To appear in the proceedings of the 3rd International Conference on Numerical Methods and Applications, Sofia, 21-26 August, 1994

A Comparison of Three Parallel Matrix Product Algorithms

F. Desprez[*][†]
Computer Science Department
The University of Tennessee
107 Ayres Hall
Knoxville, TN 37996-1301, USA

M. Pourzandi[‡]
CNRS - Laboratoire LIP, URA 1398
ENS Lyon
46, Allée d'Italie
69364 LYON Cedex 07, France

July 29, 1994

Abstract

In this paper, we present results of the study of matrix multiplication algorithms on the Intel Paragon and the Archipel VolVox IS-860 respectively with 1-port wormhole and 4-port store & forward communication protocols. We compare and analyze three algorithms on two dimensional torus/grid for both machines.

1 Introduction

The matrix product is the routine GEMM (GEneral Matrix Matrix product) of the level 3 BLAS (Basic Linear Algebra Subroutines) [6]. The matrix product is a basic routine involved in a lot of linear algebra algorithms which can be rewritten as a sequence of calls to this routine. First designed to sequential and parallel shared memory machines, the BLAS have to be recoded for distributed memory machines [3, 4] in order to get parallel versions of basic libraries such as LAPACK (Linear Algebra PACKage) [1, 3].

In this paper, we compare three basic algorithms on a two dimensional torus. The first one, designed by Fox and al. [7] for the Caltech hypercube, is based on horizontal broadcast. The second one, based on Cannon's algorithm [2], requires only neighbor to neighbor communications, but preskewings of two of the three matrices before and after execution ($C = C + AB$). Snyder's algorithm requires a transposition of matrix B before and after execution, vertical rollings of this matrix and global summations over the rows at each step [9]. These algorithms have already been implemented and compared on the Intel Delta machine by Choi, Dongarra and Walker in [4].

We have implemented and compared them on the ARCHIPEL Volvox IS-860 and Intel Paragon XPS/5. These two machines have different communication protocols, respectively *Store & forward* 4-ports and *Wormhole* 1-port but have the same kind of computation processor, i.e. the Intel i860. On the Paragon machine, the i860 is the latest version of this processor (i860 XP) and on the Volvox machine, this is the previous version (i860 XR). This explains the computation performances differences between the two machines. However, they have the same kind of behavior. Our goal is to compare matrix product algorithms and to highlight the

[*]On leave from LIP, CNRS URA 1398, ENS Lyon, 46 allée d'Italie, 69364 Lyon Cedex 07, France.

[†]This work was supported in part by the National Science Foundation Science and Technology Center Cooperative Agreement No. CCR-8809615, CNRS-NSF grant 950.223/07, PRC C^3, Archipel SA and MRE under grant 974, and DRET.

[‡]This work was supported in part by CNRS-NSF grant 950.223/07, PRC C^3, Archipel SA and MRE under grant 974, and DRET.

differences in the implementations and thus the performances due to the different models of communication. The resulting matrix product algorithms can then be used for the distributed version of level 3 BLAS library. Results of experiments are given for both machines.

For each algorithm, we assume that we start the execution with a natural full-block distribution of the matrices on the logical torus of processors. We assume that we have to compute the matrix product $C = C + AB$ where A, B and C are $N \times N$ matrices. If we store matrices A, B and C on a $P \times P$ two dimensional torus, then submatrices A_{ij}, B_{ij} and C_{ij} of size $N/P \times N/P$ are located on processor which coordinates are (i, j) in the torus. If the algorithm needs a redistribution of the matrices before execution starts, we take into account this communication overhead. And we assume that after computations, the submatrices must return to their original distribution on the processors for further computation.

2 Description of the algorithms

In this section, we discuss the necessary communications for each algorithm. We present the communications patterns of the algorithms in table 1.

