, Allée d'Italie
69364 LYON cedex 07
(desprez,sdomas,btouranc)@lip.ens-lyon.fr

Abstract :

This paper presents some works on the *LU* factorization from the ScaLAPACK library. First, a complexity analysis is given. It allows to compute the optimal block size for the block scattered distribution used in ScaLAPACK *LU*. It also gives the communication phases that are interesting to overlap. Second, two optimizations based on computations/communications overlap are given with experimental results on Intel Paragon system and IBM SP2 system.

Keywords :

Parallel numerical libraries, overlap of commications, pipelining, ScaLAPACK.

Thanks :

This work has been supported by the INRIA Rhône-Alpes and the EUREKA-EUROTOPS project. We thank the CDCSP of the University Claude Bernard Lyon I, France, for providing the access to their Intel Paragon machine.

B.1 Introduction

The LU factorization is the kernel of many applications. Thus, the importance of optimizing this routine has not to be proven because of the increasing demand of applications dealing with large matrices. Its efficient parallel implementation can bring real improvements in the execution speed of the whole application. The speed-up depends greatly on the kind of supercomputer chosen. Vector machines have very high performances, but a prohibitive cost. Distributed memory machines seem to be a good balance between performances and cost.

Portability is one of the key issue of computer programming. Many libraries have been designed to ensure portability and performances across multiple architectures. The BLAS [4, 5, 10] and LAPACK [6] are available on many platforms, provided by computers vendors. LU factorization was released in the LAPACK package, using levels 1, 2 and 3 BLAS. ScaLAPACK [1] is the parallel version of a subset of LAPACK. ScaLAPACK has been designed to ensure portability, performances and ease of use across many parallel machines. Matrices are distributed in a block scattered way. Parallelism is hidden in a parallel version of the BLAS called PBLAS [25]. Communications between processors on a virtual grid are done using the BLACS package [7].

The aim of this paper is to show that improvements can be obtained in the existing ScaLAPACK LU factorization routine by the use of overlap techniques.

The first section presents the parallel block LU decomposition. The second section presents a recent bibliography on the parallel LU factorization. It contains a selection of papers that reflects the different solutions for this problem. The third section presents a complexity study of the ScaLAPACK LU routines, and two possible optimizations based on communication / computation overlap. The main goal of the complexity analysis is to provide the optimal block size for the block scattered decomposition, but it also gives information about the communication phases that may be overlapped.

B.2 Parallel Block LU Decomposition

In ScaLAPACK, the parallel LU factorization uses a block scattered decomposition of matrix A on a $P \times Q$ processors grid. The $M \times N$ matrix is divided in square blocks ($r \times r$). Thus, each processor owns a local matrix with $\left\lceil \frac{M}{P \times r} \right\rceil \times \left\lceil \frac{N}{Q \times r} \right\rceil$ blocks. This distribution is really suitable for the block LU decomposition and it provides a good load balancing between processors.

0	1	2	0	1	2	0	1	2	0	1	2
3	4	5	3	4	5	3	4	5	3	4	5
0	1	2	0	1	2	0	1	2	0	1	2
3	4	5	3	4	5	3	4	5	3	4	5
0	1	2	0	1	2	0	1	2	0	1	2
3	4	5	3	4	5	3	4	5	3	4	5
0	1	2	0	1	2	0	1	2	0	1	2
3	4	5	3	4	5	3	4	5	3	4	5
0	1	2	0	1	2	0	1	2	0	1	2
3	4	5	3	4	5	3	4	5	3	4	5
0	1	2	0	1	2	0	1	2	0	1	2
3	4	5	3	4	5	3	4	5	3	4	5

FIG. B.1 – Block scattered decomposition of a 12r × 12r matrix on a 2 × 3 grid.

Figure B.1 shows how to distribute the $r \times r$ blocks of a $12r \times 12r$ matrix on a 2×3 grid in a block scattered way. Squares marked with a number represent a single block of the global matrix. For example, all grey blocks belong to processor 0.

The block LU decomposition consists in three phases, repeated as many times as there are block columns to be factorized in the global matrix : instead of working on a single column of the global matrix A at a time, r columns are factored at each step. And for convenience, the local L and U matrices are stored in place of the matrix to be decomposed.

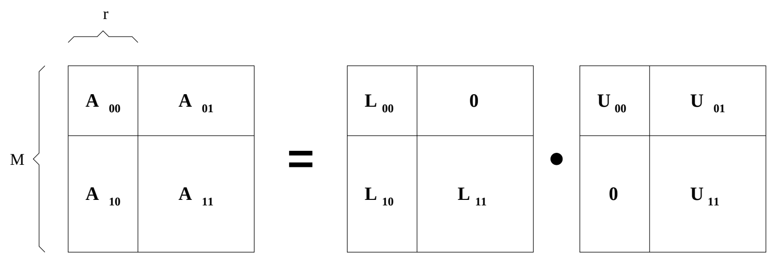


FIG. B.2 – First step of the block LU factorization.

```
$pcol = 0$
$prow = 0$
for $k = 0$ to $\min(M_b, N_b) - 1$ by step $r$ do
    for $i = 0$ to $r - 1$ do                                   }
       if ($my\_col = pcol$) then                               }
          find pivot and its position                           }
       end if                                                   }
       broadcast the two values to all processors               }
       exchange pivot rows                                      } phase 1
       if ($my\_col = pcol$) then                               }
         divide under-diag. elts. of col. i by pivot            }
         update col. i+1 to r-1 /* _GER */                      }
       end if                                                   }
    end for                                                     }

    if ($my\_row = prow$) then                                  }
       broadcast $L_{00}$ to all processors of the $prow$ row   } phase 2
       solve $L_{00}.U_{01} = A_{01}$ /* _TRSM */               }
    end if                                                      }

    broadcast $L_{10}$ on processors rows                       }
    broadcast $U_{01}$ on processor columns                     } phase 3
    update $A_{11} \leftarrow A_{11} - L_{10}.U_{01}$ /* _GEMM */ }

    $pcol = (pcol + 1) mod Q$
    $prow = (prow + 1) mod P$
end for
```

FIG. B.3 – Parallel block LU factorization using a block scattered data distribution.

According to Figure B.2, the three steps are the followings :

– in order to obtain (L_{00}, L_{10}) and U_{00}, a simple Gaussian elimination is computed on $\binom{A_{00}}{A_{10}}$.

$$L_{00}.U_{00} = A_{00} \tag{B.1}$$

$$L_{10}.U_{00} = A_{10} \tag{B.2}$$

– the U_{01} block is obtained by a triangular solve (see equation B.3).

$$L_{00}.U_{01} = A_{01} \tag{B.3}$$

– $L_{11}.U_{11}$ is given by equation B.4. A matrix product is needed ($A_{11} - L_{10}.U_{01}$).

$$L_{10}.U_{01} + L_{11}.U_{11} = A_{11} \tag{B.4}$$

The three steps are recursively computed on $L_{11}.U_{11}$ to obtain L_{11} and U_{11}.

The general parallel algorithm is given in Figure B.3. `_GER` is a level 2 BLAS, `_TRSM` is the level 3 BLAS triangular solve routine and `_GEMM` is the general matrix product routine.

The ScaLAPACK routines `PDGETRF` and `PDGETF2` execute the block *LU* factorization on a matrix distributed in a block scattered way. Figures B.4 and B.5 represent the execution schemes of these two routines :
– `PDGETF2` performs the factorization of a block column to compute L_{00},L_{10}, and U_{00} (phase 1 of the general algorithm in Figure B.3).
– `PDGETRF` calls `PDGETF2`, then updates the remaining blocks of the matrix to compute $L_{11}.U_{11}$, U_{01} (phase 2 and 3 of the general algorithm in Figure B.3).

B.3 Related Work

Various methods have been proposed to improve the parallel *LU* factorization. The corresponding papers present experiments which are sometimes corroborated by complexity studies.

In [8], a row oriented method is presented with an efficient pivot selection. It uses a reverse spanning tree broadcast to choose the real pivot among all the local pivots, owned by different processors. In ScaLAPACK, it is now achieved by the BLACS operation `_GMAX2D`. Furthermore, in [2], a load balancing strategy for the choice of the pivot is added to the row oriented method with a low cost row interchange.

Meanwhile, a row distribution supposes that the real pivot is often chosen on a different processor than the owner of the current row. Thus, there are some communications to exchange and to broadcast the pivot row.

In [11], a column-scattered distribution is chosen with the well-known pipelined ring algorithm. A complete performance estimation is provided for distributed memory parallel machines, and compared to experimental results.

In [3], a column-scattered distribution is chosen and asynchronous communications are used to overlap computations (columns updates). A theoretical model is given for the complexity on a complete network and a ring topology.

The two last methods are quite efficient because choosing the pivot on a single processor (instead of all, like in [2, 8]) is very fast and exchanging rows occurs in local memory. Therefore, it uses a fine grain parallelism.

ScaLAPACK *LU* is a block *LU* decomposition with a block scattered distribution of the data. In [25], a complexity study and different assumptions on the choice of the block size and the grid size are proposed. It appears that a rectangular grid with few rows of processors is optimal (for the reasons exposed upper). Thus, it can use the best of the precedent methods but with coarse grain computations.

The block scattered distribution used in the ScaLAPACK *LU* factorization seems to be the most interesting solution for distributed memory computers, as far as the block and the grid sizes are well chosen.

Consequently, this paper presents an optimized version of *LU* factorization with a block scattered distribution and an overlap between communications and computations. It also gives a method to compute an optimal block size without an “enquiry routine” (as previewed in [25]).

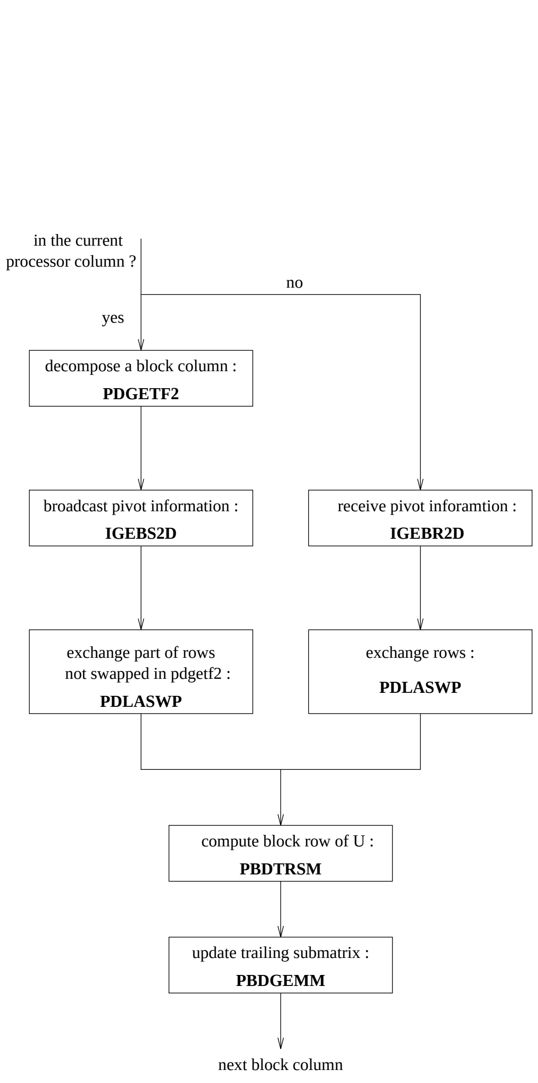

in the current processor column ?
no
yes
decompose a block column : PDGETF2
broadcast pivot information : IGEBS2D
receive pivot inforamtion : IGEBR2D
exchange part of rows not swapped in pdgetf2 : PDLASWP
exchange rows : PDLASWP
compute block row of U : PBDTRSM
update trailing submatrix : PBDGEMM
next block column

FIG. B.4 – Execution scheme of the PDGETRF routine.

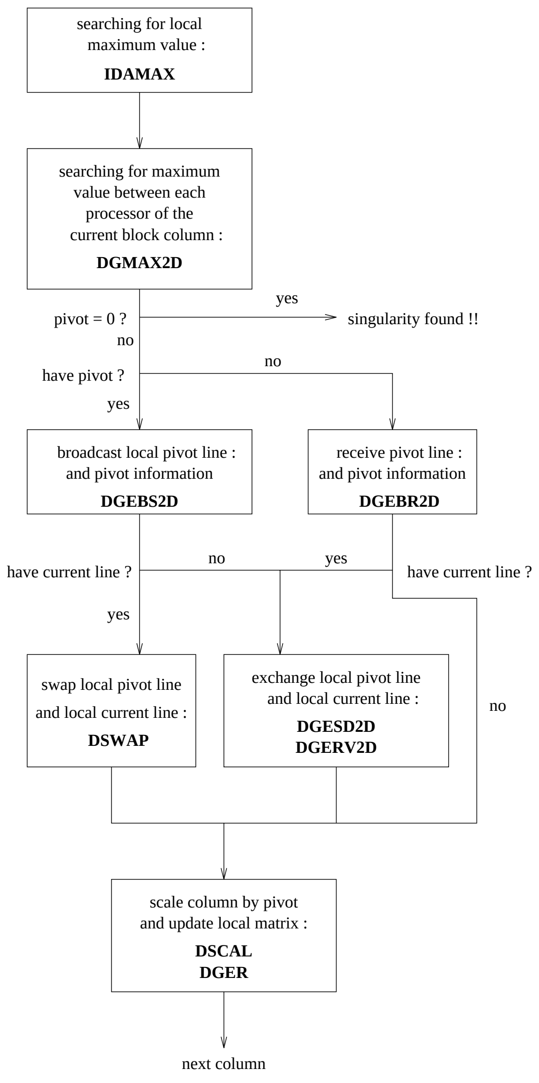

searching for local maximum value : IDAMAX
searching for maximum value between each processor of the current block column : DGMAX2D
pivot = 0 ?
yes
singularity found !!
no
have pivot ?
no
yes
broadcast local pivot line : and pivot information DGEBS2D
receive pivot line : and pivot information DGEBR2D
have current line ?
no
yes
have current line ?
yes
swap local pivot line and local current line : DSWAP
exchange local pivot line and local current line : DGESD2D DGERV2D
no
scale column by pivot and update local matrix : DSCAL DGER
next column

FIG. B.5 – Execution scheme of the PDGETF2 routine.

B.4 Analysis and Optimization of the ScaLAPACK LU Factorization

B.4.1 Complexity Analysis

A prediction of the execution time is important to confirm the experimental results. The value of the parameters which influence the execution time can be deduced from the model. Furthermore, it gives the execution time of the communication phases. Such predictions are very interesting to choose what communications to overlap.

In the ScaLAPACK LU factorization, performances depend greatly on the block size used for the block scattered decomposition. Thus, it is interesting to compute the optimal block size to avoid a number of tests. It is also interesting to compute automatically the best data distribution (for parallel compilers, for example).

Figure B.6 shows the impact of the block size on the performance of the ScaLAPACK LU factorization. On the Intel Paragon at the University of Lyon, a size of 64 is less efficient than 8, which is the optimal value.

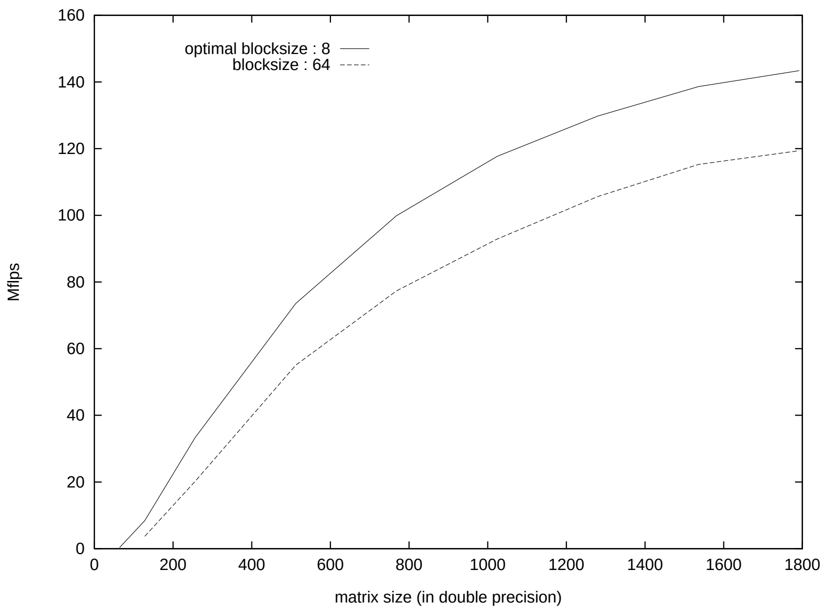


FIG. B.6 – Performance (In Mflops) of LU factorization on 4 processors with two different block sizes on Intel Paragon.

B.4.1.1 The Model

In this section, the theoretical optimal block size S_b is obtained by an interpolation, based on experimental measurements (for a given supercomputer) of the LU subroutines. For all subroutines function of S_b (see below), the execution time is expressed literally, and then derived to find the optimal S_b.

The complexities are given below with S_b as the block size, P_r and P_c as the number of processor rows and columns, and M as the matrix size. The subroutines names are the BLAS or LAPACK names expressed in Figures B.4 and B.5. We assume that the global communications (`DGEBS2D, DGMAX2D, ...`) are proportional to the basic communication time $t(L) = \beta + L\tau$. For a given global communication *op*, $t_{op}(L) = f(\beta + L\tau)$, where f is determined by the communication scheme that achieve the global communication. For example, a tree broadcast on a row of processors is achieved in $t_{tree} = \lceil log_2 P_c \rceil.[\beta + L\tau]$, and a ring broadcast in $t_{oring} = (P_c - 1).[\beta + L\tau]$. The f function used to compute the total execution time is given in the expressions below.

B.4.1.2 Subroutines used in `PDGETF2`

– **IDAMAX :** is executed S_b times for each `PDGETF2` call. There are $\lceil \frac{M}{S_b} \rceil$ `PDGETF2` calls during the factorization (Figures B.1 and B.4). The execution time of a single `IDAMAX` call is linear.

$$T_{idamax} = \sum_{i=1}^{\lceil \frac{M}{S_b} \rceil} S_b.[a_{idamax}(S_b \times \lceil \frac{i}{P_r} \rceil) + b_{idamax}]$$

– **DGMAX2D :** is executed S_b times for each `PDGETF2` call. The execution time of a single `DGMAX2D` call is linear but proportional to the number of processors

$$T_{dgmax} = \lceil \frac{M}{S_b} \rceil.S_b.f[a_{sendd} + b_{sendd}]^{1} \text{ with } f = \lceil log_2 P_r \rceil$$

– **DCOPY :** is executed $2 \times S_b$ times for each `PDGETF2` call, with a fixed data size S_b. It is first used to

[1] $a_{sendd} = \tau$ and $b_{sendd} = \beta$. These parameters are for double precision buffer send. For integers, the a_{sendi} notation is used.

copy the current pivot segment to broadcast it, and secondly to copy the real pivot segment into local matrix. The execution time of a single DCOPY call is linear.

$$T_{dcopy} = 2.\lceil \frac{M}{S_b} \rceil .S_b.[a_{dcopy}S_b + b_{dcopy}]$$

– **DGEBS2D,DGEBR2D :** is executed S_b times for each PDGETF2 call, with a fixed data size S_b.

$$T_{broadcastd} = \lceil \frac{M}{S_b} \rceil .S_b.f[a_{sendd}S_b + b_{sendd}] \text{ with } f = \lceil log_2 P_r \rceil$$

– **DGESD2D,DGERV2D :** is executed S_b times for each PDGETF2 call, with a fixed data size S_b.

$$T_{send} = \lceil \frac{M}{S_b} \rceil .S_b.f[a_{sendd}S_b + b_{sendd}]$$

– **DSCAL :** is executed S_b times for each PDGETF2 call. The execution time of a single DSCAL call is linear.

$$T_{dscal} = \sum_{i=1}^{\lceil \frac{M}{S_b} \rceil} S_b.[a_{dscal}(S_b \times \lceil \frac{i}{P_r} \rceil) + b_{dscal}]$$

– **DGER :** is executed S_b times for each PDGETF2 call, with a decreasing data size. The execution time of a single DGER call is quadratic. It is a level 2 BLAS which computes $A \leftarrow \alpha x.y^t + A$ with $A_{(m \times n)}$: $t_{dger}(m,n) = (a_{dger}m + b_{dger}).n + (c_{dger}m + d_{dger})$

$$T_{dger} = \sum_{i=1}^{\lceil \frac{M}{S_b} \rceil} \sum_{j=1}^{S_b - 1} [(a_{dger}(S_b \times \lceil \frac{i}{P_r} \rceil) + b_{dger}).j + (c_{dger}(S_b \times \lceil \frac{i}{P_r} \rceil) + d_{dger})]$$

B.4.1.3 Subroutines used in PDGETRF

– **IGEBS2D,IGEBR2D :** is executed $\lceil \frac{M}{S_b} \rceil$ time, with a fixed data size S_b.
$T_{broadcasti} = \lceil \frac{M}{S_b} \rceil .f[a_{sendi}S_b + b_{sendi}]$ with $f = \lceil log_2 P_c \rceil$

– **PDLASWP :** is executed $\lceil \frac{M}{S_b} \rceil$ time, but exchanges S_b rows at one time. We assume that there is a neglectful number of times for which the pivot is at the right place. Furthermore, the probability to find the pivot on the same processor that owns the current row is considered uniform. In the expression below, N_{mem} is the number of times where the rows are swapped within local memory, and N_{com} when the rows are swapped by communications. These parameters can be computed precisely for a given M, S_b, and P_r.

$$T_{swap} = N_{mem}.[a_{dswap}(S_b \times \left\lceil \frac{\lceil \frac{M}{S_b} \rceil}{P_r} \right\rceil) + b_{swap}] + N_{com}.[(2.a_{dcopy} + a_{sendd})(S_b \times \left\lceil \frac{\lceil \frac{M}{S_b} \rceil}{P_r} \right\rceil) + (2.b_{dcopy} + b_{sendd})]$$

– **PBDTRSM :** is executed $\lceil \frac{M}{S_b} \rceil - 1$ times, with a variable data size, multiple of S_b. It must be decomposed in two parts :

1. **broadcasting :** the factored column can be broadcast in one chunk in order to distribute the current L_{00} block (see figure B.2) for DTRSM computation, and L_{10} for later DGEMM computation. Before broadcasting, the whole column is copied in a working buffer.

$$T_{broadcastd} = \sum_{i=2}^{\lceil \frac{M}{S_b} \rceil} [(f(a_{sendd}) + a_{dcopy})(S_b^2 \times \lceil \frac{i}{P_r} \rceil) + (f(b_{sendd}) + b_{dcopy})] \text{ with } f = \lceil log_2 P_c \rceil$$

2. DTRSM **:** The execution time of a single DTRSM call is cubic. It is a level 3 BLAS which computes $B \leftarrow \alpha.A^{-1}.B$ with $A_{(m \times m)}$ and $B_{(m \times n)}$:

$$t_{dtrsm}(m,n) = (a_{dtrsm}n + b_{dtrsm}).m^2 + (c_{dtrsm}n + d_{dtrsm}).m + (e_{dtrsm}n + f_{dtrsm})$$

$$T_{DTRSM} = \sum_{i=1}^{\lceil \frac{M}{S_b} \rceil - 1} [(a_{dtrsm}.(S_b \times \left\lceil \frac{i}{P_c} \right\rceil) + b_{dtrsm}).S_b^2 + (c_{dtrsm}.(S_b \times \left\lceil \frac{i}{P_c} \right\rceil) + d_{dtrsm}).S_b + (e_{dtrsm}.(S_b \times \left\lceil \frac{i}{P_c} \right\rceil) + f_{dtrsm})]$$

– **PBDGEMM :** executed $\lceil \frac{M}{S_b} \rceil - 1$ time, with a variable data size, multiple of S_b. It must be decomposed in two parts :

1. **broadcasting :** the last factored row (see `DTRSM`) must be broadcast in order to achieve `DGEMM` computation. Before broadcasting, the whole row is copied in a working buffer.

$$T_{broadcastd} = \sum_{i=1}^{\lceil \frac{M}{S_b} \rceil - 1} [(f(a_{sendd}) + a_{dcopy})(S_b^2 \times \lceil \tfrac{i}{P_c} \rceil) + (f(b_{sendd}) + b_{dcopy})] \text{ with } f = \lceil log_2 P_r \rceil$$

2. `DGEMM` **:** The execution time of a single `DGEMM` call is cubic. It is a level 3 BLAS which computes $C \leftarrow \alpha.A.B + \beta.C$ with $A_{(m \times k)}$, $B_{(k \times n)}$ and $C_{(m \times n)}$:

$$t_{dgemm}(m,n,k) = (a_{dgemm} m.n + b_{dgemm} m + c_{dgemm} n + d).k + (e_{dgemm} m.n + f_{dgemm} m + g_{dgemm} n + h)$$

$$T_{DGEMM} = \sum_{i=1}^{\lceil \frac{M}{S_b} \rceil - 1} [(a_{dgemm}.S_b + b_{dgemm})(S_b \times \left\lceil \frac{i}{P_r} \right\rceil).(S_b \times \left\lceil \frac{i}{P_c} \right\rceil) + (c_{dgemm}.S_b + d_{dgemm})(S_b \times \left\lceil \frac{i}{P_r} \right\rceil) + (e_{dgemm}.S_b + f_{dgemm})(S_b \times \left\lceil \frac{i}{P_c} \right\rceil) + (g_{dgemm}.S_b + h_{dgemm})]$$

In order to compute the optimal block size, all the precedent equations have been derivated over S_b. The total complexity optimum is obtain when the sum of the derivatives equals zero. Hence, the optimal block size is given by the resolution of a third degree equation. We computed the three roots for different grid sizes and matrix sizes. Only one solution was positive each time. Results are given in Tables B.1 and B.3. It gives the optimal block size as a function of different grid sizes and matrix sizes (1000 signifies a 1000×1000 matrix in double precision).

B.4.1.4 Results on an Intel Paragon

The Intel Paragon is a distributed memory machine based on i860 processors connected via a 2D grid. Each processor reaches about 50 Mflops at peak performance.

Five important points have to be noticed :

– Theoretical optimal block sizes are close to the experimental ones. They range between 7.2 and 10.1 and actual tests give an optimal block size of 8 or 10 on a 4×4 grid[2].

– The theoretical optimal block size is a function of the matrix size, and it raises up to a top value around 10. But this size is a real. A "good" block size is the nearest integer value from the theoretical one.

[2] A size of 16 was found on previous tests. It has been done on the same machine but with an older version of the operating system. Results depend greatly on the machine and its system.

grid size	matrix size							
	1000	2000	3000	4000	5000	6000	7000	8000
4×4	8.31	9.28	9.64	9.83	9.94	10.0	10.1	10.1
8×8	7.38	8.38	8.79	9.03	9.17	9.27	9.35	9.41
16×16	7.26	8.0	8.38	8.61	8.77	8.88	8.96	9.03

TAB. B.1 – Optimal block size on a Paragon system.

type	matrix size			
	1000	2000	3000	4000
experimental	8 - 9	8	8	10
theoretical	8.31	9.28	9.64	9.83

TAB. B.2 – Comparison between theoretical and experimental block sizes on a Paragon system.

– The optimal block size hardly depends on the number of processors. Asymptotic values are 9 for 256 processors and 10 for 16. Again, experimental results confirm this point.
– According to experimental results, the grid shape has no real influence on the optimal block size, though it greatly influences the execution time [25]. A rectangular grid with few rows of processors works faster than a square grid. There are less communications since pivoting is achieved more often in local memory.
– Results are identical if only the subroutines `DGER`, `DTRSM`, and `DGEMM` are used for block size computation (they represent 95% of total computation time).

B.4.1.5 Results on an IBM SP2

The IBM SP2 is also a distributed memory machine based on RS6000 processors connected via a multistage network. Each processor reaches 265 Mflops at peak performance.

grid size	matrix size							
	256	512	768	1024	1280	1536	1792	2048
2×2	25.5	29.8	32.8	35.5	37.9	40.2	42.3	44.4
3×3	24.1	27.5	29.7	31.6	33.3	35.0	36.6	38.0
4×4	23.5	26.4	28.2	29.7	31.0	32.3	33.5	34.7

TAB. B.3 – Optimal block size on an IBM SP2.

Three important points have to be noticed :
– The experimental block sizes are quite difficult to obtain because the execution time can vary between two executions. The results in Table B.4 are the average optimal block sizes given by the executions.
– Some experimental values are quite far from the theoretical optimum. But in most cases, this optimum gives a performance very close from the best experimental result. This is particularly true for grids smaller than 2×4. For example, on a 2×2 grid with a matrix size of 1024, the theoretical value is 35 and the experimental is 20. The performances are respectively 280 and 270 Mflops which represents a gap of 3.7%.
– On the SP2, the optimal block size is very dependent from the grid size and the matrix size. This justifies the interest of the complexity analysis.

type	**matrix size**							
	256	512	768	1024	1280	1536	1792	2048
3×3 exp.	27	33	30	35	36	35	38	38
3×3 theo.	24.1	27.5	29.7	31.6	33.3	35.0	36.6	38.0
1×9 exp.	27	25	24	25	26	27	26	27
1×9 theo.	20.3	21.7	22.7	23.7	24.6	25.4	26.3	27.1

TAB. B.4 – Comparison between theoretical and experimental block sizes on an IBM SP2.

B.4.2 Optimizations

The ScaLAPACK version of LU has been implemented in order to be scalable : each subroutine call is a BLAS or BLACS call. These two libraries are already fully optimized for a wide range of computers. Furthermore, the minimal number of operations to achieve a LU factorization is well-known ($\frac{2}{3}n^3 + 2n^2$). Thus, only communication phases can be optimized since the computation time is fixed. Asynchronous messages are used to overlap communications with computations.

B.4.2.1 Broadcast Overlap

By looking closely at the algorithm, we can see that processors are often waiting results from other processors ! During the block column decomposition, only a processor column is working. And only one processor row is working during the triangular solve. This brings us to the first optimization solution :

"Instead of broadcasting (L_{00}, L_{10}) panel before the triangular solve (_TRSM), do it at the same time."

This means that general synchronous BLACS broadcast routines (_GEBS2D and _GEBR2D) are used on processors that do not compute _TRSM, and asynchronous communications are used to send (L_{00}, L_{10}) during _TRSM on processors that compute it.

Therefore, the single block L_{00} must be broadcast on the current processor row to perform _TRSM. But it takes less time than broadcasting (L_{00}, L_{10}).

This solution seems interesting since we gain at each step i of factorization the broadcast time of $P_b - i - 1$ blocks, compared to the original version. But, unfortunately, that gain represents only 1 to 2 percents of total factorization time on the Paragon system. This is due to the number of times this broadcast is done (only $P_b - 1$ times). Moreover, this deceiving result can be predicted by the complexity analysis. The sum of broadcasting time for all steps never exceeds 2 percents of total execution time.

B.4.2.2 Rows Pivoting Overlap

It appears that a good speed-up cannot be obtained unless a communication phase is overlapped most of the time. Thus, it is interesting to overlap the pivoting time since it is executed for each row of the matrix :

"Instead of broadcasting pivot informations and then exchanging rows after the (A_{00}, A_{10}) decomposition (PDGETF2), do it at the same time."

This means that we can use the DSCAL time of the processors column which decompose (A_{00}, A_{10}), to exchange current and real pivot rows using asynchronous communications. Then we use the DGER time to send asynchronously the pivot informations to the next processor in the processors row. Thus, as soon as this processor receives pivot informations, it can exchange the rows for pivoting and send the information to the next processor in the row. And so on, until the last processor on the pseudo-ring receives its data.

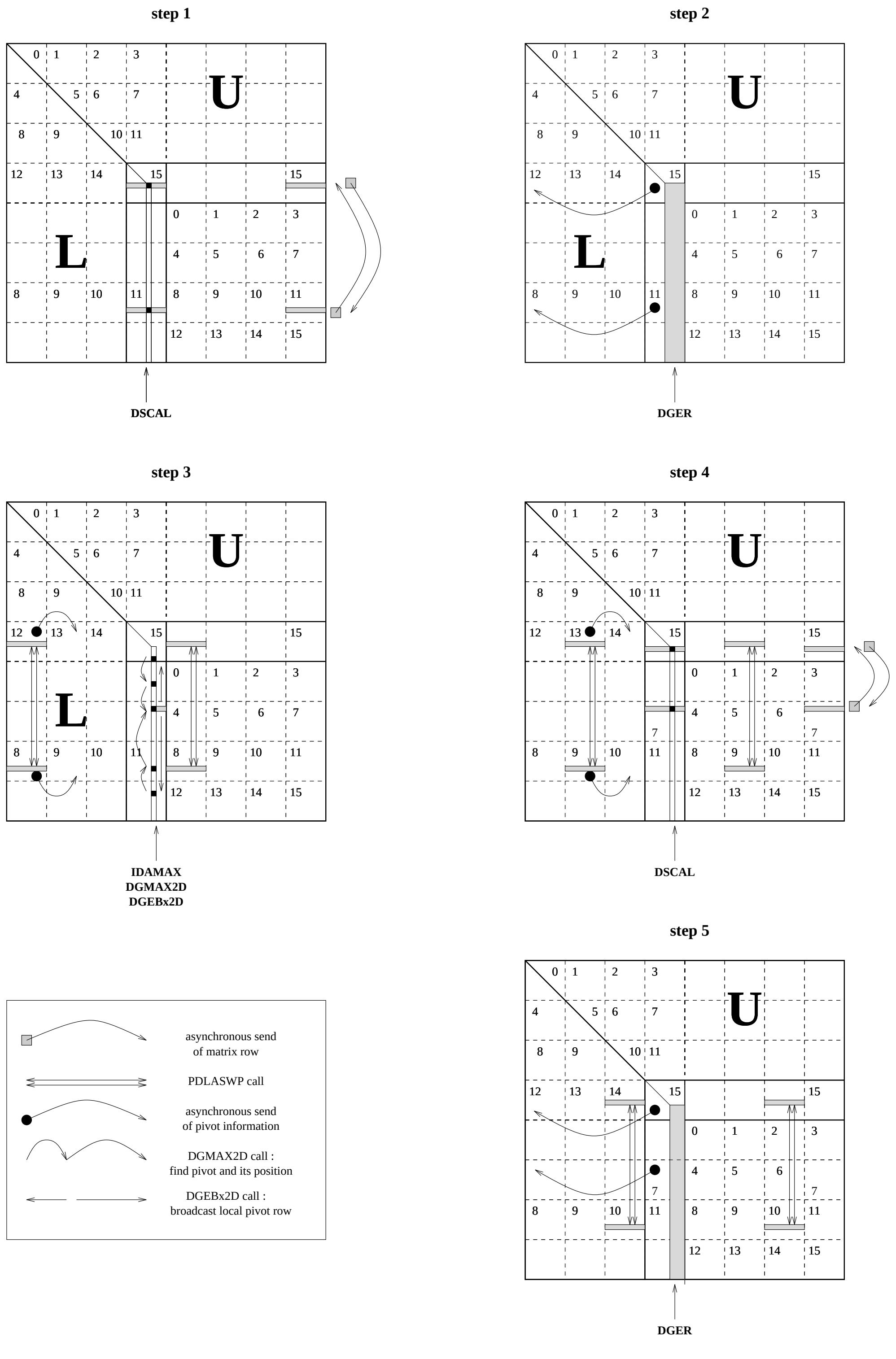

step 1
U
L
DSCAL
step 2
DGER
step 3
IDAMAX
DGMAX2D
DGEBx2D
step 4
DSCAL
step 5
DGER
asynchronous send of matrix row
PDLASWP call
asynchronous send of pivot information
DGMAX2D call : find pivot and its position
DGEBx2D call : broadcast local pivot row

FIG. B.7 – One iteration of optimized PDGETF2 routine.

During this step, the block column decomposition continues on the processor column.

Figure B.7 represents the different steps of these operations for an 64×64 matrix distributed on a 4×4 grid using a 8×8 block size. We explain this example in the following :

- **step** 1 : This is the k^{th} iteration of `PDGETF2`. The real pivot row has been found on processor 11. Then, processor 15, that owns the current pivot row k, divides the current pivot by the real pivot and asynchronously send the whole k^{th} row to processor 11. After, it proceeds with `DSCAL` and after, waits for the completion of the asynchronous send and receives the real pivot row.
  Identically, processor 11 asynchronously sends the whole pivot row to processor 15 and computes the `DSCAL` at the same time. After completion, it receives the current pivot row.
  In the best case, processors 15 and 11 do not need to wait for send completion since the communication is already over when `DSCAL` ends. In fact, a wait state appears with a very large matrix, when their size reaches memory size limits. In this case, the `DSCAL` time does not completely overlap the communication time, since `DSCAL` domain size decreases each step.
  Other processors in the pivot processor column just execute the `DSCAL` routine.
- **step** 2 : `DSCAL` and pivoting is done. Now, the local sub-matrix must be updated with `DGER`. Processors 15 and 11 use this time to send the pivot information to their right neighbor on a pseudo-ring made from the processors row. In the figure B.7, a pseudo-ring is $(12, 13, 14, 15)$, and the right neighbor of 15 is 12.
  Processors 12 and 8 are just waiting for pivot information using a blocking receive.
- **step** 3 : processors 12 and 8 have just received the pivot information. They can now exchange the k^{th} row and the pivot row, and send the pivot informations to their right neighbor. Meanwhile, processors in the pivot column continue the decomposition, finding the pivot and its position, broadcasting the local pivot row, ...
- **step** 4 : as in step 1, `DSCAL` is overlapped by an asynchronous exchange of current and real pivot row. But now, processors 13 and 9 are working, exchanging rows, instead of waiting for the completion of `PDGETF2` to work.
- **step** 5 : as in step 2, but with two more processors working.

The impact of this optimization is clearly shown on Figure B.8. On the top of the figure, we can see that processors 1 and 3 are completely idle during the non-optimized `PDGETF2` execution. After, there is a communication phase to exchange the pivot row. On the bottom of the figure, processors 1 and 3 are no more idle : they are receiving the pivot information and exchanging rows during the optimized `PDGETF2` execution. Thus, there are no more communication post-phase. They are overlapped by `PDGETF2`.

B.4.3 Experimental Results

All experimental results have been produced on two Paragon systems and a SP2 system, with various grid size. The two Paragon (ORNL, Tennessee USA, and Lyon, France) have not the same operating system and it appeared that it leads to different experimental results.There are four different grid sizes : 4×4, 6×5 (Lyon), 8×8, and 16×16 (ORNL). The 16 processors SP@ system is located at the LaBRI (Bordeaux, France).

All the results have been successfully run with the test driver included in the ScaLAPACK LU factorization package in order to test the correctness of the optimizations. This driver completes the LU factorization on a matrix A and then solves the system $LUx = B$. It also provides different possibilities to vary the execution parameters such as block size, number of processors, number of columns of the right-hand-side (RHS) matrix $B, \ldots$

The results on Paragon systems are shown on Figures B.9, B.10, B.11 and B.12. The IBM SP2 results are on Figures B.14 and B.13.

Figure B.9 shows a comparison between optimized and non-optimized results for 16×16 grid. The optimized version grows faster (in Mflops) than the non-optimized, before becoming almost parallel. Indeed, the optimization works better for small matrix sizes ($< 375 \times P_c$ with P_c, the number of processor columns) because the pivoting communications are all executed during `PDGETF2`. After this limit, the pivoting time becomes more important and communications cannot be completely overlapped by the `PDGETF2` execution time. This is confirmed by the complexity analysis. The Figure B.10 shows the predicted and the experimental gain for a 4×4 grid. For each matrix size, the predicted gain is given by the minimum of the total time

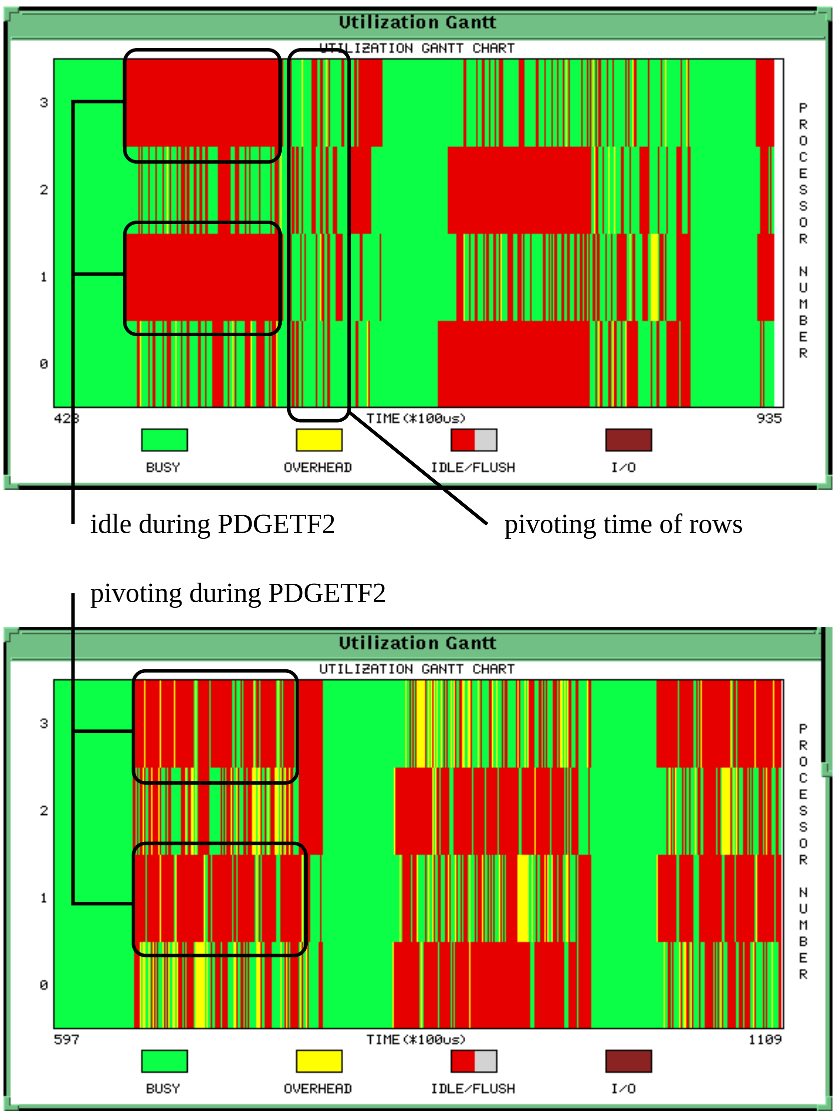


FIG. B.8 – Comparison of the Gantt diagram for optimized and non-optimized versions.

of `PDGETF2` and the total communication time for pivoting.

Figure B.11 shows the gain in percents over non-optimized version for 8×8 and 16×16 grids. It can reach 15% for small matrix sizes, and stay above 4% for the largest matrix size that can be allocated. This figure confirms that the gain progressively decreases for a matrix size greater than $375 \times P_c$.

Figure B.12 is the same as figure B.11 but the x-coordinates are expressed as a function of the sub-matrix size for a single processor. It shows that the gain does not depend on the total matrix size but only on the size of the sub-matrix that one processor owns. This is due to the block scattered distribution which provides a good load balancing.

Figure B.13 shows a comparison between the optimized and non-optimized results for a 3×3 grid on a IBM SP2. For small matrices, the optimized version works much faster than the basic version because the pivoting is completely overlapped by the `PDGETF2` execution. For large matrices, there is no gain since the pivoting is hardly overlapped. The gain can even be negative as shown on Figure B.14. In fact, asynchronous communications are often slower than blocking communications, especially for large messages.

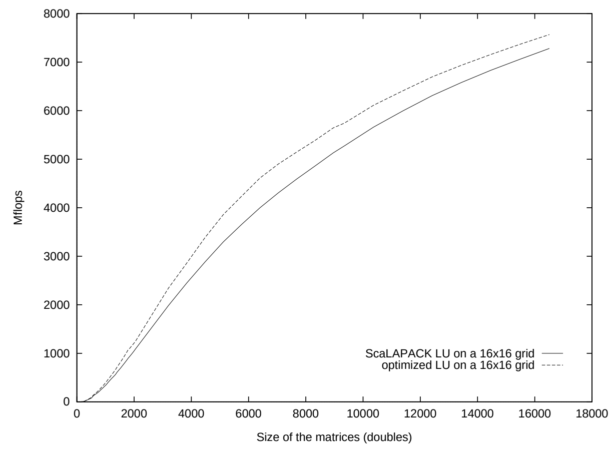


FIG. B.9 – Experimental results on a 16 × 16 grid, on a Paragon.

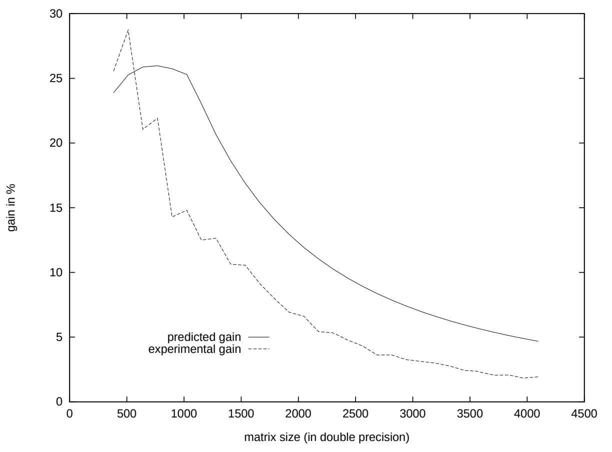


FIG. B.10 – Predicted and experimental gain on a 4 × 4 grid, on a Paragon.

