Scalable BLAS 2 and 3 Matrix Multiplication for Sparse Banded Matrices on Distributed Memory MIMD Machines

Nikos Chrisochoides [*], Mokhtar Aboelaze [†], Elias Houstis [‡]and Catehrine Houstis [§]

Abstract

In this paper, we present two algorithms for sparse banded matrix-vector and sparse banded matrix-matrix product operations on distributed memory multiprocessor systems that support a mesh and ring interconnection topology. We aslo study the scalability of these two algorithms. We employ systolic type techniques to eliminate synchronization delay and minimize the communication overhead among processors. The performance of algorithms developed for the above operations depends on the bandwidth of the matrices involved and have been currently implemented on the NCUBE II with 64 processors. Our preliminary experimental data agree with the expected theoretical behavior.

1 Introduction

In designing algorithms for multiprocessor systems, one must take into consideration the following three important issues [Bokh 90] : *i*) minimizing the edge contention; this happens when one or more links are shared between more than one paths in the computation graph, *ii*) minimizing the amount of data transfer between the different processors, *iii*) minimizing the synchronization delay; synchronization delay is the delay suffered by a process that is waiting for the completion of another process at another processor. It has been observed that the minimization of the cost functions corresponding to the above three design objectives depend on the way the underlying computation graph is decomposed which constitutes an NP-complete problem for general computational graphs [Gare 79]. Another important factor in designing algorithms for multiprocessors is known as *scalability.* Scalability is a measure of how well the algorithm

[*]Northeast Parallel Architectures Center, Syracuse University.
[†]York University, Computer Science Department, North York, Ontario, Canada M3J 1P3.
[‡]Department of Computer Science, Purdue University, West Lafayette, IN 47907.
[§]University of Crete, Computer Science Department, Heraclion, Greece.

The work of M. Aboelaze was supported in part by NSERC-09P0043688. The work of N. Chrisochoides was supported by Computer Science at Purdue University and Alex G. Nason award at Syracuse University. The work of the rest of the authors was supported in part by AFSOR 88-0234, NSF grant CCF-8619817 and ESPRIT project 2702-GENESIS-P.

behaves when the number of processors in the system increases. Some algorithm/architecture pairs are efficient when the number of processors are small. Increasing the number of processors dramatically increases the overhead and the performance degrades.

For the case of well structured computations, special purpose algorithm architecture pairs were suggested and known as systolic arrays [Kung 82], [Mold 82], and [Mira 84]. These architectures consist of simple processing elements(PEs) which are capable of performing one arithmetic operation. In systolic computations, the decrease of edge contention and the elimination of the synchronization delay is achieved by mapping the computation graph into a systolic array such that the correct data are in the correct place at the appropriate time.

In this paper we propose systolic type techniques to parallelize primitive linear algebra operations applied to sparse data. The set of these operations are known as sparse BLAS 2 and 3, which includes multiplication of banded matrices and banded matrix-vector operations. Sparse matrix operations are used in the discretization of Partial Differential Equations (PDE) with the well known finite element and difference techniques. We are using the above implementation of BLAS to develop MIMD solvers to solve PDE discretization equations. Unlike the sparse matrix operations, the dense BLAS 2 and 3 have been studied extensively [Fox 87], [Cher 88], [Bern 89]. In a previous paper [Chri 92], we showed the implementation and the performance of parallel BLAS for dense matrices. The experimental results indicate an efficiency of up to 98% for matrix-vector operations and 94% for matrix-matrix operations on a 64 processors configuration NCUBE II with one Mbyte of memory per processor. The remaining of this paper is as follows, in section 2 we discuss the concept of scalability. In section 3, we review some of the proposed ideas for the parallelization of BLAS and their complexity on various architectures. In section 4, we present our proposed algorithms for the implementation of sparse BLAS 2 and 3 on distributed memory machines with mesh and ring interconnection topologies. We also discuss the scalability of our proposed algorithms. Section 5 is a conclusion.