Algorithm	**Cannon**	**Fox**	**Snyder**
Pre and post processing	skewing of A and B	None	Transposition of B
Computation of C	locally	locally	horizontal global sums
Movements of A	horizontal rollings	horizontal broadcast	none
Movements of B	vertical rollings	vertical rollings	vertical rollings

Table 1: Communication patterns of the three algorithms.

2.1 Fox's algorithm

The algorithm, also called broadcast-multiply-roll, described by Fox in [7, 8] has been first developed for the Caltech hypercube. It broadcast horizontally a diagonal of the submatrices A_{ij} through the processors, at each step of the execution. The communication pattern of the submatrices B_{ij} is vertical rollings.

This algorithm requires a natural mapping of the submatrices, i.e. submatrices $[A, B, C]_{ij}$ are stored on the processors which coordinates in the grid are (i, j). The algorithm is given in Figure 1 and his two first steps are given in Figure 4.

2.2 Cannon's algorithm

In order to have only neighbor-to-neighbor communications, submatrices A_{ij} and B_{ij} are distributed by diagonals through the rows of processors. The submatrices on the diagonal of A are distributed on the first column of the processors and the first diagonal submatrices of B are distributed on the first row of the processors (Figure 5). Because of this repartition, we have to redistribute submatrices A_{ij} and B_{ij} after the computation of the matrix product in order to put them back to their original location. Therefore, during the computation phase, we perform vertical and horizontal rollings according to the coordinates of the processor in the two dimensional torus. Figure 2 gives Cannon's algorithm.

2.3 Transpose algorithm

This algorithm has been presented by Snyder [9]. The submatrix A_{ij} is stored on processor (i, j) while the submatrix B_{ij} is stored on processor (j, i), i.e. matrix B is transposed before the computation phase. The algorithm requires a horizontal global sum at each step on each row

```
/* No pre treatment */

/* Computation of the matrix product */
Broadcast of the first diagonal of A
for k = 1 to √P − 1 in // do
   GEMM(A, B, C)
   Vertical rollings of B
   Broadcast of the next diagonal of A
endfor
in // do
   GEMM(A, B, C)
   Vertical rollings of B
fin

/* No post treatment */
```

Figure 1: Broadcast-multiply-roll algorithm.

```
/* Pre treatments */
Rollings of A and B

/* Computation of the matrix product */
for k = 1 to √P in // do
   GEMM(A, B, C)
   Vertical rollings of B
   Horizontal rollings of A
endfor

/* Post treatment */
Rollings of A and B
```

Figure 2: Cannon's algorithm.

of processors. The communication pattern of the sub-matrices B_{ij}, as for the Fox algorithm, is vertical rollings. We also have to redistribute submatrices B_{ij} after the matrix computation by transposing matrix B backward. Figure 3 gives Snyder's algorithm for node (i, j). Figure 6 shows the two first steps of this algorihm. The submatrices C_{ij} in bold are the ones which are computed at this step.

3 Experimental results

In this section, we present the performances of the three algorithms on both machines. We show that the most efficient algorithm changes according to the parallel machine used. On both machines, we try to maximize the performances by using overlap of the rollings by the computations. This can be done easily on both machines using asynchronous communications. Broadcast and global sums are not overlap because they need many communication steps and thus their overlap will imply a reduction of the grain of computation and thus a reduction of the performances of the calls to the level 3 BLAS routine DGEMM. This has already been

```
/* Pre treatment */
transpose matrix B

/* Computation of the matrix product */
in // do
   GEMM(A, B, C)
   Vertical rollings of B
endfor
for k = 1 to √P − 1 in // do
   Global sums on the rows of processors for C_{i,(i+k−1) mod √P}
   GEMM(A, B, C)
   Vertical rollings of B
endfor
Global sums on the rows of processors for C_{i,(i+√P−1) mod √P}

/* Post treatment */
transpose matrix B
```

Figure 3: Snyder's algorithm for node (i, j).

studied in [5]. The sequential DGEMM routine is assembly coded in order to obtain the best performances.

