Parallel Computing'93, D.J. Evans, J.R. Joubert and H. Liddell Eds, Elsevier Science Publisher B.V., 1993.

Minimizing Communication Overhead Using Pipelining for Multi-Dimensional FFT on Distributed Memory Machines

C.Calvin
LMC-IMAG
INPG
Av. Félix Viallet
38012 GRENOBLE Cedex
calvin@mistral.imag.fr

F.Desprez*
LIP, CNRS URA 1398
ENS Lyon
46, Allée d'Italie
69364 LYON Cedex 07
desprez@lip.ens-lyon.fr

1 Introduction

The Fourier Transform (FT) arises in many fields of science, like signal processing, applied mathematics, image processing ... and in particular, to compute convolution, spectral analysis, to identify features in images [10], and to compute the spectral transform by solving differential equations [9]. Multi-dimensional FT are used in fluid dynamics, solution of Poisson's equations [3, 21] and also image processing via filtering techniques. The most popular algorithm for computing a FT is the Fast Fourier Transform (FFT) algorithm.

The computation of mono and multi-dimensional FFT has been extensively studied on many parallel architectures like shared-memory [2] and vector supercomputers [1], SIMD [3, 13] and MIMD machines [6, 7, 11, 12, 22, 23]. These papers show different implementations of FFT algorithms according to the physical properties of the machines or implement new schemes based on sequential ones. Chu [7] presents the implementations of different algorithms to compute efficiently the bi-dimensional FFT problem on an hypercube machine and shows that different algorithms can be of interest depending of the size of the data and the parameters of the target machine. He points out that improvements can be obtained using overlapping of communications and computations. Walker [22, 23] presents block versions of the bi-dimensional FFT using overlap but only one communication link at the same time.

We assume a linear model of communication [16]. According to most existing parallel machines, we suppose that the communication links are bidirectional and parallel (k-port assumption) [19]. For all presented algorithms, we assume that the underlying topology is an hypercube. We recall briefly that an hypercube of dimension d has $P = 2^d$ nodes and 2^{d-1} edges. The nodes are represented by the binary words of length d. Two nodes are linked each other along direction k, if they differ only from bit k. So, a node has exactly d neighbors [18].

*This work was supported by MRE grant No. 974, the CNRS-NSF grant No. 950.22/ 07 and the research program C3

2 Problem

The Discrete Fourier Transformation (DFT) of a complex vector x of size N is the vector X, of size N, defined by:

$$X_j = \sum_{k=0}^{N-1} x_k \omega^{jk} \text{ where } \omega^{jk} = e^{-\frac{2i\pi jk}{N}} \text{ and } i = \sqrt{-1}.$$

The first algorithm of Fast Fourier Transform was introduced by Runge and Konitz [17] and rediscovered by Cooley and Tuckey [8]. Since, a lot of variations have been derived from the original algorithm [20]. They only differ by the way of storing intermediate data for vector length equal to a power of two. The basic idea of these algorithms is the splitting of data entry x into two subsets at each step of the algorithm, and combined them using a "butterfly" scheme. The cost of the sequential algorithm is: $T^{seq}_{fft\_1d} = N \log{(N)} \tau_a$, where τ_a is the cost of a complex multiply and add.

2.1 Bi-dimensional FFT

The computation of the bi-dimensional DFT is given by the following equation :

$$X(j_1, j_2) = \sum_{k_1=0}^{N-1} \left(\sum_{k_2=0}^{N-1} x(k_1, k_2) \times \exp\left(\frac{-2i\pi(k_1.j_1 + k_2.j_2)}{N} \right) \right)$$

This equation can be transformed in order to come down to the computation of two classical mono-dimensional FFT (FFT1D) [3, 7]. The algorithm is simple: we just have to compute mono-dimensional FFTs on both dimensions.
The cost of a sequential bi-dimensional FFT is: $T^{seq}_{fft\_2d} = 2N^2 \log{(N)} \tau_a$

3 Parallelization

We first present the classical parallelization of the mono-dimensional FFT on an hypercube. Then, we describe a method using overlapping to compute several mono-dimensional FFT and apply this method to the bi-dimensional case.

For all the algorithms, we use an allocation function based on the classical *"Bit Reverse"* allocation. This allocation implies that all the communication during the algorithm are between direct neighbors in the hypercube [6].