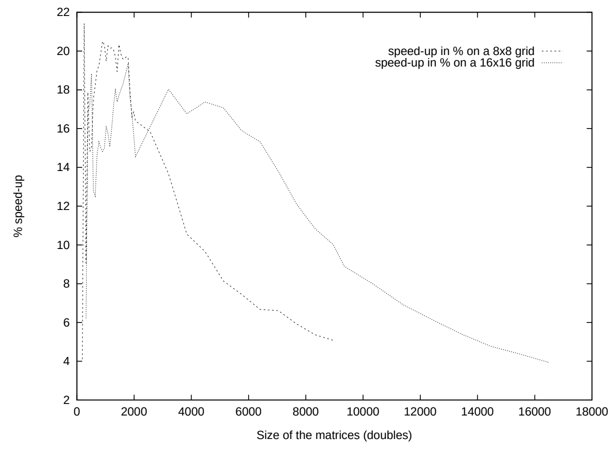


FIG. B.11 – Gain on the 8 × 8 and 16 × 16 grids, on a Paragon

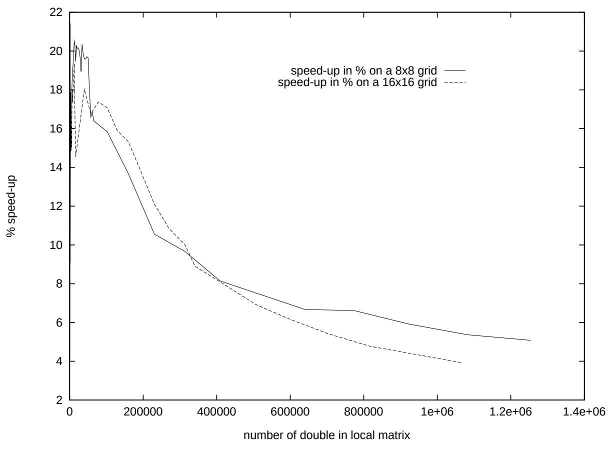


FIG. B.12 – Gain on the 8 × 8 and 16 × 16 grids, as a function of the matrix size on a single processor, on a Paragon.

Consequently, the asynchronous pivoting is often slower than in the basic version and it leads to worse performances when it is hardly overlapped.

B.5 Conclusion and future work

After a description of the *LU* algorithm in ScaLAPACK, a complete analysis of complexity has been presented. This theoretical model allows the computation of the optimal block size for the block scattered decomposition. Thus, it is possible to have the best performance with a simple pre-computation.

The second part has presented two optimizations based on communication / computation overlap. In the two cases, our aim was to “hide” the time of some big communication phases. Furthermore, it allows to reduce the idle time of some processors that are waiting results from other to continue the execution.

All experimentations have been done on Paragon systems but the methods presented in this paper are general. Thus, they can be applied to any supercomputer.

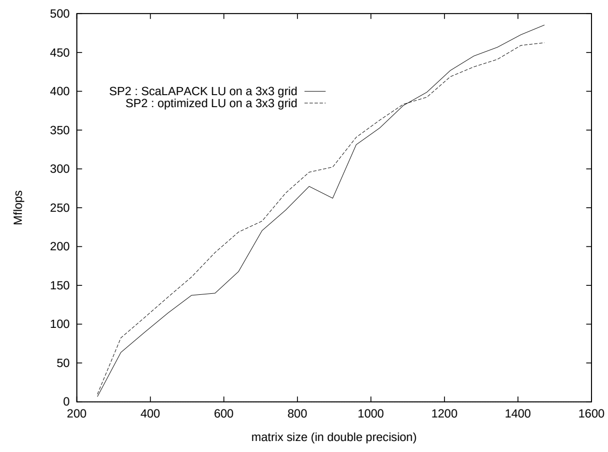


FIG. B.13 – Experimental results on a 3×3 grid, on an IBM SP2.

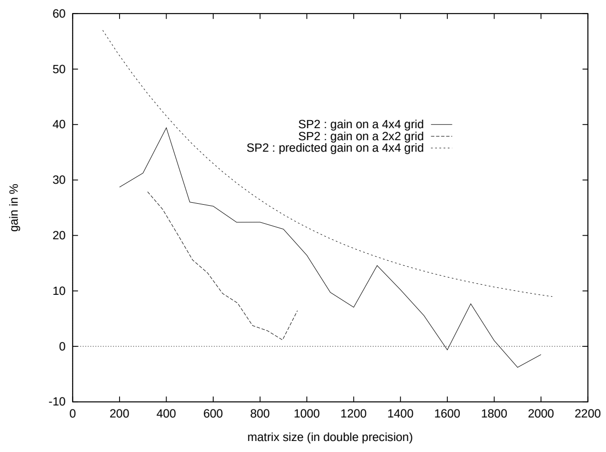


FIG. B.14 – Gain on a 2×2 and 4×4 grid, on an IBM SP2.

Meanwhile, some hints can be given as future work :

– All the updates have been done using Paragon system calls syntax because asynchronous BLACS do not exist. Consequently, our code is not fully portable. But the modifications are simple enough to be rewritten on any supercomputer.It will soon be ported on a CRAY T3D.
– The gain decreases as the matrix size grows : the `PDGETF2` time becomes not big enough to overlap the communications due to pivoting. Thus, another routine could be used to overlap more communications (like `DGEMM`).
– The optimal block size computation is also interesting as input for parallel compilers because it gives the best data distribution for a given matrix size, and number of processors. It would be interesting to include such a computation in an HPF compiler.

B.6 References

[1] E. Anderson, A. Benzoni, J. Dongarra, S. Moulton, S. Ostrouchov, B. Tourancheau, and R. Van de geijn. Lapack for distributed memory architecture progress report. In *Fifth SIAM Conference on Parallel Processing for Scientific Computing,*, 1991.

[2] E. Chu and A. George. Gaussian Elimination with Partial Pivoting and Load Balancing on a Multiprocessor. *Parallel Computing*, 5 :65–74, 1987.

[3] F. Desprez, J.J. Dongarra, and B. Tourancheau. Performance Complexity of LU Factorization with Efficient Pipelining and Overlap on a Multiprocessor. *Parallel Processing Letters*, 5-II, 1995.

[4] J.J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling. A Set of Level 3 Basic Linear Algebra Subprograms. *ACM Transaction on Mathematical Software*, 16(1) :1–17, 1990.

[5] J.J. Dongarra, J. Du Croz, S. Hammarling, and R. Hanson. An Extended Set of Fortran Basic Linear Algebra Subroutines. *ACM Transaction on Mathematical Software*, 14(1) :1–17, March 1988.

[6] J.J. Dongarra, R. Van De Geijn, and D.W. Walker. A Look at Dense Linear Algebra Libraries. Technical Report ORNL/TM-12126, Oak Ridge National Laboratory, July 1992.

[7] J.J. Dongarra, R.A. Van De Geijn, and R.C. Whaley. A User's Guide to the BLACS, March 1993.

[8] G.A. Geist and M.T. Heath. Matrix Factorization on a Hypercube Multiprocessor. Technical report, Oak Ridge National Laboratory, 1985.

[9] J.Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley. LAPACK Working Note : ScaLAPACK : A Portable Linear Algebra Library for Distributed

Memory Computers - Design Issues and Performances. Technical Report 95, Department of Computer Science - University of Tennessee, 1995.

[10] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh. Basic Linear Algebra Subprograms for Fortran Usage. *ACM Transaction on Mathematical Software*, 5 :308–323, 1979.

[11] B.V. Purushotham, A. Basu, P.S. Kumar, and L.M. Patnaik. Performance Estimation of LU Factorisation on Message Passing Multiprocessors. *Parallel Processing Letters*, 2(1) :51–60, 1992.

Chapitre **C**

Optimal Grain Size Computation for Pipelined Algorithms

Conference : EuroPAR

Date : 1996

Authors :

Frédéric Desprez
LIP (URA CNRS 1398)
INRIA Rhône-Alpes et ENS Lyon
69 364 Lyon Cedex, France
desprez@lip.ens-lyon.fr

Pierre Ramet and Jean Roman
LaBRI (URA CNRS 1304)
ENSERB et Université Bordeaux I
33 405 Talence Cedex, France
(ramet,roman)@labri.u-bordeaux.fr

Abstract :

In this paper, we present a method for overlapping communications on parallel computers for pipelined algorithms. We first introduce a general theoretical model which leads to a generic computation scheme for the optimal packet size. Then, we use the OPIUM [1] library, which provides an easy-to-use and efficient way to compute, in the general case, this optimal packet size, on the column LU factorization ; the implementation and performance measures are made on an Intel Paragon.

Keywords :

Communications overlap, pipelined algorithms, optimal packet size computation.

Thanks :

This work is supported by the INRIA Rhône-Alpes, the french GDR-PRS and the Eureka-Eurotops project.

[1]Optimal Packet sIze compUtation Methods.

C.1 Introduction

Parallel distributed memory machines improve performances and memory capacity but their use adds an overhead due to the communications. To obtain programs that perform and scale well, this overhead must be hidden. Several solutions exist. The choice of a good data distribution is the first step that can be done to lower the number and the size of communications. Depending of the dependences within the code, asynchronous communications can be used to overlap computations and communications. The call to the communication routine (send or receive) will be put as soon as possible in the code. A wait routine will then be used to check for the completion of the communication. Unfortunately, this is not always legal due to the dependences between computations and communications. Pipeline schemes are also sometimes found like the one on Figure C.1 (A). There is a sequentiality within the code, and the total execution time is higher than the sequential one because of the overhead of the communications. One first solution is to start the communications as soon as possible, i.e. as soon as one processor has computed one data; this is called a fine-grain pipeline ((B) on Figure C.1). This solution adds an higher overhead because of the communication startup time. A trade-off has to be found which minimizes the execution time. This is a coarse grain pipeline ((C) on Figure C.1).

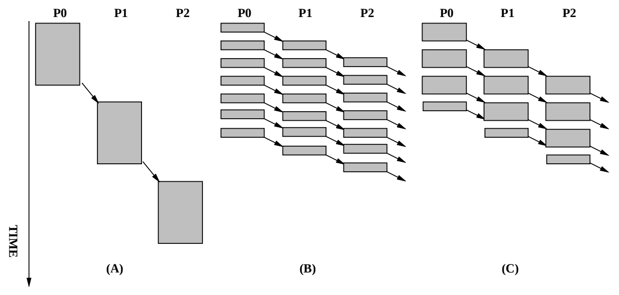


FIG. C.1 – Overlap of communications using a pipeline method.

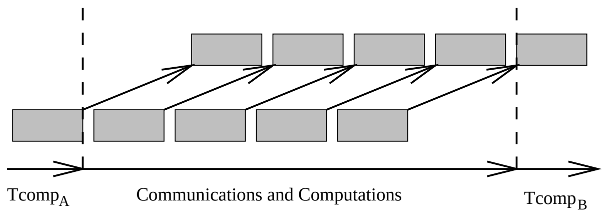


FIG. C.2 – Communication/Computation Overlap.

The linear case is an easy one [4]. This is no longer the case as soon as the complexity of the computations and communications is not a linear function of the packet size. The aim of this paper is to present a library which can be used to improve pipeline computations in the general case. This library, called OPIUM (Optimal Packet sIze compUtation Methods) will be used in another library which implements coarse grain pipelines, the LOCCS [2] [19, 6].

Some previous studies exist, especially around the compilation of data-parallel languages like High Performance Fortran. The pipelining of computation is a classical optimization of such codes [35, 1]. The linear case from which this study has started is described in [4]. The results are also used within the LOCCS library. In [8], the optimal size of packets is computed for general DOACROSS loop nests. The optimizations of the Fortran D compiler are presented in [35]. Here again, only the simple cases that arise in the compilation of simple loop nests are given. The author admits that a more accurate evaluation of the optimal size of packets should be given to get the best performances. Finally in [34], the case of a loop nest with two loops is presented on a network of workstations, using a computation of the optimal size done at run-time, which allows some kind of load-balancing.

In the general case presented in this paper, the computation of the optimal grain size is no longer trivial. It can require complicated methods and a very accurate model to obtain correct results; a library is necessary to compute the optimal size without a knowledge of the methods, to do run-time optimizations, to avoid to re-compute the size if it is not necessary, and to include this kind of optimization in a data-parallel compiler.

The rest of the paper is organized as follows : the section 2 introduces the problem formulation and the theoretical model. In section 3, we describe experiments on the Intel Paragon, based on the OPIUM library,

[2]Low Overhead Communication and Computation Subroutines.

including performance results and analysis. Then we conclude with remarks on the benefits of this study and on our future work.

C.2 Computation/Communication Overlap

C.2.1 Problem formulation and theoretical model

In this section, we first describe a theoretical model for the overlap of communications by computations. It will be used as the reference model until the end of the paper. Then, the way to resolve this model is presented and illustrated with some examples. Finally, a method to avoid the re-computation of the optimal packet sizes as far as possible is described.

Let L be the Amount of Data to be Pipelined (ADP). We split the data to be sent in μ packets of size ν (see Figure C.2). Moreover, in the following, ν_{opt} denotes the Optimal Packet Size (OPS).

DÉFINITION 1 *The "forward computation time" denotes the time spent in the forward computation on a packet of size ν. It can be defined by*

$$Tcomp_A(\nu, L) = R_A(L)\ f_A(\nu)\ \tau_A \qquad \textit{where}$$

- $R_A(L)$ *is the forward complexity constants that may depend on the ADP,*
- $f_A(\nu)$ *is the complexity function of the forward computation* implementation,
- *and τ_A denotes the elementary forward computation time.*

DÉFINITION 2 *In the same way, the "backward computation time" denotes the time spent in the backward computation on a packet of size ν with*

$$Tcomp_B(\nu, L) = R_B(L)\ f_B(\nu)\ \tau_B.$$

DÉFINITION 3 *In the following, the notation $\theta(f_A(\nu))/\theta(f_B(\nu))$ will be used to represent a pipeline algorithm with a "forward" complexity growing as $\theta(f_A(\nu))$ and a "backward" complexity growing as $\theta(f_B(\nu))$.*

Example 1

Let us study the column LU factorization. A column cyclic distribution is used to balance the computation load all over the execution [9]. With this distribution, we can extract three phases for each factorization step

- the "local scale" : the processor holding the pivot column of the current step computes its contribution, and sends it to the other processors,
- the "local update" : the same processor updates its local columns,
- the "remote update" : the other processors receive the contribution, and update their columns.

We clearly have a pipeline scheme. The "local change" must be in the "forward computation" and the "remote update" is the "backward computation". The "local update" is put in the "forward computation" to well balance the "forward" and the "backward" computations such as we can hope to obtain an optimal overlap gain. For this implementation, the complexities of the "forward" and "backward" computations are $\theta(\Delta\ \nu)$, where Δ represents the number of local columns to update. It is important to notice that these complexities depend on the implementation of the pipeline and on the data distribution. The column LU factorization is typically a $\theta(\nu)/\theta(\nu^2)$ algorithm, whereas we have a $\theta(\nu)/\theta(\nu)$ pipelined implementation.

DÉFINITION 4 $Tcomm(\nu)$ *denotes the time spent to send a packet of size ν.*

In general, the expression of $Tcomm(\nu)$ depends on the communication scheme (broadcast, all-to-all, ...), and on the computer architecture. In the following we will consider a linear communication model $Tcomm(\nu) = \beta + \nu\tau$ where β is the startup time and τ the per-word transfer time.

PROPOSITION 1
The OPS computation problem can be rewritten into the minimization problem of

$$Tcomp_A(\nu, L) + \frac{L}{\nu} Tcomm(\nu) + Tcomp_B(\nu, L) \tag{C.1}$$

$$\text{under the constraints} \begin{cases} Tcomp_A(\nu, L) < Tcomm(\nu) \\ Tcomp_B(\nu, L) < Tcomm(\nu) \\ 1 \leq \nu \leq L \end{cases} \tag{C.2}$$

Proof See [5]. □ If the computation time is always greater than the communication time then we split the

ADP with the minimal packet size and thus $\nu_{opt} = 1$.

We now want to solve the reference model. There are three steps :

1. First we compute the extremum ν_{min} of (C.1),
2. then we look for the validity domain of the constraints ; we compute ν_{bal} the highest packet size which verify the constraints (C.2),
3. and finally we deduce the optimal size ν_{opt}.

• We focus in the following on polynomial complexity functions. If the constraints are true then ν_{min} is the solution of $\frac{\partial}{\partial \nu} Ttotal(\nu, L) = 0$.

LEMMA 1 *Consider that* $\begin{cases} Tcomp_A(\nu) = R_A\, \nu^{\alpha_A}\, \tau_A \\ Tcomp_B(\nu) = R_B\, \nu^{\alpha_B}\, \tau_B \quad \textit{with}\ \alpha_A \geq 1,\ \alpha_B \geq 1. \\ Tcomm(\nu) = \beta + \nu\, \tau \end{cases}$
ν_{min} *is the unique solution of :* $\alpha_A\, R_A\, \nu^{\alpha_A - 1}\, \tau_A + \alpha_B\, R_B\, \nu^{\alpha_B - 1}\, \tau_B - \frac{L\beta}{\nu^2} = 0$.

Proof See [5]. □

Remark 1 For the most usual complexity functions that are over-linear and communication types, we can verify that the second order derivate of $Ttotal(\nu)$ is always positive on the validity domain of ν. For all these cases, the problem admits one and only one solution.

• Now that we have the minimum of equation (C.1), we must take care of the constraints. An other approach to optimize the overlap of computations by communications consists in finding, when possible, the packet size such that the communication time for one packet balances exactly the computation time for this packet [7]. As the "forward" and "backward" computations play a symmetrical part in the minimization of equation (C.1), we can assume that $Tcomp_A > Tcomp_B$. Then the constraint balance is given by

$$Tcomp_A(\nu, L) = Tcomm(\nu). \tag{C.3}$$

Let ν_{bal} be the solution of this equation. Often, ν_{bal} is not the OPS ; for a packet size upper than ν_{bal} we can rewrite the expression of $Ttotal(\nu, L)$ as

$$Ttotal(\nu, L) = Tcomp_A(\nu, L) + \frac{L}{\nu} Tcomp_A(\nu, L) + Tcomp_B(\nu, L). \tag{C.4}$$

If $Tcomp_A(\nu, L)$ is over-linear on ν then $Ttotal(\nu, L)$ increases with ν and ν_{bal} minimizes $Ttotal(\nu, L)$ for $\nu \geq \nu_{bal}$. Otherwise, we have to find ν_{bal} which minimizes (C.4).

• Finally, the following proposition leads to the computation of ν_{opt}.

PROPOSITION 2
If $1 \leq \nu_{bal} \leq L$ then two cases are possible :

– if $\nu_{bal} \leq \nu_{min}$ then the constraint occurs and $\nu_{opt} = \nu_{bal}$,

– otherwise the constraint does not occur and $\nu_{opt} = \nu_{min}$.

The end of this section is dedicated to some examples.

LEMMA 2 *Consider that* $\left\{\begin{array}{l} Tcomp_A(\nu) = R_A\, \nu\, \tau_A \\ Tcomp_B(\nu) = R_B\, \nu\, \tau_B \\ Tcomm(\nu) = \beta + \nu\, \tau \end{array}\right.$

Two cases are possible :

- *if* $R_A\, \tau_A\ - \tau \leq 0$ *then* $\nu_{opt} = \sqrt{\frac{L\,\beta}{R_A\,\tau_A + R_B\,\tau_B}}$.
- *if* $R_A\, \tau_A\ - \tau > 0$ *then*
  - *if* $L \leq \frac{\beta\,(R_A\,\tau_A + R_B\,\tau_B)}{(R_A\,\tau_A - \tau)^2}$ *then* $\nu_{opt} = \sqrt{\frac{L\,\beta}{R_A\,\tau_A + R_B\,\tau_B}}$,
  - *else* $\nu_{opt} = \frac{\beta}{R_A\,\tau_A - \tau}$.

Proof See [5]. □

Remark 2 In the case of the column LU factorization, R_A and R_B depend on the ADP ($R_A = R_B = \frac{L}{NPROC}$ within one, where $NPROC$ is the number of processors). So $\nu_{min} = \sqrt{\frac{NPROC\,\beta}{\tau_A + \tau_B}}$ and the optimal packet size is independent of the ADP.

LEMMA 3 *Consider that* $\left\{\begin{array}{l} Tcomp_A(\nu) = R_A\, \nu^2\, \tau_A \\ Tcomp_B(\nu) = R_B\, \nu\, \tau_B \\ Tcomm(\nu) = \beta + \nu\, \tau \end{array}\right.$

The validity domain for the OPS is $\nu \in \left[0, \frac{\tau + \sqrt{\tau^2 + 4\,R_A\,\tau_A\,\beta}}{2\,R_A\,\tau_A}\right]$ *and there is one and only one real positive solution for* ν_{min} *among the 3 Jordan solutions :*

$$\left\{\begin{array}{l} \nu_1 = \sqrt[3]{\frac{-q+\delta}{2}} + \sqrt[3]{\frac{-q-\delta}{2}} - \frac{A}{3} \\ \nu_2 = \sqrt[3]{\frac{-q+\delta}{2}} \cdot j + \sqrt[3]{\frac{-q-\delta}{2}} \cdot j^2 - \frac{A}{3} \\ \nu_3 = \sqrt[3]{\frac{-q+\delta}{2}} \cdot j^2 + \sqrt[3]{\frac{-q-\delta}{2}} \cdot j - \frac{A}{3} \end{array}\right. \quad with \left\{\begin{array}{l} p = -\frac{A^2}{3} \\ q = \frac{2\,A^3}{27} - B \\ \delta^2 = \frac{4\,p^3 + 27\,q^2}{27} \end{array}\right.$$

Proof See [5] □

LEMMA 4 *Consider that* $\left\{\begin{array}{l} Tcomp_A(\nu) = R_A\, \ln(\nu)\, \tau_A \\ Tcomp_B(\nu) = R_B\, \ln(\nu)\, \tau_B \\ Tcomm(\nu) = \beta + \nu\, \tau \end{array}\right.$

then $\nu_{min} = \frac{L\,\beta}{R_A\,\tau_A + R_B\,\tau_B}$.

The proof is straightforward. However, in order to compute ν_{bal}, we have to solve $R_A\, ln(\nu)\, \tau_A = \beta + \nu\, \tau$ and we cannot explicitly extract the solution. The same problem occurs when extracting ν_{min} with, for instance, a $\theta(\nu\, ln(\nu))/\theta(\nu\, ln(\nu))$ pipeline scheme or a $\theta(\nu^n)/\theta(\nu^m)$ pipeline scheme with $n, m > 4$.

We can see with this last examples that it is not always possible to explicitly extract the OPS. For more complex computations or for non linear communication types, we will have to use numerical schemes such as Newton or Brent methods.

C.2.2 Avoid re-computation of optimal packet sizes

In practical applications, pipeline are chained and only the ADP differs. So it is not always useful to re-compute the OPS for each pipeline. In fact, it is sometimes possible to find the gap between problem sizes such that optimal packet sizes are equal.

PROPOSITION 3
Assuming that the analytic expression of $\nu_{opt}(L)$ is known, the gap of problem sizes corresponding to equal optimal packet sizes is the largest integer l_{max} over all $l \geq 0$ such that

$$|\nu_{opt}(L+l) - \nu_{opt}(L)| < 1. \tag{C.5}$$

LEMMA 5 *If $\nu_{opt}(L)$ can be rewritten into $C\ L^{\gamma}$, with $C > 0$ and $\gamma \geq 1$,then $l_{max} = (L^{\gamma} + 1/C)^{1/\gamma} - L$.*

Proof See [5]. □

Remark 3 We can formulate three remarks :

- Consider the linear example (lemma 2). We have $C = \sqrt{\frac{\beta}{R_A\ \tau_A + R_B\ \tau_B}}$ and $\gamma = 1/2$, and so $l_{max} = \frac{R_A\ \tau_A + R_B\ \tau_B}{\beta} + 2\sqrt{\frac{L\ (R_A\ \tau_A + R_B\ \tau_B)}{\beta}}$.
- For the column LU factorization (see example (1)), the optimal size is independent of the ADP, so we compute ν_{opt} only one time.
- If the analytic expression of ν_{opt} can be explicitly extract but cannot be rewritten into $C\ L^{\gamma}$ (see for instance lemma 3), the l_{max} value will be computed by a numerical scheme.

C.3 Experiments

In this section, we discuss the experiments done to evaluate the validity of the theoretical model and the benefits of the overlap of communications by computations with the optimal packet size. Our experiments are based on the LOCCS library [4, 6] which brings some useful overlap management primitives. The performance measures are made on a 32 nodes Intel Paragon.

The main example is the column LU factorization (see example 1) with a good balance between the "forward" and the "backward" computations. We use both the general and linear specific OPIUM methods. A description of the OPIUM library is given in [5].

The optimal packet size predicted and computed with our model is confirmed by the experimental measures ; on Figure C.3, the vertical line corresponds to the OPS computed by the OPIUM routine. Moreover, according to the theory, this size is constant for each pipeline of the factorization until the constraint occurs. So it is computed only one time.

On Figure C.4, we see that the more important is the ADP, the more efficient is the communication/computation overlap.

The gain brought by the overlap decreases when the number of processors increases (see Figure C.5). This is due to the fact that local computations, for each processor, are not large enough to have an efficient overlap . But we can see that the ratio between the two algorithms (with or without overlap) is quite independent of the number of processors which shows the scalability of this optimization.

The gain curve (see Figure C.6) shows that for a large enough matrix, we quickly reach a good overlap (the factorization time with overlap is reduced by half, compared to the factorization without overlap) and we also see that, whatever the matrix size may be, the overhead time generated by the management of the overlap is negligible. We can remark that for less than 100 data, no splitting is done ($\nu_{opt} > L$).

We made some experiments with pipeline schemes for different "forward" and "backward" complexities to compare the generic method with specifics pipeline methods. The OPS are identical, and if the ADP

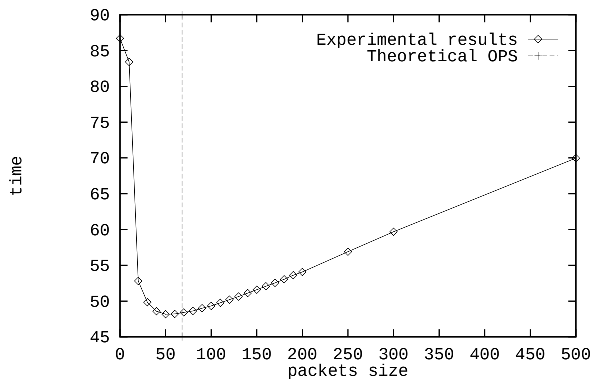


FIG. C.3 – Factorization time of a matrix of size 1024 as a function of the packet size.

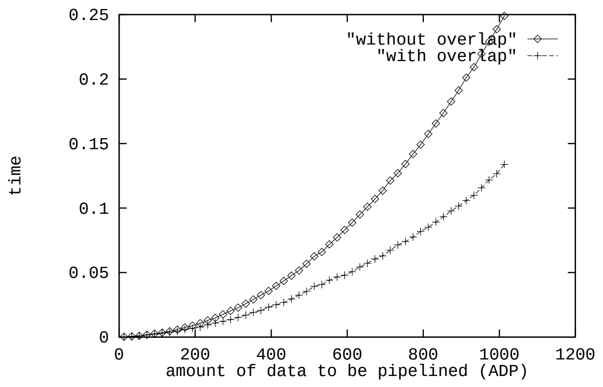


FIG. C.4 – Time spent in each pipeline of the factorization of a matrix of size 1024 (with and without overlap).

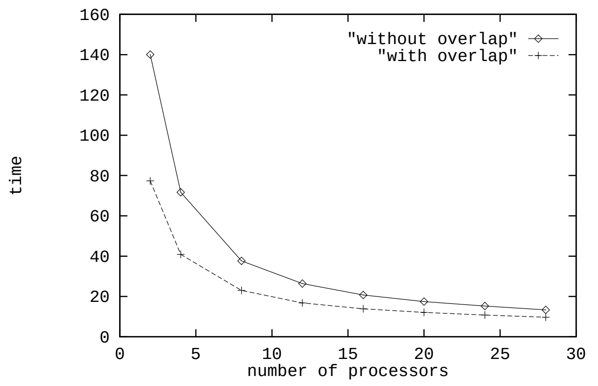


FIG. C.5 – Factorization time (with and without overlap) of a matrix of size 960 as a function of the number of processors.

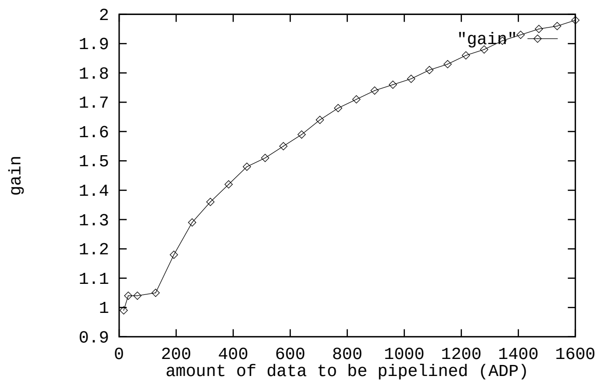


FIG. C.6 – Ratio between the factorization time, with and without overlap, as a function of the matrix size.

increases slowly, the overhead due to the iterative scheme of the generic method is negligible. Indeed, the pre-computed OPS leads to fewer iterations.

C.4 Conclusion and future works

In this paper, we have presented and experimented a general method for the computation of the optimal packet size in the framework of the overlapping communications/computations. We are currently looking into the following studies. First we will try to tackle irregular problems such as sparse Cholesky factorization ; the aim is to compute, in an adaptive scheme, the optimal packet size. Second, we want to adjust the model so that it takes care of BLAS optimizations ; indeed, the gains obtained on the *LU* factorization using a column algorithm may not be obtained for a block algorithm like the one used in ScaLAPACK [25].

There is a trade-off to be found between the computation grain size and the pipeline optimization. Finally, we wish to integrate those techniques into data parallel compilers like High Performance Fortran ones.

C.5 References

[1] T. Brandes and F. Desprez. Implementing Pipelined Computation and Communication in an HPF Compiler. Submitted to Europar'96, 1996.

[2] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley. LAPACK Working Note : ScaLAPACK : A Portable Linear Algebra Library for Distributed Memory Computers - Design Issues and Performances. Technical Report 95, Department of Computer Science - University of Tennessee, 1995.

[3] F. Desprez. A Library for Coarse Grain Macro-Pipelining in Distributed Memory Architectures. In *IFIP 10.3 Conference on Programming Environments for Massively Parallel Distributed Systems*, pages 365–371. Birkhaeuser Verlag AG, Basel, Switzerland, 1994.

[4] F. Desprez. *Procédures de Base pour le Calcul Scientifique sur Machines Parallèles à Mémoire Distribuée*. PhD thesis, Institut National Polytechnique de Grenoble, January 1994. LIP ENS-Lyon.

[5] F. Desprez, P. Ramet, and J. Roman. Optimal grain size computation for pipelined algorithms. Technical report, Laboratoire Bordelais de Recherche en Informatique, 1996.

[6] F. Desprez and B. Tourancheau. LOCCS : Low Overhead Communication and Computation Subroutines. *Future Generation Computer Systems*, 10(2&3) :279–284, June 1994.

[7] B. Tourancheau M. Pourzandi. Recouvrement Calcul/Communication dans l'Elimination de Gauss sur un iPSC/960. Technical report, Ecole Normal Supérieure de Lyon, 1992.

[8] H. Ohta, Y. Saito, M. Kainaga, and H. Ono. Optimal Tile Size Adjustement in Compiling General DOACROSS Loop Nests. In ACM Press, editor, *International Conference on Supercomputing*, pages 270–279, Barcelona, Spain, July 1995. ACM SIGARCH.

[9] Y. Saad. Communication Complexity of the Gaussian Elimination Algorithm on Multiprocessors. *Linear Algebra and Applications*, 77 :315–340, 1986.

[10] B.S. Siegel and P.A. Steenkiste. Controlling Application Grain Size on a Network of Workstations. In *Supercomputing'95*, 1995.

[11] C.W. Tseng. *An Optimizing Fortran D Compiler for MIMD Distributed-Memory Machines*. PhD thesis, Rice University, January 1993.

Chapitre **D**

Communications Optimizations and Efficient Load-balancing for a Volume Rendering Algorithm on a Cluster of PCs

Journal : Parallel and Distributed Computing Practice

Date : August 2001

Authors :

Frédérique Chaussumier
LHPC, ENS-Lyon
46 Allée d'Italie, F-69364 Lyon cedex 07
fchaussu@ens-lyon.fr

Frédéric Desprez
LIP, ENS Lyon, INRIA
46 Allée d'Italie, F-69364 Lyon cedex 07
desprez@ens-lyon.fr

Abstract :

In the medical field, volume rendering provides good quality 3D visualizations but it is still not interactive enough for a day-to-day practice. The most efficient sequential algorithm is the Shear-Warp algorithm. It renders up to 10 images per second for a small dataset. The goal of this paper is to present an efficient parallel implementation of the Shear-Warp algorithm for a distributed memory architecture, a cluster of PCs connected with a high speed network. This highly irregular algorithm led us to implement a dynamic load balancing algorithm. Furthermore, to reduce the overhead due to data redistribution, we overlap communications with computations using MPI's asynchronous communications. Using a good load-balancing and communication overlap, our implementation generates real-time 3D medical images with a good quality and a high resolution.

Keywords :

Parallel volume rendering, pile of PCs, asynchronous communications, dynamic load-balancing.

Thanks :

This work has been supported in part by the LHPC and INRIA.

D.1 Introduction and motivations

Direct volume rendering techniques are effective tools for exploring 3D scalar data. Unlike surface-rendering methods, direct volume-rendering methods can be used to visualize 3D scalar data without converting to intermediate geometric primitives. By assigning appropriate colors and opacities to the scalar data, one can render objects semi-transparently to expand the amount of 3D information available at a fixed position. Volume-rendered images can also be superimposed upon surface-oriented icons or textures thus allowing simultaneous scalar and vector field composite visualizations.

Representing a surface contained within a volumetric data set using geometric primitives can be useful in many applications, however there are several main drawbacks to this approach. First, geometric primitives can only approximate surfaces contained within the original data. Adequate approximations may require an excessive amount of geometric primitives. Therefore, a trade-off must be made between accuracy and space requirements. Second, since only a surface representation is used, much of the information contained within the data is lost during the rendering process.

Real-time rendering is an important goal in visualization applications. Most of these applications require the generation of a sequence of images for different orientations of the volume. Consequently, real-time rendering should enable a continuous visualization of the volume as its orientation changes. Moreover, higher and higher resolution datasets combined with the high computational cost of direct volume rendering makes it difficult, if not impossible, for sequential implementations to deliver the required level of performance. Therefore, such applications have been parallelized to keep a good image quality while getting a real-time visualization. Lacroute and Levoy [14] developed the Shear-Warp algorithm that exploits coherence in the volume and image space. This algorithm is currently acknowledged to be the fastest sequential volume rendering algorithm.

The goal of this paper is to present an efficient parallel implementation of the Shear-Warp algorithm for a distributed memory architecture, a cluster of PCs connected with a high-speed network and using a light weight and fast communication layer. This new parallel implementation is load balanced and overlaps communication with computation using MPI asynchronous communications.

This paper is organized as follows : in the next section, we describe and analyze the sequential version of the Shear-Warp algorithm. The third section describes previous parallelizations of this algorithm and our approach. It exhibits the main problems associated with the parallel implementation. It focuses on architecture, task partition and communications patterns. We also present a new dynamic load-balancing algorithm for the Shear-Warp algorithm tuned to interactivity. In order to improve the scalability of the algorithm, we discuss the possibility of implementing communication overlap in this algorithm. In a last section and before a conclusion, an evaluation of the communication overlap features of MPI is presented and we give our experimental results in terms of load-balancing and scalability.

D.2 Shear-Warp algorithm

Volume rendering [11] is the process of creating a 2D image directly from 3D volumetric data so that no information contained within the data is lost during the rendering process. For example, in computed tomography, scanned data useful informations are not only contained on the surfaces but also within the data. Therefore, it must have a volumetric representation, and must be displayed using volume rendering techniques. Lacroute and Levoy described a fast volume rendering algorithm called the Shear-Warp factorization [14]. It is based on an algorithm that factors the viewing transformation into a 3D shear (parallel to the data slices), a projection to form an intermediate but distorted image, and finally a 2D warp to form an undistorted final image. The Shear-Warp factorization has the property that rows of voxels in the volume are aligned with rows of pixels in the intermediate image. Consequently, a scanline-based algorithm has been constructed that traverses the volume and the intermediate image synchronously, taking advantage of the spatial coherence present in both volume and image. An established acceleration technique for volume rendering is to exploit coherence in the volume by using a spatial data structure. For a given visualization data set, there are clusters of voxels that contribute useful informations to the image and other clusters that are irrelevant. The purpose of a spatial data structure is to encode this type of coherence getting rid of irrelevant voxels. Rendering algorithms use data structures like octrees, pyramids, run-lengths encoding

(RLE) to skip transparent voxels rapidly. Lacroute and Levoy optimized the original algorithm by using spatial data structures based on run-length encoding for both the volume and the image and also taking advantage of early ray termination [13]. The RLE data structure is a sparse data structure that contains only non-transparent voxels for the object and non-saturated pixels for the image. Using a RLE, we skip empty voxels and saturated pixels. MRI (Magnetic Resonance Imaging) or CT (Computer Tomography) images contain up to 70 percent of transparent voxels. Consequently, using the RLE data-structure combined with early ray termination, the Shear-Warp algorithm developed by Lacroute is ten times faster than the original one [12]. An implementation running on an SGI Indigo workstation renders a 256^3 voxel data set in one second. This algorithm is based on three main steps : the computation of the shading lookup table, the projection of the volume data into the intermediate image, and finally the warping of the intermediate image. The projection of the volume data into the intermediate image dominates the cost of the sequential algorithm. It takes over 80% of the total amount of time for a whole execution [1]. Therefore, in this paper we focus on the compositing step which is the projection of the volume data into the intermediate image. The use of an RLE data-structure implies that scanlines may have widely different amount of data associated with them. Data repartitions in both object and image are highly irregular. They depend on the scene and the viewpoint. For instance, Figure D.1 illustrates how the data repartition in the intermediate image depends on the viewpoint. The default viewpoint is the zero-degree rotation angle. We compare the data distribution of respectively 5 and 20-degree rotation angles to the default viewpoint. We can notice that the bigger the rotation is, the more different the data repartition is.

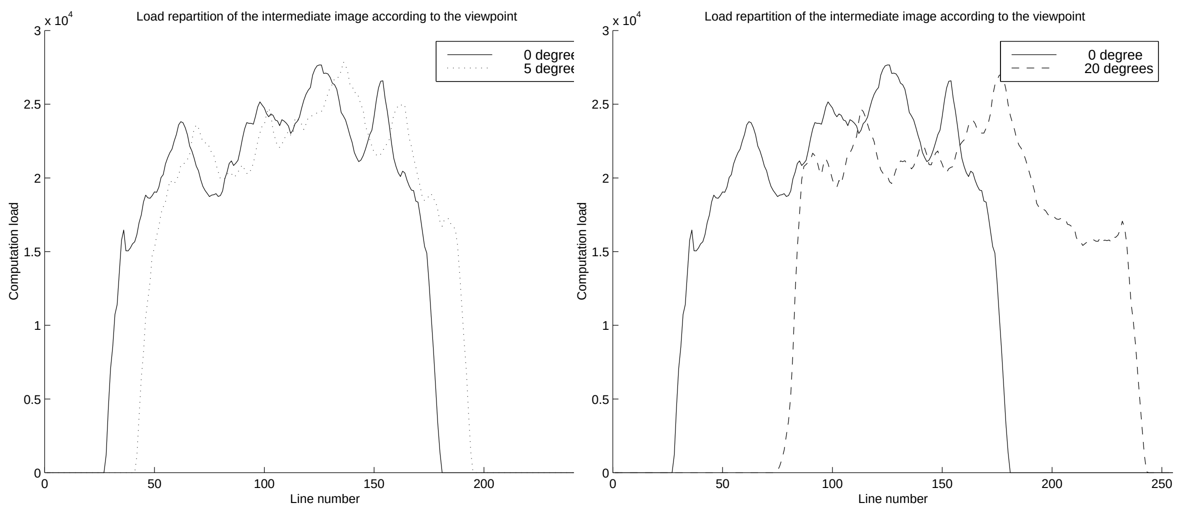


FIG. D.1 – Data repartition in the intermediate image (5-deg. and 20-deg. rotation angles).

The Shear-Warp algorithm leads to a highly irregular application. Accordingly, in the parallel algorithm, computations and communications are very irregular as well. Thus its dynamic load-balancing and the minimization of its communications have to be carefully optimized.

D.3 Parallel algorithm

In this section, we exhibit the main problems associated with the parallel implementation of the Shear-Warp algorithm. The Shear-Warp algorithm augmented with early ray termination and run-length encoding yields to an excellent per-frame sequential rendering times. It forms the basis of our parallel formulation. The critical issues in any parallel algorithm are concurrency, minimization of communication overhead, and a good load-balancing among processors.

D.3.1 Related work

Some authors have already proposed parallel formulations of this algorithm for both shared and distributed memory architectures. To get real-time performance, both Lacroute [13] and then Jiang and Singh [10] parallelized the Shear-Warp algorithm on a 16-processor SMP, the SGI Challenge. Unlike distributed memory architectures, this architecture supports fine-grain and low-latency communications adapted to the irregular communication and computation patterns of the Shear-Warp algorithm. They render a 256^3 voxel data set at over 10 frames per second. Amin et al. [1] have implemented the Shear-Warp algorithm for a distributed memory architecture. With a 128-processor TMC CM5, they could render a 256^3 voxel data set at 12 frames per second. It is comparable to the results obtained on the 16-processor shared memory architecture. However, they restricted the utilization of the Shear-Warp algorithm by allowing only one-degree rotations to change the viewpoint. Despite of this restriction, their algorithm is not scalable : the speedup is only equal to 30 for 128 processors. The two general types of task partitions in parallel volume rendering algorithms are object partitions and image partitions. In an object partition, each processor gets a specific subset of the volume data to re-sample and to composite. The partial results from each processor must then be composited together to form the image. In contrast, using an image partition, each processor has to compute a specific portion of the image. Each image pixel is computed only by one processor, but the volume data has to be moved to different processors as the viewing transformation changes. It is very important not to limit the size of the data volume. In the medical field, standard volumes are composed of 512^3 voxels, which lead to at least 135 Mbytes of raw data corresponding to several hundreds of Mbytes for the classified data. Thus, we chose to distribute the data on every processor because the replication for such volumes is impossible on standard machines. All existing implementations have designed their parallelization using an image partition that takes full advantage of the optimizations of the rendering algorithm. Moreover, for a shared memory architecture, data movement is less significant. The partitioned image is the intermediate image created during the shear step. The unit of work can be individual pixels, scanlines of pixels, or rectangular pixels. In [13], it is shown that the best shape is scanlines of pixels because it minimizes the overhead due to decoding the run-length data structure. It also maximizes the spatial locality both in the intermediate image space and object space. Given that the fundamental unit of work is a group of contiguous scanlines of the intermediate image, minimizing load-imbalances leaves three options : a static contiguous partition, a static interleaved partition and a dynamic partition. For a shared memory architecture, Lacroute chose to use a distributed task queue and a dynamic stealing. This solution is too expensive for a distributed memory architecture. It generates a prohibitive communication overhead. Consequently, for such an architecture, Amin et al. determined heuristics based on adaptative load-balancing scheme. But because their utilization restriction that considers only one-degree rotations they finally conclude that they only needed a static load balancing.

D.3.2 Our approach

Our approach is to implement the parallel version of Shear-Warp algorithm on a distributed memory architecture (a cluster of PCs interconnected with a high-speed Myrinet network from Myricom) because of its good scalability and low cost.

We implemented the overall algorithm but we only focused on the data distribution and redistribution and the composition that are written in italic in Figure D.2 (that takes most of the computation time).

On the one hand, one of our major goals is to achieve real-time performances with higher resolution data sets (particularly 512^3). On the other hand, we believe that it is important not to restrict the user utilization and to allow him to change arbitrarily the viewpoint. Thus, our new implementation proposes a dynamic load-balancing that does not depend on the previous rendering.

