The Parallelization of Level 2 and 3 BLAS Operations on Distributed Memory Machines

M. Aboelaze

N. P. Chrisochoides

Elias N. Houstis
Purdue University, enh@cs.purdue.edu

C. E. Houstis

Report Number:
91-007
THE PARALLELIZATION OF LEVEL 2 AND 3 BLAS OPERATIONS ON DISTRIBUTED MEMORY MACHINES

M. Aboelaze
N. P. Chrisochoides
E. N. Houstis
C. E. Houstis

CSD-TR-91-007
January 1991
The parallelization of level 2 and 3 \textit{BLAS} operations on distributed memory machines

M. Aboelaze ;
York University
Computer Science Department
North York, Ontario, Canada M3J 1P3

N. P. Chrisochoides † E. N. Houstis ‡ C. E. Houstis §
Purdue University
Computer Science Department
West Lafayette, IN 47907

e-mail: aboelaze@yetti.cs.yorku.ca,
cpn, enh, ceh@cs.purdue.edu,

January 7, 1991

Abstract

We present the parallelization of band matrix-vector and band matrix-matrix product operations on distributed memory multiprocessor systems that support a mesh and ring interconnection topology. Our approach eliminates synchronization delay and minimizes the communication overhead among processors. Three of these operations has
been currently implemented on the NCUBE-6400 with a 64 processor configuration. High efficiency and scalability of the algorithms has been justified. Efficiency of 98% for matrix-vector multiplication and 94% for matrix-matrix multiplication has been observed for the case of dense matrices. For the case of the band matrix-vector multiplication the communication overhead is a linear function of the number of processors.

1 Introduction

The most important objectives in designing algorithms/software for multiprocessor systems include the minimization of i) the so called edge contention (more than one path share one or more links, [Bokh 90], [Chri 90]), ii) the amount of data transferred between processors, and iii) the synchronization delay. It has been observed that the minimization of the cost functions corresponding to the above three design objectives depend on the way the associated computation graph is decomposed. For general graphs, this problem is NP-complete [Gare 79].

For the case of well structured computations, special purpose algorithm/architecture pairs were suggested known as systolic arrays [Kung 82], [Mold 82], [Mira 84], [Chen 87]. 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 synchronization 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. This scheduling strategy usually results in minimizing the time any PE spends waiting to receive the required data.

In this paper we propose to develop systolic type techniques to design faster algorithms/software for MIMD course grain computation/architecture pairs. To test these approaches we have selected to parallelize some primitive linear algebra operations, mainly band matrix operations. The dense matrix operations have been studied extensively, [Fox 87], [Cher 88], [Bern 89], and others. In section 2, we review some of the known matrix multiplication algorithms and their complexity for various architectures. Section 3 describes briefly the essential characteristics of the NCUBE-6400. For completeness in section 4 we present the matrix multiplication algorithms and their performance, for dense matrices. In section 5 the proposed matrix multiplication algorithms and their performance, for banded matrices are presented. The results indicate an efficiency up to 98% for matrix-vector
operations and 94 % for matrix-matrix operations on a 64 processors configuration Ncube-6400 with one Mbyte of memory per processor. Moreover indicate the scalability of the band matrix-vector multiplication. We expect to have performance data from iPSC2/860 by the time of publication.

2 Overview of Parallel Matrix Multiplication Algorithms

Matrix and vector operations are at the core of many important scientific computations. Many problems in physics, mathematics, engineering and chemistry can be formulated as matrix-vector operations. A lot of effort is dedicated to finding an efficient method for multiplying matrices and vectors. In this section we review some attempts in this field. Our list is by no means complete. We only summarize work closely related to our work.

Fox et al in [Fox 87] proposed techniques for matrix multiplication. Their method depends on partitioning the matrix into square or rectangular subblocks. These blocks are distributed between the processors. By the end of the matrix multiplication operation, the product matrix is distributed among the processors in the same fashion. The algorithms exploits the mesh architecture embedded in any hypercube architecture. They also use broadcasting for communicating some of these data blocks. In this paper, we tried to avoid any broadcasting, and make all communication local between neighboring processors.