We denote by $\bigoplus$ the "exclusive-or" boolean operation, and a "butterfly-operation(a, b)" is defined as $a \pm \omega b$. The algorithm of processor q is given on figure 1.

3.1 Several mono-dimensional FFT

In this section, we describe the parallelization of the computation of a set of mono-dimensional FFT.

The aim of the asynchronous algorithm (ASYNC) is to maximize the overlap of the communications, to compute several rows at the same time in order to reduce the number of startup costs, and to use the full-bandwidth of the network.

In [23], Walker gives some remarks for using overlapping and computing blocks of lines to improve performances. Thus, blocking is a way to reduce the number of startups and to allow the overlap of the communications of one block with the computations of another one. In this section, we make a generalization of the method described by Walker [23] by having more communication

```
routine fft_1d_par(x,X,N,P)
begin
   compute FFT1D on my data of size N/P
   for l = 0 to r − 1 do
      exchange block with processor q ⊕ l
      compute butterfly-operation(my block, received block)
   endfor
end
```

Figure 1: Parallel FFT1D algorithm

overlap using several links at the same time. We extend the previous method to the computations of several blocks at the same time.

To be able to work with more than one link at the same time, we must be able to find a mapping (or redistribution) of the data which allows communication of the same step of a FFT (for different blocks) to be held with different links. This mapping for s links and for r rows of FFT at the same time is given in Figure 2 for $s = 2$.

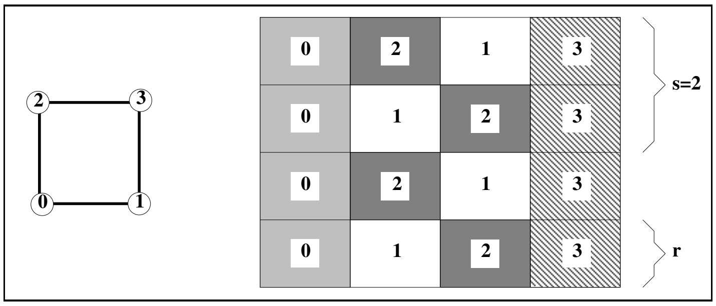

Figure 2: Mapping of a matrix using parameters $s = 2$ and r rows

In the following algorithm, we will denote:

- `FFT1D(frow,L,r)` the FFT1D algorithm on a local packet of `r` rows of length `L` starting at row `frow`
- `Update(frow,L,r,k)` the update operation on a packet received at step `k`
- `Bexchange(nbr,frow,nol,L,k)` the blocking exchange operation at the step `step` between `me` and `nbr` of `nol` rows of length `L` starting at row `frow`
- j`=NBexchange(nbr,frow,nol,L,k)` the non-blocking exchange version of the previous communication routine.
- `wait(`j`)` is the probe function

Notice that depending on the processor number, the row and the step, we are able to find the communication link which will be used for the exchange operation (corresponding to processor `nbr`).

The ASYNC algorithm is given on figure 3. Notice that this method can be used for a one-port communication protocol working on only two packets of r rows simultaneously.

```
routine fft_1d_async(f,s,r)
begin
  for i = 1 to s do
    FFT1D(f + (i - 1)r, n/P, r)
    j=NBexchange(nbr, f + (i - 1)r, r, n/P, 0)
  endfor

  for k = 1 to log(P) - 1 do
    for i = 1 to s do
      wait(j)
      Update(f + (i - 1)r, n/P, r, k)
      j=NBexchange(nbr, f + (i - 1)r, r, n/P, k)
    endfor
  endfor

  for i = 1 to s do
    wait(j)
    update(f + (i - 1)r, n/P, r, log(P))
  endfor
end
```

Figure 3: ASYNC Algorithm

In the following, we give the complexity analysis of this first algorithm. We assume that we compute the FFT1D by blocks of size r and s blocks at the same time. The total time of the algorithm is given by:

$$T_{total}^{ASYNC} = r(s+1)\frac{N}{P}\log_2\left(\frac{N}{P}\right)\tau_a + \log_2(P)\left(\max\left(\beta + r\frac{N}{P}\tau, sr\frac{N}{P}\tau_a\right) + r\frac{N}{P}\tau_a\right) \quad (1)$$