D.3.2.1 Data distribution

Because of the distributed memory architecture, we had to determine an explicit data distribution (and redistribution) that minimizes communication but keeps a good load-balancing. Explicit data distribution is a difficult problem when an image partition is used because the portion of the volume required by a particular processor depends on the viewpoint. To distribute data volume in an intermediate image partition

```
Procedure Render()
  InitialDistribution()
  Foreach viewpoint do
    Computation of a part of the shading lookup table (LUT)
    Multidistribution of the shading LUT
    Foreach voxels' slices from front to back do
      If I own the data
        Composite(data, part_image)
    EndFor
    image = Warp(part_image)
    Gather(image, root)
    If p == root
      Display(image)
    Personalized-all2all(volume)
  EndFor
End
```

FIG. D.2 – Overall algorithm

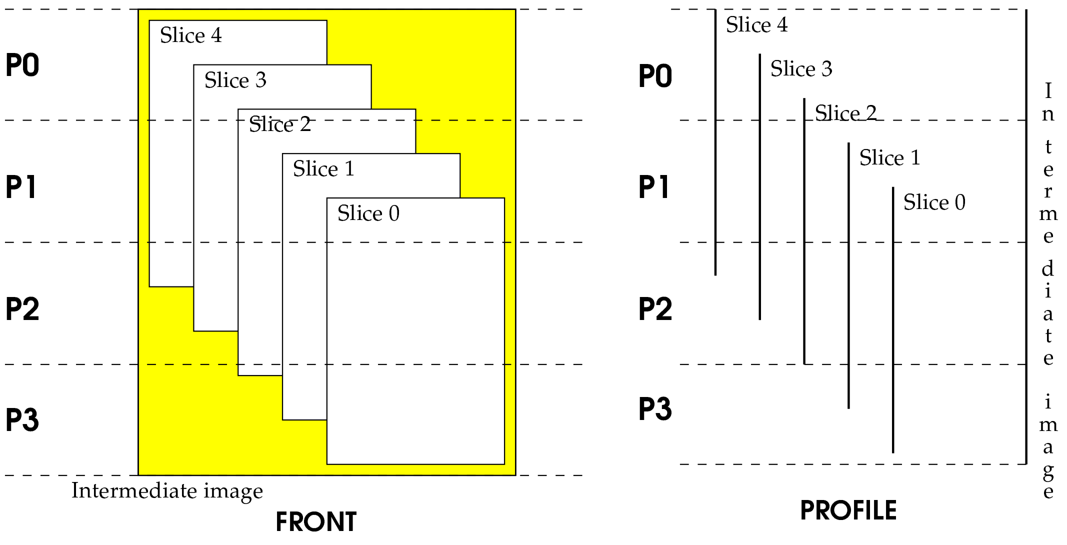


FIG. D.3 – Volume distribution in an image partition.

the volume is first sheared and then distributed by slices orthogonal to the rays. Each processor can compute its portion of the intermediate image through its assigned volume segment. The resulting intermediate images on different processors are disjoint and can be independently warped. Figure D.3 shows a simple intermediate image partition with 4 processors. The corresponding sheared volume, made up of 5 slices, is partitioned as illustrated on the figure : processor 3 owns a few scanlines in the first slice, processor 2 owns scanlines in every slice, . . .

The main overhead of this algorithm results from communications of volume data when the volume is sheared. The generated communications are shown in black in Figure D.4. Every processor receives data corresponding to the shaded scanlines from its neighbor processors except the first and the last ones.

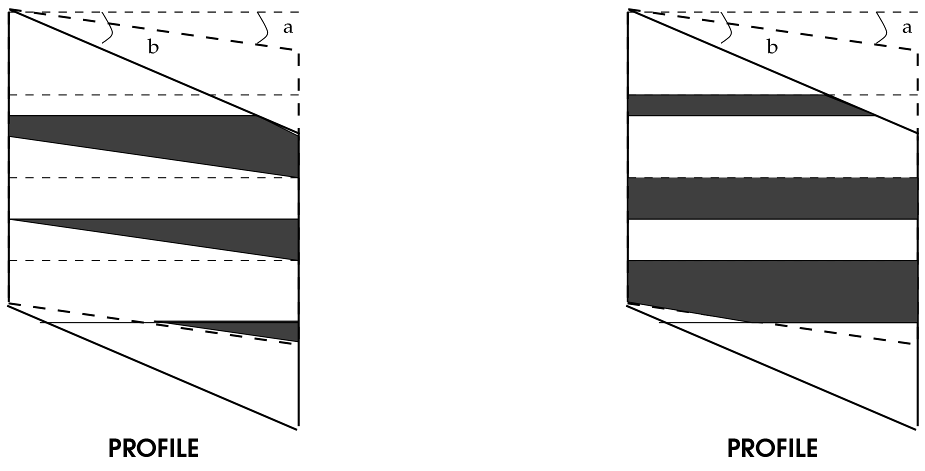


FIG. D.4 – Data received respectively from previous and next processors.

D.3.2.2 Communication patterns

In our parallel Shear-Warp algorithm, we need two types of communications : a gather of partial images into the final image, and a personalized all-to-all communication for the data redistribution when the viewpoint has changed. We implemented the personalized all-to-all communication in $p-1$ steps, where p is the number of processors, as follows : at each step each processor sends data to a step-far processor in the increasing processor number and receives data from a step-far processor in the decreasing processor number.

D.4 Optimizations

In this section, we describe our load-balancing and communication optimizations in our parallel version of the Shear-Warp algorithm.

D.4.1 Dynamic Load-Balancing

Section 2 shown that data repartitions in both object and scene are highly irregular and depend on the scene and the viewpoint (see Figure D.1). Moreover static load-balancing gave us poor results as detailed in the next section. Those are the reasons why the requirement of an arbitrary rotation of the cube of voxels implies that we have to implement a dynamic load-balancing mechanism depending on the viewpoint. In an image partition, every processor has to compute a specific portion of the image. This portion of image results from the projection of the volume data into this portion of image. As explained in section 3.1, the appropriate partitioning should be 1D linear to take advantage of the RLE data structures in both object and image. Therefore, a naive partitioning of the intermediate image that assigns an equal portion of contiguous scanlines to each processor as described in Figure D.3 yields to a bad load-balancing. We also experimented blocks of scanlines interleaving as in [15]. As a matter of fact, when the size of the blocks is one, this partitioning achieves good load-balancing. But in this case, data locality in the object becomes poor and yields to a very bad computation time. Furthermore, it is impossible to determine in a static way the accurate amount of voxels needed to generate this portion since it depends on the viewpoint arbitrarily chosen by the user.

Consequently, we used a derivation of the elastic load-balancing algorithm given in [16] to determine the load and to get a good load-balancing accordingly. This algorithm consists in computing a local partial load for each processor. Then, each processor broadcasts its partial value and adds its value with the ones received. At this moment, every processor knows the global load. By dividing this global load by the number of processors, each processor finds the elementary load. Then every processor has to get the data necessary to its computation. This elastic algorithm allows us to compute a linear partitioning of the intermediate image. Each processor will compute a block of scanlines whose size ensures equal load among the processors. This way, there is no need to interleave blocks and the data locality both in object and image reenforced by the RLE data structures is preserved.

In the Shear-Warp algorithm, every processor computes then an array containing its local contribution to each line of the intermediate image. This array is then broadcast. Each processor adds the arrays received with its own. The resulting array contains the computational load for each line of the intermediate image. They can then obtain the global load. By linearly distributing the intermediate image, processors can balance the load over the processors.

D.4.2 Overlapping communication with computation

Our second optimization is the overlap of communications by computations using non-blocking communications. A communication call is said *non-blocking* if it may return before the operation completes. A communication is said *asynchronous* if its execution proceeds as the same time as the execution of the program. Both kind of routines allow the program to continue its execution but prevent the user from re-using resources (such as buffers) specified in the call. However, a non-blocking communication is not necessarily asynchronous. The data to be communicated can be copied to a temporary buffer and the communication itself can be delayed. If the architecture of the machine has separate communication and computation processors, the communication can be started by the computation processors which gives in turn the task of sending the data over the network to the communication processor.

Several papers have presented some ways to hide communication latency [2] or to use asynchronous communications to improve the implementation of parallel algorithms [5, 7]. In [15], the authors present the parallelization of a cell-projection volume rendering algorithm that uses asynchronous communications. In [6], a good presentation of communication latency hiding is presented, including active messages [8].

In order to reduce the overhead due to data redistribution, we studied the possibility of introducing communication overlap in the compositing phase of the Shear-Warp algorithm. Because of the irregularity of the application, this presents a considerable challenge. In addition to the irregular communication and computation patterns of the Shear-Warp algorithm, we had to deal with communication layers.

So far, every communication generated by the data redistribution is done before the volume composition. Consequently it is possible to overlap the communication of slice $k+1$ with the composition of slice k. The first overlap optimization consists in starting the computation of a slice as soon as every processor sent its corresponding data. The second overlap optimization waits for a processor to send its data and begins immediately its computation corresponding to the received part of slice.

Figure D.5 shows an example of communication and computation pattern with 4 processors and 3 slices. Without overlap, the processor waits for each slice for every processor's data before starting the computation. It is represented by case a. The first overlap step (case b) consists in starting the computation of a slice as soon as every processor sent its corresponding data. Then, in the figure, we have improved the total execution time of A. The second overlap step (case c) waits for a processor to send its data and begins immediately its computation corresponding to the received part of slice. Then, in the figure, we have improved the total execution time of B ($B > A$).

D.5 Experimental results

Our target platform is the LHPC PoPC which is a cluster of 8 PowerPC 604e clocked at 200 Mhz and running Linux. Each workstation is connected with a Myricom/PCI network interface card [3] on the PCI Bus. The interface card contains a host DMA engine, which moves data from host main memory to the SRAM on the network interface, a network DMA engine, which moves the data from the SRAM into the network, and a LANai processor, which executes the low-level messaging protocol and is responsible for both coordinating the actions of the DMA engines and interfacing with the host.

The basic communication layer is BIP (Basic Interface for Parallelism) which is an optimized communication layer for the Myrinet network [17]. There is also an implementation of MPI on top of it. BIP delivers the maximal performance achievable by the hardware to the application.

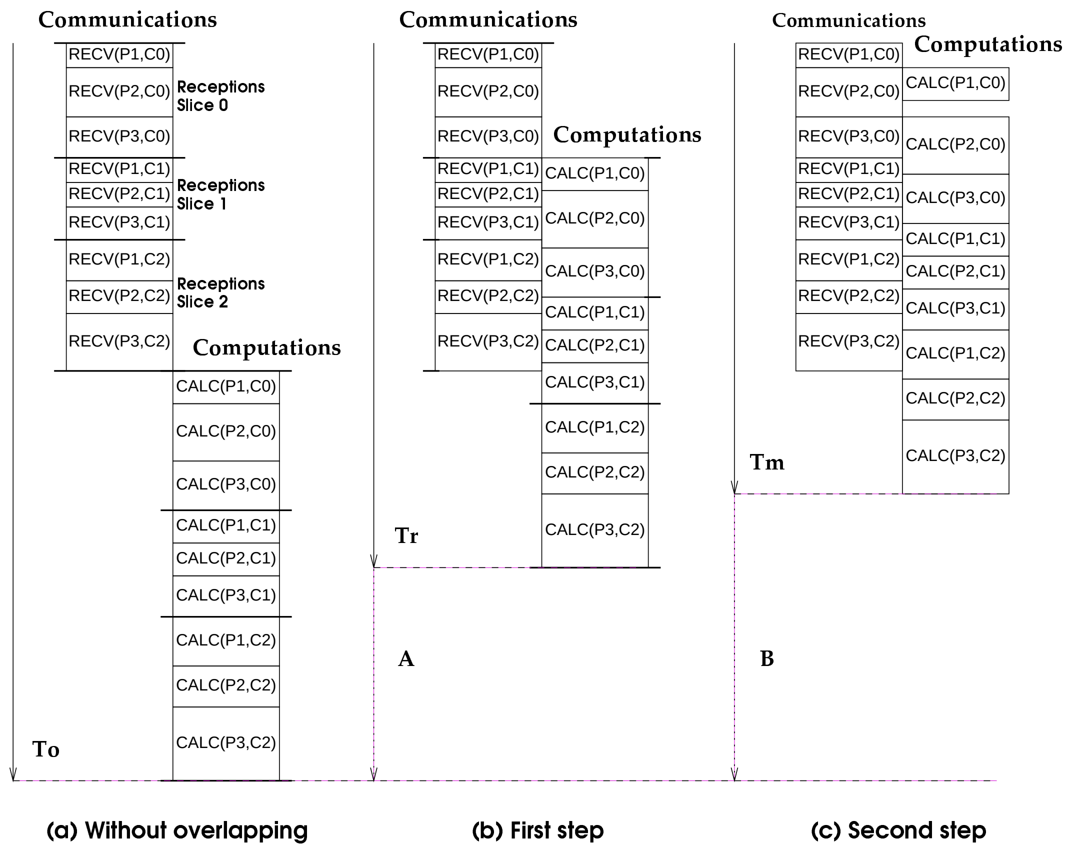


FIG. D.5 – Two possible solutions to overlap communication with computation.

D.5.1 Communication overlap in MPI

The implementations of the Message Passing Interface (MPI) [18] are now available on every kind of platforms, from SMP (Symmetric Multi Processor) to clusters of PCs. This ensures a very good portability. However the use of distributed memory machines or network of workstations adds an overhead due to the communications. To hide this overhead, non-blocking communications can be used to overlap computations and communications. However the assumption that the communication layer provides a "real" overlap and an asynchronous execution of the communication is not obvious as previously stated in [9].

Using BIP native primitives, it is theoretically possible to overlap communications and computations. We executed a test program with both blocking and non-blocking communications on two processors.

The current MPI-BIP implementation does not actually provide any overlap (see Figure D.7). The main restriction comes from the high interaction between the compute and the network processors. We observe that the total time of the non-blocking version is close to the total time of the blocking one. We executed the same test program as with BIP native primitives (see Figure D.6). Results are even worse in a non-blocking version probably because of overhead of splitting the communication in two calls.

To get some overlap with the MPI-BIP interface, we need to interrupt the user's program to switch to communication handling. The cost of the interruption must be low enough so that the improvement due to overlap is not lost. Such a design could be done by using hardware interrupts, and the use of signal handles inside the MPI-BIP implementation. But an implementation of this strategy can be done by modifying the application to periodically check the network status, to eventually launch the next step of the communication protocol. To get this effect, we tried to periodically call a "neutral" MPI primitive in the computation program which has no semantic side-effect, but which will potentially allow the background

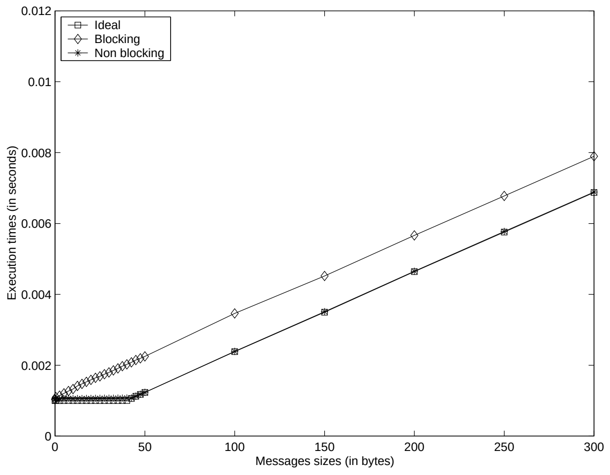


FIG. D.6 – Overlapping native BIP communications with small computations.

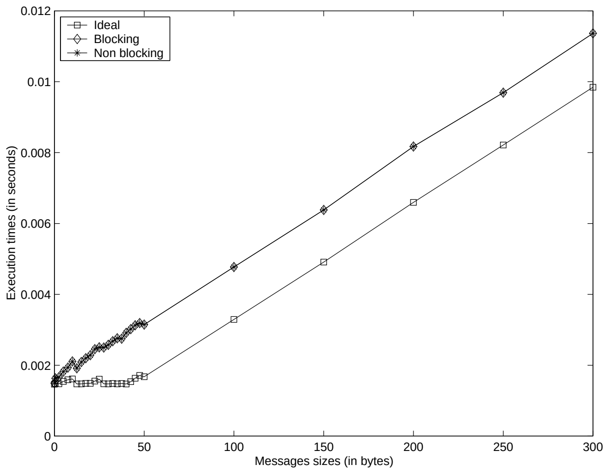


FIG. D.7 – Overlapping communication with computation using MPI-BIP.

communication to progress by handling the intermediate events.

For instance, `MPI_Iprobe` checks whether a message can be received and has no side-effect whereas `MPI_Test` consumes the MPI communication request and completes the reception. Figure D.8 gives the results for two different experiments. In the first experiment, the primitive `MPI_Iprobe` is called in the external loop. As we can see, there is no communication overlap. In contrast, in the second experiment, the primitive `MPI_Iprobe` is called in the internal loop of the computation. In this way, the communication overlap is total (note that the overhead of calling `MPI_Iprobe` is insignificant).

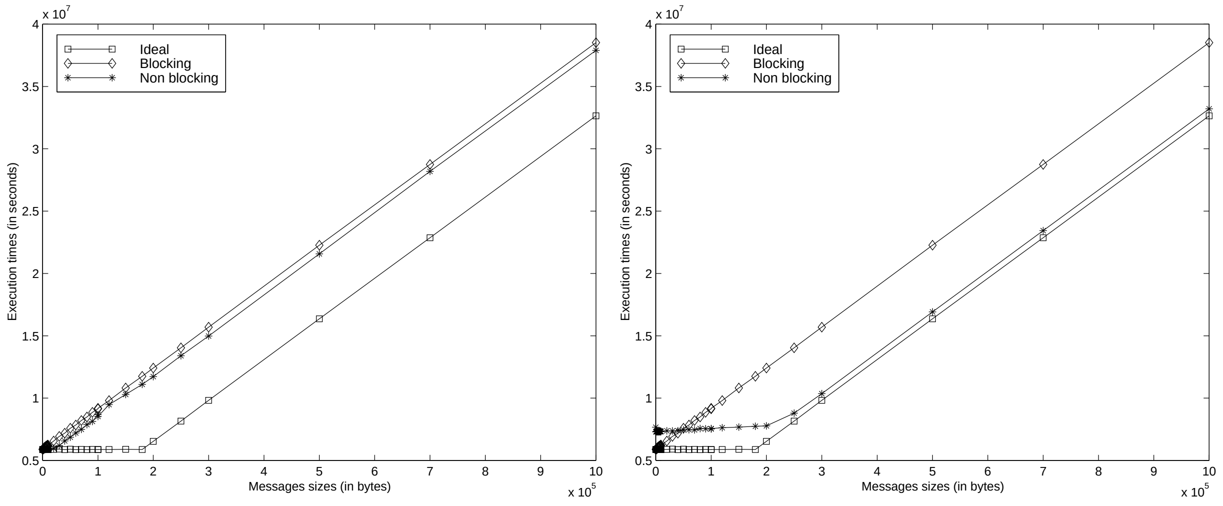


FIG. D.8 – Calling `MPI_Iprobe` in the external and internal loop of the computation.

Even if MPI has been designed to allow such an overlap, it may look surprising that in practice MPI implementations do not always allow to exploit it. We observed the same behavior on other machines and with other MPI implementations. From our study, we can actually derive a set of conditions that must fulfill MPI implementations to actually provide communication overlap. Either it must internally rely on an interrupt-driven mechanism which allow to interrupt the main computation to handle the protocol processing of the communications in the background, or there should be a second processor independent of the main compute processor, which should be able to deal with this processing. More details are given in [4] and [9].

D.5.2 Experimental results of the Shear-Warp algorithm

For our experiments, we used sets of slices provided by Lacroute (Volpack distribution). These are CT scan or MRI scan slices from the Chapel Hill Volume Rendering Test Dataset. Figure D.11 is a volume rendering example obtained with one of the sets used. This one is especially interesting because it leads to bad performances when using a naive data distribution.

Figure D.9 compares respectively the workload of each processor for an execution with processors respectively in the case of a static allocation and a dynamic redistribution.

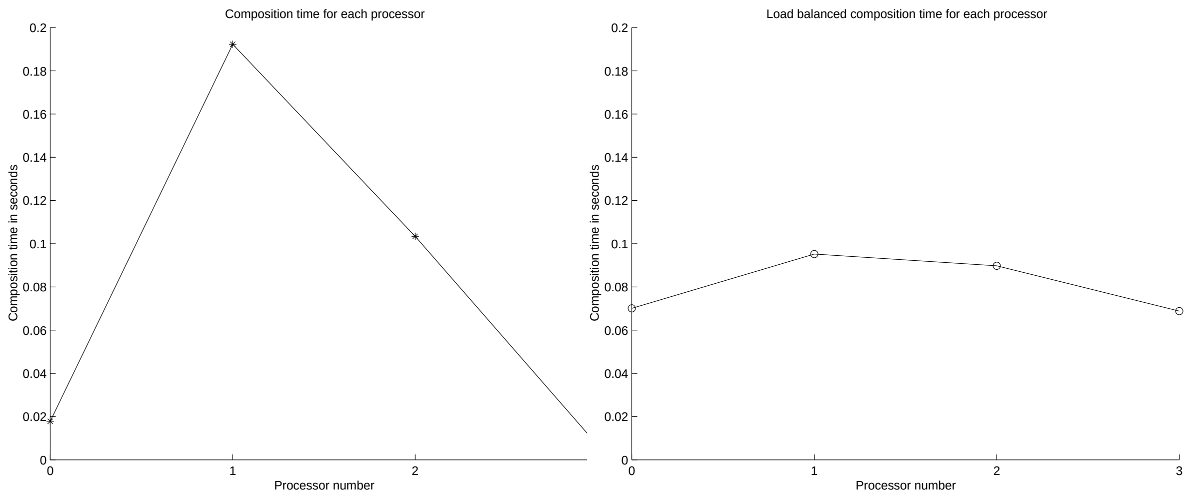


FIG. D.9 – Workload of each processor : static allocation vs dynamic redistribution.

Using a static allocation, we distribute the slices with a regular block distribution of the lines (see Figure D.3). Central processors have the whole load. On the contrary, with the dynamic load-balancing algorithm, the data is well balanced. There are some slight variations due to the granularity of the data.

In our parallel implementation, we focused on the compositing step. We first implemented the compositing step using blocking MPI primitives. The very bad scalability of this implementation, as shown in the Figure D.10, is due to data redistribution overhead. Then we added communication overlap. The figure shows that the implementation using asynchronous communications is almost perfectly scalable. We have a very good overlap of the communications because of the Myrinet technology and the BIP layer. We nevertheless had to call the `MPI_Iprobe` primitive in the computation to take advantage of the overlap inside our implementation of MPI. The curves were obtained on a colored high resolution dataset with 512^3 voxels. The total execution time for such a dataset is 1.5 s on 4 processors.

D.6 Conclusion

The first goal of this application is to provide physicians with a good quality and resolution visualization from medical datasets in real-time with a low-cost distributed memory machine. In this paper, we have presented an high performance and scalable version of the Shear-Warp algorithm implemented on a cluster of PCs. Our parallel approach of the Shear-Warp algorithm improves the interactivity of the application by using an adapted load balancing algorithm and by overlapping communications with computations. It

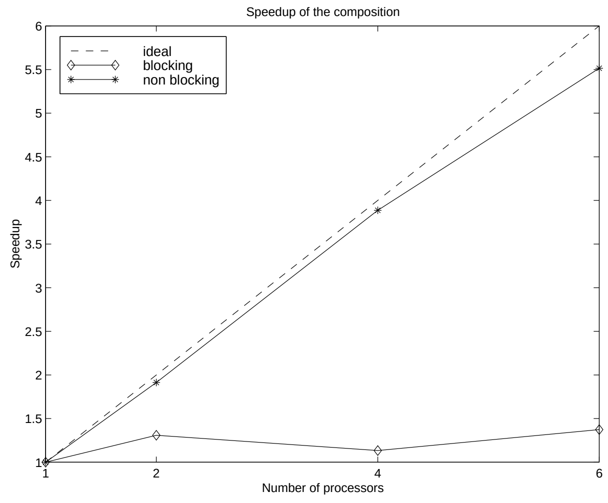


FIG. D.10 – Speedup of the compositing phase.

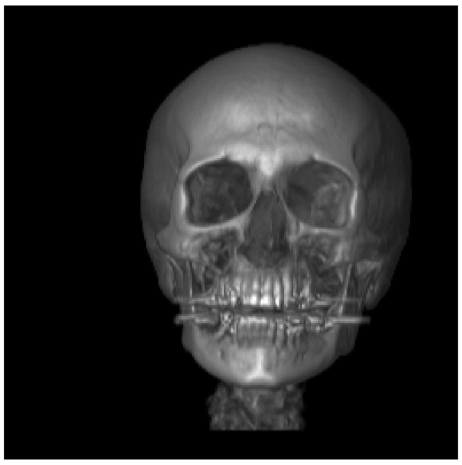


FIG. D.11 – Experimental data.

then allows the user to get a 3D representation with any viewpoint in real-time. Even with a sparse data-structure and irregular communication patterns, we are able to get performances that are comparable to implementations on "classical" parallel machines and at a lower cost. The optimizations presented in this paper can also be used in other irregular applications, such as linear algebra routines using sparse matrices, and thus we would like to create a library to overlap communications and computations for this kind of applications. We would also like to study the same load-balancing on a heterogeneous cluster connecting multiprocessor boards with a Myrinet network. A mixed algorithm could be developed, connecting the study of Lacroute with ours.

We have demonstrated that clusters of PCs can be efficiently used even for irregular applications. They offer a good alternative to higher level machines but we still need some more efficient (and portable) tools for the development of real world applications. The performance/development time ratio is still in favor of SMP machines.

D.7 References

[1] Minesh B. Amin, Ananth Grama, and Vineet Singh. Fast Volume Rendering Using an Efficient Parallel Formulation of the Shear-Warp Algorithm. In *Proceedings 1995 Parallel Rendering Symp.*, 1995.

[2] P.M. Behr, W.K. Giloi, and W. Schröder. Synchronous versus Asynchronous Communication in High-Performance Multicomputer Systems. In *Aspects of Computation on Asynchronous Parallel Processors*, pages 239–249. IFIP, Elsevier Science Publishers, 1989.

[3] Nanette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawik, Charles L. Seitz, Jakov N. Seizovic, and Wen-King Su. Myrinet – A Gigabit-per-Second Local-Area Network. *IEEE MICRO*, pages 29–36, February 1995. http://www.myri.com/research/publications/index.html.

[4] Frédérique Chaussumier, Frédéric Desprez, and Loïc Prylli. Asynchronous Communications in MPI – the BIP/Myrinet Approach. In J. Dongarra, E. Luque, and Tomas Margalef., editors, *Proceedings of the EuroPVM/MPI'99 conference*, number 1697 in Lecture Notes in Computer Science, pages 485–492, Barcelona, Spain, September 1999. Springer Verlag.

[5] M.J. Clement and M.J. Quinn. Overlapping Computations, Communications and I/O in Parallel Sorting. *Journal of Parallel and Distributed Computing*, 28 :162–172, 1995.

[6] D.E. Culler, J. Pal Singh, and A. Gupta. *Parallel Computer Architecture : A Hardware/Software Approach*. Morgan Kaufmann Publishers, 1998. ISBN 1-55860-343-3.

[7] F. Desprez, J.J. Dongarra, and B. Tourancheau. Performance Study of LU Factorization with Low Communication Overhead on Multiprocessors. *Parallel Processing Letters*, 5(2) :157–169, 1995.

[8] T. Von Eicken, D.E. Culler, S.C. Goldstein, and K.E. Schauser. Active Message : a Mecanism for Integrated Communication and Computation. In ACM IEEE Computer Society, editor, *The 19th Annual Symposium on Computer Architecture*, pages 256–266. ACM Press, May 1992.

[9] J.B. White III and S.W. Boya. Where's Overlap ? An Analysis of Popular MPI Implementations. In A. Skjeellum, P.V. Bangalore, and Y.S. Dandass, editors, *Proceedings of the Third MPI Developer's and User's Conference*, pages 1–6, Atlanda, Georgia, 1999. MPI Software Technology Press.

[10] Dongming Jiang and Jaswinder Pal Singh. Improving Parallel Shear-Warp Volume Rendering on Shared Adress Space Multiprocessors. In *Proc. of the 1997 ACM SIGPLAN Symp. on Princ. and Prac. of Par. Progr.*, June 1997.

[11] Arie E. Kaufman. Volume Visualization. *ACM Computing Surveys*, 28(1) :165–167, March 1996.

[12] Philippe Lacroute. *Fast Volume Rendering Using a Shear-Warp Factorization of The Viewing Transformation*. PhD thesis, Stanford University, 1995.

[13] Philippe Lacroute. Real-Time Volume Rendering on Shared Memory Multiprocessors Using the Shear-Warp Factorization. In *Proceedings of the 1995 Parallel Rendering Symposium*, 1995.

[14] Philippe Lacroute and Marc Levoy. Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation. In *Computer Graphics*, volume 28, pages 451–458. Stanford University, July 1994.

[15] Kwan-Liu Ma and Thomas W. Crockett. A Scalable Parallel Cell-Projection Volume Rendering Algorithm for Three-Dimensional Unstructured Data. In *Proceedings of the 1997 Parallel Rendering Symposium*, pages 95–104. ACM SIGGRAPH, October 1997.

[16] Serge Miguet and Yves Robert. Elastic Load Balancing for Image Processing Algorithms. In H.P. Zima, editor, *Par. Comput.* 1st Int. ACPC Conf., 1991.

[17] Loïc Prylli. *BIP Messages User Manual for BIP 0.94*, June 1998. http://www-bip.univ-lyon1.fr/bip.html.

[18] Marc Snir, Steve W. Otto, Steven Huss-Lederman, David W. Walker, and Jack Dongarra. *MPI : the complete reference*. MIT Press, Cambridge, MA, USA, 1996.

Chapitre **E**

Scheduling Block-Cyclic Array Redistribution

Journal : IEEE Transaction on Parallel and Distributed Computing, Vol. 9, N. 2

Date : February 1997

Authors :
Frédéric Desprez[1], Jack Dongarra[2,3], Antoine Petitet[2], Cyril Randriamaro[1] and Yves Robert[2]

[1] LIP, Ecole Normale Supérieure de Lyon, 69364 Lyon Cedex 07, France
[2] Department of Computer Science, University of Tennessee, Knoxville, TN 37996-1301, USA
[3] Mathematical Sciences Section, Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA
e-mail : [desprez, crandria]@lip.ens-lyon.fr
e-mail : [dongarra, petitet, yrobert]@cs.utk.edu

Abstract :
This article is devoted to the run-time redistribution of one-dimensional arrays that are distributed in a block-cyclic fashion over a processor grid. While previous studies have concentrated on efficiently generating the communication messages to be exchanged by the processors involved in the redistribution, we focus on the *scheduling* of those messages : how to organize the message exchanges into "structured" communication steps that minimize contention. We build upon results of Walker and Otto, who solved a particular instance of the problem, and we derive an optimal scheduling for the most general case, namely, moving from a `CYCLIC(r)` distribution on a P-processor grid to a `CYCLIC(s)` distribution on a Q-processor grid, for *arbitrary* values of the redistribution parameters P, Q, r, and s.

Keywords :
distributed arrays, redistribution, block-cyclic distribution, scheduling, MPI, HPF.

Thanks :
This work was supported in part by the National Science Foundation Grant No. ASC-9005933 ; by the Defense Advanced Research Projects Agency under contract DAAH04-95-1-0077, administered by the Army Research Office ; by the Department of Energy Office of Computational and Technology Research, Mathematical, Information, and Computational Sciences Division under Contract DE-AC05-84OR21400 ; by the National Science Foundation Science and Technology Center Cooperative Agreement No. CCR-8809615 ; by the CNRS–ENS Lyon–INRIA project *ReMaP*; and by the Eureka Project *EuroTOPS*. Yves Robert was on leave from Ecole Normale Supérieure de Lyon and is partly supported by DRET/DGA under contract ERE 96-1104/A000/DRET/DS/SR. The authors acknowledge the use of the Intel Paragon XP/S 5 computer, located in the Oak Ridge National Laboratory Center for Computational Sciences, funded by the Department of Energy's Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research.

E.1 Introduction

Run-time redistribution of arrays that are distributed in a block-cyclic fashion over a multidimensional processor grid is a difficult problem that has recently received considerable attention. This interest is motivated largely by the HPF [12] programming style, in which scientific applications are decomposed into phases. At each phase, there is an optimal distribution of the data arrays onto the processor grid. Typically, arrays are distributed according to a `CYCLIC(r)` pattern[1] along one or several dimensions of the grid. The best value of the distribution parameter r depends on the characteristics of the algorithmic kernel as well as on the communication-to-computation ratio of the target machine [5]. Because the optimal value of r changes from phase to phase and from one machine to another (think of a heterogeneous environment), run-time redistribution turns out to be a critical operation, as stated in [9, 20, 21] (among others).

Basically, we can decompose the redistribution problem into the following two subproblems :

Message generation The array to be redistributed should be efficiently scanned or processed in order to build up all the messages that are to be exchanged between processors.

Communication scheduling All the messages must be efficiently scheduled so as to minimize communication overhead. A given processor typically has several messages to send, to all other processors or to a subset of these. In terms of MPI collective operations [15], we must schedule something similar to an `MPI_ALLTOALL` communication, except that each processor may send messages only to a particular subset of receivers (the subset depending on the sender).

Previous work has concentrated mainly on the first subproblem, message generation. Message generation makes it possible to build a different message for each pair of processors that must communicate, thereby guaranteeing a volume-minimal communication phase (each processor sends or receives no more data than needed). However, the question of how to efficiently schedule the messages has received little attention. One exception is an interesting paper by Walker and Otto [20] on how to schedule messages in order to change the array distribution from `CYCLIC(r)` on a P-processor linear grid to `CYCLIC(Kr)` on the same grid. Our aim here is to extend Walker and Otto's work in order to solve the **general** redistribution problem, that is, moving from a `CYCLIC(r)` distribution on a P-processor grid to a `CYCLIC(s)` distribution on a Q-processor grid.

The general instance of the redistribution problem turns out to be much more complicated than the particular case considered by Walker and Otto. However, we provide efficient algorithms and heuristics to improve the scheduling of the communications induced by the redistribution operation. Our main result is the following : For **any** values of the redistribution parameters P, Q, r and s, we construct an **optimal** schedule, that is, a schedule whose number of communication steps is minimal. A communication step is defined so that each processor sends/receives at most one message, thereby optimizing the amount of buffering and minimizing contention on communication ports. The construction of such an optimal schedule relies on graph-theoretic techniques such as the edge coloring number of bipartite graphs. We delay the precise (mathematical) formulation of our results until Section E.4 because we need several definitions beforehand.

Without loss of generality, we focus on one-dimensional redistribution problems in this article. Although we usually deal with multidimensional arrays in high-performance computing, the problem reduces to the "tensor product" of the individual dimensions. This is because HPF does not allow more than one loop variable in an `ALIGN` directive. Therefore, multidimensional assignments and redistributions are treated as several independent one-dimensional problem instances.

The rest of this article is organized as follows. In Section E.2 we provide some examples of redistribution operations to expose the difficulties in scheduling the communications. In Section E.3 we briefly survey the literature on the redistribution problem, with particular emphasis given to the Walker and Otto paper [20]. In Section E.4 we present our main results. In Section E.5 we report on some MPI experiments that demonstrate the usefulness of our results. Finally, in Section E.6, we state some conclusions and future work directions.

[1]The definition is the following : let an array $X[0...M-1]$ be distributed according to a block-cyclic distribution `CYCLIC(r)` onto a linear grid of P processors. Then element $X[i]$ is mapped onto processor $p = \lfloor i/r \rfloor \bmod P$, $0 \leq p \leq P-1$. See Section E.4.1 for further details.

Notations

The main variables used in the next sections are listed in Table E.1. The abbreviations "N.M." and "S/R" are used in a few communication tables. They respectively mean "Number of Messages" and "Sender/Receiver".

TAB. E.1 – Main notations in the paper.

variable	definition
P	The number of processors in the original grid
Q	The number of processors in the target grid
r	The block factor of the original distribution
s	The block factor of the target distribution
X	The array to be redistributed
M	The (global) size of X
L	The least common multiple of Pr and Qs
m	The number of slices of L data elements in array X
$\mathcal{N}$	The number of steps in the communication schedule
$\mathcal{T}$	The total communication cost

E.2 Motivating Examples

Consider an array $X[0...M-1]$ of size M that is distributed according to a block cyclic distribution `CYCLIC(r)` onto a linear grid of P processors (numbered from $p = 0$ to $p = P - 1$). Our goal is to redistribute X using a `CYCLIC(s)` distribution on Q processors (numbered from $q = 0$ to $q = Q - 1$).

For simplicity, assume that the size M of X is a multiple of $L = lcm(Pr, Qs)$, the least common multiple of Pr and Qs : this is because the redistribution pattern repeats after each slice of L elements. Therefore, assuming an even number of slices in X will enable us (without loss of generality) to avoid discussing side effects. Let $m = M \div L$ be the number of slices.

Example 2

Consider a first example with $P = Q = 16$ processors, $r = 3$, and $s = 5$. Note that the new grid of Q processors can be identical to, or disjoint of, the original grid of P processors. The actual total number of processors in use is an unknown value between 16 and 32. All communications are summarized in Table E.2, which we refer to as a *communication grid*. Note that we view the source and target processor grids as disjoint in Table E.2 (even if it may not actually be the case). We see that each source processor $p \in \mathcal{P} = \{0, 1, \ldots, P-1\}$ sends 7 messages and that each processor $q \in \mathcal{Q} = \{0, 1, \ldots, Q-1\}$ receives 7 messages, too. Hence there is no need to use a full all-to-all communication scheme that would require 16 steps, with a total of 16 messages to be sent per processor (or more precisely, 15 messages and a local copy). Rather, we should try to schedule the communication more efficiently. Ideally, we could think of organizing the redistribution in 7 steps, or communication phases. At each step, 16 messages would be exchanged, involving 16 disjoint pairs of processors. This would be perfect for one-port communication machines, where each processor can send and/or receive at most one message at a time.

TAB. E.2 – Communication grid and communication steps for $P = Q = 16$, $r = 3$, and $s = 5$. Message lengths are indicated for a vector X of size $L = 240$.

$\boldsymbol{P = Q = 16, r = 3, s = 5}$, and $\boldsymbol{L = 240}$

S/R	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	N.M.
0	3b	-	-	3a	-	-	3c	-	-	2e	1g	-	1f	2d	-	-	7
1	2d	1f	-	1g	2e	-	-	3c	-	-	3b	-	-	3a	-	-	7
2	-	3b	-	-	3c	-	-	2e	1g	-	1f	2d	-	-	3a	-	7
3	-	1g	2e	-	-	3a	-	-	3b	-	-	3c	-	-	2d	1f	7
4	-	-	3c	-	-	2e	1g	-	1f	2d	-	-	3b	-	-	3a	7
5	2e	-	-	3c	-	-	3a	-	-	3b	-	-	2d	1f	-	1g	7
6	3c	-	-	2e	1g	-	1f	2d	-	-	3a	-	-	3b	-	-	7
7	-	3c	-	-	3b	-	-	3a	-	-	2d	1f	-	1g	2e	-	7
8	-	2e	1g	-	1f	2d	-	-	3c	-	-	3a	-	-	3b	-	7
9	-	-	3a	-	-	3b	-	-	2d	1f	-	1g	2e	-	-	3c	7
10	1g	-	1f	2d	-	-	3b	-	-	3c	-	-	3a	-	-	2e	7
11	3a	-	-	3b	-	-	2d	1f	-	1g	2e	-	-	3c	-	-	7
12	1f	2d	-	-	3a	-	-	3b	-	-	3c	-	-	2e	1g	-	7
13	-	3a	-	-	2d	1f	-	1g	2e	-	-	3b	-	-	3c	-	7
14	-	-	3b	-	-	3c	-	-	3a	-	-	2e	1g	-	1f	2d	7
15	-	-	2d	1f	-	1g	2e	-	-	3a	-	-	3c	-	-	3b	7
N.M.	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	

Note that we may ask something more : we can try to organize the steps in such a way that at each step, the 8 involved pairs of processors exchange a message of the same length. This approach is of interest because the cost of a step is likely to be dictated by the length of the longest message exchanged during the step. Note that message lengths may or may not vary significantly. The numbers in Table E.2 vary from 1 to 3, but they are for a single slice vector. For a vector X of length $M = 240000$, say, $m = 1000$ and message lengths vary from 1000 to 3000 (times the number of bytes needed to represent one data-type element).

A schedule that meets all these requirements, namely, 7 steps of 16 disjoint processor pairs exchanging messages of the same length, will be provided in Section E.4.3.2. We report the solution schedule in Table E.2. Entry in position (p, q) in this table denotes the step (numbered from a to g for clarity) at which processor p sends its message to processor q.

In Table E.3, we compute the cost of each communication step as (being proportional to) the length of the longest message involved in this step. The total cost of the redistribution is then the sum of the cost of all the steps. We further elaborate on how to model communication costs in Section E.4.3.1.

TAB. E.3 – Communication costs for $P = Q = 16, r = 3$, and $s = 5$.

Communication costs for $\boldsymbol{P = Q = 16, r = 3}$, and $\boldsymbol{s = 5}$

Step	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**Total**
Cost	3	3	3	2	2	1	1	**15**

Example 3

The second example, with $P = Q = 16, r = 7$, and $s = 11$, shows the usefulness of an efficient schedule even when each processor communicates with every other processor. As illustrated in Table E.4, message lengths vary with a ratio from 2 to 7, and we need to organize the all-to-all exchange steps in such a way that messages of the same length are communicated at each step. Again, we are able to achieve such a goal (see Section E.4.3.2). The solution schedule is given in Table E.4 (where steps are numbered from a to p), and its cost is given in Table E.5. (We do check that each of the 16 steps is composed of messages of the same length.)

TAB. E.4 – Communication grid and communication steps for $P = Q = 16, r = 7$, and $s = 11$. Message lengths are indicated for a vector X of size $L = 1232$.

$\boldsymbol{P = Q = 16, r = 7, s = 11}$, and $\boldsymbol{L = 1232}$

S/R	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	N.M.
0	7c	6f	2n	6g	7d	2p	5i	7e	3m	4k	7a	4j	3l	7b	5h	2o	16
1	4k	3m	7d	5h	2o	7c	6f	2n	6g	7e	2p	5i	7b	3l	4j	7a	16
2	5h	7b	3l	4k	7a	4j	3m	7d	5i	2n	7e	6f	2o	6g	7c	2p	16
3	6f	2n	6g	7e	2p	5i	7d	3l	4j	7c	4k	3m	7a	5h	2o	7b	16
4	3l	7e	5i	2n	7c	6f	2o	6g	7d	2p	5h	7a	3m	4k	7b	4j	16
5	7b	3l	4j	7c	4k	3m	7e	5i	2n	7d	6f	2o	6g	7a	2p	5h	16
6	2o	6g	7c	2p	5i	7d	3l	4k	7b	4j	3m	7e	5h	2n	7a	6f	16
7	7a	5i	2p	7d	6f	2n	6g	7b	2o	5h	7c	3l	4j	7e	4k	3m	16
8	3m	4k	7b	4j	3l	7a	5h	2p	7e	6f	2n	6g	7d	2o	5i	7c	16
9	6g	7a	2o	5i	7e	3l	4j	7c	4k	3m	7b	5h	2p	7d	6f	2n	16
10	5i	2p	7e	6f	2n	6g	7a	2o	5h	7b	3l	4k	7c	4j	3m	7d	16
11	4j	7d	4k	3m	7b	5h	2p	7a	6f	2o	6g	7c	2n	5i	7e	3l	16
12	7d	2o	5h	7b	3m	4k	7c	4j	3l	7a	5i	2p	7e	6f	2n	6g	16
13	2p	7c	6f	2o	6g	7e	2n	5h	7a	3l	4j	7b	4k	3m	7d	5i	16
14	7e	4j	3m	7a	5h	2o	7b	6f	2p	6g	7d	2n	5i	7c	3l	4k	16
15	2n	5h	7a	3l	4j	7b	4k	3m	7c	5i	2o	7d	6f	2p	6g	7e	16
N.M.	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	16	

TAB. E.5 – Communication costs for $P = Q = 16, r = 7$, and $s = 11$.

Communication costs for $\boldsymbol{P = Q = 16, r = 7}$, and $\boldsymbol{s = 11}$

Step	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**	**n**	**o**	**p**	**Total**
Cost	7	7	7	7	7	6	6	5	5	4	4	3	3	2	2	2	**77**

Example 4

Our third motivating example is with $P = Q = 15$, $r = 3$, and $s = 5$. As shown in Table E.6, the communication scheme is severely unbalanced, in that processors may have a different number of messages to send and/or to receive. Our technique is able to handle such complicated situations. We provide in Section E.4.4 a schedule composed of 10 steps. It is no longer possible to have messages of the same length at each step (for instance, processor $p = 0$ has messages only of length 3 to send, while processor $p = 1$ has messages only of length 1 or 2), but we do achieve a redistribution in 10 communication steps, where each processor sends/receives at most one message per step. The number of communication steps in Table E.6 is clearly optimal, as processor $p = 1$ has 10 messages to send. The cost of the schedule is given in Table E.7.

TAB. E.6 – Communication grid and communication steps for $P = Q = 15$, $r = 3$, and $s = 5$. Message lengths are indicated for a vector X of size $L = 225$.

$\boldsymbol{P = Q = 15, r = 3, s = 5, L = 225}$

S/R	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	N.M.
0	3c	-	-	3d	-	-	3e	-	-	3a	-	-	3b	-	-	5
1	2f	1h	-	2b	1i	-	2j	1d	-	2g	1c	-	2e	1a	-	10
2	-	3e	-	-	3d	-	-	3c	-	-	3a	-	-	3b	-	5
3	-	1i	2f	-	1j	2c	-	1g	2e	-	1h	2b	-	1d	2a	10
4	-	-	3c	-	-	3e	-	-	3d	-	-	3a	-	-	3b	5
5	3d	-	-	3e	-	-	3a	-	-	3b	-	-	3c	-	-	10
6	2e	1j	-	2a	1h	-	2b	1i	-	2c	1f	-	2d	1g	-	5
7	-	3d	-	-	3e	-	-	3a	-	-	3b	-	-	3c	-	10
8	-	1g	2d	-	1b	2f	-	1j	2a	-	1i	2e	-	1h	2c	5
9	-	-	3e	-	-	3a	-	-	3b	-	-	3c	-	-	3d	10
10	3b	-	-	3f	-	-	3c	-	-	3d	-	-	3a	-	-	5
11	2a	1b	-	2c	1g	-	2d	1h	-	2e	1j	-	2f	1i	-	10
12	-	3c	-	-	3a	-	-	3b	-	-	3d	-	-	3e	-	5
13	-	1a	2b	-	1c	2d	-	1e	2f	-	1g	2h	-	1j	2i	10
14	-	-	3a	-	-	3b	-	-	3c	-	-	3d	-	-	3e	5
N.M.	6	9	6	6	9	6	6	9	6	6	9	6	6	9	9	

TAB. E.7 – Communication costs for $P = Q = 15$, $r = 3$, and $s = 5$.