We first compare the three algorithms with pre and post treatments on a 6×6 torus on the Archipel Volvox machine (Figure 7). Despite of the pre and the post-processing stages, Cannon's algorithm remains the most efficient. The difference with the two other algorithms is more important if we do not redistribute matrices at the beginning and at the end of the computation (Figure 8). Notice that the costs of the broadcast and global sums are almost the same and the difference between Fox's and Snyder's algorithms comes from the pre and pos-transpositions.

On the Intel Paragon, communication startups are smaller and the broadcast algorithm is very efficient using wormhole routing (Figures 9 and 10). Because of the 1-port assumption, the Intel Paragon seems very sensitive to the algorithms with intensive communications even if they are grid-shaped. Fox's algorithms is thus a bit more performant on that machine. If we do not pre and post-redistribute matrices, Cannon and Fox are almost equivalent. Notice also that, due to an heavy operating system, Paragon curves are more "chaotic".

On both machines, Snyder's algorithm is the most expensive one. It requires a horizontal global sum at each step, with a prep and post-transposition.

Finally, we have compared both machines using a 6×6 torus/grid (Figures 11 and 12). Due to a limitation in the memory space (the operating system is using a large amount of memory), we stopped our experimentations on the Paragon for 1800×1800 matrices. With its more performant processor and communication network, the Intel Paragon is more performant for his algorithms. However, we can see that the two machines have two different behaviors. Cannon's algorithm is always more performant on the Archipel machine and less performant on the Intel machine.

4 Conclusion

In this paper we show clearly that the cost of the communication algorithm is not a simple function of the amount of data to be communicated but also depends on the machine's features which fit the communication schema of the algorithm well or are able to overlap the communications by the computations.

It seems that the 4-ports parallel machines are particularly adapted to Cannon's algorithm. This algorithm, using the communications on the all 4 ports of the 2D-torus topology in parallel, squeezes the best performance out of the Archipel VolVox machine. Neither Fox's algorithm nor Snyder's one, achieve the performances as well as Cannon's algorithm on these machines. This is not the case for 1-port parallel machines. These ones do not benefit from the simultaneous communications.

Actually, a broadcast based algorithm with a distance free wormhole protocol without communications intensive pre and post-treatments, is more efficient for matrix product algorithms. According to our results, we can predict Cannon's algorithm to be the most performant on wormhole 4-port machines using the i860 XP.

References

[1] C.H. Bischof. Fundamental Linear Algebra Computations on High-Performances Computers. Technical report, Mathematics and Computer Science Division - Argonne National Laboratory, 1988.

[2] L.E. Cannon. *A Cellular Computer to Implement the Kalman Filter Algorithm*. PhD thesis, Montana State University, 1969.

[3] J. Choi, J.J. Dongarra, and D.W. Walker. The Design of Scalable Software Libraries for Distributed Memory Concurrent Computers. In J.J. Dongarra and B. Tourancheau, editors, *Environments and Tools For Parallel Scientific Computing*. Elsevier, 1992.

[4] J. Choi, J.J. Dongarra, and D.W. Walker. PUMMA: Parallel Universal Matrix Multiplication Algorithms on Distributed Memory Concurrent Computers. Technical Report ORNL/TM-12252, Oak Ridge National Laboratory, April 1993.

[5] 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*. Birkhaeuser Verlag AG, Basel, Switzerland, 1994.

[6] 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.

[7] 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.

[8] G.C. Fox and S.W. Otto. Matrix Algorithms on a Hypercube I: Matrix Multiplication. *Parallel Computing*, 4:17–31, 1987.

