A Parallel Implementation of a "Symmetric Matrix Product"

S. Domas
LIP, CNRS URA 1398,
École Normale Supérieure de Lyon
46, Allée d'Italie
69364 LYON cedex 07 *

February 12, 1997

1 Introduction

The PBLAS library [2] is the parallel conterpart of level 1, 2 and 3 BLAS. It contains several subroutines that compute basic parallel linear algebra operations such as the general matrix multiplication (`PDGEMM` $:\alpha AB + \beta C \rightarrow C$). Sometimes, this subroutine is too general, especially when the result matrix is symmetric. In this case, there are redundancies of computations since the result matrix is completely computed. Furthermore, this matrix can be not perfectly symmetric because of rounding errors. For these two reasons, we have implemented a new subroutine in a "ScaLAPACK way"[1] that computes the multiplication of two symmetric commuting matrices. It works like a PBLAS but only computes the upper triangular part of the result matrix. The lower triangular part is obtained by transposition.

The first section presents the parallel algorithm used. The second section gives experimental results on the Intel Paragon, a comparison with the `PDGEMM` subroutine and a theoretical execution time evaluation.

2 Parallel algorithm

We want to parallelize the sequential operation $\alpha AB + \beta C \rightarrow C$ where A, B and C are $M \times M$ symmetric matrices (with A and B commuting), α and β some reals. The code must be ScaLAPACK compatible, that means portable and scalable. It also implies a block scattered distribution of the matrices over a virtual grid of $P_r \times P_c$ processors. Blocks are sub-matrices of size $S_b \times S_b$. Figure 1 shows an example of a block scattered distribution of a $8.S_b \times 8.S_b$ matrix over a 2×3 grid. Each block is stored in the local memory of a particular processor. For example, all gray blocs are on processor 4.

Let $A^{x,y}$ be the sub-matrix of A owned by the processor (x, y) in the grid. On Figure 1, processor 4 has the $(1, 1)$ coordinates. Thus, $A^{1,1}$ refers to its own sub-matrix of A and is composed of all gray blocks.

Let $A^{x,y}_{i\rightarrow j,k\rightarrow l}$ be the sub-matrix of $A^{x,y}$ beginning at the block (i, k) and ending at the block (j, l). In the example, the gray zone of $A^{1,1}$ begins at the block $(1, 1)$ and ends at the block $(3, 2)$.

*This research has been supported by the CNRS–ENS Lyon–INRIA *ReMaP* project and the EUREKA-EUROTOPS project.

Thus, this zone is referred by $A^{1,1}_{1\to 3,1\to 2}$. It can be simplified in $A^{1,1}_{1\to,1\to}$ because $(3,2)$ refers to the right bottom corner of the local matrix $A^{1,1}$. A single block is referred by $A^{x,y}_{i,k}$.

According to the PBLAS paradigms, A, B and C matrices are entirely stored in memory even if they are symmetric. Furthermore, the whole result matrix must be produced, not only one triangular part. The goal of our algorithm is to compute the upper triangular part of the result matrix and then to "transpose" it to obtain a complete and perfectly symmetric result. The computation must be divided in two parts since the diagonal blocks must be "half" computed (only the upper triangular part of the block) but not the non-diagonal blocks. From these hints, we construct a three steps algorithm presented in Figure 4. Each processor on the grid executes it concurrently.

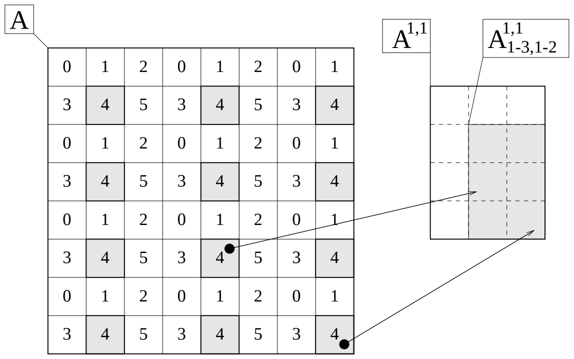

Figure 1: Block scattered distribution of an $8.S_b \times 8.S_b$ matrix over a 2×3 grid.