Deckel, Nassimi, and Sahni in [Deke 81] proposed a matrix multiplication algorithm for cube connected and perfect shuffle computers. They used \( N^2 \) processors to multiply two \( N \times N \) matrices in \( O(\frac{N}{m} \log m) \) time. They also showed 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 will not be very efficient for a vector or a band matrix.

Johnson in [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 \( T_{latency} = O(n) \) of an n cube system. In [John 89], Johnson et al presented a data parallel matrix multiplication algorithm. Their algorithm was implemented on the Connection Machine CM-2, their implementation has a peak overall performance of 5.8 GFLOPS.

Independently Cherakasky et al in [Cher 88], Berntsen in [Bern 89] and Aboelaze [Aboe 89] improved Fox's algorithm, for dense matrix multiplica-
tion, reducing the time complexity from
\[ T = \frac{2N}{P} \tau + \frac{2N^2}{\sqrt{P}} t_{\text{transf}} + \sqrt{P}(\sqrt{P} - 1)t_{\text{start}} \]
to
\[ T = \frac{2N}{P} \tau + \frac{2N^2}{\sqrt{P}} t_{\text{transf}} + (\sqrt{P} - 1)t_{\text{start}} \]
where \( P \) is the number of processors, \( \tau \) is the time for one addition and multiplication, and \( t_{\text{transf}}, t_{\text{start}} \) 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 final matrix. His idea also reduced the asymptotic communication to \( \frac{N^2}{P} \) on the expense of having \( \frac{N^2}{P} \) extra bytes of memory per processor.

Most of the previous work on this subject is not efficient for band-matrix operations. The algorithms for dense matrices presented in [Fox 88], [Cher 88], [Bern 89], and [Aboe 89] require \( P \) and \( \sqrt{P} \) iteration steps to compute the \( c = Ab \) and \( C = C + AB \) respectively; each iteration step requires \( \frac{N}{P} t_{\text{transf}} + t_{\text{start}} \) and \( \frac{N^2}{P} t_{\text{transf}} + t_{\text{start}} \) communication time respectively. In this paper, we present two algorithms for operation on band matrix \( A \in \mathbb{R}^{N \times N} \), with bandwidth \( w \). The first algorithm is to multiply \( A \) by \( b \), where \( b \in \mathbb{R}^N \). The second algorithm is to multiply \( A \) by \( B \), where \( B \in \mathbb{R}^{N \times N} \), with bandwidth \( \delta \). The first algorithm requires \( w \) iteration steps with each iteration steps requiring \( \frac{N}{P} t_{\text{transf}} + t_{\text{start}} \) communication time. The second algorithm requires \( w + \delta - 1 \) iteration steps with each iteration step requiring \( \frac{N}{P} t_{\text{transf}} \min(w, \delta) \) communication time. The two algorithms result in communication between neighboring processors, and minimize synchronization delay.

3 The NCUBE-6400

In this section we review characteristics of the NCUBE-6400, [NCUBE 90], that are relevant to our analysis. We also give performance measurements taken on the NCUBE. NCUBE-6400 is hypercube interconnection multiprocessor system, a unique binary ID is assigned to each processor of the network. Two vertices in the network are called neighbors if their binary representation differ in exactly one bit. Each processor has its own memory and works independently from the others. Processors exchange data by
sending messages to each other. The exchange of messages is based on a
circuit switching logic. When two nodes have to communicate, a fixed path
is set up between them. The message flow through this path is without inter­
rupting the intervening processors. The path between the two nodes (source
and destination) is established by a fixed routing strategy. The source node
sends an address packet of 32 bits to a channel. The address is passed from
node to node. Each processor compares his own binary ID with the address
packed, if the processor is the destination the path has been established
otherwise forwards the packet to his neighbor processor whose binary ID
most closely matches the binary ID of the destination. The time (in $\mu$sec)
to communicate a zero byte, in a network without edge contention is :

$$T_{\text{initial}} = 2.84 \times d + 132.87$$  \hspace{1cm} (3.1)

where $d$ is the length of the communication path, [Chri 90]. The claimed
[NCUBE 90] time to transfer $k$ real numbers through an established path
without edge contention is :

$$T_{\text{transf}} = k \times T_{\text{packed}}$$  \hspace{1cm} (3.2)

where $T_{\text{packed}}$ = port speed of a link in the path = 36 machine cycles With
a 40MHz external clock [NCUBE 90], each cycle is about 50 ns. Hence

$$T_{\text{transf}} = 1.8\mu\text{sec}$$  \hspace{1cm} (3.3)