[9] C. Lin and L. Snyder. An Algorithm of Choice for Solving QR Factorization. In E.D. Brooks, B.J. Heston, K.H. Warren, and L.J. Woods, editors, *The 1992 MPCI Yearly Report: Harnessing the Killer Micros*, pages 276–289. Lawrence Livermore National Laboratory, 1992.

$$
\begin{array}{cccc} C00 & C01 & C02 & C03 \\ C10 & C11 & C12 & C13 \\ C20 & C21 & C22 & C23 \\ C30 & C31 & C32 & C33 \end{array}
+=
\begin{array}{cccc} A00 & A00 & A00 & A00 \\ A11 & A11 & A11 & A11 \\ A22 & A22 & A22 & A22 \\ A33 & A33 & A33 & A33 \end{array}
\text{x}
\begin{array}{cccc} B00 & B01 & B02 & B03 \\ B10 & B11 & B12 & B13 \\ B20 & B21 & B22 & B23 \\ B30 & B31 & B32 & B33 \end{array}
$$

$$
\begin{array}{cccc} C00 & C01 & C02 & C03 \\ C10 & C11 & C12 & C13 \\ C20 & C21 & C22 & C23 \\ C30 & C31 & C32 & C33 \end{array}
+=
\begin{array}{cccc} A01 & A01 & A01 & A01 \\ A12 & A12 & A12 & A12 \\ A23 & A23 & A23 & A23 \\ A30 & A30 & A30 & A30 \end{array}
\text{x}
\begin{array}{cccc} B10 & B11 & B12 & B13 \\ B20 & B21 & B22 & B23 \\ B30 & B31 & B32 & B33 \\ B00 & B01 & B02 & B03 \end{array}
$$

Figure 4: The two first steps of the broadcast-multiply-roll algorithm.

$$
\begin{array}{cccc} C00 & C01 & C02 & C03 \\ C10 & C11 & C12 & C13 \\ C20 & C21 & C22 & C23 \\ C30 & C31 & C32 & C33 \end{array}
+=
\begin{array}{cccc} A00 & A01 & A02 & A03 \\ A11 & A12 & A13 & A10 \\ A22 & A23 & A20 & A21 \\ A33 & A30 & A31 & A32 \end{array}
\text{x}
\begin{array}{cccc} B00 & B11 & B22 & B33 \\ B10 & B21 & B32 & B03 \\ B20 & B31 & B02 & B13 \\ B30 & B01 & B12 & B23 \end{array}
$$

Figure 5: Data distribution for Cannon's algorithm.

$$
\begin{array}{cccc} \mathbf{C00} & C01 & C02 & C03 \\ C10 & \mathbf{C11} & C12 & C13 \\ C20 & C21 & \mathbf{C22} & C23 \\ C30 & C31 & C32 & \mathbf{C33} \end{array}
+=
\begin{array}{cccc} A00 & A01 & A02 & A03 \\ A10 & A11 & C12 & A13 \\ A20 & A21 & A22 & A23 \\ A30 & A31 & A32 & A33 \end{array}
\text{x}
\begin{array}{cccc} B00 & B10 & B20 & B30 \\ B01 & B11 & B21 & B31 \\ B02 & B12 & B22 & B32 \\ B03 & B13 & B23 & B33 \end{array}
$$

$$
\begin{array}{cccc} C00 & \mathbf{C01} & C02 & C03 \\ C10 & C11 & \mathbf{C12} & C13 \\ C20 & C21 & C22 & \mathbf{C23} \\ \mathbf{C30} & C31 & C32 & C33 \end{array}
+=
\begin{array}{cccc} A00 & A01 & A02 & A03 \\ A10 & A11 & C12 & A13 \\ A20 & A21 & A22 & A23 \\ A30 & A31 & A32 & A33 \end{array}
\text{x}
\begin{array}{cccc} B01 & B11 & B21 & B31 \\ B02 & B12 & B22 & B32 \\ B03 & B13 & B23 & B33 \\ B00 & B10 & B20 & B30 \end{array}
$$

Figure 6: The two first steps of Snyder's algorithm.