• **Phase 1 :** The diagonal blocks are computed one after another. Using the symmetry, the i^{th} diagonal block is the product of the i^{th} column of blocks of A with the i^{th} column of blocks of B. These columns are distributed on the same processors. Thus, there is no communication before the computation and P_c diagonal blocks can be computed at a time. Nevertheless, each processor that owns a part of the i^{th} column of blocks only computes a part of the i^{th} diagonal block. Thus, a global sum over these processors is necessary to reconstruct the final result. Furthermore, only the upper triangular part of the diagonal blocks must be computed. A new routine has been developed since no BLAS 3 can do this work. It is referred as `MUL` in figure 4.

• **Phase 2 :** The upper triangular part is computed one row of block after another. The i^{th} row is the product of the i^{th} column of blocks of A (using the symmetry) and the last $\left\lceil \frac{M}{S_b} \right\rceil - i - 1$ columns of blocks of B. Minus 1 insures that the i^{th} diagonal block is not recomputed. This means that the i^{th} column of blocks of B must not be multiplied (from which the δ introduced in Figure 4. As B is distributed in a block scattered way, the i^{th} column of blocks of A is not on all the processors that own B. Thus, it must be broadcasted before the multiplication (`DGEBS2D`). Then each processor multiplies (`DGEMM`) the part it received (`DGEBR2D`) with the local sub-matrix of B it owns. After the computation, a global sum (`DGMAX2D`) is done to reconstruct the final result. It can be noticed that the i^{th} row is computed in one step and not block after block. It increases the granularity of the computation and leads to a better efficiency of the level 3 BLAS `DGEMM`.

• **Phase 3 :** The upper triangular part of C is "transposed" in C to produce a perfectly symmetric matrix. Actually, it is only a replication, using point-to-point communications between extra-diagonal processors (from the matrix point of view), and not a real transposition.

3 Experimental results

Measures presented in Figures 2 and 3 have been produced on the Intel Paragon system from the IRISA in Rennes. Other tests are in progress on the IBM SP2 from the LaBRI in Bordeaux. Figure 2 gives the execution time as a function of the matrix size for `PDGEMM` and our routine `PDSCMM`.

PDGEMM is the ScaLAPACK subroutine that computes the general matrix multiplication in double precision without exploiting the symmetry. Its algorithm is quite similar to the phase 2 of our algorithm except that the whole B matrix is multiplied at each step. The theoretical time has been computed from a modelization (that will be given in the full paper) of the execution time of the BLAS and BLACS (DGEMM, DGEBS2D...) used in each methods. It gives very accurate estimations (less than 10% from the experimental results). The levels at fixed intervals are characteristic of the algorithm. For matrices of size $i.(P_c \times S_b) + S_b$, the number of blocks to be computed increases drastically. This increase is more important for PDGEMM. It leads to an oscillating gain presented in Figure 3. The gain is the ratio between the execution time of the two subroutines. As expected, it raises to 2.

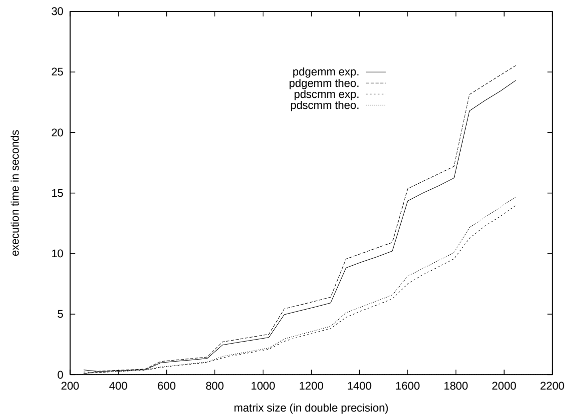

Figure 2: Theoretical and experimental execution times on a 4×4 grid.