Communication costs for $\boldsymbol{P = Q = 15, r = 3}$, and $\boldsymbol{s = 5}$

Step	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**Total**
Cost	3	3	3	3	3	3	2	2	2	2	**26**

Example 5

Our final example is with $P \neq Q$, just to show that the size of the two processor grids need not be the same. See Table E.8 for the communication grid, which is unbalanced. The solution schedule (see Section E.4.4) is composed of 4 communication steps, and this number is optimal, since processor $q = 1$ has 4 messages to receive.

TAB. E.8 – Communication grid and communication steps for $P = 12$, $Q = 8$, $r = 4$, and $s = 3$. Message lengths are indicated for a vector X of size $L = 48$.

$\boldsymbol{P = 12, Q = 8, r = 4, s = 3, L = 48}$

S/R	0	1	2	3	4	5	6	7	N.M.
0	3a	1c	-	-	-	-	-	-	2
1	-	2b	2a	-	-	-	-	-	2
2	-	-	1c	3a	-	-	-	-	2
3	-	-	-	-	3a	1c	-	-	2
4	-	-	-	-	-	2b	2a	-	2
5	-	-	-	-	-	-	1c	3a	2
6	3b	1d	-	-	-	-	-	-	2
7	-	2a	2b	-	-	-	-	-	2
8	-	-	1d	3b	-	-	-	-	2
9	-	-	-	-	3b	1d	-	-	2
10	-	-	-	-	-	2a	2b	-	2
11	-	-	-	-	-	-	1d	3b	2
N.M.	2	4	4	2	2	4	4	2	

TAB. E.9 – Communication costs for $P = 12$, $Q = 8$, $r = 4$, and $s = 3$.

Communication costs for $\boldsymbol{P = 12, Q = 8, r = 4}$, and $\boldsymbol{s = 3}$

Step	**a**	**b**	**c**	**d**	**Total**
Cost	3	3	1	1	**8**

E.3 Literature overview

We briefly survey the literature on the redistribution problem, with particular emphasis given to the work of Walker and Otto [20].

E.3.1 Message Generation

Several papers have dealt with the problem of efficient code generation for an HPF array assignment statement like

$$A[l_1 : u_1 : s_1] = B[l_2 : u_2 : s_2],$$

where both arrays A and B are distributed in a block-cyclic fashion on a linear processor grid. Some researchers (see Stichnoth et al.[16], van Reeuwijk et al.[18], and Wakatani and Wolfe [19]) have dealt principally with arrays distributed by using either a purely scattered or cyclic distribution (`CYCLIC(1)` in HPF) or a full block distribution (`CYCLIC(`$\lceil \frac{n}{p} \rceil$`)`, where n is the array size and p the number of processors).

Recently, however, several algorithms have been published that handle general block-cyclic `CYCLIC(k)` distributions. Sophisticated techniques involve finite-state machines (see Chatterjee et al. [3]), set-theoretic methods (see Gupta et al. [8]), Diophantine equations (see Kennedy et al. [10, 11]), Hermite forms and lattices (see Thirumalai and Ramanujam [17]), or linear programming (see Ancourt et al. [1]). A comparative survey of these algorithms can be found in Wang et al. [21], where it is reported that the most powerful algorithms can handle block-cyclic distributions as efficiently as the simpler case of pure cyclic or full-block mapping.

At the end of the message generation phase, each processor has computed several different messages (usually stored in temporary buffers). These messages must be sent to a set of receiving processors, as the examples of Section E.2 illustrate. Symmetrically, each processor computes the number and length of the messages it has to receive and therefore can allocate the corresponding memory space. To summarize, when the message generation phase is completed, each processor has prepared a message for all those processors to which it must send data, and each processor possesses all the information regarding the messages it will receive (number, length, and origin).

E.3.2 Communication Scheduling

Little attention has been paid to the scheduling of the communications induced by the redistribution operation. Simple strategies have been advocated. For instance, Kalns and Ni [9] view the communications as a total exchange between all processors and do not further specify the operation. In their comparative survey, Wang et al. [21] use the following template for executing an array assignment statement :

1. Generate message tables, and post all receives in advance to minimize operating systems overhead
2. Pack all communication buffers
3. Carry out barrier synchronization
4. Send all buffers
5. Wait for all messages to arrive
6. Unpack all buffers

Although the communication phase is described more precisely, note that there is no explicit scheduling : all messages are sent simultaneously by using an asynchronous communication protocol. This approach induces a tremendous requirement in terms of buffering space, and deadlock may well happen when redistributing large arrays.

The ScaLAPACK library [4] provides a set of routines to perform array redistribution. As described by Prylli and Tourancheau [14], a total exchange is organized between processors, which are arranged as a (virtual) caterpillar. The total exchange is implemented as a succession of steps. At each step, processors are arranged into pairs that perform a send/receive operation. Then the caterpillar is shifted so that new exchange pairs are formed. Again, even though special care is taken in implementing the total exchange, no attempt is made to exploit the fact that some processor pairs may not need to communicate.

The first paper devoted to *scheduling* the communications induced by a redistribution is that of Walker and Otto [20]. They review two main possibilities for implementing the communications induced by a redistribution operation :

Wildcarded nonblocking receives Similar to the strategy of Wang et al. described above, this asynchronous strategy is simple to implement but requires buffering for all the messages to be received (hence, the total amount of buffering is as high as the total volume of data to be redistributed).

Synchronous schedules A synchronized algorithm involves communication phases or steps. At each step, each participating processor posts a receive, sends data, and then waits for the completion of the receive. But several factors can lead to performance degradation. For instance, some processors may have to wait for others before they can receive any data. Or hot spots can arise if several processors attempt to send messages to the same processor at the same step. To avoid these drawbacks, Walker and Otto propose to *schedule* messages so that, at each step, each processor sends no more than one message and receives no more than one message. This strategy leads to a synchronized algorithm that is as efficient as the asynchronous version, as demonstrated by experiments (written in MPI [15]) on the IBM SP-1 and Intel Paragon, while requiring much less buffering space.

Walker and Otto [20] provide synchronous schedules only for some special instances of the redistribution problem, namely, to change the array distribution from `CYCLIC(r)` on a P-processor linear grid to `CYCLIC(Kr)` on a grid of same size. Their main result is to provide a schedule composed of K steps. At each step, all processors send and receive exactly one message. If K is smaller than P, the size of the grid, there is a dramatic improvement over a traditional all-to-all implementation.

Our aim in this article is to extend Walker and Otto's work in order to solve the general redistribution problem, that is, moving from a `CYCLIC(r)` distribution on a P-processor grid to a `CYCLIC(s)` distribution on a Q-processor grid. We retain their original idea : schedule the communications into steps. At each step, each participating processor neither sends nor receives more than one message, to avoid hot spots and resource contentions. As explained in [20], this strategy is well suited to current parallel architectures. In Section E.4.3.1, we give a precise framework to model the cost of a redistribution.

E.4 Main Results

E.4.1 Problem Formulation

Consider an array $X[0...M-1]$ of size M that is distributed according to a block-cyclic distribution `CYCLIC(r)` onto a linear grid of P processors (numbered from $p=0$ to $p=P-1$). Our goal is to redistribute X by using a `CYCLIC(s)` distribution on Q processors (numbered from $q=0$ to $q=Q-1$). Equivalently, we perform the HPF assignment $Y=X$, where X is `CYCLIC(r)` on a P-processor grid, while Y is `CYCLIC(s)` on a Q-processor grid[2].

The block-cyclic data distribution maps the global index i of vector X (i.e., element $X[i]$) onto a processor index p, a block index l, and an item index x, local to the block (with all indices starting at 0). The mapping $i \longrightarrow (p,l,x)$ may be written as

$$i \longrightarrow (p = \lfloor i/r \rfloor \bmod P,\ l = \frac{\lfloor i/r \rfloor}{P},\ x = i \bmod r). \tag{E.1}$$

We derive the relation

$$i = (Pl+p)r+x. \tag{E.2}$$

Similarly, since Y is distributed `CYCLIC(s)` on a Q-processor grid, its global index j is mapped as $j \longrightarrow (q,m,y)$, where $j=(Qm+q)s+y$. We then get the redistribution equation

$$i = (Pl+p)r+x = (Qm+q)s+y. \tag{E.3}$$

Let $L = lcm(Pr, Qs)$ be the least common multiple of Pr and Qs. Elements i and $L+i$ of X are initially distributed onto the same processor $p = \lfloor i/r \rfloor \bmod P$ (because L is a multiple of Pr, hence r divides L, and P divides $L \div r$). For a similar reason, these two elements will be redistributed onto the same processor $q = \lfloor i/s \rfloor \bmod Q$. In other words, the redistribution pattern repeats after each slice of L elements. Therefore, we restrict the discussion to a vector X of length L in the following. Let $g = \gcd(Pr,Qs)$ (of course $Lg = PrQs$). The bounds in equation (E.3) become

$$\left\{ \begin{array}{ll} 0 \le p < P & 0 \le q < Q \\ 0 \le l < \frac{L}{Pr} = \frac{Qs}{g} & 0 \le m < \frac{L}{Qs} = \frac{Pr}{g} \\ 0 \le x < r & 0 \le y < s. \end{array} \right. \tag{E.4}$$

DÉFINITION 5 *Given the distribution parameters r and s, and the grid parameters P and Q, the* **redistribution problem** *is to determine all the messages to be exchanged, that is, to find all values of p and q such that the redistribution equation (E.3) has a solution in the unknowns l, m, x, and y, subject to the bounds in Equation (E.4). Computing the number of solutions for a given processor pair (p,q) will give the* length *of the message.*

We start with a simple lemma that leads to a handy simplification :

LEMMA 6 *We can assume that r and s are relatively prime, that is,* $\gcd(r,s)=1$.

Proof The redistribution equation (E.3) can be expressed as

$$pr - qs = z + (Prl - Qsm), \tag{E.5}$$

where $z = y-x \in [1-r, s-1]$. Let $\Delta = \gcd(r,s)$, $r = \Delta r'$ and $s = \Delta s'$. Equation (E.3) can be expressed as

$$\Delta(pr'-qs') = z + \Delta(Pr'l - Qs'm).$$

[2]The more general assignment $Y[a:..] = X[b:..]$ can be dealt with similarly.

If it has a solution for a given processor pair (p, q), then Δ divides z, $z = \Delta z'$, and we deduce a solution for the redistribution problem with r', s', P, and Q. ∎

Let us illustrate this simplification on one of our motivating examples :

Back to Example 4

Note that we need to scale message lengths to move from a redistribution operation where r and s are relatively prime to one where they are not. Let us return to Example 4 and assume for a while that we know how to build the communication grid in Table E.6. To deduce the communication grid for $r = 12$ and $s = 20$, say, we keep the same messages, but we scale all lengths by $\Delta = \gcd(r, s) = 4$. This process makes sense because the new size of a vector slice is ΔL rather than L. See Table E.10 for the resulting communication grid. Of course, the scheduling of the communications will remain the same as with $r = 3$ and $s = 5$, while the cost in Table E.7 will be multiplied by Δ.

TAB. E.10 – Communications for $P = Q = 15, r = 12$, and $s = 20$. Message lengths are indicated for a vector X of size $L = 900$.

$$\boldsymbol{P = Q = 15, r = 12, s = 20, L = 900}$$

S/R	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	N.M.
0	12	-	-	12	-	-	12	-	-	12	-	-	12	-	-	5
1	8	4	-	8	4	-	8	4	-	8	4	-	8	4	-	10
2	-	12	-	-	12	-	-	12	-	-	12	-	-	12	-	5
3	-	4	8	-	4	8	-	4	8	-	4	8	-	4	8	10
4	-	-	12	-	-	12	-	-	12	-	-	12	-	-	12	5
5	12	-	-	12	-	-	12	-	-	12	-	-	12	-	-	5
6	8	4	-	8	4	-	8	4	-	8	4	-	8	4	-	10
7	-	12	-	-	12	-	-	12	-	-	12	-	-	12	-	5
8	-	4	8	-	4	8	-	4	8	-	4	8	-	4	8	10
9	-	-	12	-	-	12	-	-	12	-	-	12	-	-	12	5
10	12	-	-	12	-	-	12	-	-	12	-	-	12	-	-	5
11	8	4	-	8	4	-	8	4	-	8	4	-	8	4	-	10
12	-	12	-	-	12	-	-	12	-	-	12	-	-	12	-	5
13	-	4	8	-	4	8	-	4	8	-	4	8	-	4	8	10
14	-	-	12	-	-	12	-	-	12	-	-	12	-	-	12	5
N.M.	6	9	6	6	9	6	6	9	6	6	9	6	6	9	6	

E.4.2 Communication Pattern

LEMMA 7 *Consider a redistribution with parameters r, s, P, and Q, and assume that* $\gcd(r, s) = 1$. *Let $g = \gcd(Pr, Qs)$. The communication pattern induced by the redistribution operation is a complete all-to-all operation if and only if*

$$g \leq r + s - 1.$$

Proof We rewrite Equation (E.5) as $pr - qs = z + \lambda g$ because $Prl - Qsm$ is an arbitrary multiple of g. Since z lies in the interval $[1 - r, s - 1]$ whose length is $r + s - 1$, it is guaranteed that a multiple of g can be found within this interval if $g \leq r + s - 1$. Conversely, assume that $g \geq r + s$: we will exhibit a processor pair (p, q) exchanging no message. Indeed, $p = P - 1$ and $q = 0$ is the desired processor pair. To see this, note that $pr - qs = -r \bmod g$ (because g divides Pr) ; hence, no multiple of g can be added to $pr - qs$ so that it lies in the interval $[1-r, s-1]$, Therefore, no message will be sent from p to q during the redistribution.[3] ■

In the following, our aim is to characterize the pairs of processors that need to communicate during the redistribution operation (in the case $g \geq r+s$). Consider the following function f :

$$\left\{ \begin{array}{ccc} [0..P-1] \times [0..Q-1] & \longrightarrow & \mathbb{Z}_g \\ (p,q) & \longrightarrow & f(p,q) = pr - qs \bmod g \end{array} \right. \tag{E.6}$$

Function f maps each processor pair (p,q) onto the congruence class of $pr-qs$ modulo g. According to the proof of Lemma 7, p sends a message to q if and only if $f(p,q) \in [1-r, s-1] \pmod{g}$. Let us illustrate this process by using one of our motivating examples.

Back to Example 5

In this example, $P=12$, $Q=8$, $r=4$ and $s=3$. We have $g=24$. Take $p=11$ (as in the proof of Lemma 7). If $q=0$, $f(p,q)=-4 \notin [-3,2]$, and q receives no message from p. But if $q=6$, $f(p,q)=2 \in [-3,2]$, and q does receive a message (see Table E.8 to check this).

DÉFINITION 6 *For* $0 \leq k < g$, *let* $class(k) = f^{-1}(k)$, *that is,*

$$f^{-1}(k) = \{(p,q) \in [0..P-1] \times [0..Q-1]; \;\; f(p,q) = k\}.$$

To characterize classes, we introduce integers u and v such that

$$ru - sv = 1$$

(the extended Euclid algorithm provides such numbers for relatively prime r and s). We have the following result.

PROPOSITION 4
Assume that $\gcd(r,s)=1$. For $0 \leq k < g$,

$$class(k) = \{ \begin{pmatrix} p \\ q \end{pmatrix} = \lambda \begin{pmatrix} s \\ r \end{pmatrix} + k \begin{pmatrix} u \\ v \end{pmatrix} \bmod \begin{pmatrix} P \\ Q \end{pmatrix}; \;\; 0 \leq \lambda < \frac{PQ}{g} \}.$$

Proof First, to see that $\frac{PQ}{g}$ indeed is an integer, note that $PQ = PQ(ru-sv) = PrQu - QsPv$. Since g divides both Pr and Qs, it divides PQ.

Two different classes are disjoint (by definition). It turns out that all classes have the same number of elements. To see this, note that for all $k \in [0, g-1]$,

$$(p,q) \in class(0) \Longleftrightarrow (p+ku \bmod P, \;\; q+kv \bmod Q) \in class(k).$$

Indeed, $p+ku \bmod P = p+ku+dP$ for some integer d, $q+kv \bmod Q = q+kv+d'Q$ for some integer d', and

$$\begin{array}{rcl} f(p+ku \bmod P, q+kv \bmod Q) & = & (p+ku+dP)r - (q+kv+d'Q)s \bmod g \\ & = & pr - qs + k + dPr + d'Qs \bmod g \\ & = & f(p,q) + k \bmod g. \end{array}$$

Since there are g classes, we deduce that the number of elements in each class is $\frac{PQ}{g}$.

Next, we see that $(p_\lambda, q_\lambda) = (\lambda s \bmod P, \;\; \lambda r \bmod Q) \in class(0)$ for $0 \leq \lambda < \frac{PQ}{g}$ (because $p_\lambda r - q_\lambda s = 0 \bmod g$).

Finally, $(p_\lambda, q_\lambda) = (p_{\lambda'}, q_{\lambda'})$ implies that P divides $(\lambda - \lambda')s$ and Q divides $(\lambda - \lambda')r$. Therefore, both Pr and Qs divide $(\lambda - \lambda')rs$; hence, $L = lcm(Pr, Qs) = \frac{PrQs}{g}$ divides $(\lambda - \lambda')rs$. We deduce that $\frac{PQ}{g}$ divides $(\lambda - \lambda')$; hence all the processors pairs (p_λ, q_λ) for $0 \leq \lambda < \frac{PQ}{g}$ are distinct. We have thus enumerated $class(0)$. ■

[3]For another proof, see Petitet [13].

DÉFINITION 7 *Consider a redistribution with parameters r, s, P, and Q, and assume that* $\gcd(r, s) = 1$. *Let* $length(p, q)$ *be the length of the message sent by processor p to processor q to redistribute a single slice vector* X *of size* $L = lcm(Pr, Qs)$.

As we said earlier, the communication pattern repeats for each slice, and the value reported in the communication grid tables of Section E.2 are for a single slice ; that is, they are equal to *length*(p, q). Classes are interesting because they represent homogeneous communications : all processor pairs in a given class exchange a message of same length.

PROPOSITION 5
Assume that $\gcd(r, s) = 1$, and let $L = lcm(Pr, Qs)$ be the length of the vector X to be redistributed. Let $vol(k)$ be the piecewise function given by Figure E.1 for $k \in [1 - r, s - 1]$.
– If $r + s - 1 \leq g$, then for $k \in [1 - r, s - 1]$,

$$(p, q) \in class(k) \Rightarrow length(p, q) = vol(k)$$

(recall that if $(p, q) \in class(k)$ where $k \notin [1 - r, s - 1]$, then p sends no message to q).
– If $g \leq r + s$, then for $k \in [0, g - 1]$,

$$(p, q) \in class(k) \Rightarrow length(p, q) = \sum_{k' \in [1-r, s-1];\ k' \bmod g = k} vol(k').$$

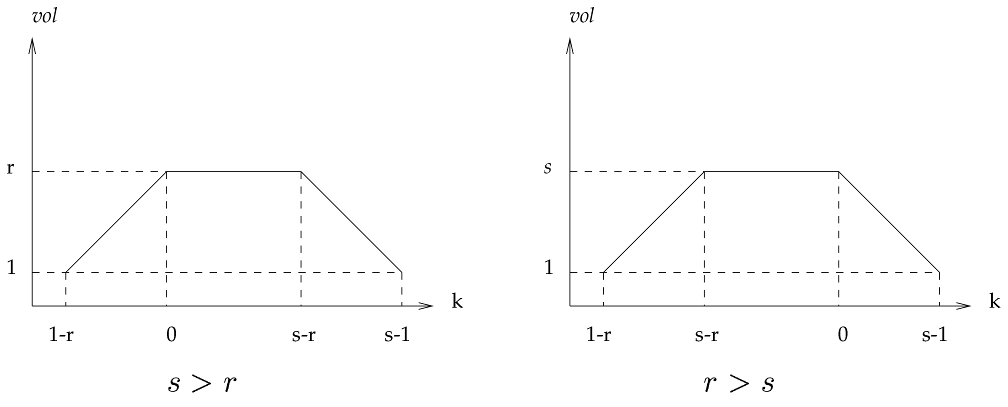


FIG. E.1 – The piecewise linear function *vol*.

Proof We simply count the number of solutions to the redistribution equation $pr - qs = y - x \bmod g$, where $0 \leq x < r$ and $0 \leq y < s$. We easily derive the piecewise linear *vol* function represented in Figure E.1. ■

We now know how to build the communication tables in Section E.2. We still have to derive a schedule, that is, a way to organize the communications as efficiently as possible.

E.4.3 Communication Schedule

E.4.3.1 Communication Model

According to the previous discussion, we concentrate on schedules that are composed of several successive steps. At each step, each sender should send no more than one message ; symmetrically, each receiver should receive no more than one message. We give a formal definition of a schedule as follows.

DÉFINITION 8 *Consider a redistribution with parameters r, s, P, and Q.*

- *The* **communication grid** *is a PQ table with a nonzero entry $length(p, q)$ in position (p, q) if and only if p has to send a message to q.*
- *A* **communication step** *is a collection of pairs $\{(p_1, q_1), (p_2, q_2), \ldots, (p_t, q_t)\}$ such that $p_i \neq p_j$ for $1 \leq i < j \leq t$, $q_i \neq q_j$ for $1 \leq i < j \leq t$, and $length(p_i, q_i) > 0$ for $1 \leq i \leq t$. A communication step is* **complete** *if $t = \min(P, Q)$ (either all senders or all receivers are active) and is incomplete otherwise. The cost of a communication step is the maximum value of its entries, in other words, $\max\{length(p_i, q_i);\ 1 \leq i \leq t\}$*
- *A schedule is a succession of communication steps such that each nonzero entry in the communication grid appears in one and only one of the steps. The cost of a schedule may be evaluated in two ways :*
  1. *the* **number of steps** *$\mathcal{N}$, which is simply the number of communication steps in the schedule ; or*
  2. *the* **total cost** *$\mathcal{T}$, which is the sum of the cost of each communication step (as defined above).*

The communication grid, as illustrated in the tables of Section E.2, summarizes the length of the required communications for a single slice vector, that is, a vector of size $L = lcm(Pr, Qs)$. The motivation for evaluating schedules via their number of steps or via their total cost is as follows :

- The number of steps $\mathcal{N}$ is the number of synchronizations required to implement the schedule. If we roughly estimate each communication step involving all processors (a permutation) as a measure unit, the number of steps is the good evaluation of the cost of the redistribution.
- We may try to be more precise. At each step, several messages of different lengths are exchanged. The duration of a step is likely to be related to the longest length of these messages. A simple model would state that the cost of a step is $\alpha + \max\{length(p_i, q_i)\}\tau$, where α is a start-up time and τ the inverse of the bandwidth on a physical communication link. Although this expression does not take hot spots and link contentions into account, it has proven useful on a variety of machines [4, 6]. The cost of a redistribution, according to this formula, is the affine expression

$$\alpha\mathcal{N} + \beta\mathcal{T}$$

with motivates our interest in both the number of steps and the total cost.

E.4.3.2 A Simple Case

There is a very simple characterization of processor pairs in each class, in the special case where r and Q, as well as s and P, are relatively prime.

PROPOSITION 6
Assume that $\gcd(r, s) = 1$. If $\gcd(r, Q) = \gcd(s, P) = 1$, then for $0 \leq k < g$,

$$(p, q) \in class(k) \Longleftrightarrow q = s^{-1}(pr - k) \bmod g \Longleftrightarrow p = r^{-1}(qs + k) \bmod g$$

(s^{-1} and r^{-1} respectively denote the inverses of s and r modulo g).

Proof Since $\gcd(r, s) = \gcd(r, Q) = 1$, r is relatively prime with Qs, hence with g. Therefore the inverse of r modulo g is well defined (and can be computed by using the extended Euclid algorithm applied to r and g). Similarly, the inverse of s modulo g is well defined, too. The condition $pr - qs = k \bmod g$ easily translates into the conditions of the proposition. ■

In this simple case, we have a very nice solution to our scheduling problem. Assume first that $g \geq r + s - 1$. Then we simply schedule communications class by class. Each class is composed of $\frac{PQ}{g}$ processor pairs that are equally distributed on each row and column of the communication grid : in each class, there are exactly $\frac{Q}{g}$ sending processors per row, and $\frac{P}{g}$ receiving processors per column. This is a direct consequence of Proposition 6. Note that g does divide P and Q : under the hypothesis $\gcd(r, Q) = \gcd(s, P) = 1$, $g = \gcd(Pr, Qs) = \gcd(P, Qs) = \gcd(P, Q)$.

To schedule a class, we want each processor $p = \alpha g + p'$, where $0 \leq \alpha < \frac{P}{g}$, $0 \leq p' < g$, to send a message to each processor $q = \beta g + q'$, where $0 \leq \beta < \frac{Q}{g}$, $0 \leq q' < g$, and $q' = s^{-1}(p'r - k) \bmod g$ (or equivalently, $p' = r^{-1}(q's + k) \bmod g$ if we look at the receiving side). In other words, the processor in position p' within each block of g elements must send a message to the processor in position q' within each block of g elements. This can be done in $\frac{\max(P,Q)}{g}$ complete steps of $\min(P, Q)$ messages. For instance, if there are five blocks of senders ($P = 5g$) and three blocks of receivers ($Q = 3g$), we have 5 steps where 3 blocks of senders send messages to 3 blocks of receivers. We can use any algorithm for generating the block permutation ; the ordering of the communications between blocks is irrelevant.

If $g = r + s - 1$, we have an all-to-all communication scheme, as illustrated in Example 3, but our scheduling by classes leads to an algorithm where all messages have the same length at a given step. If $g < r + s - 1$, we have fewer classes than $r + s - 1$. In this case we simply regroup classes that are equivalent modulo g and proceed as before.

We summarize the discussion by the following result

Proposition 7
Assume that $\gcd(r, s) = 1$. If $\gcd(r, Q) = \gcd(s, P) = 1$, then scheduling each class successively leads to an optimal communication scheme, in terms of both the number of steps and the total cost.

Proof Assume without loss of generality that $P \geq Q$. According to the previous discussion, if $g \geq r+s-1$, we have $r + s - 1$ (the number of classes) times $\frac{P}{g}$ (the number of steps for each class) communication steps. At each step we schedule messages of the same class k, hence of same length $vol(k)$. If $g < r + s - 1$, we have g times $\frac{P}{g}$ communication steps, each composed of messages of the same length (namely, $\sum_{k' \in [1-r, s-1];\ k' \bmod g = k} vol(k')$ when processing a given class $k \in [0, g - 1]$. ■

Remark 4 Walker and Otto [20] deal with a redistribution with $P = Q$ and $s = Kr$. We have shown that going from r to Kr can be simplified to going from $r = 1$ to $s = K$. If $\gcd(K, P) = 1$, the technique described in this section enables us to retrieve the results of [20].

E.4.4 The General Case

When $\gcd(s, P) = s' > 1$, entries of the communication grid may not be evenly distributed on the rows (senders). Similarly, when $\gcd(r, Q) = r' > 1$, entries of the communication grid may not be evenly distributed on the columns (receivers).

Back to Example 4

We have $P = 15$ and $s = 5$; hence $s' = 5$. We see in Table E.6 that some rows of the communication grid have 5 nonzero entries (messages), while other rows have 9. Similarly, $Q = 15$ and $r = 3$; hence $r' = 3$. Some columns of the communication grid have 6 nonzero entries, while other columns have 10.

Our first goal is to determine the maximum number of nonzero entries in a row or a column of the communication grid. We start by analyzing the distribution of each class.

Lemma 8 *Let* $\gcd(s, P) = s'$ *and* $\gcd(r, Q) = r'$. *Let* $P = P's'$ *and* $Q = Q'r'$, *and* $g_0 = \gcd(P', Q')$. *Then* $g = r's'g_0$, *and in any class* class(k), $k \in [0, g - 1]$, *the processors pairs are distributed as follows :*

– *There are* $\frac{P'}{g_0}$ *entries per column in* Q' *columns of the grid, and none in the remaining columns.*

– *There are* $\frac{Q'}{g_0}$ *entries per row in* P' *rows of the grid, and none in the remaining rows.*

Proof First let us check that $g = r's'g_0$. We write $r = r'r"$ and $s = s's"$. We have $Pr = (P's')(r'r") = (r's')(P'r")$. Similarly, $Qs = (r's')(Q's")$. Thus $g = \gcd(Pr, Qs) = r's' \gcd(P'r", Q's")$. Since $r"$ is relatively prime with Q' (by definition of r') and with $s"$ (because $\gcd(r, s) = 1$), we have $\gcd(P'r", Q's") = \gcd(P', Q's")$. Similarly, $\gcd(P', Q's") = \gcd(P', Q') = g_0$.

There are $\frac{PQ}{g}$ elements per class. Since all classes are obtained by a translation of *class*(0), we can restrict ourselves to discussing the distribution of elements in this class. The formula in Lemma 4 states that $class(0) = \{\begin{pmatrix} p \\ q \end{pmatrix} = \lambda \begin{pmatrix} s \\ r \end{pmatrix} \bmod \begin{pmatrix} P \\ Q \end{pmatrix}\}$ for $0 \leq \lambda < \frac{PQ}{g}$. But $\lambda s \bmod P$ can take only those values that are multiple of s' and $\lambda r \bmod Q$ can take only those values that are multiple of r', hence the result. To check the total number of elements, note that $\frac{PQ}{g} = \frac{(P's')(Q'r')}{r's'g_0} = \frac{P'Q'}{g_0}$. ■

Let us illustrate Lemma 8 with one of our motivating examples.

Back to Example 4

Elements of each class should be located on $\frac{P'}{g_0} = \frac{3}{1} = 3$ rows and $\frac{Q'}{g_0} = \frac{5}{1} = 5$ columns of the processor grid. Let us check *class*(1) for instance. Indeed we have the following :

$$\begin{aligned} class(1) = \{ & (2,1),(7,4),(12,7),(2,10),(7,13),(12,1), \\ & (2,4),(7,7),(12,10),(2,13),(7,1),(12,4),(2,7),(7,10),(12,13)\ \} \end{aligned}$$

Lemma 8 shows that we cannot use a schedule based on classes : considering each class separately would lead to incomplete communication steps. Rather, we should build up communication steps by mixing elements of several classes, in order to use all available processors. The maximum number of elements in a row or column of the communication grid is an obvious lower bound for the number of steps of any schedule, because each processor cannot send (or receive) more than one message at any communication step.

PROPOSITION 8
Assume that $\gcd(r, s) = 1$ and that $r + s - 1 \leq g$ (otherwise the communication grid is full). If we use the notations of Lemma 8,

1. the maximum number m_R of elements in a row of the communication grid is $m_R = \frac{Q'}{g_0}\lceil \frac{r+s-1}{s'} \rceil$; and
2. the maximum number m_C of elements in a column of the communication grid is $m_C = \frac{P'}{g_0}\lceil \frac{r+s-1}{r'} \rceil$.

Proof According to Lemma 4, two elements of *class*(k) and *class*(k') are on the same row of the communication grid if $\lambda s + ku = \lambda' s + k'u \bmod P$ for some λ and λ' in the interval $[0, \frac{PQ}{g} - 1]$. Necessarily, s', which divides P and $(\lambda - \lambda')s$, divides $(k - k')u$. But we have $ru - sv = 1$, and s is relatively prime with u. A fortiori s' is relatively prime with u. Therefore s' divides $k - k'$.

Classes share the same rows of the processor grid if they are congruent modulo s'. This induces a partition on classes. Since there are exactly $\frac{Q'}{g_0}$ elements per row in each class, and since the number of classes congruent to the same value modulo s' is either $\lfloor \frac{r+s-1}{s'} \rfloor$ or $\lceil \frac{r+s-1}{s'} \rceil$, we deduce the value of m_R. The value of m_C is obtained similarly. ■

It turns out that the lower bound for the number of steps given by Lemma 8 can indeed be achieved.

THÉORÈME 1
Assume that $\gcd(r, s) = 1$ and that $r+s-1 \leq g$ (otherwise the communication grid is full), and use the notations of Lemma 8 and Lemma 8. The optimal number of steps $\mathcal{N}_{opt}$ for any schedule is

$$\mathcal{N}_{opt} = \max\{m_R, m_C\}.$$

Proof We already know that the number of steps $\mathcal{N}$ of any schedule is greater than or equal to $\max\{m_R, m_C\}$. We give a constructive proof that this bound is tight : we derive a schedule whose number of steps is $\max\{m_R, m_C\}$. To do so, we borrow some material from graph theory. We view the communication grid as a graph $G = (V, E)$, where

– $V = \mathcal{P} \cup \mathcal{Q}$, where $\mathcal{P} = \{0, 1, \ldots, p-1\}$ is the set of sending processors, and $\mathcal{Q} = \{0, 1, \ldots, q-1\}$ is the set of receiving processors ; and

– $e = (p, q) \in E$ if and only if the entry (p, q) in the communication grid is nonzero.

G is a bipartite graph (all edges link a vertex in $\mathcal{P}$ to a vertex in $\mathcal{Q}$). The degree of G, defined as the maximum degree of its vertices, is $d_G = \max\{m_R, m_C\}$. According to König's edge coloring theorem, the edge coloring number of a bipartite graph is equal to its degree (see [7, vol. 2, p.1666] or Berge [2, p. 238]). This means that the edges of a bipartite graph can be partitioned in d_G disjoint edge matchings. A constructive proof is as follows : repeatedly extract from E a maximum matching that saturates all maximum degree nodes. At each iteration, the existence of such a maximum matching is guaranteed (see Berge [2, p. 130]). To define the schedule, we simply let the matchings at each iteration represent the communication steps. ■

Remark 5 The proof of Theorem 1 gives a bound for the complexity of determining the optimal number of steps. The best known algorithm for weighted, bipartite matching has cost $O(|V|^3)$ (Hungarian method, [7, vol. 1, p.206]). Since there are at most $\max(P, Q)$ iterations to construct the schedule, we have a procedure in $O((|P| + |Q|)^4)$ to construct a schedule whose number of steps is minimal.

E.4.5 Schedule Implementation

Our goal is twofold when designing a schedule :

– minimize the number of steps of the schedule, and

– minimize the total cost of the schedule.

We have already explained how to view the communication grid as a bipartite graph $G = (V, E)$. More accurately, we view it as an edge-weighted bipartite graph : the edge of each edge (p, q) is the length *length*(p, q) of the message sent by processor p to processor q.

We adopt the following two strategies :

stepwise If we specify the number of steps, we have to choose at each iteration a maximum matching that saturates all nodes of maximum degree. Since we are free to select any of such matchings, a natural idea is to select among all such matchings one of maximum weight (the weight of a matching is defined as the sum of the weight of its edges).

greedy If we specify the total cost, we can adopt a greedy heuristic that selects a maximum weighted matching at each step. We might end up with a schedule having more than $\mathcal{N}_{opt}$ steps but whose total cost is less.

To implement both approaches, we rely on a linear programming framework (see [7, chapter 30]). Let A be the $|V| \times |E|$ incidence matrix of G, where

$$a_{ij} = \begin{cases} 1 \text{ if edge } j \text{ is incident to vertex } i \\ 0 \text{ otherwise} \end{cases}$$

Since G is bipartite, A is totally unimodular (each square submatrix of A has determinant 0, 1 or -1). The matching polytope of G is the set of all vectors $x \in \mathbb{Q}^{|E|}$ such that

$$\begin{cases} x(e) \geq 0 & \forall e \in E \\ \sum_{e \ni v} x(e) \leq 1 & \forall v \in V \end{cases} \tag{E.7}$$

(intuitively, $x(e) = 1$ iff edge e is selected in the matching). Because the polyhedron determined by Equation E.7 is integral, we can rewrite it as the set of all vectors $x \in \mathbb{Q}^{|E|}$ such that

$$x \geq 0,\ Ax \leq b \text{ where } b = \begin{pmatrix} 1 \\ 1 \\ \dots \\ 1 \end{pmatrix} \in \mathbb{Q}^{|V|}. \tag{E.8}$$

To find a maximum weighted matching, we look for x such that

$$\max\{c^t x;\ x \geq 0,\ Ax \leq b\}, \tag{E.9}$$

where $c \in \mathbb{N}^{|E|}$ is the weight vector.

If we choose the *greedy* strategy, we simply repeat the search for a maximum weighted matching until all communications are done. If we choose the *stepwise* strategy, we have to ensure that, at each iteration, all vertices of maximum degree are saturated. This task is not difficult : for each vertex v of maximum degree in position i, we replace the constraint $(Ax)_i \leq 1$ by $(Ax)_i = 1$. This translates into $Y^t Ax = k$, where k is the number of maximum degree vertices and $Y \in \{0,1\}^{|V|}$ whose entry in position i is 1 iff the ith vertex is of maximum degree. We note that in either case we have a polynomial method. Because the matching polyhedron is integral, we solve a rational linear problem but are guaranteed to find integer solutions.

To see the fact that the greedy strategy can be better than the stepwise strategy in terms of total cost, consider the following example.

Example 6

Consider a redistribution problem with $P = 15$, $Q = 6$, $r = 2$, and $s = 3$. The communication grid and the stepwise strategy are illustrated in Table E.11 : the number of steps is equal to 10, which is optimal, but the total cost is 20 (see Table E.12). The greedy strategy requires more steps, namely, 12 (see Table E.13), but its total cost is 18 only (see Table E.14).

TAB. E.11 – Communication grid and communication steps (stepwise strategy) for $P = 15$, $Q = 6$, $r = 2$, and $s = 3$. Message lengths are indicated for a vector X of size $L = 90$.

Stepwise strategy for $\boldsymbol{P = 15}$, $\boldsymbol{Q = 6}$, $\boldsymbol{r = 2}$, and $\boldsymbol{s = 3}$

S/R	0	1	2	3	4	5	N.M.
0	2a	-	2b	-	2c	-	3
1	1h	1i	1j	1e	1g	1f	6
2	-	2b	-	2c	-	2a	3
3	2b	-	2c	-	2a	-	3
4	1j	1e	1i	1h	1f	1g	6
5	-	2c	-	2a	-	2b	3
6	2c	-	2a	-	2b	-	3
7	1i	1h	1f	1g	1j	1d	6
8	-	2a	-	2b	-	2c	3
9	2f	-	2e	-	2d	-	3
10	1g	1j	1d	1f	1i	1h	6
11	-	2g	-	2d	-	2e	3
12	2d	-	2h	-	2e	-	3
13	1e	1f	1g	1j	1h	1i	6
14	-	2d	-	2i	-	2j	3
N.M.	10	10	10	10	10	10	

TAB. E.12 – Communication costs (stepwise strategy) for $P = 15, Q = 6, r = 2$, and $s = 3$.

Stepwise strategy for $\boldsymbol{P = 15, Q = 6, r = 2}$, and $\boldsymbol{s = 3}$

Step	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**Total**
Cost	2	2	2	2	2	2	2	2	2	2	**20**

TAB. E.13 – Communication grid and communication steps (greedy strategy) for $P = 15, Q = 6, r = 2$, and $s = 3$. Message lengths are indicated for a vector X of size $L = 90$.

Greedy strategy for $\boldsymbol{P = 15, Q = 6, r = 2}$, and $\boldsymbol{s = 3}$

S/R	0	1	2	3	4	5	N.M.
0	2a	-	2b	-	2c	-	3
1	1j	1k	1l	1h	1g	1i	6
2	-	2b	-	2c	-	2a	3
3	2b	-	2c	-	2a	-	3
4	1i	1g	1h	1f	1e	1j	6
5	-	2c	-	2a	-	2b	3
6	2c	-	2a	-	2b	-	3
7	1h	1e	1g	1i	1j	1d	6
8	-	2a	-	2b	-	2c	3
9	2e	-	2f	-	2d	-	3
10	1f	1i	1d	1g	1h	1k	6
11	-	2f	-	2d	-	2e	3
12	2d	-	2e	-	2f	-	3
13	1g	1h	1i	1j	1k	1l	6
14	-	2d	-	2e	-	2f	3
N.M.	10	10	10	10	10	10	

E.4.5.1 Comparison with Walker and Otto's Strategy

Walker and Otto [20] deal with a redistribution where $P = Q$ and $s = Kr$. We know that going from r to Kr can be simplified to going from $r = 1$ to $s = K$. If $\gcd(K, P) = 1$, we apply the results of Section E.4.3.2 (see Remark 4). In the general case ($s' = \gcd(K, P) \geq 1$), classes are evenly distributed among the columns of the communication grid (because $r' = r = 1$), but not necessarily among the rows. However, all rows have the same total number of nonzero elements because s' divides $r + s - 1 = K$. In other words, the bipartite graph is regular. And since $P = Q$, any maximum matching is a perfect matching.

Because $r = 1$, all messages have the same length : *length*$(p, q) = 1$ for every nonzero entry (p, q) in the communication grid. As a consequence, the stepwise strategy will lead to an optimal schedule, in terms of both the number of steps and the total cost. Note that $\mathcal{N}_{opt} = K$ under the hypotheses of Walker and Otto : using the notations of Lemma 8, we have $g = P = Q$. Since $r = r' = 1$, $Q' = Q$; $s' = \gcd(K, P)$, $P = s'P'$,

TAB. E.14 – Communication costs (greedy strategy) for $P = 15, Q = 6, r = 2$, and $s = 3$.

Greedy strategy for $\boldsymbol{P = 15, Q = 6, r = 2}$, and $\boldsymbol{s = 3}$

Step	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**Total**
Cost	2	2	2	2	2	2	1	1	1	1	1	1	**18**

and $g_0 = P'$. We have

$$m_R = \frac{Q'}{g_0}\lceil\frac{r+s-1}{s'}\rceil = s = K,$$

$$m_C = \frac{P'}{g_0}\lceil\frac{r+s-1}{r'}\rceil = \frac{P}{P'}\frac{s}{s'} = s = K.$$

Note that the same result applies when $r = 1$ and $P \neq Q$. Because the graph is regular and all entries in the communication grid are equal, we have the following theorem, which extends Walker and Otto main result [20].

PROPOSITION 9
Consider a redistribution problem with $r = 1$ (and arbitrary P, Q and s). The schedule generated by the stepwise strategy is optimal, in terms of both the number of steps and the total cost.

The strategy presented in this article makes it possible to directly handle a redistribution from an arbitrary `CYCLIC(r)` to an arbitrary `CYCLIC(s)`. In contrast, the strategy advocated by Walker and Otto requires two redistributions : one from `CYCLIC(r)` to `CYCLIC(lcm(r,s))` and a second one from `CYCLIC(lcm(r,s))` to `CYCLIC(s)`.

E.5 MPI Experiments

This section presents results for runs on the Intel Paragon for the redistribution algorithm described in Section E.4.

E.5.1 Description

Experiments have been executed on the Intel Paragon XP/S 5 computer with a C program calling routines from the MPI library. MPI is chosen for portability and reusability reasons. Schedules are composed of steps, and each step generates at most one send and/or one receive per processor. Hence we used only one-to-one communication primitives from MPI.

Our main objective was a comparison of our new scheduling strategy against the current redistribution algorithm of ScaLAPACK [14], namely, the "caterpillar" algorithm that was briefly summarized in Section E.3.2. To run our scheduling algorithm, we proceed as follows :

1. Compute schedule steps using the results of Section E.4.
2. Pack all the communication buffers.
3. Carry out barrier synchronization.
4. Start the timer.
5. Execute communications using our redistribution algorithm (resp. the caterpillar algorithm).
6. Stop the timer.
7. Unpack all buffers.

The maximum of the timers is taken over all processors. We emphasize that we do not take the cost of message generation into account : we compare communication costs only.

Instead of the caterpillar algorithm, we could have used the `MPI_ALLTOALLV` communication primitive. It turns out that the caterpillar algorithm leads to better performance than the `MPI_ALLTOALLV` for all our experiments (the difference is roughly 20% for short vectors and 5% for long vectors).

We use the same physical processors for the input and the output processor grid. Results are not very sensitive to having the same grid or disjoint grids for senders and receivers.

E.5.2 Results

Three experiments are presented below. The first two experiments use the schedule presented in Section E.4.3.2, which is optimal in terms of both the number of steps $\mathcal{N}$ and the total cost $\mathcal{T}$. The third experiment uses the schedule presented in Section E.4.4, which is optimal only in terms of $\mathcal{N}$.

Back to Example 2

The first experiment corresponds to Example 2, with $P = Q = 16$, $r = 3$, and $s = 5$. The redistribution schedule requires 7 steps (see Table E.3). Since all messages have same length, the theoretical improvement over the caterpillar algorithm, which as 16 steps, is $7/16 \approx 0.44$. Figure E.2 shows that there is a significant difference between the two execution times. The theoretical ratio is obtained for very small vectors (e.g., of size 1200 double-precision reals). This result is not surprising because start-up times dominate the cost for small vectors. For larger vectors the ratio varies between 0.56 and 0.64. This is due to contention problems : our scheduler needs only 7 step, but each step generates 16 communications, whereas each of the 16 steps of the caterpillar algorithm generates fewer communications (between 6 and 8 per step), thereby generating less contention.