2 Scalability

Scalability is a measure of the degree of matching between the architecture, and the application program (algorithm), or, it is a measure of how fast the algorithm runs if we increase the number of processors. An algorithm is said to be scalable on a particular architecture, if the performance of the computer system grows linearly with the number of processors [Hwan 93]. Scalability analysis is a crucial factor in determining the performance of a specific algorithm on a specific machine. A specific machine could be very efficient for a particular algorithm, but not as efficient for another algorithm. On the other hand, a specific algorithm may run very efficiently on a specific machine, but not as efficient if we increase the number of processors in this machine. That is why scalability analysis is crucial in choosing the algorithm/architecture pair [Kuma 87], [Kuma 91].

The performance of a specific algorithm on a particular machine is determined by many factors, such as machine size (number of processors), machine speed, problem size (workload), communication overhead, I/O overhead, and many other factors. In this paper, we investigate the scalability of the algorithms on an NCUBE II. In this paper we choose the scalability measure proposed in [Gust 88], where the speedup is defined as $S(N, P) = \frac{T_1}{T_P}$, where T_P is the execution time of the computation in a P processors machine, and scaled speedup is defined as

$$Scl\_SpUp1 = \frac{Mflops \quad using \quad P \quad processors}{Mflops \quad using \quad single \quad processor} \tag{2.1}$$

or

$$Scl\_SpUp2 = P \times \frac{T_{Work\_done\_by\_P\_proces} - T_{Work\_wouldn't\_done\_by\_serial\_proces}}{T_{Work\_done\_by\_P\_proces}} \tag{2.2}$$

where $T_{Work\_done\_by\_P\_proces}$ indicates the total elapse time using P processors, and $T_{Work\_wouldn't\_done\_by\_serial\_proces}$ indicates the overhead due to communication and synchronization.

3 Overview of Parallel Matrix Multiplication Algorithms

The development and implementation of scalable and portable scientific algorithms across a number of parallel machines is an important and challenging problem. One of the approaches that have been extensively explored is the so called BLAS approach. The basic idea is to identify a kernel of high level primitive mathematical operations, implement them on a variety of machines, and use them to develop more complex applications on these target machines. Two well known linear algebra kernels have identified which are referred through out as BLAS 2 and 3. In this section we review some of the ideas proposed for their parallelization in the case of dense data structures and various architectures.

Fox et al in [Fox 87] proposed techniques for the multiplication of matrices decomposed into square or rectangular subblocks on hypercube architectures. These blocks are distributed between the processors. The product matrix is distributed among the processors in the same fashion. The algorithms exploit the mesh architecture embedded in any hypercube architecture. They also use broadcasting for communicating some of these data blocks.

Deckel, Nassimi, and Sahni in [Deke 81] proposed a matrix multiplication algorithm for cube connected and perfect shuffle computers. They use N^2m processors to multiply two $N \times N$ matrices in $O(\frac{N}{m} + logm)$ time. They also show how $m^2, 1 \leq m \leq N$, processors can be used to multiply two $N \times N$ matrices in $O(\frac{N^2}{m} + m(\frac{N}{m})^{2.61})$ time. This method is efficient for multiplying dense matrices, but, it appears to be inefficient for sparse BLAS 2 and 3 operations.

Johnson [John 85] presented algorithms for dense matrix multiplication and for Gauss-Jordan and Gaussian elimination. His algorithm can run on any boolean cube or torus computers. It achieves a 100% processor utilization except for a latency period $T_{latency} = O(n)$ on an ncube system. In [John 89], Johnsson et al presented a data parallel matrix multiplication algorithm which was implemented on the Connection Machine CM-2. They report 5.8 GFLOPS overall performance.

Independently, Cherakasky et al in [Cher 88], Berntsen in [Bern 89], and Aboelaze [Aboe 89] improved Fox's algorithm for dense matrix multiplication, reducing the time complexity from

$$T = \frac{2N^3}{P}\tau + \frac{2N^2}{\sqrt{P}}t_{transf} + \sqrt{P}(\sqrt{P}-1)t_{start}$$

to

$$T = \frac{2N^3}{P}\tau + \frac{2N^2}{\sqrt{P}}t_{transf} + (\sqrt{P}-1)t_{start}$$

where P is the number of processors, τ is the time for one addition and multiplication, and t_{transf}, t_{start} are machine dependent communication parameters. Berntsen's second idea was to partition the hypercube into a set of subcubes and using the cascaded sum algorithm to add up the contributions to the product matrix. His idea also reduced the asymptotic communication to $\frac{N^2}{P^{\frac{2}{3}}}$ at the expense of having $\frac{N^2}{P^{\frac{2}{3}}}$ extra bytes of memory per processor.