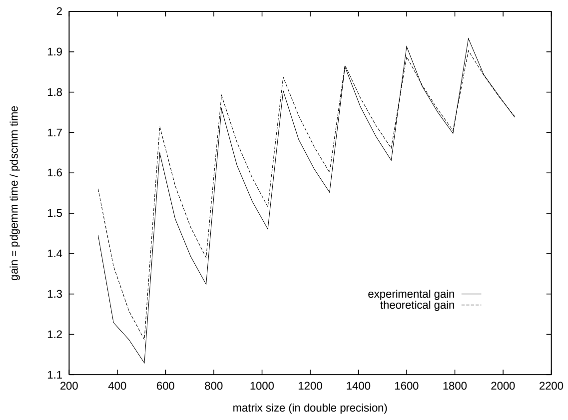

Figure 3: Ratio between the execution times given in Figure 4

4 Conclusion

Our subroutine PDSCMM is efficient since the execution time is divided by at most 2 compared with the full multiplication. Moreover, it insures a perfect symmetry of the result matrix. Thus, it can be used in a computation chain without rounding errors problems. Furthermore, the theoretical model gives an accurate estimation of the execution time and the optimal block size for a given matrix size and grid size.

References

[1] 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.

[2] 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, Department of Computer Science - University of Tennessee, 1995.

(mr, mc) are my coordinates in the grid.
$Work$ et $Work'$ are buffers

/* diagonal block computation */
For i $= 0$ *to* $\left\lfloor \frac{M}{S_b} \right\rfloor$ *do*
$cr =$ i mod P_r ; $cc =$ i mod P_c
If ($mc = cc$) *then*
compute $A^T{}^{mr,mc}_{0\rightarrow,\left\lfloor \frac{i}{P_c} \right\rfloor} \times B^{mr,mc}_{0\rightarrow,\left\lfloor \frac{i}{P_c} \right\rfloor} \rightarrow Work$ /* MUL */
global sum of $Work$ on col. cc /* DGSUM2d */
result is left on processor (cr, cc). phase 1
If ($mr = cr$) *then*
add $Work$ to $\beta.C^{mr,mc}_{\left\lfloor \frac{i}{P_r} \right\rfloor,\left\lfloor \frac{i}{P_c} \right\rfloor}$ /* DMATADD */
Endif
Endif
Endfor

/* non diagonal block computation */
For i $= 0$ *to* $\left\lfloor \frac{M}{S_b} \right\rfloor - 1$ *do*
$cr =$ i mod P_r ; $cc =$ i mod P_c
If ($mc = cc$) *then*
broadcast $A^{mr,mc}_{0\rightarrow,\left\lfloor \frac{i}{P_c} \right\rfloor}$ on row mr of processors /* DGEBS2D */
Else
receive from processor (mr, cc), $A^{mr,cc}_{0\rightarrow,\left\lfloor \frac{i}{P_c} \right\rfloor} \rightarrow Work$ /*DGEBR2D
*/
Endif
/* diagonal blocks are already computed then: */ phase 2
If $(mc = cc)$ *then* $\delta = 1$ *Else* $\delta = 0$
Endif
compute $Work^T \times B^{mr,mc}_{0\rightarrow,\left\lfloor \frac{i}{P_c} \right\rfloor+\delta\rightarrow} \rightarrow Work'$ /* DGEMM */
global sum of $Work'$ on col. mc /* DGSUM2d */
result is left n processor (cr, mc).
If ($mr = cr$) *then*
add $Work'$ to $\beta.C^{mr,mc}_{\left\lfloor \frac{i}{P_r} \right\rfloor,\left\lfloor \frac{i}{P_c} \right\rfloor+\delta\rightarrow}$ /* DMATADD */
Endif
Endfor

transpose each diagonal of the upper triangular part of C phase 3

Figure 4: Parallel algorithm of the symmetric commuting matrix multiplication.