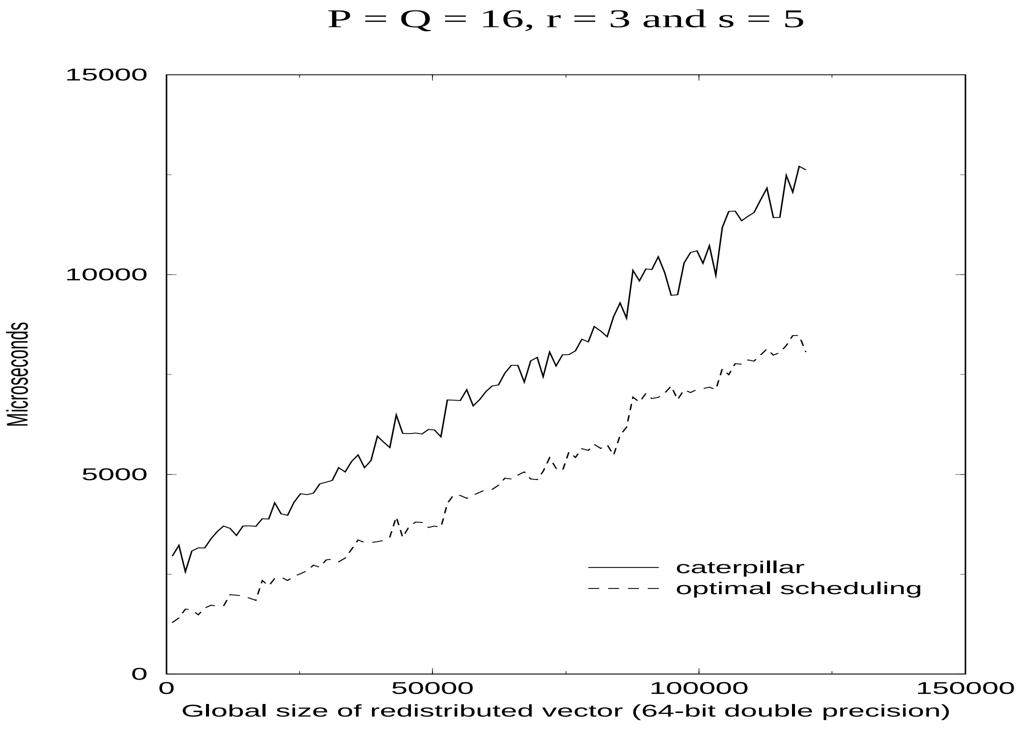


FIG. E.2 – Comparing redistribution times on the Intel Paragon for $P = Q = 16, r = 3$ and $s = 5$.

Back to Example 3

The second experiment corresponds to Example 3, with $P = Q = 16$ processors, $r = 7$, and $s = 11$. Our redistribution schedule requires 16 steps, and its total cost is $\mathcal{T} = 77$ (see Table E.5). The caterpillar algorithm requires 16 steps, too, but at each step at least one processor sends a message of length (proportional to) 7, hence a total cost of 112. The theoretical gain $77/112 \approx 0.69$ is to be expected for very long vectors only (because of start-up times). We do not obtain anything better than 0.86, because of contentions. Experiments on an IBM SP2 or on a Network of Workstations would most likely lead to more favorable ratios.

Back to Example 5

The third experiment corresponds to Example 5, with $P = 12$, $Q = 8$, $r = 4$, and $s = 3$. This experiment is similar to the first one in that our redistribution schedule requires much fewer steps (4) than does the caterpillar (12). There are two differences, however : $P \neq Q$, and our algorithm is not guaranteed to be

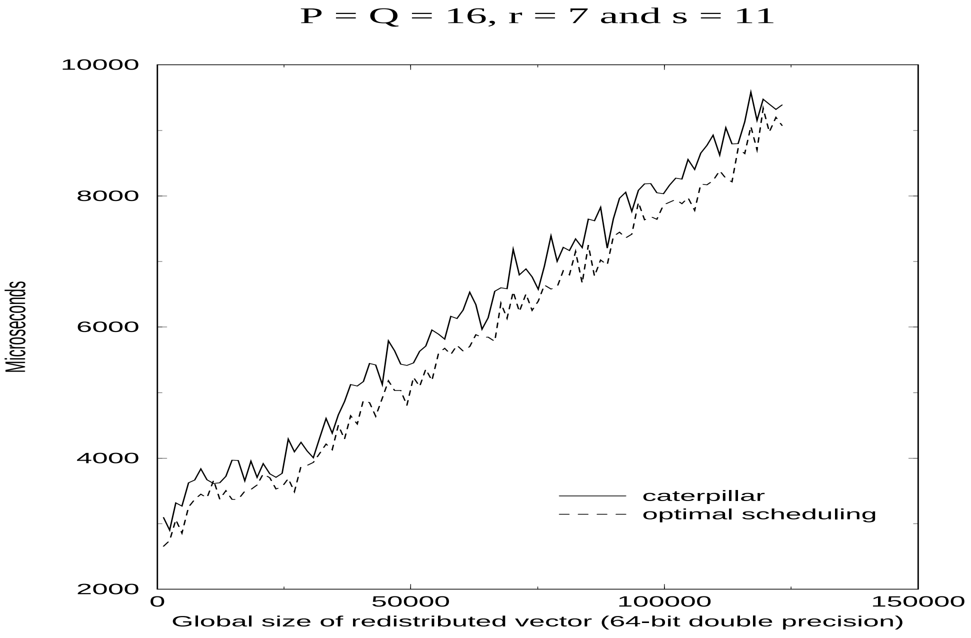


FIG. E.3 – Time measurements for the caterpillar and greedy schedules, for different vector sizes, redistributed from $P = 16, r = 7$ to $Q = 16, s = 11$.

optimal in terms of total cost. Instead of obtaining the theoretical ratio of $4/12 \approx 0.33$, we obtain results close to 0.6. To explain this, we need to take a closer look at the caterpillar algorithm. As shown in Table E.15, 6 of the 12 steps of the caterpillar algorithm are indeed empty steps, and the theoretical ratio rather is $4/6 \approx 0.66$.

TAB. E.15 – Communication costs for $P = 12$, $Q = 8$, $r = 4$, and $s = 3$ with the caterpillar schedule.

Caterpillar for $\boldsymbol{P = 12, Q = 8, r = 4}$, and $\boldsymbol{s = 3}$

Step	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**Total**
Cost	3	0	0	0	3	3	3	0	0	0	3	3	**18**

E.6 Conclusion

In this article, we have extended Walker and Otto's work in order to solve the **general** redistribution problem, that is, moving from a `CYCLIC(r)` distribution on a P-processor grid to a `CYCLIC(s)` distribution on a Q-processor grid. For **any** values of the redistribution parameters P, Q, r, and s, we have constructed a schedule whose number of steps is optimal. Such a schedule has been shown optimal in terms of total cost for some particular instances of the redistribution problem (that include Walker and Otto's work). Future work will be devoted to finding a schedule that is optimal in terms of both the number of steps and the total cost for arbitrary values of the redistribution problem. Since this problem seems very difficult (it may prove NP-complete), another perspective is to further explore the use of heuristics like the greedy algorithm that we have introduced, and to assess their performances.

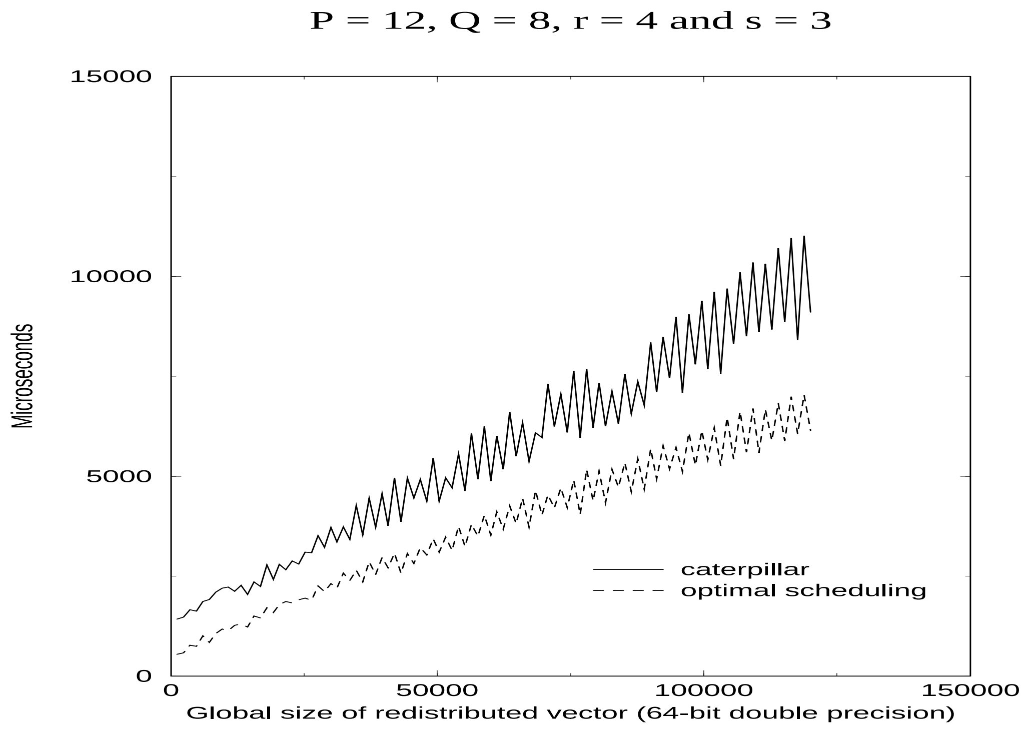


FIG. E.4 – Time measurements for the caterpillar and greedy schedules, for different vector sizes, redistributed from $P = 15$, $r = 4$ to $Q = 6$, $s = 3$.

We have run a few experiments, and these generated optimistic results. One of the next releases of the ScaLAPACK library may well include the redistribution algorithm presented in this article.

Acknowledgments

We thank the reviewers whose comments and suggestions have greatly improved the presentation of the paper.

E.7 References

[1] Corinne Ancourt, Fabien Coelho, François Irigoin, and Ronan Keryell. A linear algebra framework for static HPF code distribution. *Scientific programming*, to appear. Avalaible as CRI–Ecole des Mines Technical Report A-278-CRI, and at http://www.cri.ensmp.fr.

[2] Claude Berge. *Graphes et hypergraphes*. Dunod, 1970. English translation by Elsevier, Amsterdam (1985).

[3] S. Chatterjee, J. R. Gilbert, F. J. E. Long, R. Schreiber, and S.-H. Teng. Generating local addresses and communication sets for data-parallel programs. *Journal of Parallel and Distributed Computing*, 26(1) :72–84, 1995.

[4] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. ScaLAPACK : A portable linear algebra library for distributed memory computers - design issues and performance. *Computer Physics Communications*, 97 :1–15, 1996. (also LAPACK Working Note #95).

[5] J. J. Dongarra and D. W. Walker. Software libraries for linear algebra computations on high performance computers. *SIAM Review*, 37(2) :151–180, 1995.

[6] Gene H. Golub and Charles F. Van Loan. *Matrix computations*. Johns Hopkins, 2 edition, 1989.

[7] R.L. Graham, M. Grötschel, and L. Lovász. *Handbook of combinatorics*. Elsevier, 1995.

[8] S. K. S. Gupta, S. D. Kaushik, C.-H. Huang, and P. Sadayappan. Compiling array expressions for efficient execution on distributed-memory machines. *Journal of Parallel and Distributed Computing*, 32(2) :155–172, 1996.

[9] E. T. Kalns and L. M. Ni. Processor mapping techniques towards efficient data redistribution. *IEEE Trans. Parallel Distributed Systems*, 6(12) :1234–1247, 1995.

[10] K. Kennedy, N. Nedeljkovic, and A. Sethi. Efficient address generation for block-cyclic distributions. In *1995 ACM/IEEE Supercomputing Conference*. `http ://www.supercomp.org/sc95/proceedings`, 1995.

[11] K. Kennedy, N. Nedeljkovic, and A. Sethi. A linear-time algorithm for computing the memory access sequence in data-parallel programs. In *Fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 102–111. ACM Press, 1995.

[12] Charles H. Koelbel, David B. Loveman, Robert S. Schreiber, Guy L. Steele Jr., and Mary E. Zosel. *The High Performance Fortran Handbook*. The MIT Press, 1994.

[13] Antoine Petitet. *Algorithmic redistribution methods for block cyclic decompositions*. PhD thesis, University of Tennessee at Knoxville, December 1996.

[14] Loïc Prylli and Bernard Tourancheau. Efficient block-cyclic data redistribution. In *EuroPar'96*, volume 1123 of *Lectures Notes in Computer Science*, pages 155–164. Springer Verlag, 1996.

[15] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra. *MPI the complete reference*. The MIT Press, 1996.

[16] J. M. Stichnoth, D. O'Hallaron, and T. R. Gross. Generating communication for array statements : design, implementation, and evaluation. *Journal of Parallel and Distributed Computing*, 21(1) :150–159, 1994.

[17] A. Thirumalai and J. Ramanujam. Fast address sequence generation for data-parallel programs using integer lattices. In C.-H. Huang, P. Sadayappan, U. Banerjee, D. Gelernter, A. Nicolau, and D. Padua, editors, *Languages and Compilers for Parallel Computing*, volume 1033 of *Lectures Notes in Computer Science*, pages 191–208. Springer Verlag, 1995.

[18] K. van Reeuwijk, W. Denissen, H. J. Sips, and E. M.R.M. Paalvast. An implementation framework for HPF distributed arrays on message-passing parallel computer systems. *IEEE Trans. Parallel Distributed Systems*, 7(9) :897–914, 1996.

[19] A. Wakatani and M. Wolfe. Redistribution of block-cyclic data distributions using MPI. *Parallel Computing*, 21(9) :1485–1490, 1995.

[20] David W. Walker and Steve W. Otto. Redistribution of block-cyclic data distributions using MPI. *Concurrency : Practice and Experience*, 8(9) :707–728, 1996.

[21] Lei Wang, James M. Stichnoth, and Siddhartha Chatterjee. Runtime performance of parallel array assignment : an empirical study. In *1996 ACM/IEEE Supercomputing Conference*. `http ://www.supercomp.org/sc96/proceedings`, 1996.

Chapitre **F**

HPFIT : A Set of Integrated Tools for the Parallelization of Applications Using High Performance Fortran

Journal : Parallel Computing Vol. 23, N. 1-2

Date : 1997

Authors :

T. Brandes
GMD
SCAI
Schloss Birlinghoven
53754 St. Augustin, Germany

S. Chaumette, M.-C. Counilh
and J. Roman
LaBRI, URA CNRS 1304
ENSERB
and Université Bordeaux I
33405 Talence, France

A. Darte, F. Desprez
and J.-C. Mignot
LIP, URA CNRS 1398
INRIA Rhône-Alpes
ENS Lyon
69364 Lyon Cedex 07, France

Abstract :

In this paper, we present the HPFIT project whose aim is to provide a set of interactive tools integrated in a single environment to help users to parallelize scientific applications to be run on distributed memory parallel computers. HPFIT is built around a restructuring tool called TransTOOL which includes an editor, a parser, a dependence analysis tool and an optimization kernel. Moreover, we provide a clean interface to help developers of tools around High Performance Fortran to integrate their software within our tool.

Keywords :

Semi-automatic parallelization, High Performance Fortran, Development Tools.

Thanks :

This work is partly supported by the CNRS–ENS Lyon–INRIA project *ReMaP* and by the *LHPC–EuroTOPS* project.

F.1 Introduction

Developing large scientific applications on distributed memory machines is a difficult task, especially for newcomers or people without a deep knowledge of parallelism. Currently, we notice that parallelism not only enters industrial areas but also other fields of science. The aim of European projects, like EUROPORT 1 and 2 is clear : to prove the validity of a parallel solution for large industrial codes. Moreover, in order to solve larger problems, scientists need more and more powerful computers in terms of Mflops and memory size. During the last two years, big efforts have been put in the definition of libraries and languages. These efforts lead to standards like BLAS (Basic Linear Algebra Subroutines), MPI (Message Passing Interface) and HPF (High Performance Fortran). Some parallel libraries like ScaLAPACK or NAG start to be widely used. Finally, many tools for the development of applications using data-parallel languages begin to appear. Nevertheless, almost none of them offers a complete collection of portable tools and almost none of them is available as a free software.

It is unlikely that we will ever see a “black box” able to parallelize a non-trivial serial code into a performing parallel code. The user needs to help the compiler by giving information about his code. This can be done for example via directives inserted in the source file like in HPF. But it is also now admitted that the insertion of these directives is also a trivial task for average users, who do not have a deep knowledge of parallelization techniques. Therefore, we think that interactive parallelization tools have a great importance in parallel computing, even in the limited field of numerical problems.

This paper is the first of two papers, presenting the HPFIT project and the developments made around it. This first paper presents the HPFIT project in general and its kernel, TransTOOL. The second paper [8] presents the data structure visualization tool VisIt and HPF extensions for irregular problems. These two papers describe the first results of our development effort.

The remainder of this paper is organized as follows. Section F.2 gives a short survey of tools for data-parallel programming. Section F.3 presents the development of a parallel application and describes the HPFIT project. In Section F.4, we present TransTOOL which contains the editor, the parser, the dependence analysis tool, and an optimization kernel. Section F.5 presents two research directions in the TransTOOL optimization kernel and Section F.6 offers some conclusions and ideas for future work.

F.2 Previous Work

A lot of work has been done since the early eighties around tools for supercomputer programming. These early tools need to be enhanced to follow the development of parallel computing. In this section, we present some tools for the parallelization of applications written in Fortran 77 and HPF. For a comprehensive survey of HPF tools, see [33].

One of the first projects around an “intelligent” editor for the parallelization of applications written in Fortran77 was the ParaScope editor (PED) from Rice University [29]. PED was designed for shared memory machines. It has been built from different other projects in Rice like $\mathbb{R}_n$, PFC, and PTOOL. PED allowed the interrogation of a dependence graph whose size was limited by some filtering information. Several optimizations were added to the tool like loop restructuring, loop parallelization, dependence deletion and memory optimization. PED has been using an incremental analysis to update its text and dependences panes.

The D-Editor [23] has been partly developed from PED also at Rice University. This editor has been designed for the parallelization of applications written in Fortran D, a data-parallel language which turned out to be a major input to the design of HPF. The D-Editor contains an interprocedural analysis tool, the Fortran D compiler and tools for automatic data distribution, data-race detection, static performance estimation, and performance profiling. The graphical display is derived from PED and contains five panes : an overview pane provides a summary of the loops and subroutines in the program (loops which restrict parallelism are highlighted) ; a dependence pane displays the data dependences carried on the selected loop ; the communication pane displays all the communications associated with the selected loop ; the data layout pane displays the data decomposition information for each array of the loop ; and finally the source pane shows the actual program code. The performance analysis environment Pablo has also been integrated within the D-Editor [2].

The Vienna FORTRAN Compilation System (VFCS) [37] is a source-to-source compilation system based on Vienna Fortran, another extension set for Fortran, similar to FortranD and HPF. It contains a compiler, an interactive performance estimator P3T [13], a performance measuring system (VFPMS), and a knowledge based tool, eXPert Advisor (XPA) to help the user to insert the directives [5].

ForgeExplorer is an interactive parallelizer based on an interactive source code browser. It also performs loop level transformations.

The Annai tool environment [14], developed by CSCS in a collaboration with NEC, uses MPI as the communication interface for various distributed memory plateforms : NEC Cenju-3, Cray T3D, Intel Paragon, and Unix multiprocessor/networked workstations. It consists of an extended HPF compiler (with extensions for irregularly structured computations), a parallel debugger and performance monitor and analyzer, designed with important feedback from application developers.

The Computer Aided Parallelization Tools (CAPTools [24]) is a recent set of tools developed at the University of Greenwich. This tool can show dependences using a graphical display [31]. The user can interact inside the parallelization process by giving information about values or ranges of variables, by "deleting" dependences, and so on. It has a Partitioner window to partition his code and data structures, a communication browser to see the generated communication routines calls. The code generated by CAPTools is written in F77 and explicit communications routines calls.

Some other important compilation projects exist like SUIF and Paradigm.

The SUIF compiler [4], developed by the Stanford Compiler Group, is an infrastructure designed to support collaborative research in optimizing and parallelizing compilers. Independently developed compilation passes work together by using a common intermediate format to represent programs. The system consists of a core library, support libraries, and a variety of passes.

The PARADIGM (PARAllelizing compiler for DIstributed-memory General-purpose Multicomputers) project [6], in the Center for Reliable and High-Performance Computing at the University of Illinois at Urbana-Champaign, strives to provide a fully automated means to parallelize programs for efficient execution on a wide range of distributed-memory multicomputers. To achieve this goal, the research being performed in the PARADIGM project addresses automatic techniques for exploiting available data and task parallelism for both regular and irregular applications. Through a combination of compile-time techniques and run-time library support, the PARADIGM framework will be able to compile a wide range of applications for execution on a variety of high-performance multicomputers.

F.3 The HPFIT Project

The development of a "real" application is conducted in a cycle involving several steps. This cycle is shown on Figure F.1.

Tools can be designed to help users especially during the distribution phase. The programmer must find a "good" distribution of data structures. This distribution should allow for the most parallelism and should reduce the number and volume of communications to its minimum. This can be evaluated using various tools (monitoring tools, profilers). From this distribution, the user is able to write HPF directives in the declaration part of the code. These distributions should be propagated inside the routines. They can sometimes differ and the programmer is allowed to redistribute the data inside the code. The code is then compiled to obtain a source code in Fortran 77 with message-passing calls. It is sometimes possible to modify the resulting code to insert optimizations. The code can then be executed or simulated. If traces have been generated, the user can have an idea of the behavior of his code. The quality of the distribution (and the optimizations) can be improved. This cycle can be executed several times to obtain the best performance. One question is : how portable is the resulting code ?

One problem with HPF is that a HPF compiler is not required to follow the user's advices (stated as directives). This can be a problem because the user can have a false view of his code, "corrupted" by the compiler. That is why we think that a strong interaction between the compiler and the parallelization tool is necessary.

The aim of the **HPFIT**[1] project is to provide a set of interactive tools integrated in a single environment

[1]**H**igh **P**erformance **F**ortran **I**ntegrated **T**ools.

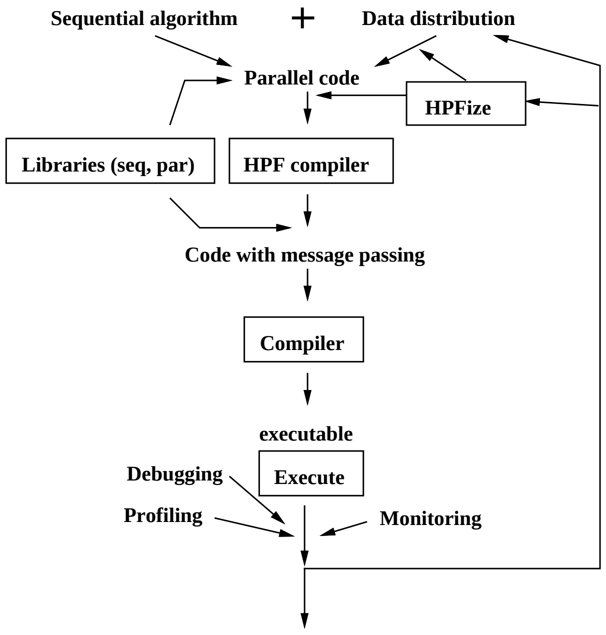


FIG. F.1 – Development cycle of an application on distributed memory platforms.

to help users to parallelize scientific applications to be run on distributed memory platforms. This tool will also be used as an interface to a large number of existing tools like HPF compilers, computation libraries, simulators, data visualization tools, monitoring systems, profilers, and so on. These tools will interact in a coherent environment. This environment should allow scientists with sequential source codes to produce good parallel versions. Furthermore HPFIT will enable users to control the parallelization of their application, in order to make it even better.

HPFIT will not be a black box taking a sequential application as input and magically producing an efficient parallel application as output. Rather, HPFIT will be a tool environment to support and ease the development, tuning and maintenance of HPF applications. Though it is intended to be an open environment that allows integration of new tools, we focus in the first version on the following items :

– source editing with analysis information,
– dependence analysis of particular loop nests,
– automatic detection of `INDEPENDENT` loops,
– automatic detection of pipelined computations patterns and code generation,
– support for data mapping (adding data distribution directives),
– visualization and evaluation of data mappings,
– interfaces to existing parallel libraries (e.g. ScaLAPACK, ...),
– interfaces to HPF compilers,
– simulation, monitoring, performance analysis and interface with profiling tools,
– compilation and execution interface.

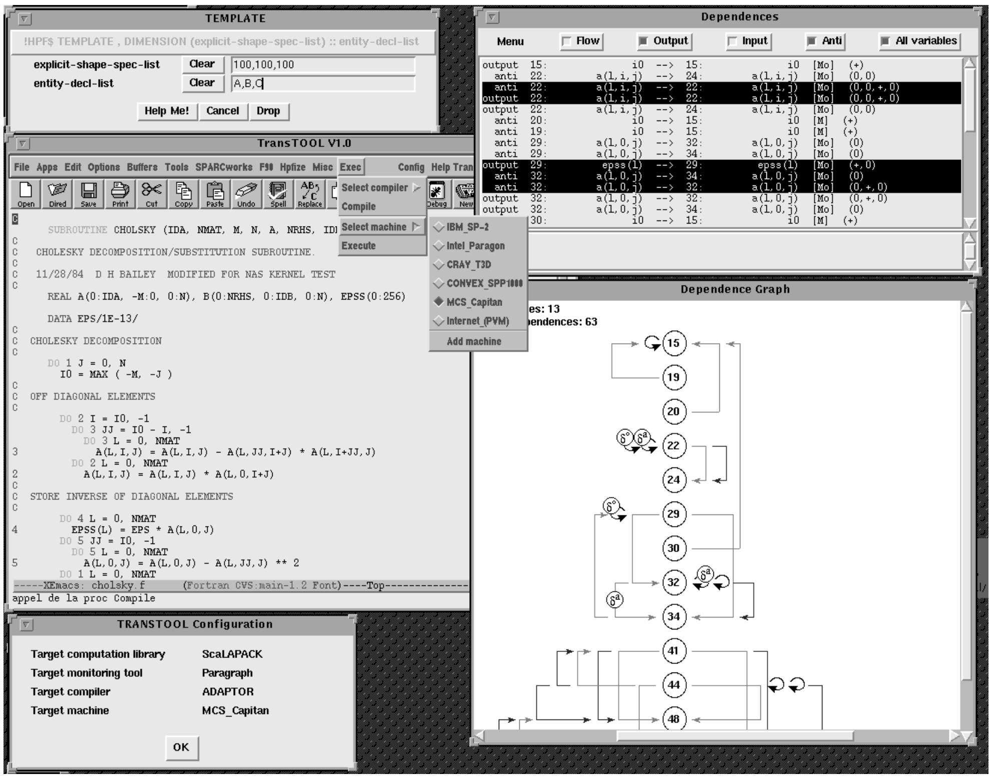
The parallelized code is a data parallel program with explicit data mapping and explicit data parallelism. As the data parallel paradigm might not be the most efficient one in some situations, we will provide a library interface to codes written in other languages or other parallel programming styles, in particular message passing libraries, and efficient parallel computation libraries.

As most applications considered for parallelization are written in Fortran, and a lot of work has been done for automatic parallelization of this language, our target language is HPF. Since HPF provides the `EXTRINSIC` mechanism, we can interface it with existing parallel libraries and other programming styles. Some HPF compilers are being released. These compilers are designed by software companies (like pghpf from PGI, DEC and IBM HPF compilers, ...) or universities (like ADAPTOR from GMD/SCAI [12], VFCS from the University of Vienna, sHPF from the University of Southampton, Fortran D from Rice University, PARADIGM from the Center for Reliable and High-Performance Computing at the University of Illinois at Urbana-Champaign, HPFC from the Ecole des Mines de Paris, ...). HPFIT should allow the user to use any of these HPF or HPF-like compilers, and all those to come, even if most of them will not allow functionalities like monitoring, and translation of sequential library calls. We intend to make use of this variety of possible "post-compilers" in order to run the HPF codes on nearly all kinds of architectures. For instance the ADAPTOR compilation system not only generates message passing code for MIMD system with distributed memory, but can also generate code for MIMD systems with shared or distributed shared memory. The first version of HPFIT (Version 1.0) will support two compilers : ADAPTOR from GMD/SCAI and pghpf from Portland Group.

The HPFIT project is based on several other projects developed in different universities. At the moment, 4 research groups have decided to work on the project and to design shared interfaces. These groups are the **LIP** in Lyon, France, the **LaBRI** in Bordeaux, France, the **LIFL** in Lille, France, and the **GMD/SCAI** in Bonn, Germany. The developed tools can be used either in a stand-alone fashion or within the HPFIT interface.

F.4 TransTOOL

TransTOOL [15], developed at the LIP, is the kernel of the HPFIT project. At the moment, it contains a powerful editor (XEmacs), the F77 parser (from the **ADAPTOR** compiler developed at the GMD/SCAI lab.), the dependence analyzer (**Petit** at the University of Maryland) and an optimization kernel. At the moment, this kernel allows to do some parallelism detection and optimizations of pipelined computations (see Section F.5). TransTOOL provides an interface to be able to get the results of the parsing and of the dependence analysis. Figure F.2 gives a snapshot of the TransTOOL screen.



explicit-shape-spec-list	100,100,100
entity-decl-list	A,B,C

```
output  15:         i0  -->  15:        i0  [Mo]  (+)
  anti  22:   a(l,i,j)  -->  24:  a(l,i,j)  [Mo]  (0,0)
  anti  22:   a(l,i,j)  -->  22:  a(l,i,j)  [Mo]  (0,0,+,0)
output  22:   a(l,i,j)  -->  22:  a(l,i,j)  [Mo]  (0,0,+,0)
output  22:   a(l,i,j)  -->  24:  a(l,i,j)  [Mo]  (0,0)
  anti  20:         i0  -->  15:        i0  [M]  (+)
  anti  19:         i0  -->  15:        i0  [M]  (+)
  anti  29:   a(l,0,j)  -->  32:  a(l,0,j)  [Mo]  (0)
  anti  29:   a(l,0,j)  -->  34:  a(l,0,j)  [Mo]  (0)
output  29:    epss(l)  -->  29:   epss(l)  [Mo]  (+,0)
  anti  32:   a(l,0,j)  -->  34:  a(l,0,j)  [Mo]  (0)
  anti  32:   a(l,0,j)  -->  32:  a(l,0,j)  [Mo]  (0,+,0)
output  32:   a(l,0,j)  -->  32:  a(l,0,j)  [Mo]  (0,+,0)
output  32:   a(l,0,j)  -->  34:  a(l,0,j)  [Mo]  (0)
        30:         i0  -->  15:        i0  [M]  (+)
```

```
es: 13
endences: 63
```

```
C
      SUBROUTINE CHOLSKY (IDA, NMAT, M, N, A, NRHS, IDI
C
C   CHOLESKY DECOMPOSITION/SUBSTITUTION SUBROUTINE.
C
C   11/28/84  D H BAILEY  MODIFIED FOR NAS KERNEL TEST
C
      REAL A(0:IDA, -M:0, 0:N), B(0:NRHS, 0:IDB, 0:N), EPSS(0:256)

      DATA EPS/1E-13/
C
C  CHOLESKY DECOMPOSITION
C
      DO 1 J = 0, N
        I0 = MAX ( -M, -J )
C
C  OFF DIAGONAL ELEMENTS
C
        DO 2 I = I0, -1
          DO 3 JJ = I0 - I, -1
            DO 3 L = 0, NMAT
3             A(L,I,J) = A(L,I,J) - A(L,JJ,I+J) * A(L,I+JJ,J)
          DO 2 L = 0, NMAT
2           A(L,I,J) = A(L,I,J) * A(L,0,I+J)
C
C  STORE INVERSE OF DIAGONAL ELEMENTS
C
        DO 4 L = 0, NMAT
4         EPSS(L) = EPS * A(L,0,J)
        DO 5 JJ = I0, -1
          DO 5 L = 0, NMAT
5           A(L,0,J) = A(L,0,J) - A(L,JJ,J) ** 2
        DO 1 L = 0, NMAT
-----XEmacs: cholsky.f      (Fortran CVS:main-1.2 Font)----Top-------------
appel de la proc Compile
```

TRANSTOOL Configuration

Target computation library	ScaLAPACK
Target monitoring tool	Paragraph
Target compiler	ADAPTOR
Target machine	MCS_Capitan

FIG. F.2 – Snapshot of TransTOOL.

F.4.1 The XEmacs Editor

The sequential program source is displayed by an XEmacs editor. Using a powerful editor enables us to share previously developed modes, to let the user configure its editor with his preferences (by using his regular .emacs file). The TransTOOL edition is suited to the target language using a modified F90 mode. HPF keywords are highlighted. The user has the opportunity to click on a program component to trigger some actions (like choosing a loop nest for the dependence analysis).

F.4.2 HPFize

Numerous applications have been developed on sequential, vector or parallel machines. These codes have to be modified to be executed on distributed memory machines. Writing a program in HPF consists in including compilation directives into the source code. It is interesting to simplify the way the user inserts such directives. This is achieved using a graphical environment that is able to get the necessary pieces of information from the user and from the program itself (after parsing and dependence analysis).

To summarize, the first functionality of this part of TransTOOL is to help the user to insert HPF basic components into his "old Fortran" source code (and give defaults values as much as possible) : let the user insert, with some assistance, directives and constructions like template, processor, align, distribute, forall and calls to intrinsic procedures.

The main research topic around the semi-automatic insertion of HPF directives is the automatic distribution of matrices using dependence analysis, previous distributions and target machine parameters.

F.4.3 Dependence Analysis

Dependence analysis is a crucial part of the semi-automatic parallelization of a code. Many tools for dependence analysis have been designed and Petit [27] is one of them. Petit is a version of Michael Wolfe's Tiny tool extended by the Omega Project at the University of Maryland. This tool uses the Omega library [28] to compute the dependences.

We use Petit via its batch interface to compute the dependences of selected loop nests. The user can choose a loop nest and ask for the dependences. Then, a graphical interface allows the user to see the dependences, to select some dependences according to some criterion (for example, choosing only the flow dependences), to see the sink and target of dependences on the editor.

When a loop nest is chosen, the corresponding sub-program is rebuilt from the Abstract Syntax Tree and transformed into the Petit language using f2p.

Figure F.2 shows some of the dependence analysis windows of TransTOOL.

F.4.4 Parsing and Unparsing

ADAPTOR (Automatic Data Parallelism Translator) is a public domain compilation system developed at GMD for compiling data parallel HPF programs to equivalent message passing programs [12]. The compiler tools used for ADAPTOR can be retrieved and used to build other tools. In TransTOOL, we use the front end which is able to parse a Fortran 77 source file, to generate the Abstract Syntax Tree (AST) and to unparse the AST in a output file. We use the AST and the unparsing functionality to build the sub-programs for the dependence analysis. If the source file is an HPF source code, the directives are also in the AST. We will use them for the semi-automatic parallelization.

We also use ADAPTOR for the compilation of the code generated by HPFIT.

F.4.5 Translation of Calls to Sequential Libraries

Many real applications are currently using sequential libraries like BLAS or LAPACK. These libraries are available on many existing machines. Parallel versions of these libraries are available, like the PBLAS [26] and ScaLAPACK [25]. They are highly optimized, and reach very high efficiencies close to peak performance together with a good scalability. It is impossible to reach the same efficiency using usual compilers. A problem appears when one wants to transform a source code into HPF : these libraries are added during the link phase, and thus their source code is not available for automatic parallelization during the upstream operations. Moreover, even if we have the source of the sequential routine, its automatic parallelization will lead to poor performance. If a parallel version of the library exists, we must use it for the parallelization of the application.

Some work have been done around HPF interfaces of parallel libraries [11, 32]. This is the best way to obtain a portability between HPF compilers and parallel libraries. One of the goals of TransTOOL is to offer the users the opportunity to automatically translate library calls from sequential to parallel versions (via their HPF interfaces). Furthermore, TransTOOL will allow the user to insert redistribution phases if it appears to be necessary. It will be a semi-automatic translation linked with the compilation. Moreover, TransTOOL will allow to insert the source code of subroutines which do not have their parallel implementation. Then the source will be "HPFized" and distribution will be inherited. One problem is that HPF handles many more distributions than those supported by ScaLAPACK. This will imply the development of conversion routines.

F.4.6 Execution Interface

TransTOOL has been designed to be a self-contained environment. From the XEmacs editor, the user should be able, taking a sequential Fortran 77 code, to semi-automatically generate an HPF source code, to compile this code and to execute the SPMD program on the different machines he has access to. To this purpose, we have designed an interface to give the parameters of the machines (how to start a computation on the machine, how you allocate nodes, and so on), to compile the HPF code by choosing among available HPF compilers, and to start the program on a remote machine.

F.4.7 Developer's Toolkit

HPFIT will provide a standard interface to many other tools used in the parallelization of applications like performance monitors, traces analyzers, and simulation tools. In this first version, HPFIT is not a totally new environment built from scratch but an "intelligent" interface into which existing or new tools are being plugged. Thanks to this interfacing, we will be able to add new tools as they appear.

The TransTOOL Developer's Toolkit (T^3) [20] is the set of interfaces which can be used to build new tools from TransTOOL, or to integrate new functionalities inside the editor. Currently, the Toolkit has interfaces to :

– the XEmacs editor,
– the parser,
– the dependence analysis tool.

These interfaces are written with either C, TCL or Lisp, so they can be used in a C program, a tcl script or within the XEmacs editor.

The first version (V 1.0) of TransTOOL and its developer's Toolkit is available on the Web[2].

F.4.8 Other Tools

HPFbuilder [18] from the LIFL which allows the user to insert HPF distribution directives using a graphical interface will be integrated soon.

F.5 Optimization Kernel

Our main interest in TransTOOL is to validate recent research results on real applications, and to be able to integrate the corresponding (limited size) software developments within existing, more complete and more powerful tools.

[2]URL `http ://www.ens-lyon.fr/~desprez/FILES/RESEARCH/SOFT/TransTOOL/`

In this section, we present two research topics, i.e. parallelism detection for automatic generation of `independent` directives, and the pipelined loops detection for the automatic generation of calls to the LOCCS library.

F.5.1 Parallelism Detection in Nested Loops

One of the objectives of the TransTOOL project is to develop and integrate strategies for transforming automatically (or semi-automatically) sequential Fortran pieces of codes into codes with HPF directives. The goal is to help the programmer to recognize parallelism at the loop level, and to automate the corresponding loops transformations for him.

Since our target language is HPF, we have to keep in mind that only transformations that can be expressed in HPF and that can be efficiently compiled by an HPF compiler are suitable. In particular, we do not currently address the following topics : parallelism exploited in doacross loops, software pipelining, minimization of synchronization barriers : the first two topics because such parallelism cannot be easily and efficiently implemented in a data-parallel language like HPF (it is easier to exploit it when *compiling* HPF), the third topic because HPF codes are usually compiled into SPMD codes, synchronized by nature.

Our main goal is to expose to the programmer the maximal parallelism that can be detected. We are interested only in understanding if a large set of independent computations can be detected, and if they can be described by parallel loops. In other words, we aim at detecting loops that, in HPF, can be preceded by the directive `!HPF$ independent` (denoted by DOPAR in the pseudo-code below).

F.5.1.1 Fine-grain Parallelism

In many applications, there is no need to use sophisticated dependence analysis techniques and parallelization algorithms for detecting full parallelism. A simple algorithm such as Allen and Kennedy's algorithm [3] is sufficient for most of the loops. Our implementation of Allen and Kennedy's algorithm will be used as a comparison base to evaluate how often more sophisticated algorithms are needed. In this context, we recently showed that, as long as dependence level is the only information available, Allen and Kennedy's algorithm detects maximal parallelism (see [16]).

In some loops however, some more accurate representation of dependences is needed. Techniques based on the hyperplane method [30] have been developed in the past so as to exploit a more accurate description of dependences such as the description by direction vectors (see Wolf and Lam's algorithm [36]). This last algorithm is able to take into account the information given on all components of distance vectors (which is not possible with Allen and Kennedy's algorithm and level of dependences), but it is not able to use the information concerning the structure of the dependence graph (which is the basis of Allen and Kennedy's algorithm for applying loop distribution).

We thus proposed a novel algorithm, Darte and Vivien's algorithm, that combines and subsumes both algorithms [17]. We found that this algorithm exploits optimally the structure of the graph and the information on direction vectors. It is even optimal for a more accurate representation of dependences that we called PRDG (polyhedral reduced dependence graph), roughly speaking, approximations of dependences by non parameterized polyhedra, defined by vertices, rays and lines.

F.5.1.2 Medium-grain Parallelism

In HPF, codes with single innermost parallel loops are often not parallel enough to offer good performance. In this case, the grain of parallelism must be increased, either by trying to move up the parallel loop to the outermost possible level, or by using blocking (tiling) techniques.

We studied this tiling problem in [7] in the simple case of uniform loop nests, and it turns out that Darte and Vivien's algorithm can be easily adapted to the tiling technique, as Wolf and Lam's algorithm that was developed with a "tiling spirit". Actually, the detection of parallel loops and the detection of maximal tiling, related to maximal sets of permutable loops, are two equivalent problems.

We are currently implementing in TransTOOL, a tiling version of Darte and Vivien's algorithm : it informs the programmer of the maximal parallelism he can hope, and proposes loop transformations that reveal maximal parallelism, either as fine-grain parallelism, or as medium-grain parallelism.

A lot of problems remain to be solved such as the choice of the block size for tiling, the choice of a suitable mapping (with possibly temporary arrays), . . . As the reader can notice, the above algorithms are able to generate `!HPF$ independent` directives, but they do not address the generation of directives such as align or distribute. This is still left to the programmer : he has to choose the mapping that exploits at best the parallelism that has been detected.

We conclude this section by a very simple example, that illustrates the type of codes that can be generated by our parallelism detection algorithm. Figure F.3(a) shows the original code, and Figure F.3(b) the code with fine-grain parallelism. Note that Allen and Kennedy's algorithm would also find one parallel loop in this example. However, the fact that loop distribution can be avoided cannot be found by Allen and Kennedy's algorithm.

```
DO i = 1, n
 DO j = 1, n
  a(i, j) = a(i - 1, j + 1) + b(i - 1, n)
  b(i, j) = a(i, j - 1) + b(i - 1, j - 1)
 ENDDO
ENDDO
```

(a)

```
DO i = 1, n
 b(i, 1) = a(i, 0) + b(i - 1, 0)
 DOPAR j = 2, n
  a(i, j - 1) = a(i - 1, j) + b(i - 1, n)
  b(i, j) = a(i, j - 1) + b(i - 1, j - 1)
 ENDDOPAR
 a(i, n) = a(i - 1, n + 1) + b(i - 1, n)
ENDDO
```

(b)

FIG. F.3 – Original code and code with fine-grain parallelism

F.5.2 Detection of Pipelined Loops and Code Generation

Parallel distributed memory machines improve performance and memory capacity but their use adds an overhead due to the communications. To obtain programs that perform and scale well, this overhead must be minimized. Part of the job is devoted to communication libraries, which should provide efficient point-to-point and macro-communications. Another important issue is to "hide" communication as much as possible, by overlapping them with independent communications.

Asynchronous communications can be used to overlap computations and communications. The call to the communication routine (send or receive) is then issued as soon as possible in the code. A wait routine will then be used to check for the completion of the communication. Unfortunately, this is not always legal due to the dependences between computations and communications. Pipeline schemes are also sometimes found within the code. These schemes lead to a sequentiality in the execution of the whole algorithm.

The optimization we have added is what we call *Macro-pipeline Overlap*. There is a sequentiality within the code (see Figure F.4 (A)). Processor P1 must wait for processor P0 to complete his computation and send the results, to receive the data and start to work. As soon as it has finished, it sends the results to processor P2 which, in turn, starts to work on the received data. The total execution time is higher than the sequential one because of the overhead of the communications. One first solution is to start the communications as soon as possible, i.e. as soon as one processor has computed one data item. For each data item computed, an other one is sent to the following processors so they can start as soon as possible. This is called a fine-grain pipeline ((B) on Figure F.4). This solution adds an overhead because of the communication startup time. This time is usually higher than the cost of the communication of one element. Thus, the total time can be higher than the one without pipelining. A trade-off has to be found which minimizes the execution time. This is a coarse grain pipeline ((C) on Figure F.4).

A typical example of a code that may benefit from a macro-pipeline optimization is the ADI algorithm given on Figure F.5.

We have designed a library for the optimization of pipelined computations called the LOCCS [19, 22] [3]. This library has been integrated in the ADAPTOR compiler [9] and we are currently working on its integration within TransTOOL.

[3]Low Overhead Communication and Computation Subroutines.

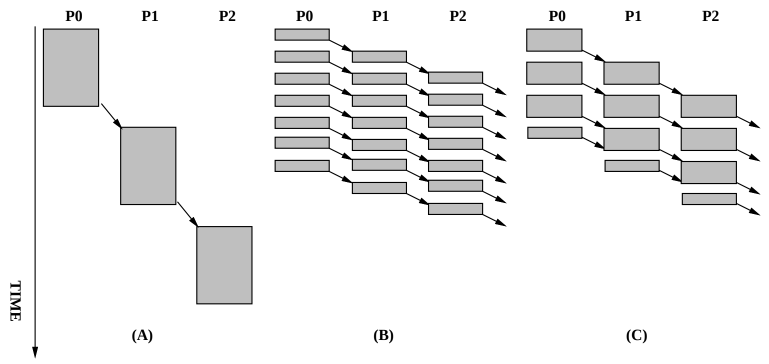


FIG. F.4 – Macro-pipeline.

```
      PARAMETER (N=...)
      REAL, DIMENSION (N,N)  :: A, B
!HPF$ DISTRIBUTE (*,BLOCK) :: A, B
      ...
!     sweep along the columns
      DO I = 2, N
        DO J = 1, N
          A(I,J) = A(I,J) - A(I-
1,J)*B(I,J)
        END DO
      END DO
!     sweep along the rows
      DO J = 2, N
        DO I = 1, N
          A(I,J) = A(I,J) - A(I,J-
1)*B(I,J)
        END DO
      END DO
```

FIG. F.5 – High Performance Fortran Version of the ADI algorithm.