The algorithms for dense matrices presented in [Fox 87], [Cher 88], [Bern 89], and [Aboe 89] require P and $\sqrt{P}$ iteration steps to compute the c = c + Ab and C = C + AB respectively; each iteration step requires $\frac{N}{P}t_{transf} + t_{start}$ and $\frac{N^2}{P}t_{transf} + t_{start}$ communication time respectively. In this paper, we present two algorithms for operation on band matrix $A \in \mathbf{R}^{N\times N}$, with bandwidth w. The first algorithm is to multiply A by b, where $b \in \mathbf{R}^N$. The second algorithm is to multiply A by B, where $B \in \mathbf{R}^{N\times N}$, with bandwidth δ. The first algorithm requires w iteration steps with each iteration requiring $\frac{N}{P}t_{transf}+t_{start}$ communication time. The second algorithm requires $\min(w, \delta)$ iteration steps with each iteration step requiring $\frac{N\min(w,\delta)}{P}t_{transf}+t_{start}$ communication time.

4 Parallelization of some level 2 and 3 BLAS for banded matrices

4.1 Band Matrix Vector Multiplication

First we consider the parallelization of the operation c = βc + αA b on a linear array of P processors when A is a banded matrix with w_1, w_2 upper and lower bandwidths. Throughout the paper we assume that matrices are stored using a sparse scheme [Rice 85]. Figure 1 shows the decomposition of the matrix into the processors's memory. In Figure 1 a row in the matrix represent strip of rows in the original matrix. The matrix is divided intp P strips($P = 4$ in Figure 1), each strip is stored in a different processor in a linear array or processors. The b vector is also divided into P strips and stored in the different processors.

The proposed implementation is based on a decomposition of matrix A into an upper U (including the diagonal of A) and lower L triangular matrices such as A = L + U. Furthermore, we assume that row strip $\{a_{i,j}\}_{j=1}^{P}$ and b_i are stored in processor i. Without any loss of generality we can assume and $\alpha = 1$ $beta = 1$. Then the vector c can be expressed as c = c + (Lb + Ub). The products Ub and Lb are computed within $w_1 + 1$ and w_2 iterations respectively. The algorithm operates first by calculating Lb, which is accomplished by performing accumulation and shifting the b vector to the left. Then the b vector is restored, and Ub is calculated by accumulation and shifting b to the right. The computation involved is described in Figure 2. In order to compute the complexity of the above algorithm we assume, that A has K non-zero elements, and $N >> w_1 + w_2 + 1$. Then it can be shown that the time complexity is

$$T_P = \frac{K}{P}\tau + (w_1 + w_2 + 1)\{\gamma + \delta\frac{N}{P}\} \qquad (4.1)$$

and the memory space required for each subdomain is $O(\frac{K}{P} + 3\frac{N}{P})$.

$$\begin{bmatrix} a_{11} & a_{12} & & \\ a_{21} & a_{22} & a_{23} & \\ & a_{32} & a_{33} & a_{34} \\ & & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