The floating-point performance of the node was measured to be in the range

$$t_{\text{flop}} = 1.28\mu\text{sec} \rightarrow 1.40\mu\text{sec}$$  \hspace{1cm} (3.4)

The time to perform the operation $c(i) = c(i) + a(i,j) \times b(j)$ was measured,
to be equal to

$$t_{\text{1oper}} = 3.852\mu\text{sec}$$  \hspace{1cm} (3.5)

The time to perform the operation $c(i) = c(i) + a(i,j) \times b(id(i,j))$ was
measured, to be equal to

$$t_{\text{2oper}} = 7.915\mu\text{sec}$$  \hspace{1cm} (3.6)

The time to perform the operations $rsum = rsum + a(i,j) \times b(j,k)$, and
$c(i,j) = c(i,j) + rsum$ was measured, to be equal to

$$t_{\text{3oper}} = 3.933\mu\text{sec} \ and \ t_{\text{4oper}} = 7.632E-03\mu\text{sec}$$  \hspace{1cm} (3.7)
4 PARALLELIZATION OF LEVEL 2 AND 3 BLAS FOR DENSE MATRICES

The time to perform the operations $rsum = rsum + a(i, k) \cdot b(ida(i,k), j)$, and $c(i,j) = c(i,j) + rsum$ was measured, to be equal to

$$t_{5\text{oper}} = 7.639\mu\text{sec} \quad \text{and} \quad t_{6\text{oper}} = 7.632E^{-03}\mu\text{sec} \quad (3.8)$$

For distributed memory multiprocessor systems fixing the size problem creates a constraint, since large size data cannot fit on a single processor. In such cases the scaled speedup can be computed [Gust 88] either as:

$$S_{\text{cl.SpUp}}1 = \frac{M \text{ flops using } P \text{ processors}}{M \text{ flops using single processor}} \quad (3.9)$$

or as:

$$S_{\text{cl.SpUp}}2 = P \times \frac{T_{\text{Work done by P processors}} - T_{\text{Work wouldn't done by serial proces}}} {T_{\text{Work done by P proces}}} \quad (3.10)$$

4 Parallelization of level 2 and 3 BLAS for dense matrices

The algorithms we are describing in this paper are suitable for multiprocessor systems with the following properties:

1. $P$ identical processors, each with local memory,
2. the interconnection of the processors supports at least mesh and ring topology with wrap around,
3. the time to communicate $k$ real words between two processors is $t_{\text{start}} + t_{\text{trans}}k$ where $t_{\text{start}} = t_0 + t_1 d$, and $d$ is the distance of the processors in the interconnection topology,
4. the time to perform a floating-point multiply or add is $\tau$

4.1 Dense Matrix $\times$ Vector Multiplication

In this section we examine the matrix-vector multiplication $c = \alpha c + \beta Ab$, where $A \in \mathbb{R}^{N \times N}$, a column vector $b \in \mathbb{R}^N$, and a column vector $c \in \mathbb{R}^N$. $\alpha, \beta$ are scalers. The interconnection of PEs is a wrap around linear array. The interconnection of PEs and the distribution of input are shown in
Figure 1. The interconnection network and distribution of the input

Figure 1. Each PE $i$ computes the corresponding $c_i$. As can be seen the matrix $A$ is divided into submatrices $A_{i,j} \in \mathbb{R}^{\frac{N}{P} \times \frac{N}{P}}$, where $P$ is the number of processors, and the vector $b$ is divided into subvectors $b_i \in \mathbb{R}^{\frac{N}{P}}$. Each processor contains one row of submatrices and one subvector.