Within ADAPTOR, the LOCCS library consists in a driver routine that takes as parameters information about the distributed matrices, the distributed dimension(s) and a routine which is called at each step of the macro-pipeline. Within the driver routine, choices are made to use macro-pipelining or not, and also on the way of doing this optimization. There are several reasons to use a library instead of generating the code directly in the SPMD source code. The first one is the ease of use for the programmer of an HPF compiler. Instead of generating several lines of code, the compiler only has just to generate a subroutine call, to fill the parameters and to generate the computation routine. Another reason is to be able to perform run-time optimizations like, for example, the dynamic computation of the optimal grain size as a function of the network load, cache effects, and so on.

We have obtained very good results using this library in the ADAPTOR compiler, for example with the ADI algorithm given in Figure F.5. There are two strategies to solve this problem, one using a redistribution (transposition) and the other one using our library to have an optimized pipelined execution. The pipelined execution achieves nearly the optimal speed-up and a dynamic data remapping is not necessary in this case.

```
      PARAMETER (N=...)
      REAL, DIMENSION (N,N)  :: A, B
!HPF$ DISTRIBUTE (*,BLOCK) :: A, B
      ...
      DO I = 2, N    ! parallel execution
         DO J = 1, N
            A(I,J) = A(I,J) - A(I-
1,J)*B(I,J)
         END DO
      END DO

      CALL DALIB_LOCCS_DRIVER (BLOCK, 2, 0,
           A(:,2:N), [0,1], B(:,2:N), [0,0])
      ...

      EXTRINSIC (HPF_LOCAL) SUBROU-
TINE BLOCK (A, B)
      REAL A(:,:), B(:,:)
!HPF$ DISTRIBUTE *(*,BLOCK) :: A, B
      DO J=lbound(A,2),ubound(A,2)
        DO I=lbound(A,1),ubound(A,1)
           A(I,J) = A(I,J) - A(I,J-
1)*B(I,J)
        END DO
      END DO
      END
```

FIG. F.6 – ADI algorithm using the LOCCS library.

Other applications of the library will be given in [10].

Now we need to integrate the LOCCS inside TransTOOL. First we need to find what Tseng called *Cross Processor Loops* (CP loop) in [35]. A loop is a CP loop if it has a true dependence carried by the loop and of course if its iteration crosses the processors boundaries. If a loop is a CP loop, one processor needs the results of the computation of its left or right neighbor to start to work. However, telling that a loop is a CP loop is not sufficient to say that a macro-pipeline execution is efficient. We are working on an algorithm that detects loops that can benefit from a macro-pipeline execution.

For the computation of the optimal granularity of the pipeline, we will use the OPIUM library [21]. The granularity will be also tuned at run-time depending of cache effects or network trafic (this has also been suggested in [1] and [34]).

The user will be able to give to TransTOOL the parameters of the target machine (parameters of a communication, costs of an average computation). These parameters will be used for the optimization of the code generation. For example, when using PVM on a network of workstations connected via Ethernet, no macro-pipeline should be used because of the huge costs of communication startups.

F.6 Conclusion and Future Work

In this paper, we have presented the first versions of the HPFIT project and of TransTOOL. HPFIT will provide one interface to many other tools used in the parallelization of applications like performance monitors, traces analyzers, and simulation tools. In this first version, HPFIT is not a totally new environment built from scratch but an “intelligent” interface into which existing or new tools are being plugged. Thanks to this interfacing, we will be able to add new tools as they appear.

Data distribution and alignment is one of the most important problem for the parallelization of applications using HPF. It is known to be a NP-complete problem in most cases ; however for classical problems, heuristics can be found that will lead to good performance. Thus we need to add a tool for semi-automatic data distribution (within HPFize). Another problem that has already been raised by Kennedy et al. in [23] is an interaction between the HPF compiler and the editor. This is not a trivial work as the compiler can make huge transformations to obtain an SPMD code with local arrays and calls to communications routines.

Converting dusty F77 to Fortran 90 seems to be a useful intermediate step in the parallelization process. For example, data parallelism could be expressed by array syntax and FORALL loops (Fortran 95). Part of this work is clearly not our job (cleaning F77) but we could add some fonctionnalities to integrate F90 constructs in the HPFize part of TransTOOL, by taking those loops nests that can be transformed.

A lot of work remains to be done in the field of tools for semi-automatic parallelization of applications. We hope that a collaboration between several laboratories will lead to an interesting and performant tool made available to the whole community.

Acknowledgments

We would like to thank Gilles Lebourgeois, Olivier Reymann, Georges-André Silber, Lionel Tricon and Julien Zory for their work within the TransTOOL project, and the referees for their helpful remarks and corrections on the paper.

F.7 References

[1] V.S. Adve, C. Koelbel, and J. Mellor-Crummey. Compiler Support for Analysis and Tuning Data Parallel Programs. Technical Report CRPC-TR95520, Center for Research on Parallel Computation, Rice University, March 1995.

[2] V.S. Adve, J.C. Wang, J. Mellor-Crummey, D.A. Reed, M. Anderson, and K. Kennedy. An Integrated Compilation and Performance Analysis Environnement for Data Parallel Programs. Technical Report CRPC-TR94513-S, Center for Research on Parallel Computation, Rice University, December 1994.

[3] J.R. Allen and K. Kennedy. Automatic Translations of Fortran Programs to Vector Form. *ACM Toplas*, 9 :491–542, 1987.

[4] P. Amarasinghe, J.M. Anderson, M.S. Lam, and C.-W. Tseng. The SUIF Compiler for Scalable Parallel Machines. In *Seventh SIAM Conference on Parallel Processing for Scientific Computing*, February 1995.

[5] S. Andel, B.M. Chapman, J. Hulman, and H.P. Zima. An Expert Advisor for Parallel Programming Environments and Its Realization within the Framework of the Vienna Fortran Compilation System. Technical report, Institute for Software Technology and Parallel Systems, Vienna, Austria, 1996.

[6] P. Banerjee, J.A. Chandy, M. Gupta, E.W. Hodges-IV, J.G. Holm, A. Lain, D.J. Palermo, S. Ramaswamy, and E. Su. The PARADIGM Compiler for Distributed-Memory Multicomputers. *IEEE Transactions on Computers*, 28(10) :37–47, October 1995.

[7] Pierre Boulet, Alain Darte, Tanguy Risset, and Yves Robert. (pen)-Ultimate Tiling ? *Integration, the VLSI Journal*, 17 :33–51, 1994.

[8] T. Brandes, S. Chaumette, M.-C. Counilh, A. Darte, F. Desprez, J.C. Mignot, and J. Roman. HPFIT : A Set of Integrated Tools for the Parallelization of Applications Using High Performance Fortran : Part II : Data Structures Visualization and HPF Extensions for Irregular Problems. In J.J. Dongarra and B. Tourancheau, editors, *Third Workshop on Environments and Tools for Parallel Scientific Computing*, Faverges, August 1996. SIAM.

[9] T. Brandes and F. Desprez. Implementing Pipelined Computation and Communication in an HPF Compiler. In *Europar'96 Parallel Processing*, volume 1123 of *Lecture Notes in Computer Science*, pages 459–462. Springer Verlag, August 1996.

[10] T. Brandes and F. Desprez. Implementing Pipelined Computation and Communication in an HPF Compiler. Technical report, LIP - ENS Lyon, 1996.

[11] T. Brandes and D. Greco. Realization of an HPF interface to ScaLAPACK with Redistributions. In H. Liddel, A. Colbrook, B. Hertzberger, and P. Sloot, editors, *High-Performance Computing and Networking (HPCN)*, volume 1067 of *Lectures Notes in Computer Science*, pages 834–839. Springer Verlag, 1996.

[12] Th. Brandes and F. Zimmermann. ADAPTOR - A Transformation Tool for HPF Programs. In K.M. Decker and R.M. Rehmann, editors, *Programming Environments for Massively Parallel Distributed Systems*, pages 91–96. Birkhäuser, April 1994.

[13] B.M. Chapman, T. Fahringer, and H.P. Zima. Automatic Support for Data Distribution on Distribution on Distributed Memory Multiprocessor Systems. Technical Report TR 93-2, Institute for Software Technology and Parallel Systems, University of Vienna, Austria, August 1993.

[14] C. Clémançon, A. Endo J. Fritsher, A. Müller, R. Rühl, and B.J.N. Wylie. The "Annai" Environment for Portable Distributed Parallel Programming. In *28th Hawaii International Conference on System Sciences (HICSS-28)*, volume II, pages 242–251. IEEE Computer Society Press, January 1995.

[15] A. Darte, F. Desprez, G. Lebourgeois, J.C. Mignot, G.A. Silber, and L. Tricon. TransTOOL Working Note # 1 :TransTOOL : An Interactive Parallelization Tool. Technical report, LIP ENS-Lyon, France, August 1996.

[16] Alain Darte and Frédéric Vivien. On the Optimality of Allen and Kennedy's Algorithm for Parallelism Extraction in Nested Loops. In *Proceedings of Europar'96*, Lyon, France, August 1996. Springer Verlag. To appear.

[17] Alain Darte and Frédéric Vivien. Optimal Fine and Medium Grain Parallelism in Polyhedral Reduced Dependence Graphs. In *Proceedings of PACT'96*, Boston, MA, October 1996. IEEE Computer Society Press. To appear.

[18] J.-L Dekeyser and C. Lefevre. HPF-Builder : A Visual Environment to Transform Fortran 90 Codes to HPF. In J.J. Dongarra and B. Tourancheau, editors, *Third Workshop on Environments and Tools for Parallel Scientific Computing*, Faverges, August 1996. SIAM.

[19] F. Desprez. A Library for Coarse Grain Macro-Pipelining in Distributed Memory Architectures. In *IFIP 10.3 Conference on Programming Environments for Massively Parallel Distributed Systems*, pages 365–371. Birkhaeuser Verlag AG, Basel, Switzerland, 1994.

[20] F. Desprez, J.C. Mignot, G. Lebourgeois, and L. Tricon. *TransTOOL Working Note #4 : T^3 : The TransTOOL Developper's Toolkit Version 1.0.* LIP ENS-Lyon, France, August 1996.

[21] F. Desprez, P. Ramet, and J. Roman. Optimal Grain Size Computation for Pipelined Algorithms. In *Europar'96 Parallel Processing*, volume 1123 of *Lecture Notes in Computer Science*, pages 165–172. Springer Verlag, August 1996.

[22] F. Desprez and B. Tourancheau. LOCCS : Low Overhead Communication and Computation Subroutines. Technical Report 92-44, LIP - ENS Lyon, December 1992.

[23] S. Hiranandani, K. Kennedy, C.-W. Tseng, and S. Warren. Design and Implementation of the D Editor. In J.J. Dongarra and B. Tourancheau, editors, *Second Workshop on Environments and Tools for Parallel and Scientific Computing*, pages 1–10, Townsend, TN, May 1994. SIAM.

[24] C.S. Ierotheou, S.P. Johnson, M. Cross, and P.F. Leggett. Computer Aided Parallelisation Tools (CAPTools) - Conceptual Overview and Performance on the Parallelization of Structured Mesh Codes. *Parallel Computing*, 22 :163–195, 1996.

[25] J.Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley. LAPACK Working Note : ScaLAPACK : A Portable Linear Algebra Library for Distributed Memory Computers - Design Issues and Performances. Technical Report 95, The Universtity of Tennessee - Knoxville, 1995.

[26] J.Choi, J. Dongarra, S. Ostrouchov, A. Petitet, D. Walker, and R.C. Whaley. LAPACK Working Note : A Proposal for a Set of Parallel Linear Algebra Subprograms. Technical Report 100, The Universtity of Tennessee - Knoxville, 1995.

[27] W. Kelly, V. Maslov, W. Pugh, E. Rosser, T. Shpeisman, and D. Wonnacott. *New User Iinterface for Petit and Other Extensions*. CS Dept, University of Maryland, April 1996. http ://www.cs.umd.edu/projects/omega.

[28] W. Kelly, V. Maslov, W. Pugh, E. Rosser, T. Shpeisman, and D. Wonnacott. *The Omega Library - Version 1.0 - Interface Guide*. CS Dept, University of Maryland, April 1996. http ://www.cs.umd.edu/projects/omega.

[29] K. Kennedy, K.S. McKinley, and C.-W. Tseng. Interactive Parallel Programming Using the ParaScope Editor. *IEEE Transactions on Parallel and Distributed Systems*, 2(3) :329–341, July 1991.

[30] Leslie Lamport. The Parallel Execution of DO Loops. *Communications of the ACM*, 17(2) :83–93, February 1974.

[31] P.F. Leggett, A.T.J. Marsh, S.P. Johnson, and M. Cross. Integrating User Knowledge with Information from Parallelization Tools to Facilitate the Automatic Generation of Efficient Parallel FORTRAN Code. *Parallel Computing*, 22 :259–288, 1996.

[32] P.A.R. Lorenzo, A. Muller, Y. Murakami, and B.J.N. Wylie. High Performance Fortran Interfacing to ScaLAPACK. Technical Report TR-96-13, Swiss Center for Scientific Computing (CSCS), Manno, Switzerland, 1996.

[33] J.-L. Pazat. Tools for High Performance Fortran : A Survey. Technical report, IRISA, Rennes, France, 1996. http ://www.irisa.fr/pampa/HPF/survey.html.

[34] B.S. Siegel and P.A. Steenkiste. Controlling Application Grain Size on a Network of Workstations. In *Supercomputing'95*, 1995. http ://www.cs.cmu.edu/afs/cs/project/nectar/WWW/gnectar_papers.html.

[35] C.-W. Tseng. *An Optimizing Fortran D Compiler for MIMD Distributed-Memory Machines*. PhD thesis, Rice University, January 1993.

[36] M.E. Wolf and M.S. Lam. A Loop Transformation Theory and an Algorithm to Maximize Parallelism. *IEEE Transactions on Parallel and Distributed Systems*, 2(4) :452–471, 1991.

[37] H. Zima and B. Chapman. Compiling for Distributed Memory Systems. Technical report, Institute for Statitics and Computer Science, Vienna, Austria, May 1994.

Chapitre **G**

Mixed Parallel Implementations of the Top Level Step of Strassen and Winograd Matrix Multiplication Algorithms

Conference : International Parallel and Distributed Processing Symposium (IPDPS'01)

Date : 2001

Authors :

Frédéric Desprez and Frédéric Suter

ReMaP/LIP
UMR CNRS - ENS Lyon - INRIA 5668
Ecole Normale Supérieure de Lyon
46, Allée d'Italie, F-69364 Lyon Cedex 07
`(desprez,fsuter)@ens-lyon.fr`

Abstract :

This paper presents parallel implementations of the top level of Strassen and Winograd algorithms for matrix multiplication that use mixed-parallelism, i.e., simultaneous exploitation of data- and task-parallelism. This paradigm allows a better task placement and reduces the communication costs. A comparison with the ScaLAPACK implementation of the matrix multiplication is given. We present a theoretical evaluation of the algorithms which is corroborated by experiments.

Keywords :

Mixed-parallelism, matrix multiplication, Strassen,Winograd.

Thanks :

This paper is partly supported by INRIA, CNRS, ENS Lyon and the ARC INRIA OURAGAN.

G.1 Introduction

Parallel scientific applications can be divided in two major classes : *data-* and *task-parallel* applications. The former consists in applying the same operation in parallel on different elements of a data set, while the later is defined to be concurrent computations on different data sets. These two classes can be combined to get a simultaneous exploitation of data- and task-parallelism, so called *mixed-parallelism*. In mixed-parallel applications, several data-parallel computations can be executed concurrently in a task-parallel way. Mixed-parallelism programming employs a M-SIMD (Multiple SIMD) or M-SPMD style. It is the combination of both task-parallelism (MIMD or MPMD) and data-parallelism (SIMD or SPMD). The exploitation of mixed-parallelism has many advantages. One of them is the ability to increase scalability because it allows the use of more parallelism when the maximal amount of data- or task-parallelism that can be exploited is reached. A good overview of this topic is given in [2]. Most of the researches about the simultaneous exploitation of data- and task-parallelism have been done in the area of programming languages to give simple high level accesses to more parallelism and in the area of compilers, where problems such as scheduling and allocation of concurrent data-parallel tasks are studied [15]. In [14], Ramaswamy introduces the Macro Dataflow Graph (MDG) structure to describe mixed-parallel programs. A MDG is an direct acyclic graph where nodes represent sequential or data-parallel computations and edges represent precedence constraints, with two distinguished nodes, one preceding and one succeeding all other nodes. Once the MDG is extracted from the code, an algorithm is applied to simultaneously place and schedule tasks on the computing resources. Here, we use a sequential language like C or FORTRAN77 with high-performance libraries like ScaLAPACK [5] and its associated communication library, the BLACS. Available processors are explicitly divided into sub-grids that receive data-parallel operations to execute.

Matrix multiplication is the kernel of many scientific applications [12, 16] and several parallel implementations have been proposed, most of them using data-parallelism. Some algorithms substitute multiplications by additions and thus reduce the number of multiplications computed. Strassen [17] and Winograd [8] are such algorithms that are best suited for a practical implementation. They have been extensively studied on monoprocessor machines to increase the computational performances of numerical applications [1, 5, 10, 11, 18]. Several parallel implementations of both algorithms have already been proposed but most of them use either data-parallelism [9] or task-parallelism [6]. Mixed-parallel versions [7, 14] also exist. The former [7] uses a monodimensional distribution of the matrices and the later [14] is an application of an automatic parallelization tool. In this paper, we consider an application of mixed-parallelism to the first level of recursion of Strassen and Winograd matrix multiplication algorithms. Strassen (or Winograd) algorithm is used at the higher level while standard high-performance level 3 PBLAS kernels are used at the lower level.

Our motivation is to build efficient parallel algorithms for client-server applications. We assume that matrices are distributed on disjoint grids of processors because of previous computations. A and B are two square matrices of dimension M distributed on two separate square grids of processors of same size. We want to compute the product $C = AB$ with C distributed on the same grid as A. In such a case, there are two "classical" ways to compute this product. First, we can redistribute B on the grid where A is located and then compute the product on this grid. But this way, we use half of the amount of available processors. Then, we can redistribute both A and B on all processors, compute the product, and then redistribute C on the grid where A is located. We propose a third and mixed way that keeps all matrices in place and evenly distributes tasks (e.g., additions[1] and multiplications on matrix quarters) on the two grids.

The remainder of this paper is organized as follows. Section G.2 recalls Strassen and Winograd algorithms. In Section G.3, we present our algorithms and their issues related to matrices distribution, temporary arrays reduction, and task placement. Section G.4 gives an evaluation of the theoretical costs of the different algorithms. Finally and before a conclusion, we give, in Section G.5, our experimental results on a cluster of PCs connected trough a Myrinet network.

[1]In the following, we denote either additions or subtractions by addition.

G.2 Strassen and Winograd algorithms

In 1969, Strassen [17] introduced an algorithm to multiply $M \times M$ matrices which has a lower complexity than the classical $O(M^3)$ (Figure G.1(top)). It is based on a scheme for the product of two 2×2 matrices which involves 7 multiplications and 18 additions instead of the usual 8 multiplications and 4 additions. This scheme can be easily applied to 2×2 block matrices. To compute the product $C = AB$, if A and B are distributed in square blocks of dimension $M/2$, we have :

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

Strassen algorithm can also be applied recursively on square matrices of dimension $M = 2^i$ to finally obtain a complexity of $O(M^{\log(7)}) = O(M^{2.807})$. Several other variations of this algorithm allow to handle matrices of arbitrary size, most of them are referenced or detailed in [11].

The Winograd variant of Strassen Algorithm, introduced in [8], uses the same number of multiplications but reduces the number of additions from 18 to 15. This algorithm is presented in Figure G.1(bottom).

Input : Matrices A, B

$T_1 = A_{11} + A_{22}$		$T_6 = B_{11} + B_{22}$
$T_2 = A_{21} + A_{22}$		$T_7 = B_{12} - B_{22}$
$T_3 = A_{11} + A_{12}$	**Phase 1**	$T_8 = B_{21} - B_{11}$
$T_4 = A_{21} - A_{11}$		$T_9 = B_{11} + B_{12}$
$T_5 = A_{12} - A_{22}$		$T_{10} = B_{21} + B_{22}$
$Q_1 = T_1 * T_6$		$Q_5 = T_3 * B_{22}$
$Q_2 = T_2 * B_{11}$	**Phase 2**	$Q_6 = T_4 * T_9$
$Q_3 = A_{11} * T_7$		$Q_7 = T_5 * T_{10}$
$Q_4 = A_{22} * T_8$		
$T_1 = Q_1 + Q_4$		$C_{11} = T_1 - T_2$
$T_3 = Q_3 + Q_1$	**Phase 3**	$C_{12} = Q_3 + Q_5$
$T_2 = Q_5 - Q_7$		$C_{21} = Q_2 + Q_4$
$T_4 = Q_2 - Q_6$		$C_{22} = T_3 - T_4$

Output : $C = (C_{ij})$

Input : Matrices A, B

$T_1 = A_{21} + A_{22}$		$T_5 = B_{12} - B_{11}$
$T_2 = S_1 - A_{11}$	**Phase 1**	$T_6 = B_{22} - T_1$
$T_3 = A_{11} - A_{21}$		$T_7 = B_{22} - B_{12}$
$T_4 = A_{12} - S_2$		$T_8 = B_{21} + T_2$
$Q_1 = A_{11} * B_{11}$		$Q_5 = T_3 * T_7$
$Q_2 = A_{12} * B_{21}$	**Phase 2**	$Q_6 = T_4 * B_{22}$
$Q_3 = T_1 * T_5$		$Q_7 = A_{22} * T_8$
$Q_4 = T_2 * T_6$		
$T_1 = Q_1 + Q_4$		$C_{11} = Q_1 + Q_2$
$T_2 = Q_2 + Q_5$	**Phase 3**	$C_{12} = T_3 + Q_6$
$T_3 = T_1 + Q_3$		$C_{21} = T_2 + Q_7$
		$C_{22} = T_2 - Q_3$

Output : $C = (C_{ij})$

FIG. G.1 – First level of recursion of Strassen algorithm (top) and its Winograd variant (bottom).

G.3 Mixed parallel algorithms

To benefit from the use of mixed-parallelism in Strassen algorithm, the strategy employed keeps matrices in place and distributes tasks among all of the processors instead of aligning matrices before computing. As we use standard parallel numerical routines from ScaLAPACK, we need to keep the data distribution imposed by these kernels (full-block or block-cyclic distributions). Our goal is thus to reduce the communications and to equilibrate as much as possible the computations among the processors. The two grids (or contexts) where matrices A and B are distributed are square and of dimension $p = 2^i$. We consider that they are sub-grids of a virtual rectangular grid of size $p \times 2p$ (thus we have a total of $2p^2$ processors). Processors of this global grid are row major numbered from $[0, 0]$ in the upper left corner to $[p-1, 2p-1]$ in the lower right. We will keep this numbering from the whole context until the end of this paper.

G.3.1 Strassen algorithm

G.3.1.1 Data distribution

Strassen algorithm is better than standard matrix multiplication algorithm because additions and subtractions of matrices can be computed in linear time without communications. But if we use a *full-block* distribution, it introduces communications. To avoid them, all the processors involved in the computation must own a part of each matrix quarters. The bidimensional *block-cyclic* distribution is the most adapted, but the block size parameter has to be carefully chosen. $M/2p$ is the maximum block size that allows the local computation of all of the accumulations. Figure G.2 shows an example of the chosen data distribution when $p = 2$. For each block, subscript gives its matrix quarter and superscript corresponds to the block-cyclic distribution.

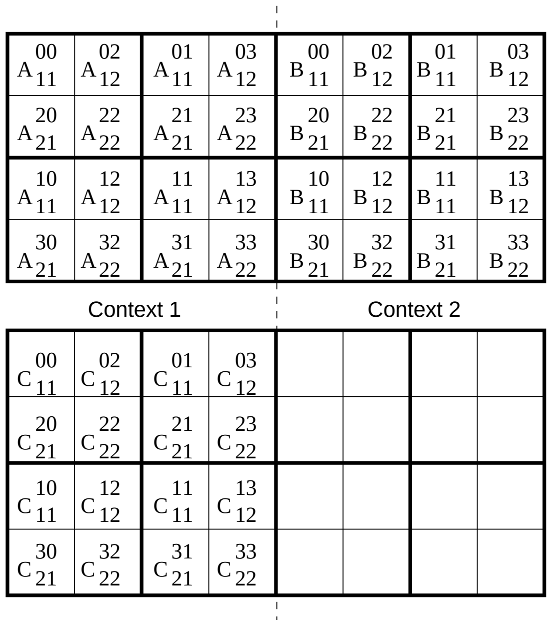

A_{11}^{00} A_{12}^{02} A_{11}^{01} A_{12}^{03} B_{11}^{00} B_{12}^{02} B_{11}^{01} B_{12}^{03}
A_{21}^{20} A_{22}^{22} A_{21}^{21} A_{22}^{23} B_{21}^{20} B_{22}^{22} B_{21}^{21} B_{22}^{23}
A_{11}^{10} A_{12}^{12} A_{11}^{11} A_{12}^{13} B_{11}^{10} B_{12}^{12} B_{11}^{11} B_{12}^{13}
A_{21}^{30} A_{22}^{32} A_{21}^{31} A_{22}^{33} B_{21}^{30} B_{22}^{32} B_{21}^{31} B_{22}^{33}
Context 1
Context 2
C_{11}^{00} C_{12}^{02} C_{11}^{01} C_{12}^{03}
C_{21}^{20} C_{22}^{22} C_{21}^{21} C_{22}^{23}
C_{11}^{10} C_{12}^{12} C_{11}^{11} C_{12}^{13}
C_{21}^{30} C_{22}^{32} C_{21}^{31} C_{22}^{33}

FIG. G.2 – Mapping of matrices A, B and C on a 2×4 processor grid.

G.3.1.2 Temporary allocation and memory usage

The results of the 18 additions and the 7 multiplications of the Strassen algorithm must be stored in temporary variables. If we consider the algorithm presented in Figure G.1(top), 10 temporary variables are needed in phase 1, 7 in phase 2, and 4 in phase 3. As we compute the product of matrices of size M on square grids of size $p \times p$, all these temporary variables are of dimension $M/2p$. So, this algorithm needs $21(M/2p)^2$ temporary elements on each processor.

Now we propose an optimization of the number of these temporary variables in which we need only two temporary variables (R^1 and T^1) on context 1 and three temporary variables (R^2, S^2 and T^2) on context 2. In Figure G.3, we propose an implementation of Strassen algorithm with this optimization. This algorithm needs five temporary variables of size M/p in addition to the three matrices A, B, and C. But since the temporary variables allocated in one context are not declared by processors of the other context, this algorithm needs, at least, $3(M/p)^2$ temporary elements on each processor.

Input : Matrix A, temporary variables R^1, T^1

Store to :	**Computation**	**Algorithmic var.**
C_{11}	$A_{11} + A_{22}$	T_1
R^1_{22}	$A_{21} + A_{22}$	T_2
R^1_{11}	$A_{11} + A_{12}$	T_3
R^1_{12}	$A_{21} - A_{11}$	T_4
R^1_{21}	$A_{12} - A_{22}$	T_5
	Send(R^1)	
T^1	Receive (R^2)	
R^1_{11}	$C_{11} * T^1_{11}$	Q_1
R^1_{21}	$A_{11} * T^1_{12}$	Q_3
R^1_{12}	$A_{22} * T^1_{21}$	Q_4
C_{11}	$R^1_{11} + R^1_{12}$	T_1
	Send(R^1_{11}, R^1_{21})	
$C_{12} + C_{22}$	Receive (S^2_{12}, S^2_{22})	
C_{11}	$C_{11} - C_{12}$	C_{11}
C_{21}	$C_{22} + R^1_{12}$	C_{21}
$C_{12} + C_{22}$	Receive (S^2_{11}, S^2_{21})	C_{12} & C_{22}

Output : $C = (C_{ij})$

Input : Matrix B, temporary variables R^2, S^2, T^2

Store to :	**Computation**	**Algorithmic var.**
R^2_{11}	$B_{11} + B_{22}$	T_6
R^2_{12}	$B_{12} - B_{22}$	T_7
R^2_{21}	$B_{21} - B_{11}$	T_8
S^2_{11}	$B_{11} + B_{12}$	T_9
S^2_{12}	$B_{21} + B_{22}$	T_{10}
	Send(R^2)	
T^2	Receive (R^1)	
R^2_{11}	$T^2_{11} * B_{22}$	Q_5
R^2_{12}	$T^2_{12} * S^2_{11}$	Q_6
R^2_{21}	$T^2_{21} * S^2_{12}$	Q_7
S^2_{22}	$T^2_{22} * B_{11}$	Q_2
S^2_{12}	$R^2_{11} - R^2_{21}$	T_2
	Send(S^2_{12}, S^2_{22})	
$T^2_{11} + T^2_{21}$	Receive (R^1_{11}, R^1_{21})	
S^2_{11}	$T^2_{21} + R^2_{11}$	C_{12}
S^2_{21}	$T^2_{21} + T^2_{11} - S^2_{22} + R^2_{12}$	C_{22}
	Send (S^2_{11}, S^2_{21})	

FIG. G.3 – Mixed Strassen algorithm for Context 1 (left) and 2 (right).

G.3.1.3 Task placement

Once contexts are defined, tasks from the MDG have to be assigned as evenly as possible. In phase 1, all operations concern only A or B. To keep locality, these tasks are mapped close to the data used. In phase 2, each multiplication involves one matrice coming from context 1 and one matrice coming from context 2. When a computation needs data that are not distributed on the right context, copies of missing data from one context to another have to be performed. As we work on an virtual grid of homogeneous processors, the choice of the locations of the computations is driven by the reduction of communication cost.

Figure G.4(top) shows where allocations and copies take place in Strassen MDG. White circles represent tasks executed on context 1, shaded circles represent tasks executed on context 2 and dotted lines mean that there is a copy needed.

In order to reduce the communication overhead, we group messages together. For example, between phases 1 and 2, we send R^2 in whole but only three quarters are needed. This avoids complicated accesses in non contiguous memory locations and thus, reduces the number of communication latencies.

G.3.2 Winograd algorithm

The main idea of the mixed-parallel implementation is kept here for the Winograd variant. We use the same contexts and the same data distribution presented in Section G.3.1.1. The number of temporary variables is also the same. However, Winograd MDG (Figure G.4(bottom)) allows a task placement that improves locality, especially during Phase 3. Computations are closer to data than in Strassen algorithm. So there are less communications. There are only two exchanges of data between the two contexts (one more copy in Strassen). Both algorithms for context 1 and 2 are presented in Figure G.5.

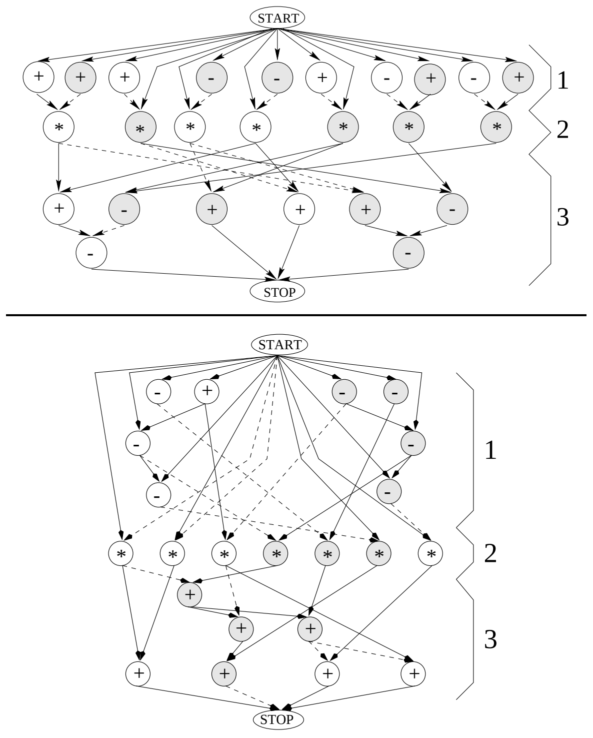


FIG. G.4 – Strassen (top) and Winograd (bottom) MDGs with repartition of allocations and copies.

Input : Matrix A, temporary variables R^1, T^1

Store to :	Computation	Algorithmic var.
C_{11}	$A_{21} + A_{22}$	T_1
R^1_{11}	$C_{11} - A_{11}$	T_2
R^1_{12}	$A_{11} - A_{21}$	T_3
R^1_{21}	$A_{12} - R^1_{11}$	T_4
	Send(R^1)	
T^1	Receive (R^2)	
R^1_{11}	$A_{11} * T^1_{11}$	Q_1
R^1_{21}	$C_{11} * T^1_{21}$	Q_3
R^1_{12}	$A_{12} * T^1_{12}$	Q_2
R^1_{22}	$A_{22} * T^1_{22}$	Q_7
	Send(R^1_{11}, R^1_{21})	
$C_{12} + C_{22}$	Receive (S^2_{12}, S^2_{22})	$C_{12} + T_2$
C_{11}	$R^1_{11} + R^1_{12}$	C_{11}
C_{21}	$C_{22} + R^1_{22}$	C_{21}
C_{22}	$C_{22} + R^1_{21}$	C_{22}

Output : $C = (C_{ij})$

Input : Matrix B, temporary variables R^2, S^2, T^2

Store to :	Computation	Algorithmic var.
R^2_{11}	B_{11}	
R^2_{12}	B_{21}	
R^2_{21}	$B_{12} - B_{11}$	T_5
S^2_{11}	$B_{22} - R^2_{21}$	T_6
S^2_{12}	$B_{22} - B_{12}$	T_7
R^2_{22}	$B_{21} - S^2_{11}$	T_8
	Send(R^2)	
T^2	Receive (R^1)	
R^2_{11}	$T^2_{11} * S^2_{11}$	Q_4
R^2_{12}	$T^2_{12} * S^2_{12}$	Q_5
R^2_{21}	$T^2_{21} * B_{22}$	Q_6
$T^2_{11} + T^2_{21}$	Receive (R^1_{11}, R^1_{21})	
S^2_{11}	$T^2_{11} + R^2_{11}$	T_1
S^2_{22}	$S^2_{11} - R^2_{12}$	T_2
S^2_{21}	$S^2_{11} + T^2_{21}$	T_3
S^2_{12}	$S^2_{21} + R^2_{21}$	C_{12}
	Send(S^2_{12}, S^2_{22})	

FIG. G.5 – Mixed Winograd variant of Strassen algorithm for Context 1 (left) and 2 (right).

G.4 Time cost models

In this section, we evaluate the theoretical costs of Strassen, Winograd and PDGEMM algorithms with different redistributions using the whole grid or only one sub-grid : mixed-parallel Strassen, mixed-parallel Winograd, PDGEMM on the whole grid with ScaLAPACK redistribution, PDGEMM on the whole grid with an optimized redistribution, and PDGEMM on one sub-grid with ScaLAPACK redistribution.

G.4.1 Parallel machine and basic operations

G.4.1.1 Inter-context communications

Our communication model is the classical $\beta + L\tau$ where β is the network latency, L the message size, and τ the inverse of the network bandwidth. As our network has a switch, we do not have to take contention into account.

G.4.1.2 Computation routines

In [11], a classification of operations involved in Strassen algorithm is given because each operation may have a different execution speed. We can distinguish two main operations : matrix multiplication and matrix addition. As our implementation uses message grouping, we introduce a supplementary basic copy operation. Let ρ_m, ρ_a, and ρ_c be the costs of elementary operations (respectively multiplication, addition, and copy). As these factors are architecture-dependent, they are determined experimentally. Since our block size is carefully chosen, we do not have cache effects. Now, we give detailed models for each of the basic operations involved in our different algorithms.

Matrix multiplication In [4], a model is given for the ScaLAPACK function `PDGEMM` used in our algorithm to compute the seven matrix multiplications at the lower level. We have the following cost model :

$$T_{Mult} = \frac{2M^3}{pq}\rho_m + M^2(\tau_q^p + \tau_p^q) + \frac{2M}{R}(\beta_q^p + \beta_p^q),$$

where R is the block size, and τ_q^p and β_q^p (resp. τ_p^q and β_p^q) are functions of the grid topology and the communication pattern for bandwidth and latency in a row (resp. column) broadcast.

Matrix addition (resp. copy) Because of our data distribution, all additions (resp.copies) are executed locally without communications as explained in Section G.3.1.1. The cost of an addition (resp. copy) is given by :

$$T_{Add} = \frac{M^2}{p^2}\rho_a \text{ (resp. } T_{Copy} = \frac{M^2}{p^2}\rho_c).$$

G.4.2 Mixed parallel Strassen algorithm

To perform inter-context communications, we use locally blocking send and receive functions. Because the two sub-grids have same size and same shape, these copies induce communications between processor pairs. Processor $[MyRow, MyCol]$ from context 1 exchanges data with processor $[MyRow, MyCol + p]$ in context 2. We have two Send/Receive phases in each context and one communication from the second context to the first one. We compute the total amount of data communicated : $L = \frac{7}{2p^2}M^2$ leading to a communication cost model equal to :

$$T_{Redist} = 5\beta + \frac{7\tau}{2p^2}M^2.$$

Then, there are 7 multiplications, 3 on the first context and 4 on the second one. As these products work on matrices of size $M/2$, we have :

$$\begin{aligned} T_{Mult} &= 4\left(\tfrac{2\rho_m}{p^2}\left(\tfrac{M}{2}\right)^3 + 2\left(\tfrac{M}{2}\right)^2\tau_p^p + 2\tfrac{M/2}{M/2p}\beta_p^p\right) \\ &= \tfrac{M^3}{p^2}\rho_m + 2M^2\tau_p^p + 8p\beta_p^p. \end{aligned}$$

Finally, there are 18 additions which are evenly distributed on both contexts. The critical path has 10 additions and 4 multiplications tasks. So, assuming that all operations work on matrices of size $M/2$, we get :

$$T_{Add} = 10\left(\frac{\left(\frac{M}{2}\right)^2}{p^2}\rho_a\right) = \frac{5M^2}{2p^2}\rho_a.$$

If we sum these costs, we get the following cost model T_S for Strassen algorithm :

$$T_S = \frac{M^3}{p^2}\rho_m + \left(\frac{5\rho_a + 7\tau}{2p^2} + 2\tau_p^p\right) M^2 + 8p\beta_p^p + 5\beta.$$

G.4.3 Mixed parallel Winograd variant

In the Winograd variant of Strassen algorithm, we introduce copies as we use message grouping. The other basic operations are the same as in Strassen version. The only changes are the number of additions and their order. If we look at both contexts algorithms (Figure G.5 (left and right)), we can define the following critical path : 4 additions and 2 copies on context 2, 4 multiplications on context 1, 4 additions on context 2, and finally 3 adds/subtracts on context 1. Concerning communications, a better data placement reduces the global cost. The total amount of data communicated (L) is equal to $\frac{3M^2}{p^2}$, leading to the following communication cost model :

$$T_{Redist} = 4\beta + \frac{3M^2}{p^2}\tau.$$

We have the same number of multiplications as in Strassen. We also have 11 additions and 2 copies. So, assuming that all operations work on matrices of size $M/2$, we get :

$$T_{Add} = \frac{11M^2}{4p^2}\rho_a \text{ and } T_{Copy} = \frac{M^2}{2p^2}\rho_c.$$

If we sum these costs, we get the following cost model T_W for Winograd algorithm :

$$\begin{aligned} T_W = {} & \tfrac{\rho_m}{p^2}M^3 + \left(\tfrac{11\rho_a}{4p^2} + \tfrac{\rho_c}{2p^2} + \tfrac{3\tau}{p^2} + 2\tau_p^p\right) M^2 \\ & +8p\beta_p^p + 4\beta. \end{aligned}$$

G.4.4 Redistribution of A and B and PDGEMM multiplication on all processors

We can split this algorithm in three phases : redistribution of A and B to the global grid, matrix multiplication on all processors, and redistribution of the result C on context 1. Our first idea was to use only ScaLAPACK routines (matrix multiplication and redistribution routines). ScaLAPACK redistribution routine [13] uses a caterpillar algorithm which is efficient for most redistribution patterns. But if we pay attention to source and destination distributions used in our algorithms, we can see an optimization is possible. Indeed, only one half of each matrix, has to be effectively communicated, the other half being copied in a temporary variable. Figure G.6 presents the caterpillar algorithm for a redistribution on a 2×4 processor grid (surrounded communications are effective).

In section G.4.1.2, we gave a model for the matrix multiplication on a $p \times p$ grid. Here we have a $p \times 2p$ grid. Using the same basic operations and communication models as for our algorithm and as the volume of data communicated by each processor is $M^2/2p^2$, we have :

$$\begin{aligned} T_{PDGEMM}^{full\ grid} = {} & \tfrac{M^3}{p^2}\rho_m + M^2\left(\tfrac{3\tau+\rho_c}{2p^2} + \tau_{2p}^p + \tau_p^{2p}\right) \\ & +3\beta + p(\beta_{2p}^p + \beta_p^{2p}). \end{aligned}$$

G.4.5 Redistribution of B and PDGEMM multiplication on one sub-grid

There are two steps in this algorithm : redistribution of B on the same sub-grid as A and multiplication on p^2 processors. Matrix C will be distributed on the same grid as A. To redistribute B, we use the

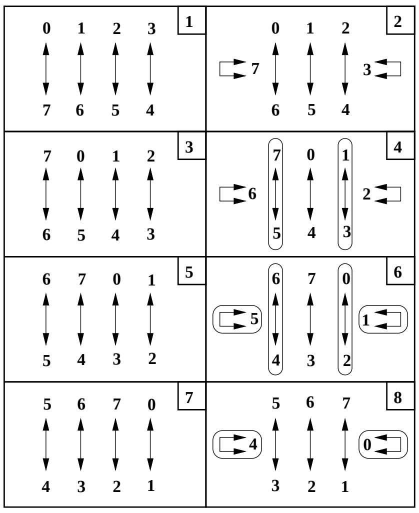


FIG. G.6 – Communication pattern of the caterpillar algorithm.

Algorithm	Computation M^3/p^2	Computation $M^2/2p^2$	Communication M^2	Communication Latency
Strassen	ρ_m	$5\rho_a$	$7\tau/2p^2 + 2\tau_p^p$	$8p\beta_p^p + 5\beta$
			$(7/2p^2 + (2\log p)/p)\tau$	$(8p\log p + 5)\beta$
Winograd	ρ_m	$11\rho_a/2 + \rho_c$	$3\tau/p^2 + 2\tau_p^p$	$8p\beta_p^p + 4\beta$
			$(3/p^2 + (2\log p)/p)\tau$	$(8p\log p + 4)\beta$
PDGEMM$^{full\ grid}$	ρ_m	ρ_c	$3\tau/2p^2 + \tau_p^{2p} + \tau_{2p}^p$	$p(\beta_p^{2p} + \beta_{2p}^p) + 3\beta$
			$(3/2p^2 + (2\log p + 1)/p)\tau$	$(p(2\log p + 1) + 3)\beta$
PDGEMM$^{sub-grid}$	$2\rho_m$	-	$\tau/p^2 + 2\tau_p^p$	$2p\beta_p^p + \beta$
			$(1/p^2 + (2\log p)/p)\tau$	$(2p\log p + 1)\beta$

TAB. G.1 – Summary of computation and communication costs.

same model used to do inter-context copies in Section G.4.4. Each pair of processors uses a Send/Receive operation to communicate M^2/p^2 elements. The cost model for matrix product will be the one given in Section G.4.1.2. Thus, we have :

$$T_{PDGEMM}^{sub-grid} = \frac{2M^3}{p^2}\rho_m + \left(\frac{\tau}{p^2} + 2\tau_p^p\right) M^2 + \beta + 2p\beta_p^p.$$

G.4.6 Analysis of the theoretical costs

Table G.1 gives a summary of the different components of the computation and communication models for our algorithms and PDGEMM. In communication costs, values of τ_p^p, τ_{2p}^p, τ_p^{2p}, β_p^p, β_{2p}^p, and β_p^{2p} can be replaced by their actual values (assuming that we have a broadcast based on a tree), respectively $\log p * \tau/p^2$, $(\log p + 1) * \tau/p$, $(\log p + 1) * \tau/p$, $\log p * \beta$, $(\log p + 1) * \beta$, and $(\log p + 1) * \beta$. For each the new communication costs are given by the second lines of Table G.1

It is easy to see that the $PDGEMM^{sub-grid}$ algorithm can not be as good as the others since its M^3 factor is 2 times bigger. Concerning the 3 other algorithms, the main difference comes from the communication links latency. PDGEMM seems to be better but the volume of communications shows PDGEMM

[2]In this paper, log is $\log_2$

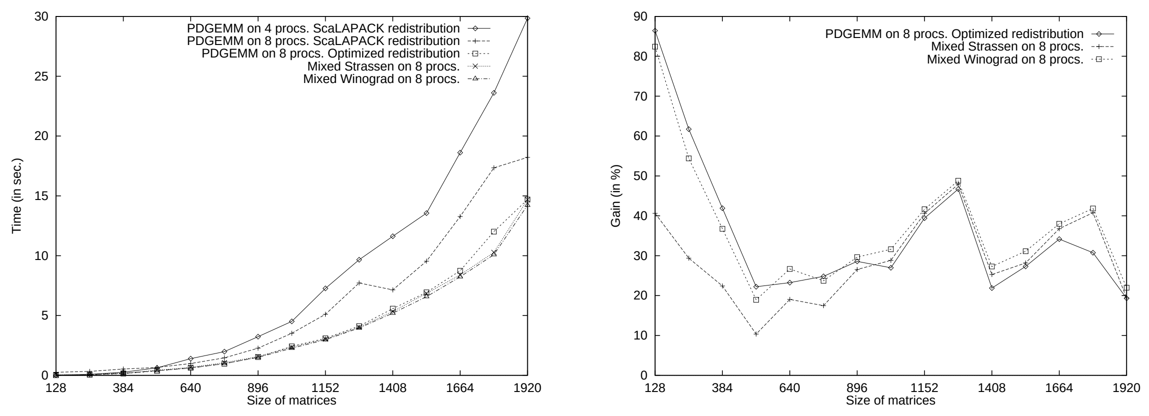


FIG. G.7 – Comparison of the execution time of our different implementations of the matrix product kernel (left) and gain obtained by Strassen, Winograd and PDGEMM with optimized redistribution over PDGEMM with the ScaLAPACK redistribution (Caterpillar) (right).

and Strassen are equivalent while Winograd is slightly better. We can also verify that the better locality in Winograd induces less communication ($1/2 * (M/p)^2$) than Strassen even if it increases the computation cost.