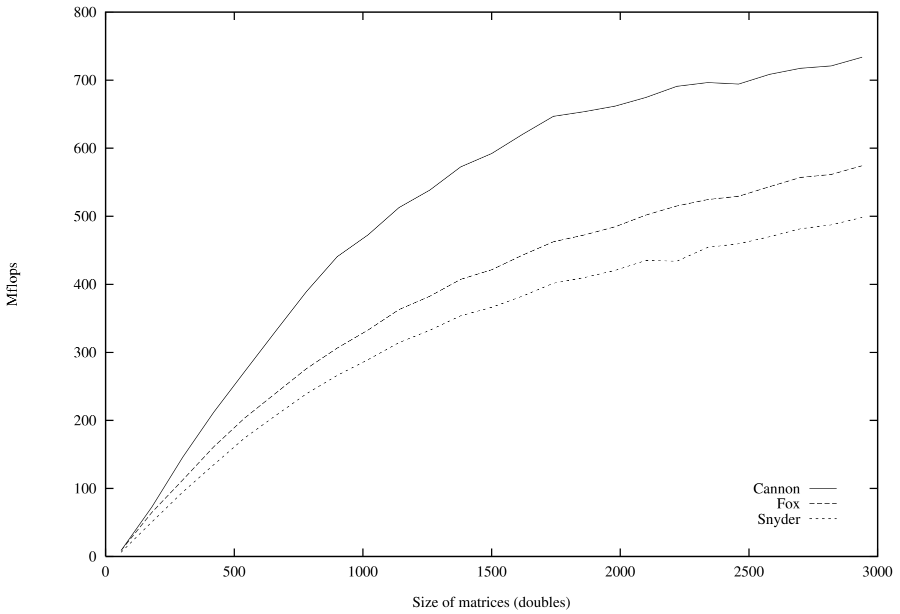

Figure 7: Performances of the matrix product with pre(post)treatments in Mflops according to the matrix size on VolVox IS-860 for a 6×6 torus

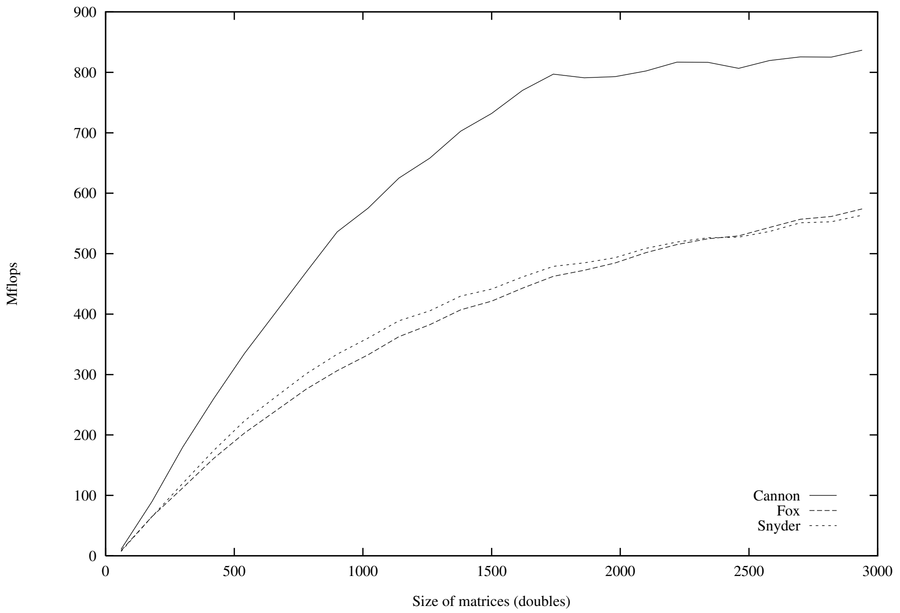

Figure 8: Performances of the matrix product without pre(post)treatments in Mflops according to the matrix size on VolVox IS-860 for a 6×6 torus

