TR/PA/92/74 (Revised)

A Parallel Block Implementation of Level 3 BLAS for MIMD Vector Processors

Michel J. Daydé[2,4], Iain S. Duff[1,3], Antoine Petitet[1]

March 25, 1993

Abstract

We describe an implementation of Level 3 BLAS based on the use of the matrix-matrix multiplication kernel (GEMM). Blocking techniques are used to express the BLAS in terms of operations involving triangular blocks and calls to GEMM. A principal advantage of this approach is that most manufacturers provide at least an efficient serial version of GEMM so that our implementation can capture a significant percentage of the computer performance. A parameter which controls the blocking allows an efficient exploitation of the memory hierarchy of the various target computers. Furthermore, this blocked version of Level 3 BLAS is naturally parallel. We present results on the ALLIANT FX/80, the CONVEX C220, the CRAY-2, and the IBM 3090/VF. For GEMM, we always use the manufacturer-supplied versions. For the operations dealing with triangular blocks, we use assembler or tuned Fortran (using looop-unrolling) codes, depending on the efficiency of the available libraries.

Keywords : Vectorization, parallelization, Level 3 BLAS, matrix-matrix kernels.

[1]CERFACS, 42 av. G. Coriolis, 31057 Toulouse Cedex, France
[2]ENSEEIHT-IRIT, 2 rue Camichel, 31071 Toulouse CEDEX, France
[3]Also Rutherford Appleton Laboratory, OXON OX11 0QX, England.
[4]also visiting scientist in the Parallel Algorithms Group at CERFACS

Contents

1 Introduction

This report describes an implementation of the Level 3 BLAS computational kernels ([9],[10]). This implementation is based on the use of the general matrix-matrix multiplication kernel GEMM. We show that this implementation is portable and is efficient if there exists a tuned version of GEMM. We also show that it is a natural way of parallelizing the BLAS kernels (a loop-level parallelism is sufficient). Therefore, it can be used as a platform both for serial and parallel implementations of the Level 3 BLAS.

The very good ratio of flops over memory references makes the matrix-matrix multiplication kernel GEMM capable of attaining nearly the peak performance on most computers. Therefore, we would like to take advantage of the efficiency of the GEMM kernel in order to improve the Level 3 BLAS when dealing with large enough matrices. Our basic idea for designing the Level 3 BLAS is to partition the computations across submatrices so that the calculations can be expressed in terms of calls to GEMM and operations involving triangular matrices. Depending on the availability of tuned kernels in the manufacturer-supplied libraries, we use assembler or tuned Fortran coded kernels for the operations on triangular submatrices. The efficiency of this implementation (especially when dealing with large enough matrices) is demonstrated on vector and parallel computers. We illustrate this on the ALLIANT FX/80, the CONVEX C220, the CRAY-2, and the IBM 3090-600/VF.

This study was originally motivated by the very poor performance of some of the Level 3 BLAS kernels supplied in the Scientific library on the ALLIANT FX/80 (especially the symmetric rank-k update that is crucial for the performance of Cholesky factorization). Our implementation combines the use of blocking and loop unrolling techniques. We believe that these codes constitute a good approach for a first implementation of the Level 3 BLAS on computers that do not provide a highly tuned version of the Level 3 BLAS.

The implementation of the kernels using both blocking and loop unrolling is described in Section 2 (examples of codes for the ALLIANT FX/80 are reported in the appendix). The corresponding results on the ALLIANT are reported in Section 3. Section 4 presents the results obtained on the other computers. We present conclusions in Section 5. All the computations are performed using 64-bit arithmetic, i.e. we have considered the double precision kernels (DGEMM, DSYMM, DSYRK, DSYR2K, DTRMM, and DTRSM) on the ALLIANT, the CONVEX, and the IBM, and the single precision version on the CRAY-2.

The ALLIANT FX/80 has 8 processors which access the shared memory through a 512 Kbytes cache memory organized into 4 banks. The cache is direct mapped and connected to the processors via a crossbar switch. The cache is connected to the main memory by a memory bus capable of delivering half the bandwidth of that between cache and processors. Each processor contains eight 64-bit vector registers of length 32. The cycle time is 85 ns. The peak performance is 23.6 Megaflops per CPU. A concurrency control bus connects the processors to effect synchronizations. This characteristic plus the cache management allows efficient parallelization of small granularity calculations using a loop-based parallelism : the overheads remain limited and the load balancing is handled by the hardware.

The CONVEX C220 has two vector processors. Each vector processor has 8 vector registers of length 128 and one scalar unit that access data through a 4 Kbytes cache. The cycle time of the machine is 40 ns, therefore the peak performance of one processor is 50 Mflops and the peak performance of the configuration is 100 Mflops. The processors access the shared memory via a crossbar switch (the memory size of our target computer is 128 Mbytes). The memory is interleaved and organized into banks (16 banks with 128 Mbytes of memory). The memory bandwidth between the processors and the memory is 200 Mbits per second.

The CRAY-2 is a four processor architecture with a relatively slow large main memory and a faster but smaller local memory. It is important for efficiency to take advantage of the local memory because it is the only way of reducing memory contention which is a crucial factor on this machine. The memory is divided into 128 banks on the CRAY-2 and bank conflict occurs if accesses are made to the same bank within 47 clock periods (we use a CRAY-2 with a dynamic RAM memory). This memory contention is more likely to occur when the computation is proceeding quickly (for example, with a high degree of vectorization or parallelism). The management of the local memory can be done efficiently only using assembler. The peak performance of each processor is 487.8 Megaflops. There are eight 64-bit vector registers of length 64.

On the IBM 3090J, each processor accesses data through a 128 Kbytes cache memory. The cost of accessing data from memory is roughly twice the cost of accessing data from the cache. The main memory is divided in cache lines consisting of 256 bytes (thirty two 64-bit words). When one word of main memory is accessed all the cache line is loaded into the cache. This cache line will flush another, so that it is important to make full use of data in the cache ([21]). There are 8 vector registers of length 256 (16 vector registers using single precision). Our experiments are performed on a configuration with six vector facilities. The clock period is 14.5 ns, providing a peak performance of 138 Mflops for each vector facility.

2 Block implementation of Level 3 BLAS

2.1 Efficient exploitation of the memory hierarchy

The ability of the memory to supply data to the processors at a sufficient rate is crucial on most modern computers. This results in complex memory organizations, where the memory is usually organized in a hierarchical manner. Therefore, the minimization of data transfers between the levels of the memory hierarchy is a key issue for performance ([12], [13]). The code performance can be substantially increased using a proper organization of the calculations (to help the detection of chaining for example) and by using loop unrolling and blocking techniques.