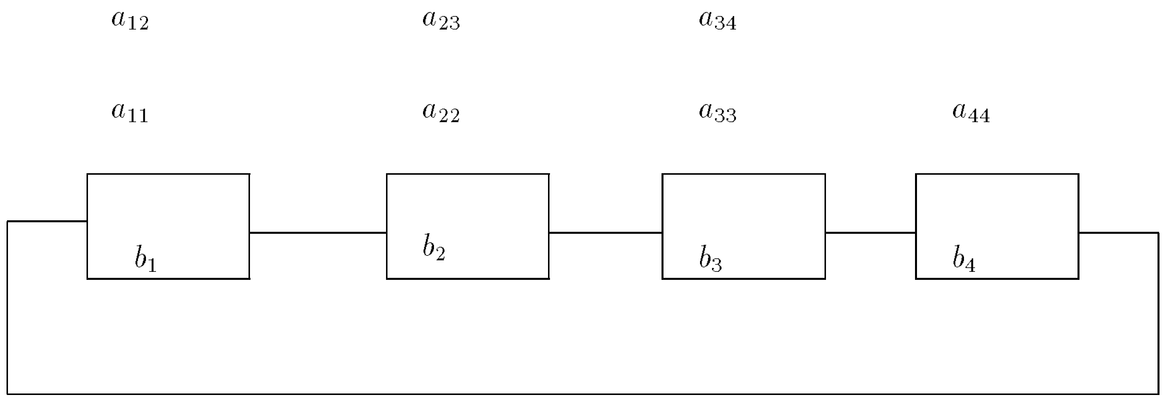

Figure 1: Band matrix vector multiplication on a 4 processor system

4.2 Banded Matrix - Matrix Multiplication

Second, we consider the implementation of $C = \beta C + \alpha AB$, on a ring of P processors when A, B are banded matrices with u_1, u_2 upper, and l_1, l_2 lower bandwidths respectively. The processor i computes a strip of columns C_i of matrix C and holds one strip of rows of matrix A (denoted by A_i) and a one strip of columns of matrix B (denoted by B_i).

The algorithm consists of two phases as in band-matrix vector multiplication. Without any loss of generality we can assume and $\alpha = 1$ $beta = 1$. In the first phase, each PE starts by calculating $c_{ii} = c_{ii} + A_i \times B_i$, then each PE i passes B_i to PE $i-1$, this phase is repeated $\min u_1, u_2 + 1$ times. In the second phase each PE restores B_i and passes it to PE $i+1$. This phase is repeated l_i times, where $u_i = \min u_1, u_2$. The initial distribution of the matrices to the processors is shown in Figure 3 for the case of $P = 4$. While The implementation proposed for this operation is described in Figure 4. Without any loss of generality, we assume that K_1, K_2 are the number of non-zero elements for the matrices A, B respectively and denote by $w_1 = u_1 + l_1 + 1$ and $w_2 = u_2 + l_2 + 1$. Then we can show that

$$T_P = \frac{\min(K_1 w_2, K_2 w_1)}{P}\tau + \{\gamma + \delta\frac{N}{P}\min(w_1, w_2)\}min(w_1, w_2) \qquad (4.2)$$

The above realization has been implemented on the NCUBE 6400.

Tables 1 and 2 shows the maximum elapsed time and the scaled speedup for band matrix vector multiplication, while Tables 3 and 4 shows the same results for band matrix matrix multiplication. We notice that in Tables 1-4 the elapsed time is almost constant with increasing the number of processors if the workload per processor is constant, which include a linear scalability (linear *isoecciciency function* [Kuma 91]. While the scaled speedup increases with

Phase 1: Multiply the Upper triangular U by b

```
temp := d
For each PE i do in parallel
  For j := 0 to w2
    if (i + j  =< P) then
       begin
        if ( i = 1 ) then do nothing
         else Send d to PE i-1
         c := c + a(i, j+i) * d
         if ( i = P ) then do nothing
         else Receive d from PE i+1
       end
    endif
  end
end
```

Phase 2: Multiply the Lower triangular L by b

```
For each PE i do in parallel
  begin
  d := temp
    For j := 1 to w2
      if (i < j) then
         begin
           if ( i = P ) then do nothing
           else Send d to PE i + 1
           if ( i = 1 ) then do nothing
           else Receive d from PE i - 1
           c := c + a(i, i-j) * d
         end
      endif
    end
end
```

Figure 2: The pseudo code for banded matrix-vector multiplication.