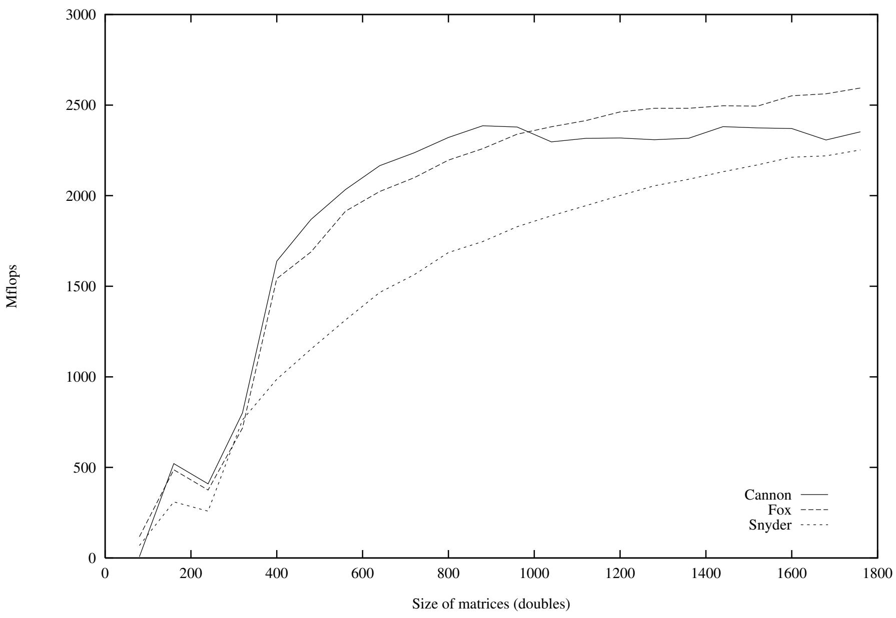

Figure 9: Performances of the matrix product with pre(post)treatments in Mflops according to the matrix size on Intel Paragon XPS/5 for a 8×8 grid

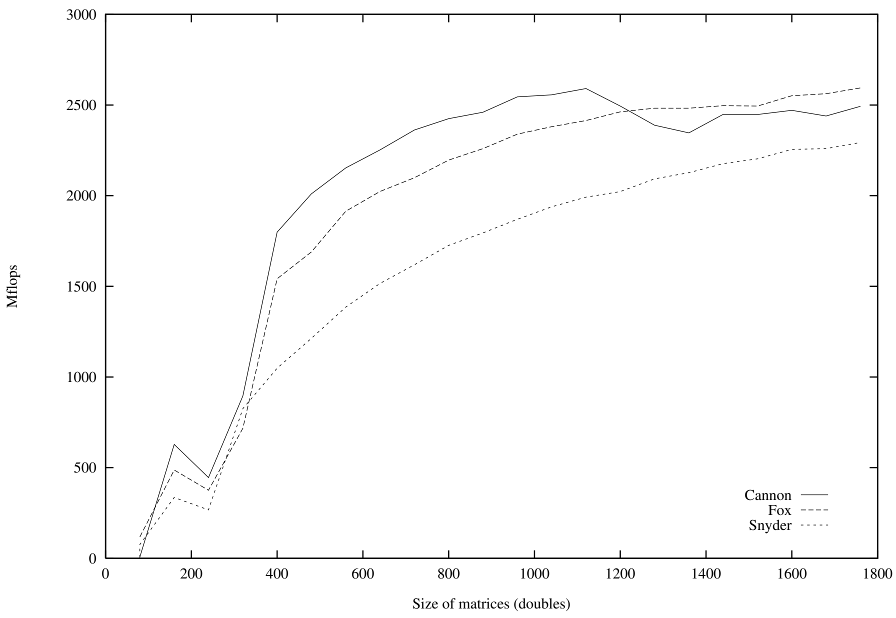

Figure 10: Performances of the matrix product without pre(post)treatments in Mflops according to the matrix size on Intel Paragon XPS/5 for a 8×8 grid