All the Level 3 BLAS kernels, except GEMM, involve upper or lower triangular matrices. Therefore a reduction in vector length during the computation cannot be avoided. Additionally, triangular matrices, as compared to rectangular matrices, do not allow as efficient access patterns to the various levels of the memory (such as cache or local memory) and

as efficient use of the data held in the vector registers. Our basic idea is to express all the Level 3 BLAS kernels in terms of subkernels dealing with NB $\times$ NB submatrices that involve GEMM operations in addition to operations dealing with triangular submatrices. Of course, the relative efficiency of this approach depends on the availability of a highly tuned GEMM and the efficiency of the implementation of the rest of the Level 3 BLAS. This approach is machine independent : only the NB parameter, corresponding to the block size, should be tuned according to the characteristics of the target machine. Both the size of the cache and the length of the vector registers determine its optimal value. Depending on the performance of the manufacturer-supplied library kernels on ALLIANT, CONVEX, CRAY, and IBM, we use library or tuned Fortran versions of the Level 3 BLAS for triangular matrices, the version of GEMM we use is always the manufacturer-supplied one.

On the ALLIANT FX/80, for example, the memory hierarchy involves the vector registers, the shared cache, and the central memory. Loop unrolling allows efficient reuse of the operands in the vector registers, while blocking is used to perform the calculations on submatrices that can fit in cache. The size of the submatrices is controlled by the blocking parameter NB. One could say that loop unrolling is used to enhance the performance of the kernels on small matrices, while blocking the computations allows us to handle large matrices efficiently.

In the following sections, we describe the block implementation of the Level 3 BLAS. Our blocked versions of the Level 3 BLAS will be termed SYMMP, TRSMP, TRMMP, SYRKP, and SYR2KP to differentiate them from the unblocked Level 3 BLAS that will be used on submatrices (in practice all these names are prefixed by S or D depending on whether the routine is single or double precision).

For the sake of clarity, we comment only on one of the variants of the kernels and we illustrate our blocking strategy on matrices that are only partitioned into four blocks. In practice, the matrices are partitioned into NB $\times$ NB blocks where NB is chosen according to the machine characteristics. For example, a 128 $\times$ 128 matrix will be partitioned into 16 blocks with NB equal to 32, while it will only be partitioned into 4 blocks if NB is equal 64.

2.2 Subroutine SYMM

SYMM performs one of the matrix-matrix operations :

$$\mathbf{C}=\alpha.\mathbf{AB}+\beta.\mathbf{C}, \text{ or } \mathbf{C}=\alpha.\mathbf{BA}+\beta.\mathbf{C}$$

where α and β are scalars, $\mathbf{A}$ is a symmetric matrix (only the upper or lower triangular parts are used) and $\mathbf{B}$ and $\mathbf{C}$ are m $\times$ n matrices.

We consider the following case (corresponding to the parameters “Left”, “Upper”):

$$\begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix} = \alpha . \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{1,2}^t & A_{2,2} \end{pmatrix} \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} + \beta . \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

1. $C_{1,1} \leftarrow \beta . C_{1,1} + \alpha . A_{1,1} B_{1,1}$ (SYMM)
2. $C_{1,1} \leftarrow C_{1,1} + \alpha . A_{1,2} B_{2,1}$ (GEMM)
3. $C_{1,2} \leftarrow \beta . C_{1,2} + \alpha . A_{1,1} B_{1,2}$ (SYMM)
4. $C_{1,2} \leftarrow C_{1,2} + \alpha . A_{1,2} B_{2,2}$ (GEMM)
5. $C_{2,1} \leftarrow \beta . C_{2,1} + \alpha . A_{2,2} B_{2,1}$ (SYMM)
6. $C_{2,1} \leftarrow C_{2,1} + \alpha . A_{1,2}^t B_{1,1}$ (GEMM)
7. $C_{2,2} \leftarrow \beta . C_{2,2} + \alpha . A_{2,2} B_{2,2}$ (SYMM)
8. $C_{2,2} \leftarrow C_{2,2} + \alpha . A_{1,2}^t B_{1,2}$ (GEMM)

Therefore, the SYMMP kernel can be expressed as a sequence of calls to the routines SYMM and GEMM. The calculations of the submatrices of C is independent so that the parallelization of this operation is straightforward. The codes corresponding to the blocked and loop-unrolled versions of the symmetric matrix-matrix multiplication are reported in the appendices.

2.3 Subroutine TRSM

TRSM solves one of the matrix equations :

$$\mathbf{A.X}=\alpha.\mathbf{B},\ \mathbf{A^t.X}=\alpha.\mathbf{B},\ \mathbf{X.A}=\alpha.\mathbf{B},\ \text{or}\ \mathbf{X.A^t}=\alpha.\mathbf{B}$$

where α is a scalar, **X** and **B** are m $\times$ n matrices and **A** is a unit, or non-unit, upper or lower triangular matrix. **B** is overwritten by **X**.

We consider the following case (corresponding to the parameters "Left", "No transpose", and "Upper"):

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ 0 & A_{2,2} \end{pmatrix} \begin{pmatrix} X_{1,1} & X_{1,2} \\ X_{2,1} & X_{2,2} \end{pmatrix} = \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix}$$

1. Solution of $A_{2,2} X_{2,1} = B_{2,1}$ and $B_{2,1}$ is overwritten by $X_{2,1}$ (TRSM)

2. Solution of $A_{2,2}X_{2,2}$=$B_{2,2}$ and $B_{2,2}$ is overwritten by $X_{2,2}$ (TRSM)

3. $B_{1,1} \leftarrow B_{1,1}$-$A_{1,2}B_{2,1}$ (GEMM)

4. $B_{1,2} \leftarrow B_{1,2}$-$A_{1,2}B_{2,2}$ (GEMM)

5. Solution of $A_{1,1}X_{1,1}$=$B_{1,1}$ and $B_{1,1}$ is overwritten by $X_{1,1}$ (TRSM)

6. Solution of $A_{1,1}X_{1,2}$=$B_{1,2}$ and $B_{1,2}$ is overwritten by $X_{1,2}$ (TRSM)

Therefore, TRSMP can be computed as a sequence of triangular solutions (TRSM) and matrix-matrix multiplications (GEMM). Previous experiments that we did on the CRAY-2 ([7]) were performed using 2 $\times$ 2 diagonal blocks (where we were using a rank-two update from the Harwell Subroutine Library). The block columns of the matrix **X** can be computed simultaneously.