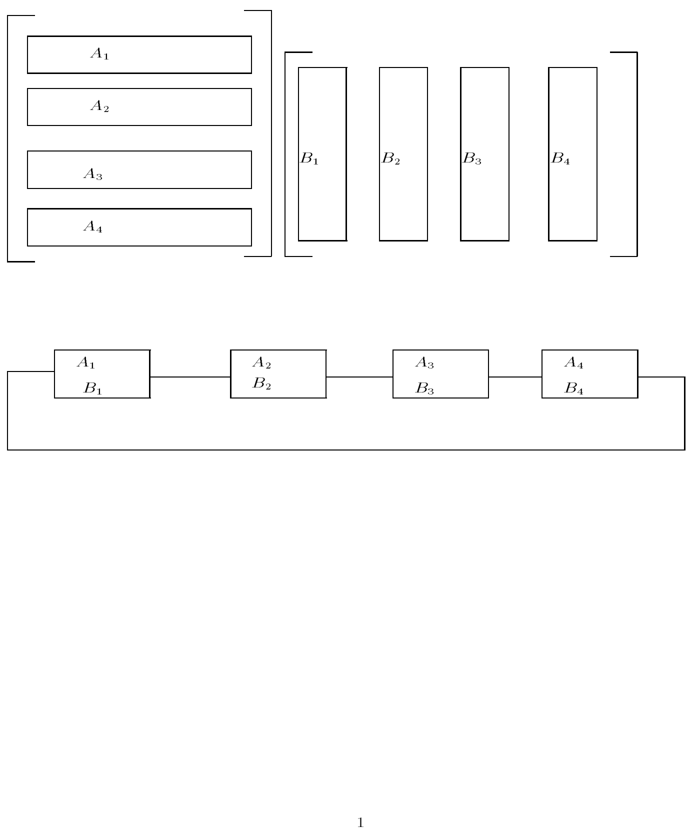

Figure 3: Band matrix vector multiplication on a 4 processor system

increasing the number of processors.
We realize that this is a limited experience. We are currently implementing direct and iterative solvers for discrete elliptic equations within the //ELLPACK environment [Hous 90] to access the performance of BLAS operations.

5 Conculsion

In this paper, we presented two scalable algorithms for band matrix vector, and band matrix matrix multiplication. The efficiency of these two algorithms is 40%, this is due to the need of minimizing the memory requirements at the different processor which means that the matrices are to be stored using a sparse matrix storage scheme. The two algorithms are scalable (performance grows linearly with increasing the number of processors), which means they are suitable for multiprocessors with a lrage number of processors.

Phase 1

```
temp := b
For each PE i do in parallel  /* each PE contain a = Ai , b = Bi */
  For j := 0 to min(u1 , u2)
    if (i + j =< N) then
       begin
         if (i = 1) do nothing
         else Send b to PE i-1
         c(i,i+j) := c(i,i+j)  +  a * b
         if (i = P) then do nothing
         else Receive b from PE i+1
    endif
  endfor
endfor
```

Phase 2

```
b := temp
For Each PE i in parallel do
  For j := 1 to min(l1 , l2) do
    if (i > j) then
       begin
         if(i = P) then do nothing
         else send b to PE i+1
         if( i = 1) then do nothing
         else receive b from PE i-1
         c(i, i-j) := c(i,i-j) +  a * b
    endif
  endfor
endfor
```

Figure 4: The pseudo code for banded matrix-matrix multiplication

Table 1: Measured maximum total elapsed time (in seconds) for block tridiagonal matrices.
for band matrix vector multiplication

matrix size / node	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
8	0.0634	0.0644	0.0644	0.0644	0.0645
16	0.0221	0.0222	0.0222	0.0222	0.0222
32	0.0847	0.0848	0.0848	0.0849	0.0849
64	0.3345	0.3346	0.3346	0.3347	0.3347

Table 2: Measured scaled speed up (2.2) for block tridiagonal matrices.
for band matrix vector multiplication.

matrix size / node	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
8	3.55	7.03	14.05	28.22	56.25
16	3.87	7.72	15.43	30.94	61.80
32	3.96	7.91	15.83	31.66	63.30
64	3.99	7.97	15.95	31.90	63.80

Table 3: Measured maximum elapsed time (in seconds) for block tridiagonal matrices.
for band matrix band matrix multiplication

matrix size / node	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
8	0.275	0.281	0.281	0.281	0.281
16	4.029	4.030	4.030	4.030	4.030

Table 4: Measured scaled speed up (2.2) for block tridiagonal matrices.
for band matrix band matrix multiplication

matrix size / node	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
8	2.78	7.31	14.62	29.24	58.50
16	2.76	7.28	14.56	29.12	58.25

References

[Aboe 89] Mokhtar Aboelaze, Unpublished manuscript, June, 1989.