Now we discuss the ratio of the communications and the computations in the max term. Of course, we want to obtain the maximum overlap of the communications by the computation to obtain the best efficiency. In order to overlap the communications by the computations, we have to verify the following constraint:

$$rs\frac{N}{P} \geq \beta + r\frac{N}{P}\tau \quad (2)$$

From the two previous equations, we can compute the number of rows r computed at the same step. This leads to:

$$r \geq \left\lceil \frac{P\beta}{n(s\tau_a - \tau)} \right\rceil \quad (3)$$

s has to be equal to the number of communication ports available in parallel (d in the case of the hypercube). The critical size of r is then given by equation 3.

Other methods using SPMD-like programming paradigm are described in [4].

3.2 Application to the parallelization of the bi-dimensional FFT

After having described the main ways of allocating data [14], we present three methods to compute bi-dimensional FFT [7]. For each method, we first describe the method without overlap. After that, we show the improvements using overlap. For the last two methods, we use the methods described in the previous section.

3.2.1 Data allocation

For each allocation, we describe the function *Alloc(element)* which associates to *element* a processor number. *element* is either a row (column) i or an element (i,j) of the matrix. We denote by $(i)_2$ the binary representation of i, and by $|$ the operator of bit concatenation.

- Row-wise consecutive allocation
$$Alloc(i) = \left(\left\lfloor \frac{(i-1)\times P}{N} \right\rfloor\right)_2 \qquad i = 1\ldots N$$
- Row-wise BR allocation
$$Alloc(i) = \mathrm{BR}\Big(\left(\left\lfloor \frac{(i-1)\times P}{N} \right\rfloor\right)_2\Big) \qquad i = 1\ldots N$$
- Block BR allocation
$$Alloc(i,j) = \mathrm{BR}\left[\left(\left\lfloor \frac{(i-1)\sqrt{P}}{N} \right\rfloor\right)_2\right] \mid \mathrm{BR}\left[\left(\left\lfloor \frac{(j-1)\sqrt{P}}{N} \right\rfloor\right)_2\right] \qquad i,j = 1\ldots N$$

3.2.2 Transpose Split Method (TS)

The matrix to transform, x, is mapped on the networks using a *row-wise allocation*. The algorithm is a direct adaptation from the sequential algorithm: after having computed FFT1D on each row, we perform a matrix transpose. After this communication, another FFT1D is done on the resulting matrix. Finally, in order to return to the original mapping of the matrix, we have to do another matrix transpose. If the bi-dimensional FFT is not used in another computation part and pulled out of the network, this last transpose is not necessary. The matrix transpose can be realized by a "personalized all-to-all" communication scheme [5, 15]. If this operation is not overlapped, the communication overhead leads to poor efficiencies. We can improve the previous algorithm by overlapping the computation of the local FFT1D and the matrix transpose. We can transpose a set of r rows as soon as it has been computed.

3.2.3 Local Distributed Method (LD)

The data are allocated using the *row-wise BR allocation* function. We compute first FFT1D on each rows, but in spite of making a matrix transpose, we compute a distributed FFT1D on each column of the resulting matrix.

To use the "several FFT" parallel algorithm, we have to divide the LD algorithm in two main phases : the first one, during which we compute local mono-dimensional FFT, is distributed; and the second one, uses parallel ASYNC algorithm.

3.2.4 Block Method (Bl)

The third method consists in computing distributed FFT1D on both dimensions of the matrix. The matrix is allocated using a derivation of the *block allocation* function. To use overlapping via pipelining for the Bl method, we can use the ASYNC algorithm on both dimensions.

4 Experiments

The experiments have been done on a iPSC/860 parallel machine, and on a network of 4 SUN workstations using PVM 3.1.

On Figure 4, we compare the three different algorithms (TS, LD and Bl) without overlapping. It is clear that the TS method is the fastest and that the performances of both LD and BL methods are almost similar because of their high communication overhead. This has already been shown by Chu on an hypercube [7].