2.4 Subroutine TRMM

TRMM performs one of the matrix-matrix operations :

$$\mathbf{B}=\alpha.\mathbf{A}.\mathbf{B},\ \mathbf{B}=\alpha.\mathbf{A}^{\mathbf{t}}.\mathbf{B},\ \text{or}\ \mathbf{B}=\alpha.\mathbf{B}.\mathbf{A},\ \mathbf{B}=\alpha.\mathbf{B}.\mathbf{A}^{\mathbf{t}}$$

where α is a scalar, **B** is an m $\times$ n matrix, **A** is a unit, or non-unit, upper or lower triangular matrix.

We consider the following case (corresponding to the parameters "Left", "No transpose", and "Upper"):

$$\begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} = \alpha. \begin{pmatrix} A_{1,1} & A_{1,2} \\ 0 & A_{2,2} \end{pmatrix} \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix}$$

1. $B_{1,1} \leftarrow \alpha.A_{1,1}B_{1,1}$ (TRMM)

2. $B_{1,1} \leftarrow B_{1,1}$+$\alpha.A_{1,2}B_{2,1}$ (GEMM)

3. $B_{1,2} \leftarrow \alpha.A_{1,1}B_{1,2}$ (TRMM)

4. $B_{1,2} \leftarrow B_{1,2}$+$\alpha.A_{1,2}B_{2,2}$ (GEMM)

5. $B_{2,1} \leftarrow \alpha.A_{2,2}B_{2,1}$ (TRMM)

6. $B_{2,2} \leftarrow \alpha.A_{2,2}B_{2,2}$ (TRMM)

TRMMP is expressed as a sequence of GEMM and TRMM operations. The computations of the submatrices of **B** within the same block row are independent.

2.5 Subroutine SYRK

SYRK performs one of the symmetric rank-k operations :

$$\mathbf{C}=\alpha.\mathbf{A}\mathbf{A}^{\mathbf{t}}+\beta.\mathbf{C}, \text{ or } \mathbf{C}=\alpha.\mathbf{A}^{\mathbf{t}}\mathbf{A}+\beta.\mathbf{C}$$

where α and β are scalars, $\mathbf{C}$ is an n $\times$ n symmetric matrix (only the upper or lower triangular parts are updated), and $\mathbf{A}$ is a n $\times$ k matrix in the first case and a k $\times$ n matrix in the second case.

We consider the following case (corresponding to "Upper", and "No transpose"):

$$\begin{pmatrix} C_{1,1} & C_{1,2} \\ 0 & C_{2,2} \end{pmatrix} = \alpha. \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \begin{pmatrix} A_{1,1}^t & A_{2,1}^t \\ A_{1,2}^t & A_{2,2}^t \end{pmatrix} + \beta. \begin{pmatrix} C_{1,1} & C_{1,2} \\ 0 & C_{2,2} \end{pmatrix}$$

1. $C_{1,1} \leftarrow \beta.C_{1,1} + \alpha.A_{1,1}A_{1,1}^t$ (SYRK)
2. $C_{1,1} \leftarrow C_{1,1} + \alpha.A_{1,2}A_{1,2}^t$ (SYRK)
3. $C_{1,2} \leftarrow \beta.C_{1,2} + \alpha.A_{1,1}A_{2,1}^t$ (GEMM)
4. $C_{1,2} \leftarrow C_{1,2} + \alpha.A_{1,2}A_{2,2}^t$ (GEMM)
5. $C_{2,2} \leftarrow \beta.C_{2,2} + \alpha.A_{2,1}A_{2,1}^t$ (SYRK)
6. $C_{2,2} \leftarrow C_{2,2} + \alpha.A_{2,2}A_{2,2}^t$ (SYRK)

Again, the symmetric rank-k update is expressed as a sequence of SYRK for updating the diagonal blocks and GEMM for the other blocks. The updates of the submatrices of $\mathbf{C}$ can be performed independently.

2.6 Subroutine SYR2K

SYR2K performs one of the symmetric rank-2k operations:

$$\mathbf{C}=\alpha.\mathbf{A}\mathbf{B}^{\mathbf{t}}+\alpha.\mathbf{B}\mathbf{A}^{\mathbf{t}}+\beta.\mathbf{C}, \text{ or } \mathbf{C}=\alpha.\mathbf{A}^{\mathbf{t}}\mathbf{B}+\alpha.\mathbf{B}^{\mathbf{t}}\mathbf{A}+\beta.\mathbf{C}$$

where α and β are scalars, $\mathbf{C}$ is an n $\times$ n symmetric matrix (only the upper or lower triangular parts are updated) and $\mathbf{A}$ and $\mathbf{B}$ are n $\times$ k matrices in the first case and k $\times$ n matrices in the second case.

We consider the following case (corresponding to "Upper", and "No transpose"):

$$\begin{pmatrix} C_{1,1} & C_{1,2} \\ 0 & C_{2,2} \end{pmatrix} = \alpha . \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \begin{pmatrix} B_{1,1}^t & B_{2,1}^t \\ B_{1,2}^t & B_{2,2}^t \end{pmatrix}$$

$$+\alpha . \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} \begin{pmatrix} A_{1,1}^t & A_{2,1}^t \\ A_{1,2}^t & A_{2,2}^t \end{pmatrix} + \beta . \begin{pmatrix} C_{1,1} & C_{1,2} \\ 0 & C_{2,2} \end{pmatrix}$$

1. $C_{1,1} \leftarrow \beta . C_{1,1} + \alpha . A_{1,1} B_{1,1}^t + \alpha . B_{1,1} A_{1,1}^t$ (SYR2K)
2. $C_{1,1} \leftarrow C_{1,1} + \alpha . A_{1,2} B_{1,2}^t + \alpha . B_{1,2} A_{1,2}^t$ (SYR2K)
3. $C_{1,2} \leftarrow \beta . C_{1,2} + \alpha . A_{1,1} B_{2,1}^t$ (GEMM)
4. $C_{1,2} \leftarrow C_{1,2} + \alpha . B_{1,1} A_{2,1}^t$ (GEMM)
5. $C_{1,2} \leftarrow C_{1,2} + \alpha . A_{1,2} B_{2,2}^t$ (GEMM)
6. $C_{1,2} \leftarrow C_{1,2} + \alpha . B_{1,2} A_{2,2}^t$ (GEMM)
7. $C_{2,2} \leftarrow \beta . C_{2,2} + \alpha . A_{2,1} B_{2,1}^t + \alpha . B_{2,1} A_{2,1}^t$ (SYR2K)
8. $C_{2,2} \leftarrow C_{2,2} + \alpha . A_{2,2} B_{2,2}^t + \alpha . B_{2,2} A_{2,2}^t$ (SYR2K)

SYR2KP is expressed as a sequence of SYR2K for updating the diagonal triangular blocks, and GEMM on on the other blocks. The update of the submatrices of **C** can be effected simultaneously.