[Bern 89] J. Berntsen Communication Efficient Matrix multiplication on hypercubes, *Parallel Computing*, Vol 12, No 3, Dec. 1989 pp335–342

[Bokh 90] S.H. Bokhari Communication Overhead on the Intel iPSC-860 Hypercube. *ICASE Interim report 10*, NASA Langeley Research Center, Hampton, Virginia 23665

[Cher 88] V. Cherkassky and R. Smith, Efficient Mapping and Implementation of Matrix algorithms on a hypercube, *The Journal of Supercomputing*, Vol 2, pp 7–27, 1988

[Chri 90] N.P. Chrisochoides, Communication overhead on the NCUBE-6400 hypercube.

[chri 92] N. P. Chrisochoides, M. A. Aboelaze, E. N. Houstis, and C. E. Houstis The Parallelization of Some Level 2 and 3 BLAS Operations on Distributed memory Machines, *7th IMACS International Conference on Computer Mentods for Partial Differential Equations*, New Bruinsick, N.J. June 1992 pp 119-126.

[Deke 81] E. Dekel, D. Nassimi, and S. Sahni, Parallel Matrix and graph algorithms, *SIAM Computing*, Nov. 1981, pp 657–675

[Fox 87] G.C.Fox, S. W. Otto, and A.J. Hey, Matrix Algorithms on a hypercube I : Matrix Multiplication, *Parallel Computing*, 1987, pp 17–31.

[Gare 79] M. R. Garey and D.S. Johnson *Computers and Intractability, A Guide to the Theory of NP-Completeness.*

[Hous 90] E.N. Houstis, J.R. Rice, N.P. Chrisochoides, H.C. Karathanases, P.N. Papachiou, M.K. Samartzis, E.A. Vavalis, Ko Yang Wang and S. Weerawarana, //ELLPACK: A numerical simulation programming environment for parallel MIMD machines, In *Supercomputing* 90, ACM Press, 3–23, 1990.

[Hwan 93] K. Hwang *Advanced Computer Architecture with parallel Programming* McGraw Hill 1993.

[John 85] S. L. Johnsson, Communication efficient Basic Linear Algebra Computations on hypercube architecture, *Technical Report YALE/CSD/RR-361*, Dept. of Computer Science, Yale University, 1985.

[John 89] S. L. Johnsson, T. Harris, and Kapil K. Mathur, Matrix Multiplication on the Connection Machine, *Proc. Supercomputing 89*, Nov 13-17 1989, Reno Nevada, ACM Press, page 326–332

[Kuma 87] V. Kumar, and N. Rao, Parallel Depth-First Search, Part II: Analysis, *International Journal of Parallel Programming*, v 16(6) 1987, pp 501-519

[Kuma 91] V. Kumar, and A. Gupta Analyzing Scalability of Parallel Algorithms and Architecture *Computer Science Department Tech. Report 91-81*, University of Minnesota May 1991.

[Kung 82] H. T. Kung, Why Systolic Architecture, *Computer*, Vol 15, No 1, Jan. 1982, pp 37–46.

[Mira 84] W. L. Miranker and A. Winkler, Space-Time representations of computational structures, *Computing*, Vol. 32, 1984, pp 93–114.

[Mold 82] D. I. Moldovan, On the analysis ans synthesis of VLSI algorithms, *IEEE Trans. on Computers*, Vol. C-31, No 11, Nov. 1982, pp 1121–1126

[Rice 85] J. Rice, and Boivster, *Solving Elliptic Problems using Ellpack*, Springer-Verlag, 1985.