**Algorithm**

The vector $c = (c_i)$ can be expressed as:

$$c_i = \sum_{j=1}^{N} A_{ij} b_j$$

Without any loss of generality we assume $\alpha = \beta = 1$.

The algorithm performs $P$ iterations. In each iteration a partial sum of equation (3.1) is accumulated. the algorithm starts by multiplying $A_{i,i}$ by $b_i$. Then every processor sends the part of the vector $b$ it stores to processor $i-1$ and receives the part of $b$ from the corresponding processor, finally multiplies it by $A_{i,(i+1)\text{mod}P}$. The algorithm can be expressed as following:

For all PE $i = 0$ to $P$-1 do
begin
4 PARALLELIZATION OF LEVEL 2 AND 3 BLAS FOR DENSE MATRICES

For $k = 1$ to $P$ do
begin
Send $b(i)$ to PE $i-1 \mod P$
$c(i) = c(i) + A(i,i+k \mod P) \times b(i)$
Receive $b(i)$ from PE $i+1 \mod P$
end

Complexity Analysis
Assume that each multiply and add takes $\tau$ seconds. Assume also that
transferring $w$ words in a network without edge contention takes $\alpha + \beta \times w$,
where $\alpha = \alpha(d)$, $d$ length of the message path. Both $\alpha$ and $\beta$ are machine
dependent parameters. Under the above assumptions the execution time for
the algorithm is:

$$T_P = P \times \left\{ \frac{N^2}{P^2} \times \tau + \alpha + \beta \times \frac{N}{P} \right\}$$

(4.2)

For a single processor (4.2) becomes:

$$T_1 = N^2 \times \tau$$

(4.3)

we get speedup equal to:

$$S(N,P) = \frac{N^2 \times \tau}{P \times \left\{ \frac{N^2}{P^2} \times \tau + \alpha + \beta \times \frac{N}{P} \right\}}$$

(4.4)

The space required for each processor is: $O\left( \frac{N^2}{P} + 2 \times \frac{N}{P} \right)$.

From the equations (4.4) and (3.1) - (3.5) we estimate the speedup for
various problem sizes and different configurations. Figure 2 depicts the
estimated speedup for problem sizes $N = 640, 3200, 32000$ and processors $P$
$= 2^i$, for $i = 0, 4, 6, 7, 8, \text{ and } 9$.

4.2 Dense Matrix Matrix Multiplication

In this section we examine the matrix-matrix multiplication $C = \alpha C + \beta AB$,
where matrix $A, B, C \in \mathbb{R}^{N \times N}$, and $\alpha, \beta$ scalars. The interconnection of PEs
is a wrap around grid. The interconnection of the PEs into a grid topol­
ygy and the distribution of input are shown in Figure 3. PE $(i,j)$ computes
the $C_{i,j}$. As can be seen the matrices $A, B$ are divided into submatrices
Figure 2: Estimated SpeedUp for P = 1, 4, 16, 64, 128, 256, and 512 processors
Figure 3: The interconnection network and distribution of the input
4 PARALLELIZATION OF LEVEL 2 AND 3 BLAS FOR DENSE MATRICES

\[ A_{i,j}, B_{i,j} \in \mathbb{R}^{N \times N}, \text{ where } P \text{ is the number of processors.} \]

There are two paths of moving data, across c-path the algorithm moves the submatrices \( C_{i,j} \) and across b-path it moves the submatrices \( B_{i,j} \).