3 Numerical results on the ALLIANT FX/80

We use Version 5.0 of the Libalgebra Library. We only present results corresponding to the best value of the blocking parameter (NB). Note that all the operations within the BLAS kernels are parallelized using the loop-level parallelism and that the versions of Level 3 BLAS supplied in the ALLIANT Scientific Library are already parallelized. Therefore, all our experiments on the ALLIANT FX/80 have been performed using eight processors.

3.1 Subroutine SYMM

We present in Table 3.1.1 and Table 3.1.2 the results we obtained for the case "Left" and "Upper" for SYMMP (the results are similar for the other cases). The advantage of using both the loop-unrolling and the blocking techniques is illustrated in the tables below. We compare the performance of the unrolled version of SYMM (see code in the appendix) with the performance of the unrolled and blocked version SYMMP. The unrolling depth is equal

	Unrolled				Blocked and unrolled			
M/N	32	64	96	128	32	64	96	128
32	10.8	12.3	12.5	12.7	10.7	11.7	11.4	11.8
64	16.5	16.9	16.9	17.1	15.8	16.2	15.6	16.2
96	19.2	19.8	20.3	20.6	22.3	22.5	22.5	23.2
128	20.8	21.1	21.7	21.9	24.0	25.8	26.2	26.7

Table 3.1.1: Comparison in Mflops of loop-unrolled and blocked plus loop-unrolled versions of SYMM ("Left", "Upper", NB = 64) using eight processors.

to 4, while the blocking parameter is set to 64 (chosen after extensive experimentation). All the executions are made on eight processors.

The blocked version is more efficient as soon as the matrices are large enough (M greater than 64), partly because of the more favourable memory access patterns and because of the very well tuned version of GEMM supplied by ALLIANT. On small matrices, the calls to GEMM and SYMM within SYMMP cause a slight performance degradation compared to the unrolled code.

M=N	ALLIANT Version	Blocked Version
32	1.5	10.7
64	2.8	16.2
128	5.2	26.7
256	8.9	37.8
512	12.7	45.3
1024	16.2	52.1

Table 3.1.2: Performance in Mflops of DSYMMP ("Left" and "Upper", NB = 64) using eight processors.

The performance improvement of our tuned version over the version supplied by ALLIANT is very impressive. Our tuned code is much more efficient on small matrices because of the loop-unrolling, while our blocking technique allows us to increase the performance when dealing with large matrices. Therefore, DSYMMP is 10 times faster on small matrices and 4 to 5 five times faster on large matrices.

3.2 Subroutine TRSM

We present in Table 3.2.1 the results we obtained for the case "Left", "No transpose", "Upper", and "Unit" :

The ALLIANT-supplied version of DTRSM is well-tuned on small blocks, therefore our blocked version is no better because of the additional overhead due to the calls to the ALLIANT version of DTRSM within DTRSMP (and additional error tests on the input parameters). Nevertheless, our blocked version is more efficient as soon as the matrices are large enough, because of the use of DGEMM. We notice that the performance of the

M=N	ALLIANT Version	Blocked Version
32	14.3	13.7
64	36.4	35.7
128	56.9	50.2
256	54.4	58.8
512	50.7	65.4
1024	48.9	68.2

Table 3.2.1: Performance in Mflops of DTRSMP ("Left", "No transpose", "Upper", and "Unit", NB = 64) using eight processors.

ALLIANT version decreases from matrices of order 128, while it continues to increase using our blocked version. This is typically due to an increase in caches misses in the ALLIANT version, while the blocked version has a more efficient reuse of data in the cache.

3.3 Subroutine TRMM

We present in Table 3.3.1 the results we obtained for the case "Left", "No transpose", "Upper", "Unit".

M=N	ALLIANT Version	Blocked Version
32	14.4	13.8
64	35.6	29.3
128	51.6	50.0
256	53.2	59.4
512	49.1	67.2
1024	48.3	70.5

Table 3.3.1: Performance in Mflops of DTRMMP ("Left", "No transpose", "Upper", and "Unit", NB = 64) using eight processors.

As for the case of DTRSMP, the overhead of additional calls to DTRMMP from ALLIANT in our blocked version makes it slightly slower on small matrices where the blocking does not help. As soon as the matrices are large enough, the blocking produces a higher performance than the ALLIANT version of DTRMM.

3.4 Subroutine SYRK

We present in Table 3.4.1 the results we obtained for the case "Upper", "No transpose".

The ALLIANT supplied DSYRK kernel is very poor. Note that the performance of this kernel is crucial for the performance of blocked Cholesky factorization. Our Fortran tuned

M=N	ALLIANT Version	Blocked Version
32	4.7	11.5
64	6.7	20.2
128	8.1	30.2
256	8.5	40.2
512	8.1	52.2
1024	7.3	59.1

Table 3.4.1: Performance in Mflops of DSYRKP ("Upper", and "No transpose", NB = 64) using eight processors.

version using loop-unrolling is very efficient on small matrices, and provides a very substantial improvement in performance. On large matrices, the blocked implementation of SYRK is 8 times faster than the ALLIANT supplied version.

3.5 Subroutine SYR2K

We present in Table 3.5.1 the results we obtained for the case "Upper", "No transpose".

M=N	ALLIANT Version	Blocked Version
32	7.5	13.3
64	10.8	20.6
128	13.1	32.5
256	13.2	43.9
512	12.2	50.0
1024	11.1	57.3

Table 3.5.1: Performance in Mflops of DSYR2KP ("Upper", and "No transpose", NB = 64) using eight processors.

DSYR2K in the ALLIANT Library is slightly more efficient than DSYRK, but still performs quite slowly. Our tuned version achieves much higher performance both on small and large matrices.

3.6 Conclusions on the ALLIANT FX/80

Our blocked version of the Level 3 BLAS behaves similarly for all the variants of the kernels. It shows a very good performance improvement over the one supplied within the ALLIANT Scientific Library. This is especially true for the symmetric rank-k updates (SYRK and SYR2K). In addition, our version is much more efficient on small matrices (SYMM, SYRK, and SYR2K) which is of special interest for block linear algebra algorithms. We did not

perform experiments using single precision, but similar performance improvement should be obtained.

As we have seen on the ALLIANT FX/80, the availability of an efficient manufacturer-supplied version of GEMM allows us to obtain a better performance using loop unrolling and blocking techniques. Our code is obviously portable, the efficiency will depend on the performance of the manufacturer supplied GEMM and on the tuned Fortran codes used for handling triangular blocks.