G.5 Experimental results

Our experimentations have been done on a cluster of Pentium processors connected through a Myrinet network. For communications, we used a version of the BLACS library on top of the BIP protocol. The theoretical bandwidth is $125MB/s$ and the latency $6\mu s$. The PBLAS v1.0 library is used for lower level matrix multiplications in Strassen and Winograd algorithms. Here we compare performances of all of the algorithms presented in Section G.4.

Figure G.7(left) corroborates our theoretical analysis. Using ScaLAPACK only (PDGEMM and redistribution), either on the whole grid or one sub-grid, is clearly less efficient than Strassen, Winograd or even a PDGEMM implementation with an optimized redistribution. Notice that the optimized redistribution has to be hand-coded which is not the case of a library implementation of Strassen or Winograd.

Figure G.7(right) presents the gains obtained by our different implementations over the pure ScaLAPACK matrix product on the whole grid.

G.6 Conclusion and future work

In this paper, we have presented a mixed-parallel implementation of Strassen and Winograd algorithms in the scope of client–server applications (i.e., the different data involved in the computations are already distributed on disjoint grids). We chose these algorithms because they are composed of several data-independent tasks which are easier to schedule and place compared to a classical matrix multiplication. After giving details about the proposed mixed-parallel algorithms and their related issues, we gave theoretical models of these algorithms. In order to validate our approach, we compared theoretical models and experimental results of Strassen and Winograd with two data-parallel algorithms using the ScaLAPACK library and experiments corroborated our theoretical analysis.

Our future work consists in adapting our algorithms to heterogeneous platforms. We believe that mixed parallelism techniques are better adapted to clusters of parallel machines than pure data-parallel versions. Separate parallel implementation of a regular matrix product algorithm can be efficiently tuned on every cluster depending of their capacity. The results presented in this paper give us the hope of obtaining good performances on such platforms.

G.7 References

[1] D. Bailey, K. Lee, and H. Simon. Using Strassen's Algorithm to Accelerate the Solution of Linear Systems. *Journal of Supercomputing*, 4(4) :357–371, Jan 1991.

[2] H. Bal and M. Haines. Approaches for Integrating Task and Data Parallelism. *IEEE Concurrency*, Jul-Sep 1998.

[3] L. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. Whaley. *ScaLAPACK Users' Guide*. SIAM, 1997.

[4] E. Caron, D. Lazure, and G. Utard. Performance Prediction and Analysis of Parallel Out-of-Core Matrix Factorization. In *Proceedings of the 7th International Conference on High Performance Computing*, Dec 2000.

[5] S. Chatterjee, A. Lebeck, P. Patnala, and M. Thottethodi. Recursive Array Layouts and Fast Parallel Matrix Multiplication. In *Proceedings of 11th Annual ACM Symposium on Parallel Algorithms and Architectures*, June 1999.

[6] C. Chou, Y. Deng, and Y. Wang. A Massively Parallel Method for Matrix Multiplication Based on Strassen's Method. Technical Report SUNYSB-AMS-93-17, Center for Scientific Computing, Univ. of Stony Brook, Nov 1993.

[7] B. Dumitrescu, J. Roch, and D. Trystram. Fast Matrix Multiplication Algorithms on MIMD Architectures. Technical Report RT-87, LMC-IMAG, INPG Grenoble, France, July 1992.

[8] P. Fischer and R. Probert. Efficient Procedures for Using Matrix Algorithms. In *Automata, Languages and Programming*, volume 14 of Lecture Notes in Computer Science, pages 413–427. Springer-Verlag, Berlin, 1974.

[9] B. Grayson and R. V. de Geijn. A High Performance Parallel Strassen Implementation. *Parallel Processing Letters*, 6(1) :3–12, 1996.

[10] N. Higham. Exploiting Fast Matrix Multiplication Within the Level 3 BLAS. Technical Report TR89-984, Dept of CS - Center of Applied Mathematics - Cornell University, Apr 1989.

[11] S. Huss-Lederman, E. Jacobson, J. Johnson, A. Tsao, and T. Turnbull. Strassen's Algorithm for Matrix Multiplication : Modeling, Analysis, and Implementation. Technical Report CCS-TR-96-147, Center for Computing Sciences, Argonne National Lab., 1996.

[12] B. Kågström, P. Ling, and C. V. Loan. GEMM-Based Level 3 BLAS : High Performance Model Implementations and Performance Evaluation Benchmark. Technical Report UMINF-95.18, Umeå University, Oct 1995.

[13] L. Prylli and B. Tourancheau. Fast Runtime Block Cyclic Data Redistribution on Multiprocessors. *Journal of Parallel and Distributed Computing*, 45, Aug 1997.

[14] S. Ramaswany. *Simultaneous Exploitation of Task and Data Parallelism in Regular Scientific Applications*. PhD thesis, Univ. of Illinois, Urbana-Champaign, 1996.

[15] T. Rauber and G. Rünger. Scheduling of Data Parallel Modules for Scientific Computing. In *Proceedings of 9th SIAM Conference on Parallel Processing for Scientific Computing*, San Antonio, Mar 1999.

[16] W. Rönsch and H. Strauß. The Level 3 BLAS Forms of Parallel Factorization Methods. In D. Evans, G. Joubert, and F. Peters, editors, *Parallel Computing 89*, pages 85–92. Elsevier Science Publisher B.V., 1989.

[17] V. Strassen. Gaussian Elimination Is Not Optimal. *Numerische Mathematik*, 14(3) :354–356, 1969.

[18] M. Thottethodi, S. Chatterjee, and A. Lebeck. Tuning Strassen's Matrix Multiplication for Memory Efficiency. In *Proceedings of Supercomputing'98*, Orlando, Nov 1998.

Chapitre **H**

Scilab to Scilab$_{//}$ – The OURAGAN Project

Journal : Parallel Computing

Date : To appear

Authors :

E. Caron[1], S. Chaumette[2], S. Contassot-Vivier[3], F. Desprez[4], E. Fleury[5], C. Gomez[7], M. Goursat[7], E. Jeannot[5][6], D. Lazure[1], F. Lombard[3], J.-M. Nicod[3], L. Philippe[3], M. Quinson[4], P. Ramet[2], J. Roman[2], F. Rubi[2], S. Steer[7], F. Suter[4], G. Utard[1]

[1] Paladin, LaRIA. 5, rue du Moulin Neuf, F-80 000 Amiens
[2] Alienor, LaBRI. Domaine Universitaire, 351, cours de la Libération, F-33405 Talence cedex
[3] SDRP, LIFC. 16, route de Gray, F-25030 Besançon cedex
[4] ReMaP, LIP-ENS Lyon. INRIA Rhône-Alpes, 46, allée d'Italie, F-69364 Lyon cedex 07
[5] Résédas, LORIA. 615, rue du Jardin Botanique, BP 101, F-54602 Villers-Les-Nancy, cedex
[6] Part of this work has been done while the author was in postdoc at LaBRI.
[7] Métalau, INRIA Rocquencourt. Domaine de Voluceau, BP 105, F-78153 Le Chesnay cedex

Abstract :

In this paper, we present the parallelization of a MATLAB-like tool called SCILAB using high performance numerical libraries and different approaches based either on the duplication of SCILAB processes or using computational servers. This tool allows to perform high level operations on distributed matrices in a metacomputing environment. We also present performance results on different architectures.

Keywords :

SCILAB$_{//}$, Parallel libraries, Computational servers, CORBA, Data Redistribution

Thanks :

This work has been supported in part by the ARC INRIA OURAGAN. URL : http://www.ens-lyon.fr/~desprez/OURAGAN/.

H.1 Introduction

Interactive parallel tools have gained a large interest since the early nineties. Many parallel versions of MATLAB are now available, both in public domain and in the commercial world. SCILAB [21], developed at INRIA in the Métalau project, is a scientific software package for numerical computations in a user-friendly environment. SCILAB is well spread in the scientific community and its popularity has been growing. It is available on several platforms and runs under different types of operating systems (Unix and Unix-like OS, Windows). There are several reasons for its success : (1) the language syntax is simple and easy to learn (MATLAB-like syntax) ; (2) SCILAB includes hundreds of built-in mathematical functions and provides a large choice of built-in libraries : numerical algorithms, control, linear algebra, signal processing, network analysis and optimization, linear system optimization ; (3) it offers a graphical interface ; (4) it includes a high level language with a syntax similar to Fortran 90 for matrix notations. Basic matrix manipulations such as concatenation, extraction or transpose are immediately performed as well as basic operations such as addition or multiplication. SCILAB also allows manipulations of high level data structures such as polynomials, rational numbers, sparse matrices, multi-variables systems, lists, ... In one or two lines of code, this language can express a computation that requires dozens of lines of C or Fortran ; (5) SCILAB can easily be extended with user-developed modules ; (6) SCILAB can easily be interfaced with other languages like C, Fortran or even Maple and Mupad ; (7) SCILAB can generate Fortran programs ; (8) and last but not least, SCILAB is a public domain software.

One possible drawback of using a sophisticated interpreter is that such a language can not give performance as good as classical compiled languages. However, the performance loss (between 1 and 10 times) should be opposed to the ease of development. All the advantages of tools like MATLAB can be found in SCILAB. It is fairly easy to modify the code, change the size of data, print variables, or modify the problem formulation interactively. The prototyping of code is then enhanced by this important feature. Moreover, for coarse grain applications, the interactive aspect of SCILAB is not a limitation and the interpretation overhead remains negligible.

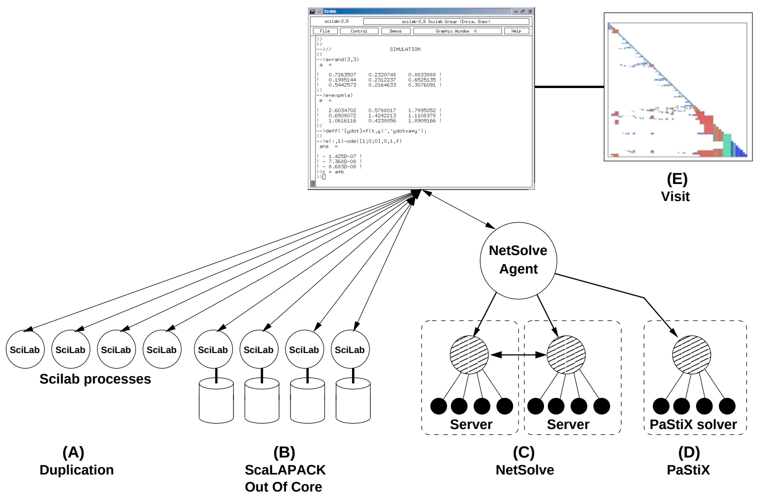
SCILAB should be considered as a "real" language allowing the development of applications. Problems developed by scientists using SCILAB have long execution times and a medium or coarse grain computation. Nowadays, many scientists tend to use a great variety of distributed computing resources such as massively parallel machines, clusters of workstations, SMP machines, and piles of PCs. A SCILAB user who would like to scale his/her application by going to a parallel machine or a network of workstations will not be able to use the SCILAB language and he/she will have to re-program the whole application in C or Fortran. Today's supercomputers still lack of simple user interfaces and access procedures. Parallel computing can then become tremendously tough to use and debug. Moreover, further developments on applications will have to be coded in C or Fortran. Since the investment for researchers or scientists to use the supercomputer facilities in the traditional way is notoriously big, the user has generally to choose between two alternatives : performance (in terms of computational and memory resources) or ease of use.

The idea of providing an access to parallel computing to MATLAB is not new. The first approach consists in compiling MATLAB scripts to an other language (like Fortran [27, 36] or C [19]) and then to apply classical optimizations for its parallelization and use high performance libraries [13, 19, 35]. The advantages of this approach are its high performance and the use of sophisticated compilation methods. However, interactivity is lost and type inference is a tough problem. The second approach keeps MATLAB interactive and provides parallel extensions. There are also two main approaches for the interactive version of high-performance MATLAB tools. The first idea is to duplicate the tool itself (or a part of it) on every node of the target machine [34, 38]. This approach has one advantage : the "master" process just sends regular commands to the "workers" which in turn interpret commands and execute them before sending the result back. Its main drawbacks are of course performance loss during command interpretations, heavy weight processes, and the need of interfacing every library that need to be added to the tool. The second approach is to rely on a parallel library server that waits for commands [11, 25, 31]. Another approach uses mixed compilation and run-time techniques. MATLAB scripts are compiled into an intermediate language which is executed on a virtual machine. This Matlab Virtual Machine from the Match project provides a high performance runtime environment [4].

All these interactive projects use either Matlab duplication or servers. In the OURAGAN project, we aim to offer both approaches. OURAGAN is a join project between several laboratories in France[1] which objective is to bring high performance and memory capacity to SCILAB users. This is a real challenge because we would like to hide as much as possible the use of parallelism to the user.

In fact, we target three different kinds of users. The first one is a parallel computing guru. He/she knows how to write parallel programs using message passing or parallel libraries. He/she wants to keep track of the way data and computations are distributed. For this user, we just provide interfaces to the communication and computation libraries. The second kind of user is a scientist. *"Parallel comput-what ? No way ! I just need a 45 Gflops workstation with 30 GBytes of memory. Could you provide me with such PC ?"*. For this kind, everything is hidden in SCILAB. The tool decides itself whether or not it should (re)distribute the data, start new processes, and so on. This is done in SCILAB by operator overloading. Finally, an intermediate level is provided. This type of user wants to have a transparent access to the libraries as much as possible but is also concerned by performance. He/she has a good knowledge of parallel computing and would like to program his/her applications using message passing and computation libraries, but in a more transparent way.

Figure H.1 shows the overall architecture of SCILAB$_{//}$. Section H.2 presents our first approach which consists in duplicating SCILAB processes on different processors. Then, these processes can exchange messages using either PVM or MPI *(Fig 1-A)*. As we target linear algebra operations, we provide interfaces to parallel libraries like ScaLAPACK [5] and its *out-of-core* prototype *(Fig 1-B)*. Then, in Section H.3, we detail our second approach, which uses computational servers. After presenting the interface between SCILAB and NETSOLVE [10], we deal with our developments to enhance NETSOLVE concerning data persistence, resource location, performance evaluation and communication layers *(Fig 1-C)*. Then, we present two target servers we interfaced with this system : PASTIX *(Fig 1-D)*, which is a parallel direct solver for sparse systems and VISIT *(Fig 1-E)*, a visualization tool for distributed data. Finally, we give a conclusion and present our future work in Section H.4.



```
>>
>>
-->//                    SIMULATION
>>
-->a=rand(3,3)
 a  =

!   0.7263507    0.2320748    0.8833888 !
!   0.1985144    0.2312237    0.6525135 !
!   0.5442573    0.2164633    0.3076091 !
>>
-->e=expm(a)
 e  =

!   2.6034702    0.5788017    1.7895052 !
!   0.6508072    1.4242213    1.1108378 !
!   1.0616116    0.4238856    1.8909166 !
>>
-->deff('[ydot]=f(t,y)','ydot=a*y');
>>
-->e(:,1)-ode([1;0;0],0,1,f)
 ans  =

! - 1.425D-07 !
! - 7.368D-08 !
! - 8.683D-08 !
>>c = a*b
>>
```

FIG. H.1 – The different approaches and developments around SCILAB$_{//}$ in the OURAGAN project.

H.2 SCILAB Processes Duplication

The first approach of our Scilab parallelization allows the user to start other remote SCILAB sessions from the SCILAB window, make them communicate and use parallel numerical libraries.

[1]Supported by INRIA.

H.2.1 Message Passing Interfaces

To be able to use SCILAB as a tool for parallel computing, the first step was to provide a "regular" message passing interface for the user. This was done by including the standard PVM [20] interface within SCILAB. This interface allows users to develop parallel programs and benefit from all the main features of SCILAB that simplify numerical computing (as they were listed in introduction). We choose PVM to implement the first message passing interface since it allows to dynamically spawn new processes which is not the case with MPI [37]. Nevertheless, we also added a message passing interface based on MPI. Using MPI implies that the user has to decide at the beginning of its SCILAB session the maximum number of processes he/she will use.

These "regular" message passing interfaces provide low level functions to get the best performance, but are reserved to "expert users". A SCILAB$_{//}$ instance is able to communicate and interact with other SCILAB$_{//}$ instances and the user can send data of any types (including matrices, lists, functions, ...) using PVM (or MPI) commands. These first low-level interfaces provide a tool to easily run parallel algorithms without loosing the power and ease of SCILAB. Indeed, one SCILAB$_{//}$ instance may send any kind of sub-matrix of a matrix A (not only consecutive blocks) with the following instruction `pvm_send(dest,A(1 :2 :N, :),tag)` or may define a function f and send it to another instance that will be able to execute it on its own data : `deff('[x]=f(y)','x = 1/y'),pvm_send(dest,f,tag)`. Among other performance results, [14] shows that (1) when the user sends a full matrix, performances obtained by SCILAB$_{//}$ are as good as a program written in C, and the interpretation of the call does not deteriorate the performance; (2) an overhead is introduced by sending sub-matrices, due to memory copies that take place in both sender and receiver SCILAB$_{//}$ processes. Indeed, an expression like `send(A(1 :2 :100,2 :2 :100),...)` send a 50×50 matrix which is not contiguous in memory, so the `send()` routine must copy all elements in a contiguous buffer before sending the data.

H.2.2 Parallel Libraries Interfaces

H.2.2.1 Parallel Linear Algebra Package

In order to keep a good portability, interoperability and efficiency, SCILAB$_{//}$ also integrates interfaces to parallel linear algebra libraries like PBLAS, SCALAPACK and the BLACS communication library. Thus, the user may distribute his/her matrices and run parallel routines in order to achieve good performance. As we said above, this level remains dedicated to "expert" users that have good parallel computing skills and that are familiar with the design of the SCALAPACK interface. SCILAB$_{//}$ simplifies calls by enabling default arguments, calling automatically the corresponding complex or double functions by checking the type of parameters, ... The following SCILAB$_{//}$ script illustrates the use of the BLACS routines and gives an example of the function `pblas_gemm` that compute $C = \alpha A * B + \beta C$. This simple example is executed by all SCILAB$_{//}$ processes in a SPMD manner. The following script begins by initializing a 2×2 grid of processors. Once the initialization of each local part of the distributed matrices A, B and C is done, the call to the parallel routine `pblas_gemm` can be executed. The SCILAB$_{//}$ API is very similar to the Fortran one. Note that some parameters are omitted and are set to default values (transposition of matrices, row and column index of the sub-matrix to operate, ...).

```
[mypnum,nprocs] = blacs_pinfo();
blacs_setup(4);
icontxt = blacs_get()
ictxt = blacs_gridinit(icontxt,'R',2,2);
M=1000;K=1000;N=1000;MB = M/2; NB=K/2;
desc_A = sca_descinit(ictxt,M,K,MB,NB,0,0,M);
desc_B = sca_descinit(ictxt,K,N,MB,NB,0,0,K);
desc_C = sca_descinit(ictxt,M,N,MB,NB,0,0,K);
A = rand(M/2,K/2)
B = rand(K/2,N/2)
C = zeros(M/2,N/2)
pblas_gemm(M,N,K,1,"A",desc_A,"B",desc_B,0,"C",desc_C)
```

Nevertheless, some numerical applications are limited by the physical memory size. To break this limit, *out-of-core* techniques may be employed like using disks as an extension of the main memory. So we add an interface to the SCALAPACK *out-of-core* prototype.

H.2.2.2 Out-of-Core Extensions

The benefit of this method is to handle huge matrices on a cheap hardware. As we are developing an *out-of-core* extension of SCILAB$_{//}$, users can work on matrices that do not fit in memory. Depending on the data size, SCILAB$_{//}$ would spawn automatically either the in-core program or the corresponding *out-of-core* program, without script modification.

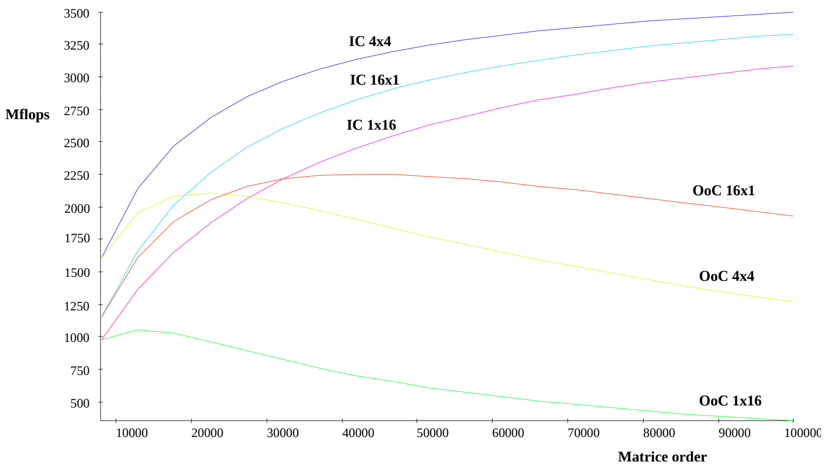


FIG. H.2 – Theoretical performance of the *LU* factorization.

SCALAPACK provides some *out-of-core* functions [18] which we interfaced with SCILAB$_{//}$ by adding a new data type to take matrix distribution over disks into account. The performance of the SCALAPACK out-of-core functions were evaluated before their integration into SCILAB$_{//}$. For instance, Figure H.2 shows theoretical results of the *out-of-core* (OoC) left-right looking algorithm for the *LU* factorization (on a cluster of 16 Celeron PCs with 96 MB/node) and compares it to theoretical performance of the right-looking in-core (IC) algorithm (with no physical memory limit). Performance are shown for 3 kinds of topologies : one row of 16 processors (1×16), a 4×4 grid, and one column of 16 processors (16×1). This figure outlines the impact of distribution on performance. For *out-of-core* computations, the best distribution is a column of processors where the communication overhead of the algorithm is avoided. This *out-of-core* function was also modified to allow the overlap of I/O by computation. Then, the performance is very close to the theoretical performance of the in-core algorithm on a (virtual) machine with no physical memory limit (see [9] for details).

Table H.1 gives performance obtained for the out-of-core *LU* factorization in SCILAB$^{ooc}_{//}$ on an Alpha cluster with 6 processors and 768 MB of memory. The overhead of the interface is negligible. In brief, the startup time relative to the time of the call to the *out-of-core* function by SCILAB$^{ooc}_{//}$ is just some seconds versus several hours or days to execute it.

Unfortunately, only few *out-of-core* routines are provided in SCALAPACK. We developed some original functions like the *out-of-core* identity matrix generator, the *out-of-core* matrix comparison and the *out-of-core* matrix inversion [8].

Whereas good performance is achieved when computing bound operations like matrix factorization, the I/O overhead can not be hidden for element-wise operations of SCILAB. Consider the following expression : `A=sin(A)+cos(B)+sqrt(A)` where `A` and `B` are out-of-core matrices. During the evaluation process, the `A` matrix is read two times, big temporary matrices are generated and re-read, using large space on disk. The I/O cost is greater than computation cost. A solution to reduce I/O cost and avoid generation of big

Matrix order	Matrix size	Execution time	Performance
12288	1.2GB	23m 15s	886 Mflops
21504	3.7GB	1h 09m	1601 Mflops
27648	6.1GB	2h 47m	1406 Mflops

TAB. H.1 – Performance in Mega Flops (Mflops) of SCILAB$_{//}^{ooc}$ *LU* factorization.

temporary matrices is to split out-of-core matrices A and B into small blocks in such a way than the whole expression is evaluated in memory block per block.

H.2.3 Semi-transparent Use of Parallel Computing Using SCILAB$_{//}$

Even if some specialists want to access an expert level, many SCILAB users do not want to spend time learning parallel programming but their main goal still remains to run their programs which become more and more time and memory consuming. In order to provide efficient parallel linear algebra operations inside the SCILAB console but dealing neither with message passing routines nor specific SCALAPACK routines, we decided to add a new distributed type inside SCILAB. This level of transparency is motivated by the need to bring the benefits of interactive environments to supercomputers while maintaining the efficiency and power of highly optimized parallel computational libraries. From the user point of view, the fact that a scalar matrix is distributed or not, will not influence the way of writing SCILAB programs. The only additional commands are :

scip_init : initializes the grid, that is, the number of processors that the user will use for this session. Note that a configuration file can be used to specify the default configuration (number of hosts and name of the computers) ;

scip_init_dist : initiates a specific distribution if the user does not want to use the default one ;

scip_distribute : is the main routine that will distribute a scalar matrix, defined into the SCILAB console or stored on a file system, on the other SCILAB$_{//}$ processes that were previously started by the `scip_init` function.

We overloaded common SCILAB functions and operations so that they work with the distributed type. Thus, operations on distributed matrices will be executed in parallel. At the moment, all operations that have their counterpart in the two parallel linear algebra libraries, PBLAS and SCALAPACK, are overloaded. The main point is that the user is still able to use the SCILAB classical matrix notations and operations to write parallel programs.

Of course, all SCILAB functions working on scalar types are not overloaded. When an operation is intended on a distributed type whereas there is no parallel function corresponding to it, the user may choose between several modes : the first (and simplest) one, is to generate an error ; the second one, is to systematically gather the distributed matrix inside the SCILAB console (if it fits in SCILAB memory) and execute the operation in it. The other case is when an operation involves both scalar and distributed data. Once again, the user may choose between several modes : to generate an error ; to gather the distributed matrix or to propagate the distribution. The last choice enables to only distribute the main matrix once and then, the interpretor will automatically propagate the distribution used to the other data involved in further parallel operations. The following example shows how to perform a very simple matrix multiplication on a $P \times Q$ grid of SCILAB$_{//}$ processes. In this example, the distribution used is a bidimensional block-cyclic one with a block size fixed to $MB \times NB$ but the user may have used default values. Then, scalar matrices A and B are distributed and the matrix multiplication is done by using the regular $*$ symbol inside the SCILAB console.

```
CTXT = scip_init(P,Q);
DIST = scip_init_dist("CC",0,0,CTXT(3),MB,NB);
A = rand(M,N); B = rand(M,N);
MatA = scip_distribute("A", DIST);
MatB = scip_distribute("B", DIST);
```

```
Res = MatA(1:1000,1:500)*MatB(1:500,1:1000);
size(Res)
ans  = !   1000.    1000. !
```

Table H.2 lists functions that support overloading for distributed matrices. As we notice on the example above, setting and retrieving array sections (using only consecutive blocs) also work transparently for distributed matrices. The overloading of major element-wise functions like `cos`, `sin` is done and easy to implement.

Funct.	Description	Funct.	Description
$+, -, *, .*, ./$	Classical matrix binary op.	`size`	Size of an object
`chol`	Cholesky decomposition	`hess`	Hessenberg form
`inv`	Matrix inverse	`linsolve`	Linear eqn. solver
`lu`	LU factor of a Gauss. elim.	`qr`	QR decomp.
`rcond`	Inverse condition number	`schur`	Schur decomp.
`spec`	Eigenvalues	`svd`	Sing. value decomp.

TAB. H.2 – Overloaded distributed matrix operations in SCILAB$_{//}$.

Figure H.3 plots performance obtained using the standard $*$ operator inside the SCILAB console on distributed matrices. The x-axis represents the size of the matrix and the y-axis the time to execute 2 matrix multiplications since we perform `Res=A*B*C` where A, B and C are square matrices. These tests were performed on a SGI Origin2000. The curve plotted with crosses $(- + -)$ is obtained by doing the computation on a single node, i.e., by using the sequential scalar operator $*$ in SCILAB. The curve plotted with $(- \times -)$ is obtained by using 4 processors and finally, the curve plotted with stars $(- * -)$ is obtained by running the test on 16 processors. The main important point is that it is possible to obtain a very good speedup on such a simple operation that appears many times in SCILAB scripts. Thus, it is worthwhile to use the distributed scalar type when dealing with matrix of size greater than 200×200. The overhead introduced by the distributed type, i.e., sending the instructions to the set of slaves, is not really a problem when computation complexity (and memory capacity) becomes the real burden.

H.3 Network-Enabled Servers

In the previous section, we have presented our first apprach to parallelize SCILAB. But this approach is reserved to expert users. In this section, we present a more transparent way to access parallel resources from SCILAB, using computational servers.

Due to the progress in networking, computing intensive problems in several areas can now be solved using networked scientific computing. In the same way that World Wide Web has changed the way that we think about information, we can easily imagine the types of applications we might construct if we had instantaneous access to a supercomputer from our desktop. The RPC approach [8, 9] is a good candidate to build Network-Enabled Servers (NES) environments on the Grid. Several tools that provide this functionnality exist like NETSOLVE [10], NINF [13], NEOS [11], OVM [7] or RCS [2].

This approach leads us to integrate an interface to NETSOLVE which is a client-agent-servers application that enables users to solve complex scientific problems remotely by accessing hardware and software resources distributed across a network. A load-balancing policy is used by NETSOLVE to ensure good performance by enabling the system to use available computational resources as efficiently as possible. The SCILAB-NETSOLVE interface allows the user to send blocking and non-blocking requests to the NETSOLVE agent which plays the role of a resource broker.

Figure H.4 plots performance obtained using non-blocking NETSOLVE routines to solve several eigenvalues problems on a set of matrices. The matrix size was fixed to 500×500. SCILAB was running on a Sparc workstation but one of the server was running on an Origin2000. The x-axis represents the number of calls and the y-axis the time. The curve is linear which in fact corresponds to the time to send and receive the

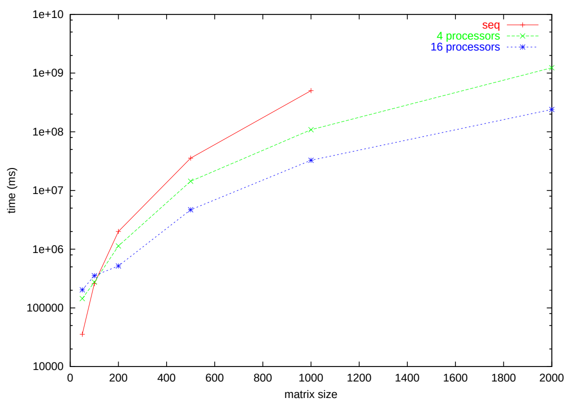


FIG. H.3 – Performance of matrix multiplication using the distributed overloaded operator $*$.

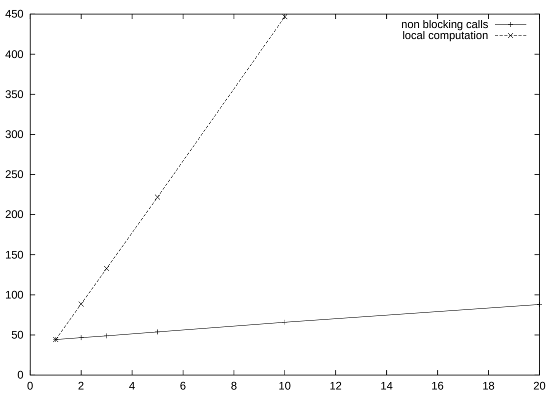


FIG. H.4 – Eigenvalues problems solving using the SCILAB-NETSOLVE interface.

results since the server was able to accept all the calls in parallel. It shows that, from SCILAB, the user is able to obtain high performance almost without having to deal with parallel computing.

H.3.1 Data Persistence and Data Redistribution

When we interfaced SCILAB and NETSOLVE, we have been confronted by two drawbacks of NETSOLVE concerning *data persistence* and *data redistribution*. When a server has computed a result, this result may be used again as an input parameter of another request on this server. Hence, it can be useful to use *data persistence*, i.e., cache data on this server. Moreover, this result can also be involved in a computation on another server, in that case it can be useful to *redistribute data* from server to server. However, NETSOLVE does not implement data redistribution between servers : when a server has completed a computation, output objects (results) are retrieved by the client. Therefore, many useless communications could be avoided. This problem as been tackled with the new request sequencing feature [3]. However, the current request sequencing implementation does not allow to handle multiple servers. Moreover, our data persistence implementation allows the client to manage its distributed data and their availability on different servers. We have modified NETSOLVE in order to implement data persistence and redistribution between servers. This has been done in a transparent way, with no change to the API. Existing client programs will work normally after recompiling. Moreover, our implementation is stand-alone : data management works without the help of any other tool. In order to implement this, we modified NETSOLVE servers so that, when a computation completes, data stay locally on the server. The server is waiting for orders from the client. There are five orders a server may receive : *exit* means that the server terminates and all its local data are lost ; *send one output object*, the server sends one of its results either to its client or to an other server ; *send one input object*, the server sends one of the problem parameter to its client or to an other server ; *send all output objects*, all the results are sent to a client or to an other server ; *send all input objects*, all the problem parameters are sent to a client or to an other server.

Communications between servers are implemented sockets. When a client wants two servers to exchange data, a socket is established between these servers. We add new functions and data structures to the NETSOLVE client library to allow the use of data persistence and redistribution features. When a client wants to use a remote data, it has only to specify the server session and the object number and type for this session.

Figure H.5 shows how a complex matrix multiplication between two distant cities may benefit from data redistribution. In this experiment, we show the time to execute a complex matrix multiplication where computation have been decomposed as follows : (1) $C_{r_1} = A_r \times B_r$; (2) $C_{r_2} = A_i \times B_i$; (3) $C_{i_1} = A_r \times B_i$; (4) $C_{i_2} = A_i \times B_r$; (5) $C_r = C_{r_1} - C_{r_2}$; (6) $C_i = C_{i_1} + C_{i_2}$. Where A_r (resp. B_r and C_r) is the real part and A_i (resp. B_i and C_i) the imaginary part of complex matrix A (resp. B and C). The four matrix multiplications were computed on one node of the IBM SP2 of the LaBRI in Bordeaux, while the two matrix additions were

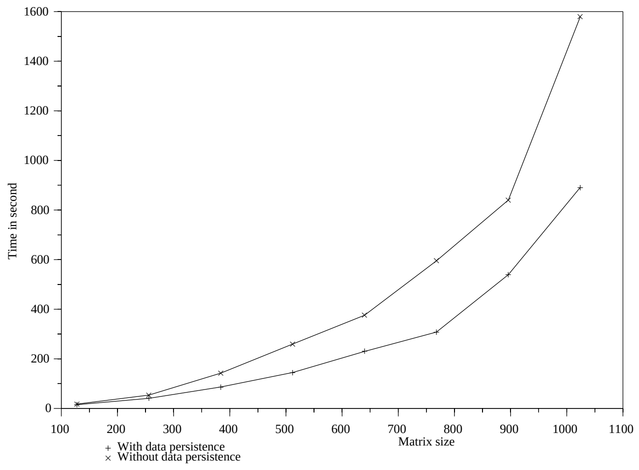


FIG. H.5 – Complex matrix multiplication using NETSOLVE between Nancy and Bordeaux.

performed locally in Nancy. One shall remark that steps 1 and 2 can be executed in parallel as well as steps 3 and 4. With data redistribution, objects A_r, B_r, A_i, and B_i are not sent back to the client between steps 1–2 and steps 3–4. We see that in that case computations are 1.77 faster for matrix of size 1024 than the same computation performed without data persistence and redistribution.

H.3.2 Software Resources Location and Performance Evaluation

To schedule computations over servers, we are facing two problems : first, we have to find which resources are able to satisfy the request, and then, we have to choose the best suited one by evaluating the performance of each proposed solution. To solve the first problem, we are developing a library called SLIM, Scientific Libraries Metaserver. Its goal is to link a problem description to implementations available on servers. In most cases, it is not a one-to-one mapping : a single problem can be solved by many implementations from several libraries, while an other problem may need more than one computational step to be solved. For example, if a user wants to solve a system of linear equations represented by sparse matrix, depending of the data themselves, it can be solved by a direct solver or by a preconditioner followed by a iterative solver. Thus, sequential and parallel implementations may be available. In the first prototype, we decided to use the name of the SCILAB built-in functions as problem description language. Even if this approach is satisfying in this context, it lacks of generality, and we are currently working on a better solution based on the Guide to Available Mathematical Software (GAMS) problem taxonomy [3].

Once SLIM has found which implementations are able to solve the given problem, the system has to evaluate the performance of each one for each machine providing it. The NETSOLVE agent scheduler has some lacks in this domain that we propose to fill. First, it considers that the characteristics of the link (bandwidth and latency) between a client and a server are the same as those of the link between itself and this server. Then, the time complexity of problems must be expressed through a simple expression such as ax^b, where a and b are constants defining the complexity, and x the size of involved data. To improve the knowledge of the metasystem needed by the scheduler to choose the best possible server, we are developing a library called FAST, Fast Agent System Timer [17]. FAST is composed of several layers and relies on low level software, as shown on Figure I.2. To address the drawback of the network performance forecast in NETSOLVE, FAST uses the Network Weather Service (NWS) [14]. It is a distributed system that periodically monitors and dynamically forecasts performance of various network and computational resources. Furthermore, the dynamic data acquisition module of FAST enhances NWS. If there is no direct NWS monitoring between two machines, FAST finds the shortest path between them in the graph of monitored links. In this case, the estimated bandwidth is the minimum of those of the path. For the latency, the sum is taken. Concerning the second drawback, FAST includes routines to model the time and space needs of a computation on a given machine as functions of the parameters of the computation. For that, it fits data resulting of benchmarks (realized at installation time with no external load) by linear regression using the least square method. The result is a polynomial function which order is automatically chosen to minimize the error. This allows to take cache and swap effects in account. Furthermore, as the modeled function is more complex, it is more expressive. To store these static data, FAST uses the Lightweight Directory Access Protocol (LDAP) [7]. LDAP was chosen for its optimizations to read and search data. Furthermore, LDAP is widely used in the Grid community. Finally, FAST also includes a user API, which is a set of functions that combine static and dynamic data acquired from lower level components to produce ready-to-use values. These functions allow the scheduler to get the time to move an amount of data between two computers and the predicted time to solve a problem on a computer taking its actual workload into account.

Figure I.2 shows an overview of SLIM, FAST and their interactions with a client application. The scheduling is done in several steps : (1) the client gives the problem description to SLIM. (2) SLIM contacts the database system and searches out the set of implementations which are able to solve the submitted problem. For example, if the problem is a multiplication of dense matrices, the `DGEMM` function of the SCALAPACK library would be a candidate. (3) This set is then sent to FAST to forecast the execution time of each solution for each server. (4) FAST acquires the static data from the database and (5) the dynamic data from NWS. (6) Finally, these data are combined by FAST in a list of couples {implementation i on server s ; estimated time t} and returned to the calling application. Then, the client uses this result to choose which server to contact.

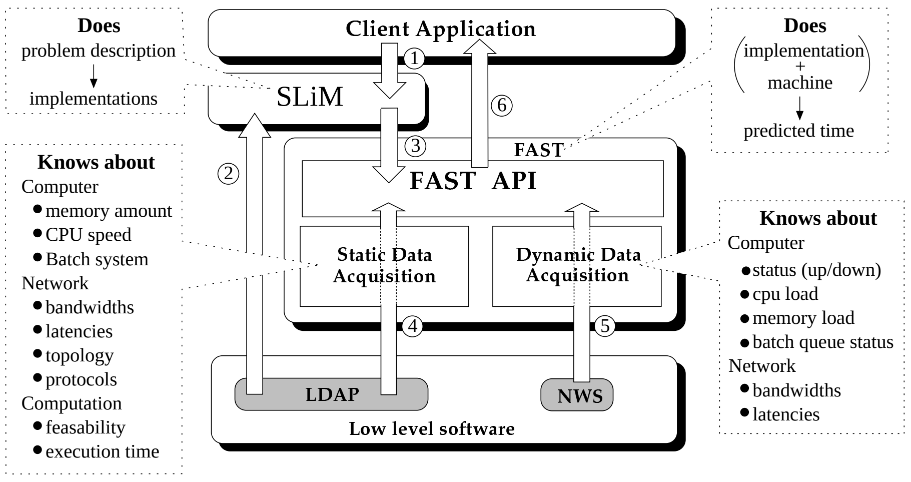


FIG. H.6 – Overview of SLIM and FAST.

H.3.3 CORBA Interface to Parallel Servers

Communications are a key issue in NES environments. The communication layer should provide both good performance and ease of development. The CORBA norm, defined by the *Object Management Group (OMG)*, aims at providing a standard and transparent interface for the development of object oriented distributed applications over heterogeneous networks. CORBA systems are built around an *Object Request Broker (ORB)*, which is a communication bus between CORBA objects. Communications are initiated by method invocations between objects which can be located on different hosts.

In this section, we give a brief overview of CORBA systems. Then, we discuss the integration and the definition of new services in order to provide metacomputing domain CORBA services. Our current implementation still relies on the NETSOLVE architecture (scheduler, problem description) but it becomes a part of the CORBA services. Thus, we define a new CORBA interface between NETSOLVE components to take advantage of CORBA high level features over the socket interface. We propose a mapping of these two communication layers on a common interface so as to make them accessible in the same way. The example in section H.3.3.3 shows how works a metacomputing session using our platform.

Some goals of the CORBA norm are : to allow a high transparency level in the communication primitives of an application ; to normalize the features of object oriented distributed systems ; to allow interoperability between these systems (i.e., transparent communications between different CORBA implementations) ; to provide a distributed programming environment that is independent from the language (i.e., communications between applications written in different languages are transparent) ; to normalize most common system processes into CORBA services ; and to reduce the development time for distributed applications.

H.3.3.1 CORBA Services for Metacomputing.

As the CORBA norm provides a transparent way to implement distributed application, its use in the domain of metacomputing should be considered. Existing metacomputing platforms are usually subject to very frequent experimental modifications and features add-ons. CORBA systems allow, with a very low communication time increase, a great ease of development and a greater maintainability of the code. Moreover, when communicating between heterogeneous architecture, CORBA can even be faster than XDR [16]. As a matter of fact, our tests [1] have shown that communication times with an ORB are equal to those with standard sockets plus a constant value. Table H.3 shows the time necessary to some free ORBs to send various sizes of characters arrays on a local network. Performance of the sockets library in the same conditions are given too. These results confirm that the overhead induced by CORBA is not significant when the data amount grows.

size in bytes	10	100	1000	10000
Mico	0.65 ms	0.73 ms	1.55 ms	9.83 ms
OmniOrb2	0.56 ms	0.68 ms	1.4 ms	9.81 ms
OrbAcus C++	0.57 ms	0.67 ms	1.54 ms	10.26 ms
OrbAcus Java	1 ms	2 ms	6 ms	47 ms
Jonathan	1.06 ms	1.23 ms	3.37 ms	24.3 ms
Sockets	0.26 ms	0.35 ms	1.24 ms	9.67 ms

TAB. H.3 – Communication times of some free ORBs compared with the socket library.

Thus, even if slightly better performances could be obtained by writing optimized socket applications, CORBA seems to be a good choice for the development of a metacomputing platform. It is well suited for resource allocation in distributed systems, which is a critical point in metacomputing. We thus propose that CORBA systems are an interesting alternative for the development of a metacomputing platform.

On such platforms, data can be moved across the network without the client being notified of their new location. No common service is still able to perform this kind of function in a metacomputing context. Moreover, the existing services are too complex to be customized to satisfy our specific needs. A set of specific services should be developed for metacomputing. We have specified a set of metacomputing services and propose to interface them with the SCILAB$_{//}$ software. We describe below each of these services and we give an overview of how they could work together to allow a transparent metacomputing process.

Our metacomputing system is composed of three kinds of entities : SCILAB$_{//}$ clients ; computational servers ; and metacomputing services. Clients are SCILAB$_{//}$ processes embedded in CORBA objects. Computational servers are numerical libraries with a CORBA frontal which allows remote invocations. Clients never send their computation requests directly to servers. The metacomputing services are in charge to transmit these requests to the appropriate servers. For each computation, a server is chosen to achieve best performances. We define two main metacomputing services, which are implemented as CORBA objects.

The *trading service*, or *trader*, is responsible for the servers pool management. It keeps up to date a table of all the available servers and their features (e.g., what problems they are able to solve, CPU, memory, . . .). A client sends its computation request to the trader which chooses a server and transmits the request to it. In order to provide good performance, the trader takes many parameters into account. In the best case, it tends to choose a server which is idle and which already holds a data involved in the computation. If we are not in the ideal case, the trader chooses a server which minimizes the sum of communication and computation time. The scheduler of our metacomputing platform is thus part of our trading service.