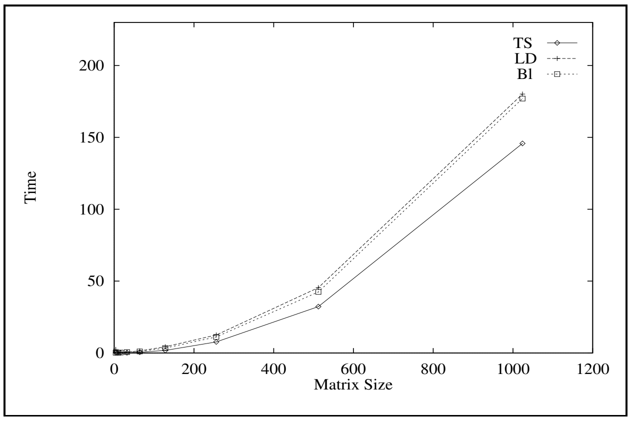

Figure 4: Comparison of the 3 methods without overlapping on 4 Sun workstations using PVM

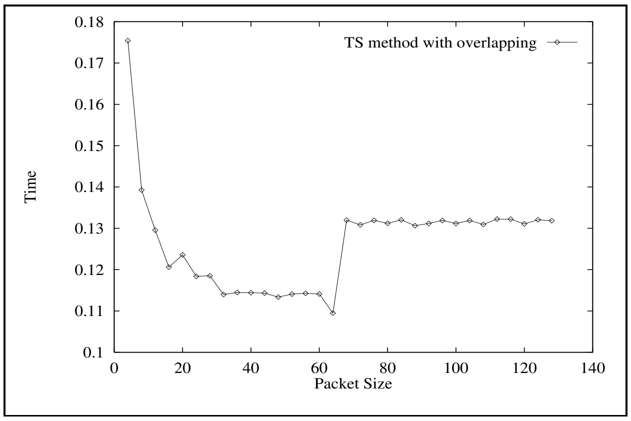

Figure 5: Execution time of the LD method using overlapping for $N = 128$ as a function of the packet size on the iPSC/860 with 8 processors

We have represented on Figure 5 the execution time of LD algorithm using overlapping as a function of the packet size r. The number of packets, s, is equal to 2 because the iPSC is a one-port machine.

As we can see, there are two phases in this graph. The first one is descendant. As the packet size increases, the time decreases until an "optimal packet size" where the overlap is maximal.

When the packet size is greater than this "optimal size", we do not have anymore overlap, and the time is equal to the time of the LD algorithm without overlapping.

Let us remark that for small sizes of packet, the time of the algorithm is greater than when we do not overlap the communications. This is due to the number of start-up factors which increases and

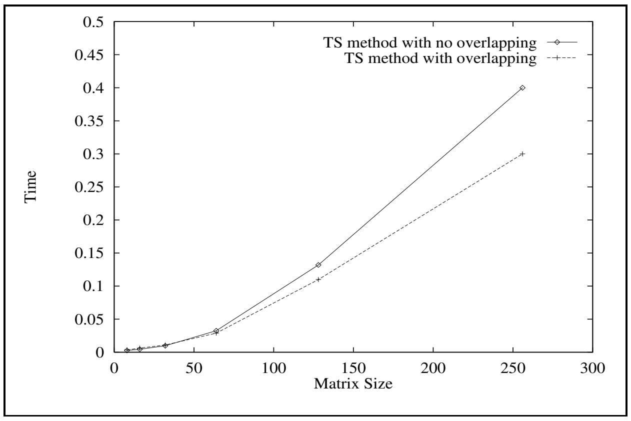

Figure 6: Comparison of the LD method with no overlapping and the LD method with overlapping on the iPSC/860 with 8 processors

so the communication time is always greater than the computation time.

Figure 6 shows the comparison between the LD algorithm with no overlapping, and the LD algorithm using overlap with optimal packet size for each dimension of the matrix. These experiments have been realized on an 8-nodes iPSC/860. For large enough matrix size, (N= 128 in this case), the method using overlap is faster than the algorithm with no overlapping.

5 Conclusion

In this paper we have presented different algorithms to compute the bi-dimensional FFT. These methods allow the overlapping of the communications by the computations and to reduce the number of start-up costs.
We have shown that the overlap is total using coarse grain pipelining. The experiments corroborate nicely this theoretical analysis.
Some other methods, using the SPMD-like programming paradigm, and other experiments are discussed in [4].

References

[1] M. Ashworth and A.G. Lyne. A Segmented FFT Algorithm for Vector Computers. *Parallel Computing*, 6:217–224, 1988.