4 Experiments on the CONVEX C220, the CRAY-2, and the IBM 3090-600J

We have run the same codes on the CONVEX C220, the CRAY-2, and the IBM 3090-600J. This means that, when no highly-tuned versions of SYMM, SYRK, SYR2K, TRSM ,and TRMM, are available for dealing with diagonal blocks within the blocked Level 3 BLAS, we have used the same tuned Fortran code with loop-unrolling as on the ALLIANT FX/80. No special attempt has been made for further tuning. Therefore, these tuned codes may be far from optimal on these computers. On all the computers, the reported experiments correspond to the same cases as on the ALLIANT FX/80.

4.1 Uniprocessor performance on the CONVEX C220

As on the Alliant FX/80, we have used a combination of tuned Fortran and manufacturer-supplied kernels depending on the efficiency of the VECLIB Library (we used Version 5.0). Note that we make no effort to adapt the tuned Fortran codes designed on the ALLIANT for the CONVEX. We always make use of the matrix-matrix multiplication (DGEMM) from VECLIB.

DSYMMP uses DSYMM and DGEMM from the VECLIB Library for the right variants, while we use DGEMM from VECLIB and the tuned Fortran code for the left ones since it is more efficient. For the same reason, we use tuned Fortran and DGEMM for the "No transpose" variants of both DSYRKP and DSYR2KP, and the VECLIB routines for the other ones. Finally, we use DTRMM and DTRSM from VECLIB.

We present in Table 4.1.1 a summary of the performance we have obtained on one processor of the CONVEX C220 using double precision.

The tuned Fortran version of DSYMM using loop-unrolling allows us to double the performance of SYMM on small matrices. The symmetric updates, DSYRKP and DSYR2KP, are more efficient on large matrices than the VECLIB supplied ones. Depending on the variant considered, our blocked version of DTRSM is sometimes slightly less efficient than the DTRSM from the VECLIB Library on small matrices because of the overhead we introduce when calling DGEMM and DTRSM from the VECLIB Library within our blocked code. Nevertheless, we always observe a gain of 2 when dealing with large enough matrices.

	CONVEX Version						Blocked Version					
M=N	32	64	128	256	512	1024	32	64	128	256	512	1024
SYMM	6.2	9.2	13.8	17.6	20.0	21.2	12.0	14.0	17.5	19.8	21.5	27.2
SYRK	5.5	8.2	11.4	11.4	12.2	12.4	8.7	13.0	20.1	25.3	30.0	33.0
SYR2K	7.6	11.5	15.8	18.6	20.3	21.4	9.3	13.1	18.9	22.4	21.5	26.2
TRSM	5.4	7.1	11.1	12.9	13.7	13.3	5.4	8.2	13.7	19.7	26.2	29.9
TRMM	5.1	7.6	10.8	12.8	13.9	14.3	5.1	7.5	13.1	19.9	26.1	30.8

Table 4.1.1: Performance comparison in Mflops of VECLIB and blocked versions of the Level 3 BLAS on one processor of the CONVEX C220 with NB=64.

DTRMMP behaves in the same way as the triangular solver DTRSMP and we observe a similar performance improvement when dealing with large enough matrices.

4.2 Performance on the CRAY-2

The CRAY SCILIB Library provides a complete set of highly tuned Level 3 BLAS kernels. Therefore, we always use the CRAY kernels. Our intention was only to prove the portability of our blocked Level 3 BLAS implementation rather than expecting an improvement over the CRAY uniprocessor version. Note that the CRAY kernels have been designed using a modular design similar to our blocking strategy. Sheik and Liu ([25]) have used a certain number of subkernels (matrix-vector, matrix-matrix kernels, ...) for implementing all the Level 2 and Level 3 BLAS. The CRAY microtasking directives have been used for parallelizing the BLAS kernels. The code corresponding to the microtasked version of SYMM is reported in Appendix A.3. The addition of a microtasking directive on the outer loop of all the codes is sufficient. Note that automatic parallelization would be sufficient, but the version of autotasking that we use does not allow the parallelization of loops involving character variables. These experiments were done using Version 5.0 of UNICOS, Version 6.0 now provides a parallel version of the Level 3 BLAS kernels.

We present in Table 4.2.1 the results corresponding to the CRAY and blocked versions of the Level 3 BLAS. **p** corresponds to the number of processors in use.

As expected, there is no performance improvement on one processor when using our blocked version but the degradation is slight. Note that the blocking strategy on this computer aims at using as efficiently as possible the local memory of each processor. The blocking of the Level 3 BLAS calculations done by CRAY code designers is obviously very efficient. Our microtasked version of Level 3 BLAS only provides speed-ups when the matrices are at least of order 128. This means that in order to guarantee speed-ups the parallel kernels should include tests to decide if it is worthwhile to use several processors. The number of processors used could also be selected according to the minimum granularity required for efficient parallelization. Finally, we believe that, rather than a static parallelization of the kernels, a self-scheduling technique would be better.

		SCILIB Version						Blocked Version					
M=N	**p**	32	64	128	256	512	1024	32	64	128	256	512	1024
SYMM	1	249	385	423	416	429	447	234	380	402	390	404	424
	4							221	365	541	721	1090	1468
SYRK	1	126	260	305	336	348	361	114	243	286	303	318	310
	4							127	261	456	666	906	1160
SYR2K	1	165	293	335	384	411	424	142	279	329	355	377	387
	4							143	271	482	785	1272	1357
TRSM	1	64	104	168	244	317	373	57	100	162	233	299	347
	4							56	101	313	430	842	1168
TRMM	1	237	386	420	436	445	450	200	372	397	406	409	411
	4							183	358	782	750	898	1073

Table 4.2.1: Performance comparison in Mflops of SCILIB and blocked uniprocessor and multiprocessor versions of the Level 3 BLAS with NB = 64.

4.3 Uniprocessor performance on the IBM 3090-600J

The IBM ESSL Library (version 3) does not provide a complete version of the Level 3 BLAS kernels. Only DGEMM and DTRSM are supplied. Therefore, we compare the performance of the standard Fortran codes for DSYMM, DSYRK, DSYR2K, and DTRMM, and the DTRSM code from ESSL, to the performance of our blocked versions. We have used the tuned Fortran codes for DTRMM, DSYMM, DSYRK, and DSYR2K, and ESSL versions of DGEMM and DTRSM. No attempt has been made to tune the Fortran codes specifically for the IBM.

	Standard Fortran/ESSL					Blocked Version				
M=N	32	64	128	256	512	32	64	128	256	512
SYMM	10.5	16.9	25.7	31.3	35.4	13.9	23.9	36.4	50.6	64.7
SYRK	9.8	16.4	23.7	27.0	28.7	18.5	32.1	47.3	62.7	68.2
SYR2K	14.5	23.8	34.0	35.7	38.6	19.0	31.9	44.1	56.3	58.6
TRSM	17.8	35.9	67.9	83.0	92.0	19.1	36.2	68.8	76.2	81.9
TRMM	9.7	15.5	22.6	26.1	27.5	9.3	15.7	20.4	35.9	50.9

Table 4.3.1: Performance comparison in Mflops of blocked and standard Fortran versions of the Level 3 BLAS (except for DTRSM from the ESSL Library), with NB = 128, on one processor.