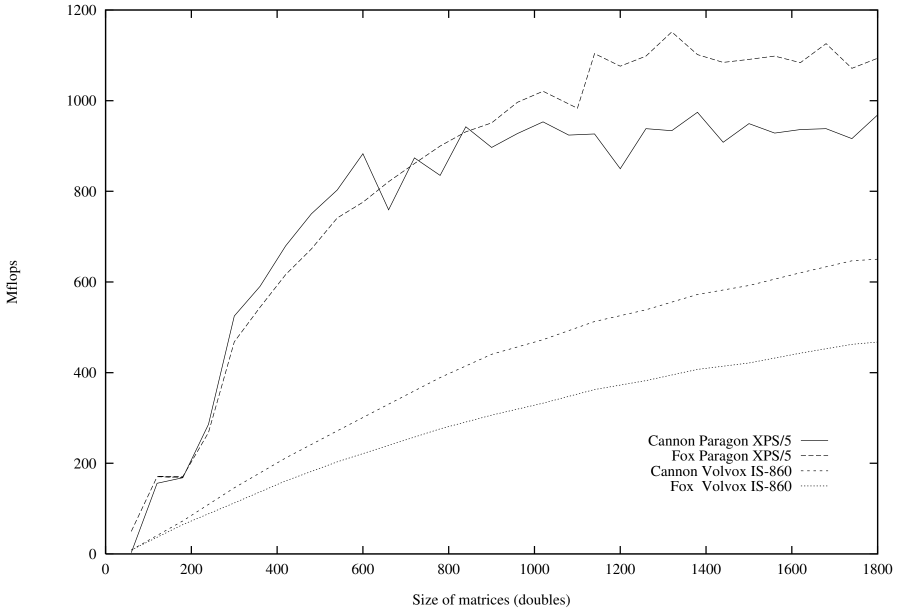

Figure 11: Performances of the matrix product with pre(post)treatments in Mflops according to the matrix size on VolVox IS-860 and Paragon XPS/5 for a 6×6 torus/grid

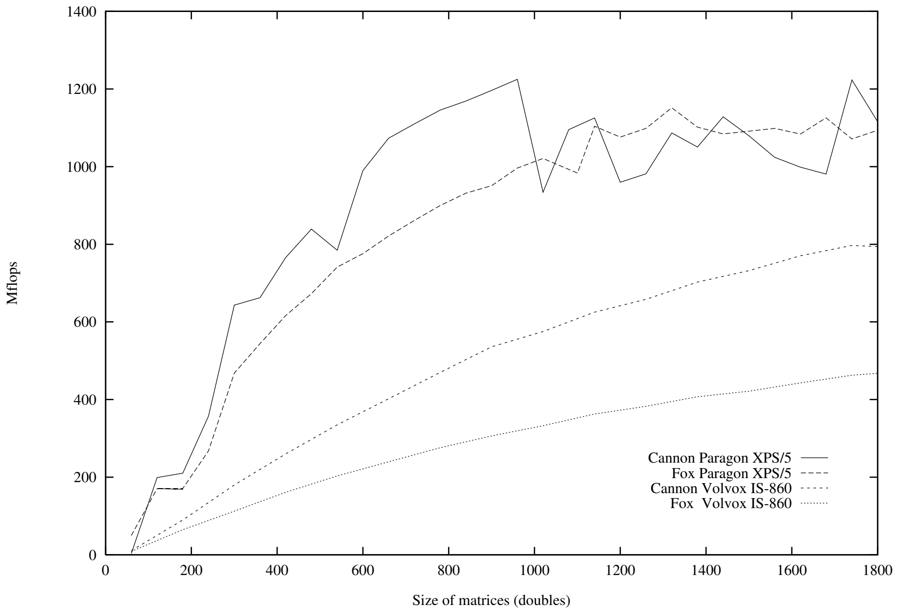

Figure 12: Performances of the matrix product without pre(post)treatments in Mflops according to the matrix size on VolVox IS-860 and Paragon XPS/5 for a 6×6 torus/grid