[2] A. Averbuch, E. Gabber, B. Gordissky, and Y. Medan. A Parallel FFT on an MIMD Machine. *Parallel Computing*, 15:61–74, 1990.

[3] A. Brass and G.S. Pawley. Two and three dimensional FFTs on highly parallel computers. *Parallel Computing*, 3:167–184, 1986.

[4] C. Calvin and F. Desprez. Coarse Grain Pipelining and Low Overhead Communications for the Multi-Dimensional FFT Algorithms. to appear as a techreport LIP ENS-LYON France, 1993.

[5] C. Calvin and D. Trystram. Matrix Transpose for Block Allocations on Processor Networks. *submitted to Journal of Parallel and Distributed Computing*, 1993.

[6] R.M. Chamberlain. Gray codes, Fast Fourier Transforms and hypercubes. *Parallel Computing*, 6:225–233, 1988.

[7] C. Y. Chu. Comparison of Two-Dimensional FFT Methods on the Hypercube. In Geoffrey Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications*, volume 2, 1988.

[8] C.W. Cooley and J.W. Tuckey. An algorithm for the machine calculation of complex Fourier series. *Math. Comput.*, 19:297–301, 1965.

[9] I. Foster and P.H. Worley. Parallelizing the Spectral Transform Method: A Comparison of Alternative Parallel Algorithms. In R.F. Sincovec, D.E. Keyes, M.R. Leuze, L.R. Petzold, and D.A. Reed, editors, *Sixth SIAM Conference on Parallel Processing for Scientific Computing*, pages 100–107. SIAM, 1993.

[10] G. Fox, M. Johnson, G. Lycenza, S. Otto, J. Salmon, and D. Walker. *Solving Problems on Concurrent Processors - General Techniques and Regular Problems*, volume 1. Prentice Hall, 1988.

[11] A. Gupta and V. Kumar. The Scalability of FFT on Parallel Computers. Technical Report TR 90-20, Department of Computer Science - University of Minnesota - Minneapolis, October 1990. Revised October 1992.

[12] Y. Huang and Y. Paker. A Parallel FFT Algorithm for Transputer Networks. *Parallel Computing*, 17:895–906, 1991.

[13] S. L. Johnsson and R. L. Krawitz. Cooley-Tuckey FFT on the Connection Machine. *Parallel Computing*, 18:1201–1221, 1992.

[14] S.L. Johnsson and C.T. Ho. Algorithms for Matrix Transposition on Boolean n-cube Configured Ensemble Architectures. Technical Report YALEU/DCS/TR-572, Department of Computer Science - Yale University, September 1987.

[15] S.L. Johnsson and C.T. Ho. Expressing Boolean Cube Matrix Algorithms in Shared Memory Primitives, 1988.

[16] S.L. Johnsson and C.T. Ho. Optimum Broadcasting and Personalized Communication in Hypercubes. *IEEE Transaction on Computers*, 8(38):1249–1268, 1989.

[17] C. Runge and H. Konitz. *Vorlesungen uber Numerisches Rechnen.* Berlin, 1924.

[18] Y. Saad and M.H. Schultz. Topological Properties of Hypercubes. *IEEE Transaction on Computers*, 37(7):867–871, July 1988.

[19] Q.F. Stout and B. Wagar. Intensive Hypercube Communications, Prearranged Communications in Link-Bound Machines. *Journal of Parallel and Distributed Computing*, (10):167–181, 1990.

[20] P. N. Swarztrauber. Multiprocessors FFTs. *Parallel Computing*, 5:197–210, 1987.

[21] R. A. Sweet, W. L. Briggs, S. Oliveira, and J. Porsche. FFTs and Three-Dimensional Poisson Solvers for Hypercubes. *Parallel Computing*, 17:121–131, 1991.

[22] D. W. Walker. Portable Programming within a Message-Passing Model: the FFT as an Example. In Geoffrey Fox California Institute of Technology, editor, *The Third Conference On Hypercube Concurrent Computers and Applications*, volume II - Applications, 1988.

[23] D.W. Walker, P.H. Worley, and J.B. Drake. Parallelizing the Spectral Transform Method-Part II. Technical Report ORNL/TM-11855, Oak Ridge National Laboratory, July 1991.