As expected, our blocked codes are more efficient than the standard Fortran version of Level 3 BLAS as soon as the matrices are large enough. Note that Kågström and Ling ([16], [20]), and Mayes and Radicati ([21], [22]) have developed more efficient versions of the Level 3 BLAS for the IBM 3090 using a similar blocking technique. They use highly tuned Fortran versions of the kernels for the IBM, so that they reach much higher performance than us. Our blocked version of DTRSM is slightly less efficient then the DTRSM from the ESSL Library because of the overhead we introduce especially on small matrices. We observe a gain of 2 using the other kernels when dealing with large enough matrices.

5 Conclusion

The Level 3 BLAS is a set of computational kernels targeted at matrix-matrix operations with the aim of providing efficient and portable implementations of algorithms on high-performance computers, especially vector and parallel computers. The linear algebra package LAPACK ([2]), for example, makes extensive use of the Level 3 BLAS.

We have described an efficient and portable implementation of the Level 3 BLAS on parallel vector computers with a global shared memory. The development of a highly tuned version of the complete set of Level 3 BLAS may be a considerable task. We have shown here that significant MFlop rates can be achieved, only requiring a tuned version of the matrix-matrix multiplication kernel GEMM. Our approach is based on the use of a blocking technique that allows us to map the computational kernels efficiently into a memory hierarchy. Therefore, the Level 3 BLAS are expressed as a sequence of matrix-matrix multiplications (GEMM) and operations involving triangular blocks. An efficient version of GEMM is all that is needed to obtain high performance on large matrices. On small matrices (of order less than the blocking factor) the performance of the kernel dealing with triangular matrices is crucial. Depending on the availability of highly tuned manufacturer-supplied kernels, we use tuned Fortran or library kernels. In their paper on GEMM-based Level 3 BLAS, Kågström, Ling, and Van Loan ([17]) use similar ideas. They describe algorithms for the implementation of two Level 3 BLAS routines : SYRK and TRSM. They use highly tuned matrix-matrix and GEMV-based operations and report on experiments on a single processor of the IBM 3090/VF.

This blocked version of Level 3 BLAS was initially tested on the ALLIANT FX/80. The performance improvement over the ALLIANT supplied version varies from 1.5 for TRSM and TRMM to 10 for SYRK. Note that these performance improvements, especially on small matrices, have a strong effect on the performance of, for example, blocked linear solvers that make use of these kernels ([2], see experiments in [23]).

We have also demonstrated the portability of such an approach on the CONVEX C220, the CRAY-2, and the IBM 3090J. We have been able to improve the uniprocessor performance of the Level 3 BLAS on the CONVEX C220 compared to the versions supplied in the VECLIB Library. On the CRAY-2, due to the high efficiency of the manufacturer-supplied kernels we have not been able to improve performance on a single processor. Nevertheless, we have demonstrated that a parallel version of the Level 3 BLAS, obtained by the insertion of microtasking directives in our blocked codes was able to provide high performance and significant speed-ups.

This implementation of the Level 3 BLAS is suitable as a platform for developing a tuned version of the BLAS. It is also a straightforward way of parallelizing the Level 3 BLAS. It basically only requires an efficient GEMM kernel often supplied by the manufacturer. The combination of the blocking and loop unrolling techniques allows an efficient exploitation of the memory hierarchy and only the blocking parameter is machine dependent. This blocked implementation has been successfully used for developing both serial and parallel tuned versions of the Level 3 BLAS for a 30-node BBN-TC2000 ([1]).

6 Availability of codes

The codes described in the present paper are available using ftp anonymous on orion.cerfacs.fr (138.63.200.33). The software is located in pub/blas/par_blocked. A compressed tarfile called vector.mimd.tar.Z contains the following codes :

- A set of test routines that check the correct execution and compute the Mflop rates of the block implementation compared with the available uniprocessor version of the Level 3 BLAS.
- The block implementation of the Level 3 BLAS.
- Tuned versions of the unblocked Level 3 BLAS.

We believe that these codes constitute a good platform for developing a complete set of Level 3 BLAS for shared memory multiprocessors. The tuned uniprocessor code is really the machine-dependent part of the software. Therefore, we advise the user to check the availability of tuned serial codes before using our tuned Fortran codes. Note that the design of a tuned uniprocessor Level 3 BLAS on computers where the processor accesses data through a cache generally requires blocking. As a consequence, a tuned version of the Level 3 BLAS for this type of architecture may use the same blocking ideas as in the present paper, except that the ordering of loops will depend on considerations concerning the efficient reuse of data held in cache. This may lead to a blocked implementation that looks like the one we have described but where the ordering of loops is different and not necessarily appropriate for parallelization. We are currently experimenting with these ideas on a range of RISC-based workstations and will make any resulting code available on anonymous ftp.

References

[1] Amestoy, P.R., Daydé, M.J., Duff, I.S., Morère, P. (1992). Linear Algebra Calculations on the BBN TC2000. CERFACS Report TR/PA/92/69.

[2] Anderson, E., Bai, Z., Bischof, C., Demmel, J.W., Dongarra, J.J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S., and Sorensen, D.C. (1992). LAPACK Users' Guide. SIAM, Philadelphia.

[3] Berger, Ph., Daydé, M.J., Morère, P. (1991). Implementation and Use of Level 3 BLAS kernels on a Transputer T800 Ring Network. CERFACS Report TR/PA/91/54.

[4] Bischof, C. and Van Loan, C. (1987). The WY representation for products of Householder matrices. *SIAM J. Sci. Stat. Comput.* (8), 2-13.

[5] Calahan, D.A. (1986). Block-oriented, local-memory-based linear equation solution on the CRAY-2: uniprocessor algorithms. In *Proceedings international conference on parallel processing.* Washington, D.C. : IEEE Computer Society Press, 375-378.

[6] Daydé, M.J., and Duff, I.S. (1989). Use of Level 3 BLAS in LU factorization on the CRAY-2, the ETA-10P, and the IBM 3090/VF. *Int. J. of Supercomputer Applics.* **3** (2), 40-70 .

[7] Daydé, M.J., and Duff, I.S. (1990). Use of Level 3 BLAS in LU factorization in a multiprocessing environment on three vector multiprocessors, the ALLIANT FX/80, the CRAY-2, and the IBM 3090/VF. *Int. J. of Supercomputer Applics.* **5** (3), 92-110.