The *location service* is responsible for keeping track of the data migrations across the platform. Every data is associated with a unique *identifier* or *reference*. The *location service* keeps up to date a table which associates each data identifier with the server or client owning it. This service should be warned of every data migration. Then, it is able to be called by a client to retrieve a result or by the trader when data locations are needed to choose a server.

H.3.3.2 Integration into NETSOLVE.

In order to benefit of the existing developments made over NETSOLVE in the OURAGAN project, we choose to integrate our metacomputing services into NETSOLVE. The resulting version of NETSOLVE should be able to use both sockets and CORBA communication layers. For technical reasons, our two services are included in the same CORBA object which is built upon the NETSOLVE agent. The resulting CORBA service is called metacomputing agent as well.

In order to implement our CORBA layer in parallel with the existing socket layer, a *common interface* has been designed. This interface is a set of functions which can be executed by both layers [15]. This interface acts like a wrapper which allows the developers of the core of the agent to use the two communications layers in the same way. This common interface is designed to allow data persistence and further extensions like data duplication. In the next, we give an overview of the resulting architecture and of the interactions between the components of the platform.

H.3.3.3 Metacomputing Session Example.

Figure H.7 shows the architecture of the metacomputing platform in a general way. The purpose of this section is to show how a metacomputing session that uses CORBAworks.

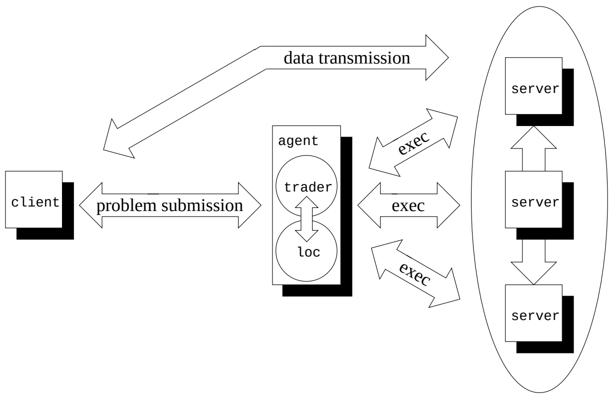


FIG. H.7 – An example of a metacomputing session using the common interface (`c=a*b`).

First of all, servers have to be registered within the agent with details about their features (e.g., problems to solve, CPU, memory available, …). When a client begins a metacomputing session, it has first to look for the reference of the agent upon the system using the CORBA *naming service*. Now, it is able to export data. Exported data have a reference in the *location service* of the agent. Then, when the client submits a problem to the metacomputing platform, it invokes a method of the agent. Parameters of this submission are the problem to solve and the involved exported data. This way, the agent is able to choose a server to perform the submitted problem using both its *trading* and *location services*. The components that own involved data are also invocated to send them to the chosen server. The computation is launched when all data are received. The return of these different methods is a reference to the result. Then the client is able to retrieve this result, if necessary, or to submit another problem using it. So, this kind of session allows the system to reduce the transmission data over the network.

This example illustrates the ease of the design of data persistence. In the same way, several extensions can be specified for a more complex data management, e.g., data duplication, lazy copy, ... Moreover, CORBA allows a flexible client/server programming by allowing communications without permanent connection between components of the system.

H.3.4 Parallel Direct Solver for Sparse Symmetric Positive Definite Systems

As a first target of the computational servers approach in SCILAB$_{//}$, we chose an efficient parallel software processing chain able to solve large linear systems with direct method called PASTIX. This project is developed by the ALiENor team from LaBRI. Solving large sparse symmetric positive definite systems $Ax = b$ of linear equations is a crucial and time-consuming step, arising in many scientific and engineering applications. PASTIX focuses on the block partitioning and scheduling problem for high performance sparse LDL^T factorization without pivoting on parallel/distributed architectures ; we consider a parallel supernodal version of sparse LDL^t factorization with total local aggregation. In [22], a first version describing a mapping and scheduling algorithm for 1D distribution of blocks was presented. Then, an original algorithm based on a mixed 1D/2D block distribution has been presented in [23] ; it computes an efficient static scheduling that fully drives the block computations of the parallel solver. Parallel experiments were run on an IBM SP2[2], whose nodes are 120 MHz Power2SC thin nodes. These results show that our PASTIX software compares very favorably to PSPASES [26].

We have implemented a PASTIX server for NETSOLVE, and defined an interface to perform globally or separately the whole steps of our parallel software chain. The integration of PASTIX into SCILAB$_{//}$ is now effective and we are currently working to improve data persistence into that server. First experiments on irregular industrial problems are promising. However, they show that the initialization of the coefficients of the matrix is a time consuming step. Indeed, the storage format used by the SCILAB$_{//}$ platform must be more compatible with the PASTIX data structures ; we currently use the RSA sparse matrix format as input for the server.

H.3.5 Visualization of Distributed Data

Industrial applications mainly use standard data structures such as matrices, but most of the time provide a specific problem-oriented implementation, e.g., Compressed Sparse Column (CSC). Specific implementations are used especially often when dealing with large sparse and irregular data structures, such as matrices coming from the domain of finite elements. The gap between the implementation and the abstract data structure it implements is even bigger when considering parallel applications. Hence, there is a need for tools that make it possible for developers to visualize both their data, their structure, and the operation that are applied to it, whatever their effective implementation and distribution are. These tools must carry the semantics of the application and provide synthesis or filtering mechanisms that make it possible to focus on specific aspects of the problem. Our project was first to define a framework to support the development of such tools, and then to implement a set of software components using this framework. VISIT is one of these tools which is developed at LaBRI. In the OURAGAN project, our goal was to integrate VISIT as a visualization server for SCILAB$_{//}$. In the following, for illustrative purpose, we focus on its integration with the PASTIX computational server.

Our approach consists in providing support for using sparse and irregular data-structures inside applications : we want to provide tools dealing with such data structure at a high level of abstraction. To achieve this goal, we use a model with four levels : (1) the *Implementation Graph* describes the implementation of the data structure in terms of data items and access functions, i.e., the way they are accessed within the application, for instance CSC storage for a matrix ; (2) the *Abstraction Graph* describes the abstract data structure, e.g., a matrix ; (3) the *Mapping Graph* describes the relationship between the implementation graph and the abstraction graph. In this graph, a node is defined as a pair containing both a node of the abstraction graph and a set of nodes of the implementation graph (an empty set matches a hole in a sparse data structure) ;

[2]The IBM SP2 of the CINES, located in Montpellier, France.

(4) the *View Graph*, this level is used for synthesis and filtering of information, e.g., it makes it possible to focus on a column of a matrix.

Based on this model, we have developed libraries to abstract from implementation and distribution of data structures. These libraries allow the manipulation of the data structures at any level contained within our framework, while keeping the same efficiency when working at the mapping graph level or at the implementation level. Using our libraries, high level tools work only with graphs, which they can for instance go through to achieve some operations. It is possible to attach information to any node of any of the graphs. This feature is used by VISIT (see [12] for details). For the visualization of large matrices, VISIT uses the MatView software [30].

The PASTIX code is basically made up of a set of tasks, one for each matrix block. The distribution of these data blocks infers the distribution of the tasks on the processors of the parallel computer. It seems interesting to observe the execution of PASTIX in terms of distributed data blocks. It should be noted that the distribution is irregular and is computed in an initial step. We have described the structure of the data storage used in PASTIX as an implementation graph and a mapping graph. Figure H.8 shows the distribution of data blocks, the abstraction graph being the full matrix; each processor has its own grayscale. Figure H.9 is a MatView zoom of the lower right corner of the matrix. The data that are used come from the Oilpan of the Harwell-Boeing Sparse Matrix Collection and is of size 75000 × 75000. The target parallel computer is an IBM SP2 with 16 processors.

We are currently instrumenting PASTIX in order to generate traces in terms of access to data blocks. This will make it possible to show the dependencies between data tasks/blocks in terms of remote read, remote write, local read an local write operations.

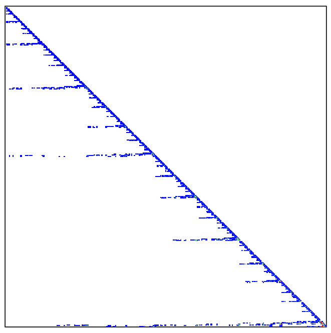


FIG. H.8 – Matrix blocks distribution.

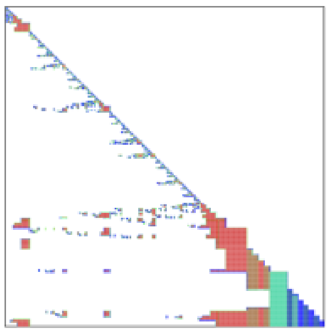


FIG. H.9 – A MatView zoom.

H.4 Conclusion and future work

Matlab is an interesting approach to Problem Solving Environments. In this paper, we have presented a parallel version of SCILAB, a Matlab-like tool developed at INRIA. Two approaches have been presented. The first one duplicates SCILAB processes on different processors and then uses either message passing with PVM or MPI or high performance numerical routines with ScaLAPACK (in-core and out-of-core and with or without operator overloading). The second one uses an improved version of a high performance Network Enabled Server, NETSOLVE. We added an accurate evaluation of the performance of the metacomputing platform and data persistence on the servers which avoids too many exchanges between the client and the servers and allows redistribution of data between servers. We also presented how we manage two interfaces for the communications between the different components of the tool, i.e., sockets and CORBA.

Concerning the SCILAB processes duplication, the SCILAB interpreter should be able to cast between different data-types when needed. For example, this is the case when computing the product of one (in-core) distributed matrix with one out-of-core matrix. Similarly, the product of two in-core (resp. out-of-core) distributed matrices may lead to one out-of-core (resp. in-core) matrix. Cast can be implemented in two ways. The first way consists in writing computation routines for the different combinations of data-types and call them appropriately. The second way consists in defining new cast operators to promote the lower data-types (e.g., distributed in-core) to bigger ones (e.g., out-of-core) and then do the computation. This solution is easier to implement because there are few cast operators to develop. To achieve a fully functional out-of-core extension of SCILAB, the main job is to develop a lot of out-of-core routines and interface them. Another way consists in defining new operators which allow selection of in-core (distributed or not) blocks of out-of-core matrices. Then, with these new operators, new out-of-core functionalities can be directly written using the built-in programming language of SCILAB. The prototyping and development of new out-of-core functions should be made easier.

Another future work consists in designing a scalable, portable and hierarchical set of agents to improve the Network-Enabled Servers part of our developments. To achieve this goal, we are currently designing DIET, a Distributed Interactive Engeeniering Toolbox [16]. We also would like to add parallel libraries (like ScaLAPACK or PETSc) as computational servers and to handle the redistribution of distributed data between servers on a heterogeneous platform. This is mandatory if we want to be independent of SCILAB, and port our tools on a real grid platform involving different clusters and different networks, as the one connecting several research centers (and their clusters) from INRIA with a 2.5 Gb/s network[3].

Acknowledgements

We would like to thank all the people that helped us in the OURAGAN project in different ways like D. Goudin, P. Hénon, J.-M. Lepine, F. Pellegrini and N. Viollet.

H.5 References

[1] J.-L. Anthoine, P. Chatonnay, D. Laiymani, J.-M. Nicod, and L. Philippe. Parallel Numerical Computing Using Corba. In H. R. Arabnia, editor, *Parallel and Distributed Processing Techniques and Applications (PDPTA'98)*, volume III, pages 1221 – 1228, Las Vegas, july 1998.

[2] P. Arbenz, W. Gander, and J. Moré. The Remote Computational System. *Parallel Computing*, 23(10) :1421–1428, 1997.

[3] D. Arnold, D. Bachmann, and J. Dongarra. Request Sequencing : Optimizing Communication for the Grid. In A. Bode, T. Ludwig, W. Karl, and R. Wismuller, editors, *Proc. of the Sixth European Conference on Parallel Computing (Euro-Par 2000)*, volume 1900 of *LNCS*, pages 1213–1222, Munich, Germany, August 2000. Springer Verlag.

[4] P. Banerjee, N. Shenoy, A. Choudhary, S. Hauck C. Bachmann, M. Haldar, P. Joisha, A. Jones, A. Kanhare, A. Nayak, S. Periyacheri, M. Walkden, and D. Zaretsky. A MATLAB Compiler For Distributed, Heterogeneous, Reconfigurable Computing Systems. In *Proceedings of the Int. Symp. on FPGA Custom Computing Machines (FCCM-2000)*, April 2000.

[5] S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley. *ScaLAPACK Users' Guide.* SIAM, 1997.

[6] R.F. Boisvert. The Architecture of an Intelligent Virtual Mathematical Software Repository. *Mathematics and Computers in Simulation*, 36 :269–279, 1994.

[7] G. Bosilca, G. Fedak, and F. Cappello. "ovm : Out-of-order execution parallel virtual machine". In *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGRID2001)*, May 2001.

[8] E. Caron, D. Lazure, and G. Utard. Inversion of Huge Matrix on Cluster. In *Cluster 2000. IEEE International Conference on Cluster Computing. Chemnitz, Germany.*, 28 nov - 2 dec 2000.

[9] E. Caron, D. Lazure, and G. Utard. Performance Modeling and Analysis of Parallel Out-of-Core Matrix Factorization. In *HiPC'2000. 7th International Conference on High Performance Computing. Bangalore, India*, December 17-20 2000.

[3]RNRT VTHD project (http://www.telecom.gouv.fr/rnrt/projets/pres_d74_ap99.htm).

[10] H. Casanova and J. Dongarra. A Network-Enabled Server for Solving Computational Science Problems. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(3) :212–223, 1997. and proceedings of Supercomputing'96, Pittsburgh.

[11] H. Casanova and J. Dongarra. Providing Uniform Access to Numerical Software. In M. Heath, A. Ranade, and R. Schreiber, editors, *Algorithms for Parallel Processing*, volume 105 of *IMA Volumes in Mathematics and its Applications*, pages 345–355. Springer-Verlag, 1998.

[12] S. Chaumette, F. Rubi, and J.M. Lépine. A Graph Based Framework for the Definition of Tools Dealing with Sparse and Irregular Distributed Data Structures. In IEEE, editor, *Proc. of the 3rd Int. Workshop on High-Level Parallel Programming Models and Supportive Environments*, pages 62–70, March 1998.

[13] S. Chauveau and F. Bodin. Menhir : An Environment for High Performance Matlab. In David O'Hallaron, editor, *Languages, Compilers and Run-Time Systems for Scalable Compilers (LCR'98*, volume 1511 of *LNCS*, pages 27–40, Pittsburgh, PA, May 1998. Springer Verlag.

[14] F. Desprez, E. Fleury, C. Gomez, S. Steer, and S. Ubéda. Bringing metacomputing to scilab. In *Computer Aided Control System Design (CACSD 99)*, Hawai'i, USA, August 1999. IEEE.

[15] F. Desprez, E. Fleury, E. Jeannot, J.M. Nicod, and F. Suter. Interactive Parallel Computing Using Scilab. *Submitted to Parallel Computing, Special issue on Parallel Matrix Algorithms and Applications*, 2001.

[16] F. Desprez, E. Fleury, F. Lombard, J.-M. Nicod, M. Quinson, and F. Suter. A Scalable Approach to Network-Enabled Servers. OURAGAN Whitepaper, available at http://www.ens-lyon.fr/~desprez/OURAGAN/, 2001.

[17] F. Desprez, M. Quinson, and F. Suter. Dynamic Performance Modeling for Network Enabled Solvers in a Metacomputing Environment. In *Submitted to the Seventh European Conference on Parallel Computing (Euro-Par 2001)*, 2001.

[18] J. Dongarra and E. D'Azevedo. The Design and Implementation of the Parallel Out-of-core ScaLAPACK LU, QR, and Cholesky Factorization Routines. Technical Report UT-CS-97-347, Department of Computer Science, University of Tennessee, January 1997.

[19] P. Drakenberg, P. Jacobson, and B. Kågström. A CONLAB compiler for a distributed memory multicomputer. In *Proc. of the 6th SIAM Conf. on Parallel Processing for Scientific Computing*, pages 814–821. SIAM Press, March 1993.

[20] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM : Parallel Virtual Machine - A Users' Guide and Tutorial for Networked Parallel Computing*. The MIT Press, 1994.

[21] C. Gomez, editor. *Engineering and Scientific Computing with Scilab*. Birkhaüser, 1999.

[22] P. Hénon, P. Ramet, and J. Roman. A Mapping and Scheduling Algorithm for Parallel Sparse Fan-In Numerical Factorization. In *EuroPAR'99, LNCS 1685*, pages 1059–1067. Springer Verlag, 1999.

[23] P. Hénon, P. Ramet, and J. Roman. PaStiX : A Parallel Sparse Direct Solver Based on a Static Scheduling for Mixed 1D/2D Block Distributions. In *Proceedings of Irregular'2000, LNCS 1800*, pages 519–525. Springer Verlag, May 2000.

[24] I. Howes, M. Smith, and G. Good. *Understanding and deploying LDAP directory services*. Macmillian Technical Publishing, 1999.

[25] P. Husbands, C. Isbell, and A. Edelman. Interactive Supercomputing with MITatlab. http://www-math.mit.edu/~edelman.

[26] M. Joshi, G. Karypis, V. Kumar, A. Gupta, and Gustavson F. PSPASES : Scalable Parallel Direct Solver Library for Sparse Symmetric Positive Definite Linear Systems. Technical report, University of Minnesota and IBM Thomas J. Watson Research Center, May 1999.

[27] A. Malishevsky, N. Seelam, and M.J. Quinn. Translating MATLAB Scripts into Parallel Programs Utilizing ScaLAPACK and Other Parallel Numerical Libraries. *Submitted to IEEE Transactions on Sotfware Engineering*, 1999. Available at http://www.cs.orst.edu/~quinn/matlab.html.

[28] S. Matsuoka and H. Casanova. Network-Enabled Server Systems and the Computational Grid. http://www.eece.unm.edu/~dbader/grid/WhitePapers/GF4-WG3-NES-whitepaper%-draft-000705.pdf, July 2000. Grid Forum, Advanced Programming Models Working Group whitepaper (draft).

[29] S. Matsuoka, H. Nakada, M. Sato, , and S. Sekiguchi. Design Issues of Network Enabled Server Systems for the Grid. http://www.eece.unm.edu/~dbader/grid/WhitePapers/satoshi.pdf, 2000. Grid Forum, Advanced Programming Models Working Group whitepaper.

[30] MatView. Scalable sparse matrix viewer. http://www.epm.ornl.gov/~kohl/MatView/.

[31] G. Morrow and R. van de Geijn. A Parallel Linear Algebra Server for Matlab-like Environments. In *Supercomputing'98*, Orlando, Nov 1998.

[32] NEOS. http://www-neos.mcs.anl.gov/.

[33] NINF. http://ninf.etl.go.jp/.

[34] S. Pawletta, A. Westphal, T. Pawletta, W. Drewelow, and P. Duenow. *Distributed and Parallel Application Toolbox (DP Toolbox) for use with MATLAB(r).* Institute of Automatic Control, University of Rostock and Department of Mechanical Engineering, FH Wismar, Germany, version 1.4 edition, March 1999.

[35] S. Ramaswamy, E.W. Hodges IV, and P. Banerjee. Compiling MATLAB Programs to ScaLAPACK : Exploiting Task and Data Parallelism. In *International Parallel Processing Symposium (IPPS'96)*, pages 613–619, April 1996.

[36] L. De Rose and D. Padua. A MATLAB to Fortran 90 Translator and its Effectiveness. In *Proceedings of the 10th ACM International Conference on Supercomputing - ICS'96*, Philadephia, PA, May 1996.

[37] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. J. Dongarra. *MPI : The Complete Reference.* The MIT Press, 1996.

[38] A. Trefethen, V. Menon, C. Chang, G. Czajkowski, C. Myers, and L. Trefethen. MultiMATLAB : MATLAB on multiple processors. Technical Report 96-239, Cornell Theory Center, 1996.

[39] R. Wolski, N. T. Spring, and J. Hayes. The Network Weather Service : A Distributed Resource Performance Forecasting Service for Metacomputing. *Future Generation Computing Systems, Metacomputing Issue*, 15(5–6) :757–768, Oct. 1999.

Chapitre **I**

A Scalable Approach to Network Enabled Servers

Conference : Submitted

Authors :

F. Desprez, M. Quinson, F. Suter	F. Lombard, J.M. Nicod	E. Fleury
LIP - ENS Lyon	LIFC	LORIA, INRIA Lorraine
46, allée d'Italie	16, route de Gray	615, rue du Jardin Botanique - BP 101
69364 Lyon Cedex 07, France	25016 Besançon Cedex, France	F-54602 Villers-Les-Nancy, Cedex
desprez@ens-lyon.fr	lombard@univ-fcomte.fr	eric.fleury@loria.fr

Abstract :
This paper presents the architecture and the algorithms used in DIET (Distributed Interactive Engineering Toolbox), a hierarchical set of components to build Network Enabled Server applications in a Grid environment. This environment is built on top of different tools which are able to locate an appropriate server depending of the client's request, the data localization (which can be anywhere on the system, because of previous computations) and the dynamic performance characteritics of the system.

Keywords :
Network Enabled Solvers, Parallel libraries, Computational servers, CORBA, Data Redistribution

Thanks :
This work has been supported in part by the ARC INRIA OURAGAN. URL : `http ://www.ens-lyon.fr/~desprez/OURAGAN/`.

I.1 Introduction

Huge problems can now be computed over the Internet thanks to Grid Computing Environments [5]. Because most of current applications are numerical, the use of libraries like BLAS, LAPACK, ScaLAPACK or PETSc is mandatory. The integration of such libraries in high level applications using languages like Fortran or C is far from being easy. Moreover, the computational power and memory needs of such applications may of course not be available on every workstation. Thus, the RPC [8, 9] seems to be a good candidate to build Problem Solving Environments [6] on the Grid. Several tools following this approach exist, like Netsolve [12], NINF [13], NEOS [11], or RCS [2].

This paper presents the architecture of DIET (Distributed Interactive Engineering Toolbox), a hierarchical set of components to build Network Enabled Server (NES) applications. Our target platform is the fast network VTHD connecting several research centers (and their clusters) from INRIA. This document is organized as follows : Section I.2 presents the overall architecture of the DIET platform and its main components. Section I.3 shows how CORBA can be used to connect the different components of DIET. In Section I.4, we give the algorithms used to discover software and hardware resources that are able to solve a problem sent by a client. We conclude our discussion in Section I.5 with an summary of the DIET architecture, and of the major challenges in defining such systems and implementing a prototype.

I.2 DIET architecture and related tools

In this section, we give some details about the architecture of DIET and we present the different components involved in its hierarchy. In [9], the authors give a general overview of NES environments. Usually, such environments have five different components : *clients* that submit problems to servers, *servers* that solve problems sent by clients, a *database* that contains informations about software and hardware resources, a *scheduler* that chooses an appropriate server depending of the problem sent and the informations contained in the database, and finally *monitors* that get informations about the status of the computational resources.

In DIET, a server is built upon *Computational Resources Daemons* (CRD) and a *Server Daemon* (SeD). We have a hierarchical set of agents including *Leader Agents* (LA) and *Master Agents* (MA). A *redirector* (ReD) is used to choose a master agent which is close to the client. Now we detail the basic functionalities of these different components. An overview of tools used in DIET (SLIM & FAST) is also given.

I.2.1 DIET components

Figure I.1 shows the hierarchical organization of DIET. The different parts of this architecture are the following :

Computational Resources Daemon (CRD) : A computational resource is a set of hardware and software components that can perform sequential or parallel computations on data sent by a client (or an other server). On an interactive machine, each node should have a CRD. In other cases (like batch systems), computations are launched directly from the Server Daemon.

Server Daemon (SeD) : A SeD is the point of entry of a computational server. It manages a pool of CRDs. The informations stored on a SeD are the list of the data available on a server (eventually with their distribution and the way to access them), the list of problems than can be solved on it, and every informations concerning its load (available memory, number of available CRDs, . . .).

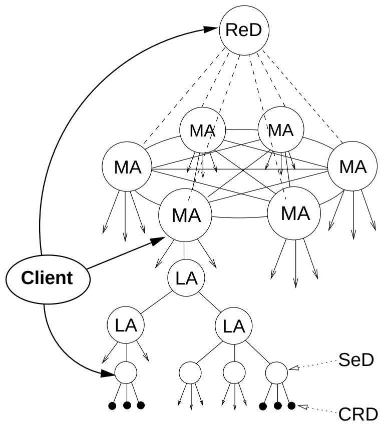


FIG. I.1 – DIET overview.

Leader Agent (LA) : A LA can be the link between a Master Agent and a SeD or another LA, between another LA and a SeD or between two LAs. Its goal is to transmit requests and informations between MAs and SeDs. The informations stored on a LA are the list of requests and, for each of its subtrees, the number of servers that can solve a given problem and informations about the data distributed in this subtree.

Master Agent (MA) : A MA is directly linked with client components. Its receives computation requests from clients and choose the SeD which can solve the problem the fastest. A MA owns the same informations as a LA, but it has a complete view of the problem that can be solved and the data that are in all its subtrees.

Redirector (ReD) : The ReD is the main point of entry of DIET. It is unique and used by the clients at their connection to know the address of the most appropriate MA.

Client : Many kinds of clients should be able to connect to the DIET system. A problem can be submitted from a web page, a problem solving environment such as SCILAB, a free implementation of Matlab, or a compiled program.

I.2.2 SLIM : Scientific Libraries Metaserver

SLIM's goal is to make the junction between problems submitted by the clients and the implementations available on the servers. In most case, there is no one-to-one mapping : a single problem can be solved by many implementations from several libraries, while another problem may need more than one computational step to be solved. For example, if the user wants to solve a system of linear equations with a sparse matrix, depending of the data themselves, it can be solved by a direct solver or by a preconditioner followed by a iterative solver. Sequential and/or parallel version of the routines may be available.

The main problem of this approach is to find an unified way to express the problems and data descriptions. One could use the description problem language used in NETSOLVE, but this not a standard, and it lacks a way to express parallel functions. The CORBA interface repository would be an alternative for this, but it was mainly designed for applications as finances or medicine. No service for scientific computation exists so far in the CORBA norm either. Indeed, SLIM could be seen as a good candidate for a CORBA service to discover available resources in a grid environment.

As our first prototype was based on SCILAB, we decided to use the name of the built-in functions as first problem description metalanguage. Even if this approach is satisfying in this context, it lacks of generality, and we are thus currently working on defining a better solution based on the GAMS [3] problem taxonomy.

All needed informations are stored in a LDAP [7] tree. LDAP is a distributed database protocol which was chosen for its read and search optimizations. Furthermore, LDAP is widely used in the grid community.

I.2.3 FAST : Fast Agent's System Timer

FAST is a tool for dynamic performance forecasting in a Grid environment. As shown in Figure I.2, FAST is composed of several layers and relies on low level softwares. First, it uses a network and CPU monitoring software to handle dynamically changing resources, like workload or bandwidth. FAST uses the Network Weather Service (NWS [14]), a tool started at the University of California San Diego and now based at the University of Tennessee Knoxville. This is a distributed system that periodically monitors and dynamically forecasts the performances of various network and computational resources. The dynamic data storage is also handled by NWS. The dynamic data acquisition module of FAST uses and enhances NWS. In fact, if there is no direct NWS monitoring between two machines, FAST searches automatically for the shortest path between them in the graph of monitored links. It estimates the bandwidth as the minimum of those in the path and the latency as the sum of those measured.

Moreover, FAST includes routines to model the time and space needs for each triplet { problem ; machine ; parameters set }. They are based on benchmarking at installation time on each machine for a representative set of parameters and polynomial data fitting. To store these static data, FAST uses the same LDAP-tree as SLIM. The user API of FAST is composed of a few set of functions that combine static and dynamic data acquired from low level software to produce ready-to-use values. Thus, FAST clients like DIET components, can get for example the time needed to move an amount of data between two SeDs, the

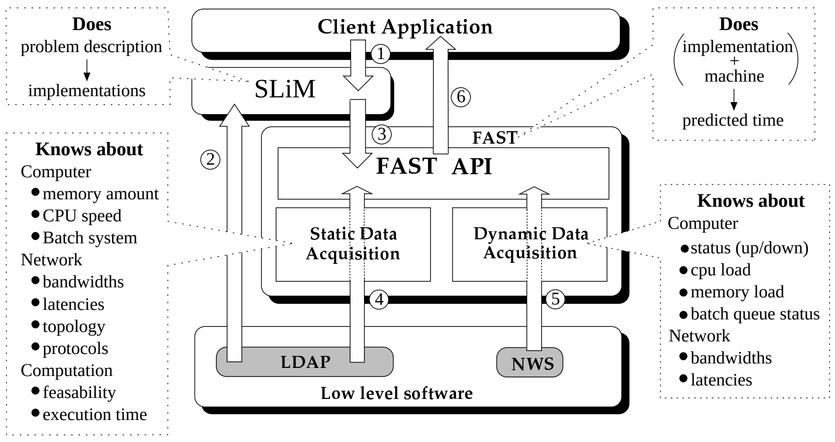


FIG. I.2 – SLIM-FAST overview.

time to solve a problem with a given set of CRDs managed by a SeD, or the addition of these two times. The static data acquisition in FAST concerns the theoretical parameters of computers and communication links of the system.

I.2.4 SLIM-FAST interactions

To give a first overview of interactions between SLIM and FAST, we consider a basic system composed of a unique agent with several servers. The real algorithm with several agents organized in a DIET hierarchy will be given in Section I.4.3.2. Figure I.2 shows the different steps of a problem submission in this case.

The agent send the descriptions of the problem to solve and the involved data to SLIM(1). SLIM contacts the database system and searches out the set of implementations which are able to solve the submitted problem. For example, if the problem is a multiplication of dense matrices, the `dgemm` function of the *ScaLAPACK* library would be a candidate (2). This set is then sent to FAST to forecast the execution time of each solution for each server (3). FAST acquires the static data from the database (4) and the dynamic data from NWS (5). Finally, these data are combined by FAST in a list of couples {server ; estimated time} and returned to the calling application (6).

I.3 CORBA based communication layer

In order to implement a network enabled problem solver, one can choose between many communication layers. Low level layers like the socket interface eventually allow the best performances. Higher level layers such has ones complying with the CORBA norm although provide interfaces for a easier and quicker development. In this section, we give an overview of the CORBA norm and discuss the opportunity to use a CORBA implementation in high performances distributed applications.

I.3.1 The CORBA norm

The CORBA norm, defined by the *Object Management Group* (OMG), provides a standard and transparent interface for the development of object oriented distributed applications over heterogeneous networks. CORBA systems are built around an *Object Request Broker* (ORB), which is a communication bus between CORBA objects. Communications are initiated by method invocations between objects that can be located on different hosts.

Different CORBA implementations are available, each one running on a wide variety of platforms. Interoperability between these implementations is granted by the *General Inter ORB Protocol* (GIOP). Thus, CORBA applications are highly portable and many users are likely to be able to access CORBA services. From the developer's point of view, CORBA can be used with most common programming languages, including C, C++ and Java.

I.3.2 CORBA features

From the developer's point of view, CORBA provides a high level object oriented interface for the development of distributed applications. Every object that wishes to export some of its methods on the network registers itself to the ORB. It then gets an *Object Reference* that allows other object to invoke it without being concerned by its localization. To allow this transparency on heterogeneous networks, an *Interface Description Language* (IDL) compiler is provided with all ORBs. This compiler automatically generates code that allows the marshaling and demarshaling of the application's data structures. This mechanism is more optimized than the XDR protocol, the data being sent without modifications when sending them between to a compatible architecture. Dynamically typed data and types descriptions downloading are also supported by the generated code.

Thus, CORBA systems provide a remote method invocation facility with a high level of transparency. This transparency should not dramatically affect the performances, communication layers being well optimized in most CORBA implementations. In fact, with a low bandwidth network, performances can be better with CORBA than with the use of lower level protocols like RPC/XDR. Moreover, in such an environment, the communication time with CORBA is the same as with sockets plus a constant value [1]. We currently project to extend these benchmarks to high performances networks.

I.3.3 Conclusions about CORBA

Grid computing is a very active research domain. Existing platforms are usually subject to frequent experimental modifications and feature add-ons. CORBA systems allow an easier development and a greater maintainability of the code. It is well suited to resource allocation in distributed systems, which is a key point in Grid environments. The performances do not seem dramatically affected by these features. We thus propose that CORBA systems are one of the alternatives of choice for the development of Grid specific services.

I.4 DIET initialization and operation

In order to design a multi-agents Grid system, one should specify the order in which components should be run and the administration policy of the system and the distributed algorithms which allow the placement of the computations on the pool of servers.

In this section, we detail the former issue by giving an example of a simple Grid system initialization. Then, we discuss the way a server is chosen to solve a given problem by taking into account the communication and computation times.

I.4.1 DIET initialization

Figure I.3 shows each step of the initialization of a simple Grid system. The architecture is build in the hierarchical order, each component contacting its father : being the entry point of the Grid system, the ReD is the first entity to be started (1). Once running, it waits for the connection of a MA (2).

Then, a LA can be launched and connected to the MA (3). At this point of the system initialization, two kinds of components can connect to the LA : a SeD (4), which manage some computation resource, or another LA (5), to add a hierarchical level in this branch. The system is then up, and a client can contact the Redirector to find the nearest MA (6).

The father node being a parameter of each component, and each component transmitting the local configuration change to its father, the system administration can also be hierarchical. For example, a MA can manage a domain like an university, providing a privileged access to the users of this domain. Then each laboratory can run a LA, while each team of the laboratory running some others child-LA to administrate

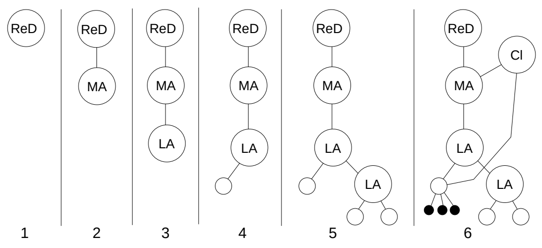


FIG. I.3 – Initialization of a DIET system.

its own servers. This hierarchical administration of the system permit to change the configuration of a part without interfering with the rest.

I.4.2 NES scenario

It is of the Redirector's responsibility to collect informations about MAs and to find which one is best suited for a new client. Once this MA is identified, the client can submit a problem through it. To choose the most appropriate server to solve the problem, the MA propagates a request in its subtrees to find both involved data and capable servers. This process is detailed in the next section. Then, the MA returns the address of the chosen server to the client and performs the transfer of persistent data involved in the computation. The client communicates its local variables to the server which can then solve the problem. Results may be sent back to the client or kept in place for the next computational step.

I.4.3 Solving a problem

We assume that the architecture described in the previous section owns several servers which are able to solve the problem and that every data needed by the computation is available on only one server over the whole architecture. The example presented in Figure I.4 considers a submission of problem `F()` using data `A` and `B`.

The algorithm presented here allows a MA to choose one of the servers it manages to execute a computation. This decision is taken in three steps :

– locating the data involved and the capable servers by sending a request to computational servers, while propagating a MA request to its subtrees ;
– evaluating the computation time on all capable servers, while sending the answer to the request back to the MA ;
– choosing a server and ordering it to perform the computation. This is done by the MA once it has got the answers.

In order to simplify the discussion, this section only refers to a system involving only one MA. The algorithms presented here are easily extended to the general case by broadcasting computation request to others MAs.

I.4.3.1 Data and server localization

Request structure In order to choose a server, the MA must locate the servers that are able to solve the submitted problem and the data involved in the computation. This is done by sending a request structure to the servers concerned. This structure, shown on Figure I.5(a), contains two fields : the `problemNickname`, and a list of `attributes` of the involved data, including details on their size and properties in order to evaluate the computation and communication times.

Algorithm When the MA receives a request from a client, it builds the *request structure* and sends it to all its children which either own one of the needed data or are able to solve the problem. The request is

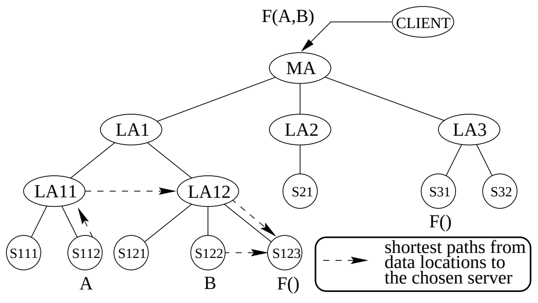


FIG. I.4 – Problem submission example.

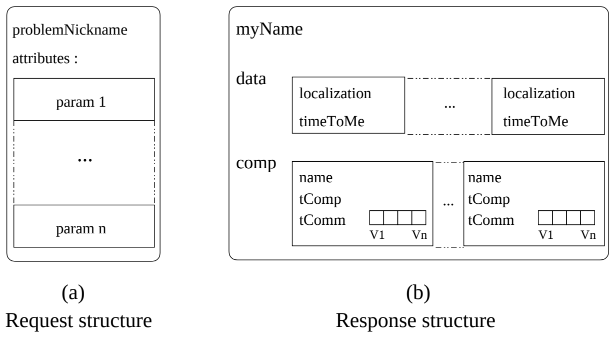


FIG. I.5 – Computation request and response structures.

transmitted from father to child in the tree following this scheme down to SeDs. Each LA labels its children reached by the request and waits for their response, which we discuss in the next section.

I.4.3.2 Computation and communication times evaluation

Once the request structure reaches the concerned SeDs, they initiate a response structure and send it back to their father.

The response structure Described in Figure I.5(b), it contains three fields :

myName : the name of the component sending back this response ;

data : this array has an entry for each variable involved in the computation, each one containing two fields :

localization : the name of the component owning the data, if it is known (this field takes a null value otherwise) ;

timeToMe : estimation of the communication time to bring the variable from the component sending the structure, if its localization is known ;

comp : this array contains an entry for every server (able to satisfy the request) known at this point of the tree. Three informations are kept for each server : its `name`; `tComp`, the estimated computation time to satisfy the request; `tComm`, an array containing the estimated time to bring each variable involved to the server. These value are computed dynamically while sending back the result to the MA with the following algorithm.

Algorithm This algorithm is divided in three parts :

1. **Initialization :** When a computation server receives a request, it sends a response structure to its father. It fills the `data` field for the variables it owns, leaving a null value for the others. If the server can solve the problem, it also puts an entry in the `comp` array with its evaluated computation time (in the `tComp` field). This is a call to FAST.
2. **Aggregation :** Every LA gathers its children's responses and aggregates them into one structure. The fields concerning communication times are gradually filled as the structures come back to the MA. For this, FAST compute the transfer time of data to the capable servers, combining informations from the monitoring and the data attributes. This transfer will use the shortest path among those which are monitored. This path only uses the links between a father and its children or between two brothers as shown on Figure I.3. Figure I.6 gives the complete algorithm used by LAs to aggregate their children's responses.

3. **Use :** When the responses come back to the MA, it can use it to take a decision. The evaluated computation and communication times are used to find the server with the lowest response time to perform the computation.

   However, we have to consider the case where a server is chosen twice whereas the first computation has not already started when the second problem was submitted. The penalty of the first computation problem could not be considered in the computation evaluation time of the second one. This a classical problem in dynamic performance evaluation. We propose to delay the evaluation of the computation time at the level of the master when it has to choose the server. In this case, we know the history of the problem submissions, so we can take them into account in the final evaluation.

```
for each data D do
  if none of my descendants owns D then
    timeToMe = 0
  else if one of my children references D then
    timeToMe = timeToMe for this child + time to send D form this child to me
  else if D is not known by any of my children then
    if a server S of my sub-tree can solve the problem then
      D will be sent through me to S if it's selected.
      ⇒ Increase D's tComm for each server
    else
      D will be sent to a capable server S following a path in which I am not involved.
      ⇒ End tComm's computation for my descendants.
    end if
  end if
end for
```

FIG. I.6 – Complete algorithm of a LA to aggregate the results.

I.4.4 Fault tolerance in DIET

Fault tolerance is a key issue in problem solving environments. In the DIET system, two kinds of faults can occur. First, an agent can die unexpectedly. The knowledge of any agent being derived from its children's knowledge, it can be reconstructed easily. So, the agent can be started again, or its children can be attached to another agent. The second kind of failure is the unexpected death of a computational server. When a server fails, data and temporary results may be lost. To decrease this risk, a checkpointing system can be implemented by the computation servers. A checkpoint corresponds to an intermediate result in the process of solving a problem. When a checkpoint is reached, the intermediate results are saved to disk (or sent to a data repository). So, if the server fails later, the computation can be restarted from the last checkpoint reached.

I.4.5 NES Management

The great variety of metacomputer components (computer, networks, files, software services) and their dynamically behavior poses special problems for the resource management point of view. In such architecture, management will play a major role and it appears very important to be able to offer a unified framework for the management of network, services and applications. The framework implies the definition of an architecture that guaranties the interoperability between applications through the portability of management informations wherever they are originated.

We claim that middleware layers based on Java technologies, and more especially JMX (Java Management Extensions) [10], offer new opportunities to support network enabled server application in a grid environment. Our approach is based on WEBM (Web-Based enterprise Management) which the standardization effort is relayed by the DMTF (Distributed Management Task Force) in order to take in account :

– Offering an homogeneous view of all managed resources whatever is their nature, location and/or access methods ;
– Keeping the legacy by integrating information models, management architectures and deployed protocols ;
– Offering an infrastructure to exchange management information between applications.

We need to define an information model based on CIM (Common Information Model) related to the grid network approach. This model will allow to define :

– The organizational model ; specify a functional architecture by specifying the active participants in the management process, their repartition, their function and their respective role ;
– Informational model ;
– Functional model ;
– Model of communication.

From on our model based on CIM, the implementation of the management components is done by using JMX that takes advantage of both Java's features and agent-based architecture. Indeed, the management system have the following features :

– Extensibility : since JMX is based on JavaBean, JMX agents can dynamically incorporate new functionalities from remote class server ;
– Interoperability : JMX agents can be integrated in CIM/WEBM and CORBA ;
– Reusability : one can integrate externally available functionalities developed as JavaBean ;
– Platform independence : JMX can be deployed in heterogeneous environments where several platforms coexist.

I.5 Conclusion and future work

In this paper, we have presented our view of a scalable Network Enabled Server system. We believe that hierarchy is mandatory when building such environments for the Grid. When thinking about Grid Computing, scalability should be one of the main concerns of developers. We propose a hierarchical approach to Network Enabled Servers using existing software like NWS or CORBA.

Our future work is to first test this approach on real applications. As our target platform allows 2.5 Gb/s communications between several INRIA research centers in France, connecting several clusters of PCs and parallel machines, we think that tightly coupled applications written in a RPC mode could benefit of such an approach. Another problem we would like to address is the optimization of data distributions for parallel library calls using a mixed data and task parallel approach. We also would like to connect our developments to infrastructure toolkits like Globus to benefit from the development of security, accounting, and interoperability services.

A concerted effort is on his way to define the overall architecture and basic functionalities of Grid environments [4]. A working group is dedicated to Advanced Programming Models which include, of course, Network Enabled Solvers [8, 9]. We think that this effort is very important to be able to get efficient software infrastructure soon and we would like to be part of it.

I.6 References

[1] J.-L. Anthoine, P. Chatonnay, D. Laiymani, J.-M. Nicod, and L. Philippe. Parallel Numerical Computing Using Corba. In H. R. Arabnia, editor, *Parallel and Distributed Processing Techniques and Applications (PDPTA'98)*, volume III, pages 1221 – 1228. CSREA Press, july 1998.

[2] P. Arbenz, W. Gander, and J. Moré. The Remote Computational System. *Parallel Computing*, 23(10) :1421–1428, 1997.

[3] R.F. Boisvert. The Architecture of an Intelligent Virtual Mathematical Software Repository. *Mathematics and Computers in Simulation*, 36 :269–279, 1994.

[4] Grid Forum. http://www.gridforum.org/.

[5] I. Foster and C. Kesselman (Eds.). *The Grid : Blueprint for a New Computing Infrastructure.* Morgan-Kaufmann, 1998.

[6] E. N. Houstis and J. R. Rice. On the future of problem solving environments. http://www.cs.purdue.edu/homes/jrr/pubs/kozo.pdf, 2000.

[7] I. A. Howes, M. C. Smith, and G. S. Good. *Understanding and deploying LDAP directory services.* Macmillian Technical Publishing, 1999.

[8] S. Matsuoka and H. Casanova. Network-Enabled Server Systems and the Computational Grid. http://www.eece.unm.edu/~dbader/grid/WhitePapers/GF4-WG3-NES-whitepaper%-draft-000705.pdf, July 2000. Grid Forum, Advanced Programming Models Working Group whitepaper (draft).

[9] S. Matsuoka, H. Nakada, M. Sato, , and S. Sekiguchi. Design Issues of Network Enabled Server Systems for the Grid. http://www.eece.unm.edu/~dbader/grid/WhitePapers/satoshi.pdf, 2000. Grid Forum, Advanced Programming Models Working Group whitepaper.

[10] Sun Microsystems. The Java Management Extensions (Draft 2.0). http://java.sun.com/products/JavaManagement/, 1999.

[11] NEOS. http://www-neos.mcs.anl.gov/.

[12] Netsolve. http://www.cs.utk.edu/netsolve/.

[13] NINF. http://ninf.etl.go.jp/.

[14] R. Wolski, N. T. Spring, and J. Hayes. The Network Weather Service : A Distributed Resource Performance Forecasting Service for Metacomputing. *Future Generation Computing Systems, Metacomputing Issue*, 15(5–6) :757–768, Oct. 1999.