[8] Dongarra, J.J., Du Croz, J., Hammarling, S., and Hanson, R.J. (1988). An extended set of Fortran Basic Linear Algebra Subprograms. *ACM Trans. Math. Softw.* (14), 1-17 and 18-32.

[9] Dongarra, J.J., Du Croz, J., Duff, I.S., and Hammarling, S. (1990). A set of level 3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Softw.* (16), 1-17.

[10] Dongarra, J.J., Du Croz, J., Duff, I.S., and Hammarling, S. (1990). Algorithm 679. A set of level 3 Basic Linear Algebra Subprograms: model implementation and test programs. *ACM Trans. Math. Softw.* (16), 18-28.

[11] Dongarra, J.J., Gustavson, F.G., and Karp, A. (1984). Implementing linear algebra algorithms for dense matrices on a vector pipeline machine. *SIAM Rev.*(26), 91-112.

[12] Gallivan, K., Jalby, W., and Meier, U. (1987). The use of BLAS3 in linear algebra on a parallel processor with a hierarchical memory. Timely communications *SIAM J. Sci. Stat. Comput.* (8), 1079-1084.

[13] Gallivan, K., Jalby, W., Meier, U., and Sameh A. (1988). Impact of hierarchical memory systems on linear algebra algorithm design. *Int. J. of Supercomputer Applics.* 2(1),12-48.

[14] Gallivan, K., Plemmons, R.J., and Sameh, A.H. (1990). Parallel algorithms for dense linear algebra computations. *SIAM Rev.*(32), 54-135.

[15] IBM (1986). Engineering and Scientific Subroutine Library. Program Number: 5668-863, *IBM.*

[16] Kågström, B., and Ling, P. (1988). Level 2 and Level 3 BLAS routines for IBM 3090 VF/400 : implementations and experiences. Report UMINF 154.88, University of Umeå, Sweden.

[17] Kågström, B., Ling, P., and Van Loan, C. (1991). High performance GEMM-Based Level-3 BLAS : sample routines for double precision real data. Report UMINF 91.09, University of Umeå, Sweden.

[18] Lawson, C.L., Hanson, R.J., Kincaid, D.R., and Krogh, F.T. (1979). Basic linear algebra subprograms for Fortran usage. *ACM Trans. Math. Softw.* (5), 308-323.

[19] Lawson, C.L., Hanson, R.J., Kincaid, D.R., and Krogh, F.T. (1979b). Algorithm 539. Basic linear algebra subprograms for Fortran usage. *ACM Trans. Math. Softw.* (5), 324-325.

[20] Ling, P. (1990). A set of high performance Level 3 BLAS structured and tuned for the IBM 3090 VF and implemented in Fortran 77. Report UMINF 179.90, University of Umeå, Sweden.

[21] Mayes, P., and Radicati Di Brozolo, G. (1989). Portable and efficient factorization algorithms on the IBM 3090/VF. Proceedings International Conference on Supercomputing, June 1989, ACM, 263-270.

[22] Mayes, P., and Radicati Di Brozolo, G. (1989). Banded Cholesky factorization using Level 3 BLAS. Report TM-134, Mathematics and Computer Science Division, Argonne National Laboratory.

[23] Petitet, A. (1990). Implémentation bloquée du BLAS 3 sur quatre calculateurs vectoriels et parallèles : l'ALLIANT FX/80, le CONVEX C220, le CRAY-2, et l'IBM 3090J/6VF. DEA Informatique, ENSEEIHT.

[24] Robert, Y. and Sguazzero, P. (1987). The LU decomposition algorithm and its efficient Fortran implementation on the IBM 3090 vector multiprocessor. Report ICE-0006, *ECSEC*.

[25] Sheikh, Q., and Liu, J. (1989). Basic Linear Algebra Subprogram Optimization on the CRAY-2 System. CRAY Channels, Spring 1989.

A Appendix

We present here the double precision code for the blocked and the loop-unrolled versions of SYMM which performs the operation :

$$\mathbf{C}=\alpha.\mathbf{A}.\mathbf{B}+\beta.\mathbf{C}$$

where α and β are scalars, $\mathbf{A}$ is a symmetric matrix and $\mathbf{B}$ and $\mathbf{C}$ are m $\times$ n matrices, in the case where only the upper triangular part of the symmetric matrix $\mathbf{A}$ is to be referenced ("Left" and "Upper").

A.1 Blocked code for DSYMMP

```
      DO 70 I = 1, M, NB

         NCOLA=MIN(M-I+1,NB)
         DO 45 K=1,N,NB

            NCOLB=MIN(N-K+1,NB)
            CALL DSYMM(SIDE,UPLO,NCOLA,NCOLB,ALPHA,A(I,I),
     $         LDA,B(I,K),LDB,BETA,C(I,K),LDC)

45       CONTINUE
         DO 60 J=1,I-NB,NB
```

```
              NLIGA=MIN(M-J+1,NB)
              DO 55 K=1,N,NB

                 NCOLB=MIN(N-K+1,NB)
                 CALL DGEMM('T','N',NCOLA,NCOLB,NLIGA,ALPHA,
     $           A(J,I),LDA,B(J,K),LDB,ONE,C(I,K),LDC)
                 CALL DGEMM('N','N',NLIGA,NCOLB,NCOLA,ALPHA,
     $           A(J,I),LDA,B(I,K),LDB,ONE,C(J,K),LDC)

55            CONTINUE

60         CONTINUE

70      CONTINUE
```

This blocked code calls either a tuned Fortran code or a manufacturer supplied kernel for DSYMM. DGEMM is in all cases manufacturer-supplied.

A.2 Unrolled version of SYMM on the ALLIANT FX/80

We present here a fraction of the Fortran code for the unrolled version of the subroutine DSYMM on the ALLIANT FX/80. The code is unrolled to a depth of MODULO, which is equal here to 4.

```
      MODULO=4
      M_MODULO=MOD(M,MODULO)

cvd$ nodepchk
cvd$ noconcur

      DO 70, I = 1,M-M_MODULO,MODULO

cvd$ nodepchk
cvd$ assoc
cvd$ concur

         DO 60, J = 1,N
```

```
                TEMP10(J) = ALPHA*B(I ,J)
                TEMP11(J) = ALPHA*B(I+1,J)
                TEMP12(J) = ALPHA*B(I+2,J)
                TEMP13(J) = ALPHA*B(I+3,J)
                TEMP20(J) = ZERO
                TEMP21(J) = ZERO
                TEMP22(J) = ZERO
                TEMP23(J) = ZERO


cvd$ nodepchk
cvd$ assoc

                DO 50, K = 1, I-1


                   C(K,J) = C(K,J)+TEMP10(J)*A(K,I)
      $            +TEMP11(J)*A(K,I+1)+TEMP12(J)*A(K,I+2)
      $            +TEMP13(J)*A(K,I+3)
                   TEMP20(J)=TEMP20(J)+B(K,J)*A(K,I )
                   TEMP21(J)=TEMP21(J)+B(K,J)*A(K,I+1)
                   TEMP22(J)=TEMP22(J)+B(K,J)*A(K,I+2)
                   TEMP23(J)=TEMP23(J)+B(K,J)*A(K,I+3)


50              CONTINUE


60           CONTINUE


cvd$ nodepchk
cvd$ assoc
cvd$ concur


             DO 61, J = 1,N


                IF( BETA.EQ.ZERO )THEN


                   C(I ,J) = TEMP10(J)*A(I ,I )+ALPHA*TEMP20(J)
                   C(I ,J) = C(I ,J)+TEMP11(J)*A(I ,I+1)
                   TEMP21(J) = TEMP21(J )+ B(I ,J)*A(I ,I+1)
                   C(I+1,J) = TEMP11(J)*A(I+1,I+1)+ALPHA*TEMP21(J)
                   C(I ,J) = C(I ,J)+TEMP12(J)*A(I ,I+2)
                   TEMP22(J) = TEMP22(J )+ B(I ,J)*A(I ,I+2)
                   C(I+1,J) = C(I+1,J)+TEMP12(J)*A(I+1,I+2)
                   TEMP22(J) = TEMP22(J )+ B(I+1,J)*A(I+1,I+2)
```

```
                C(I+2,J) = TEMP12(J)*A(I+2,I+2)+ALPHA*TEMP22(J)
                C(I ,J) = C(I ,J)+TEMP13(J)*A(I ,I+3)
                TEMP23(J) = TEMP23(J )+ B(I ,J )*A(I ,I+3)
                C(I+1,J) = C(I+1,J)+TEMP13(J)*A(I+1,I+3)
                TEMP23(J) = TEMP23(J )+ B(I+1,J )*A(I+1,I+3)
                C(I+2,J) = C(I+2,J)+TEMP13(J)*A(I+2,I+3)
                TEMP23(J) = TEMP23(J )+ B(I+2,J )*A(I+2,I+3)
                C(I+3,J) = TEMP13(J)*A(I+3,I+3)+ALPHA*TEMP23(J)

             ELSE

                C(I ,J) = BETA*C(I ,J) +
     $          TEMP10(J )*A(I ,I )+ALPHA*TEMP20(J)
                C(I ,J) = C(I ,J)+TEMP11(J)*A(I ,I+1)
                TEMP21(J) = TEMP21(J)+ B( I ,J)*A(I ,I+1)
                C(I+1, J) = BETA*C(I+1,J) +
     $          TEMP11(J)*A(I+1,I+1)+ALPHA*TEMP21(J)
                C(I ,J) = C(I ,J)+TEMP12(J)*A(I ,I+2)
                TEMP22(J) = TEMP22(J)+ B( I ,J)*A(I ,I+2)
                C(I+1,J) = C(I+1,J)+TEMP12(J)*A(I+1,I+2)
                TEMP22(J) = TEMP22(J)+ B( I+1,J)*A(I+1,I+2)
                C(I+2,J) = BETA*C(I+2,J) +
     $          TEMP12(J)*A(I+2,I+2)+ALPHA*TEMP22(J)
                C(I ,J) = C(I ,J)+TEMP13(J)*A(I ,I+3)
                TEMP23(J) = TEMP23(J)+ B(I , J)*A(I ,I+3)
                C(I+1,J) = C(I+1,J)+TEMP13(J)*A(I+1,I+3)
                TEMP23(J) = TEMP23(J)+ B(I+1, J)*A(I+1,I+3)
                C(I+2,J) = C(I+2,J)+TEMP13(J)*A(I+2,I+3)
                TEMP23(J) = TEMP23(J)+ B(I+2, J)*A(I+2,I+3)
                C(I+3,J) = BETA*C(I+3,J) +
     $          TEMP13(J)*A(I+3,I+3)+ALPHA*TEMP23(J)

             ENDIF

61         CONTINUE

70      CONTINUE

        IF (M_MODULO.NE.0) THEN

cvd$ nodepchk
cvd$ noconcur

             DO 79, I = M-M_MODULO+1,M
```

```
cvd$ nodepchk
cvd$ assoc
cvd$ concur

                DO 77, J = 1, N

                   TEMP10(J) = ALPHA*B( I,J)
                   TEMP20(J) = ZERO

cvd$ nodepchk
cvd$ assoc

                   DO 75, K = 1,I - 1

                      C(K,J) = C(K,J) + TEMP10(J)*A( K, I )
                      TEMP20(J) = TEMP20(J) + B( K,J)*A( K, I )

75                 CONTINUE

77              CONTINUE

cvd$ nodepchk
cvd$ assoc
cvd$ concur

                DO 78, J = 1, N

                   IF( BETA.EQ.ZERO )THEN

                      C(I,J) = TEMP10(J)*A(I,I)+ALPHA*TEMP20(J)

                   ELSE

                      C(I,J) = BETA*C(I,J) +
     $                 TEMP10(J)*A(I,I)+ALPHA*TEMP20(J)

                   ENDIF

78              CONTINUE

79           CONTINUE

          END IF
```

A.3 Microtasked version of SSYMMP on the CRAY-2

We present here the microtasked version of SSYMMP on the CRAY-2.

```
CMIC$ DO ALL SHARED(SIDE,UPLO,A,B,C,M,N,NB,ALPHA,BETA,LDA,LDB,LDC)
CMIC$1PRIVATE( I, K, J, NBCOLA, NBCOLB, NBLIGA )


      DO 45 K=1,N,NB


         NBCOLB=MIN(N-K+1,NB)
         DO 70 I = 1, M ,NB


                  NBCOLA = MIN(M-I+1,NB)
                  CALL SSYMM (SIDE,UPLO,
     $               NBCOLA,NBCOLB,ALPHA, A(I,I),
     $               LDA,B(I,K), LDB ,BETA,C(I,K),LDC)

                  DO 65 J=1,I-NB,NB



                     NBLIGA=MIN(M-J+1,NB)
                     CALL SGEMM('T','N',NBCOLA,NBCOLB,
     $                  NBLIGA, ALPHA, A(J,I),LDA,
     $                  B(J,K),LDB,ONE,C(I,K),LDC)
                     CALL SGEMM('N','N',NBLIGA,NBCOLB,
     $                  NBCOLA, ALPHA, A(J,I),LDA,
     $                  B(I,K),LDB,ONE,C(J,K),LDC)


65                CONTINUE


70       CONTINUE


45    CONTINUE
```