**Algorithm**

The matrix \( C = (C_{i,j}) \) can be expressed as:

\[
C_{i,j} = \sum_{k=1}^{N} A_{jk} B_{kj}
\]

For square matrices the interconnection is organized as folded square grid \( P \times P \). In each processor \((i,j)\) we store the submatrices \( A_{i,j}, B_{i,j} \in \mathbb{R}^{N \times N} \); initialize \( C_{i,j} \) to zero; throughout this section we will refer to them as \( A, B, C \). In processor \((i,j)\) the submatrix \( C_{i,j} \) is computed after \( \sqrt{P} \) iterations. Each iteration consists of the following three steps: (1) Send \( B, C \), across b/c-paths respectively to processors \((i, (j-1) \mod P), \) and \(((i+1) \mod P, j)\). (2) Compute: \( C = C + A \times B \). (3) Receive \( B,C \) from processors \((i, (j+1) \mod \sqrt{P}), \) and \(((i-1) \mod \sqrt{P}, j)\) respectively. The algorithm can be expressed as follows:

For each PE \((i,j)\) do in parallel

For \( \text{iter} := 1, \sqrt{P} \) do

begin

Send \( B \) across b-path to \((i, (j-1) \mod \sqrt{P}))\)
Send \( C \) across c-path to \(((i+1) \mod \sqrt{P}), j)\)
\( C := C + A \times B \)
Receive \( B \) from processor \((i, (j+1) \mod \sqrt{P}))\)
Receive \( C \) from processor \(((i-1) \mod \sqrt{P}, j)\)
end

**Complexity Analysis**

Under the same assumptions on the time required to communicate and multiply/add on a datum we get:

\[
T_P = \sqrt{P} \times \left\{ \frac{N^3}{P^2} \times \tau + 2 \times (\alpha + \beta \times \frac{N^2}{P}) \right\} \tag{4.5}
\]

Since

\[
T_1 = N^3 \times \tau \tag{4.6}
\]
we get speedup equal to:

\[ S(N, P) = \frac{N^3 \times \tau}{\sqrt{P} \times \left\{ \frac{N^2}{P} \times \tau + 2 \times (\alpha + \beta \times \frac{N^2}{P^3}) \right\}} \]  

(4.7)

The space required is: \( O(3 \times \frac{N^2}{P}) \)

From the equations (4.7) and (3.1) - (3.4), (3.7) we get the speedup for various problem sizes and different configurations. Figure 4 depicts the estimated speedup for problem sizes \( N = 160, 560, 1200 \) and processors \( P = 2^i \), for \( i = 0, 4, 6, 7, 8, \) and 9.

Figure 4: Estimated speedup for \( N = 160, 560, 1200 \), and \( P = 1, 16, 64, 128, 256, 512 \) processors, and Dense Matrix \( \times \) Dense Matrix Multiplication

4.3 Performance on NCUBE-6400
5 Parallelization of level 2, and 3 BLAS for band matrices

The band matrix operations have significant application in the solution of Partial Differential Equations (PDE's). Iterative methods for the solution of the linear algebraic system can be viewed as a matrix vector multiplication operations. In the following, we present these algorithms.

5.1 Band Matrix \times Vector Multiplication

In this section we investigate the operation of \( c = c + A \times b \), where \( c, b \in \mathbb{R}^N \), and \( A \in \mathbb{R}^{N \times N} \) is a banded matrix with \( w_1 \) be the upper bandwidth, and \( w_2 \) the lower bandwidth of the matrix \( A \). We will explain this algorithm for the case \( N = P \). Usually in practice \( N >> P \). However, the case \( N >> P \) can be easily generalized by replacing each element \( a_{i,j} \) by a submatrix \( A_{i,j} \in \mathbb{R}^{\frac{N}{P} \times \frac{N}{P}} \). The interconnection of the PEs is a linear array. The interconnection of PEs and the distribution of input are shown in Figure
5. Each PE $i$ computes $c_i$. As it can be seen the matrix $A$ is splitted into two submatrices, the strictly lower triangular submatrix of $A$, let us call it $L$, and the upper triangular submatrix of $A$, let us call it $U$, such that $A = L + U$. Each processor contains one row of elements (in the general case a strip of rows), and one element of vector $b$ (in the general case a strip of rows).

**Algorithm**

The vector $c$ can be expressed as: $c = c + L \times b + U \times b$. The algorithm consists of 2 phases. In the first phase it multiplies $U \times b$ and $w_1 + 1$ iterations are required; in the second phase it multiplies $L \times b$ and $w_2$ iterations are required. In the first phase each processor $i$ multiplies $a_{ii} \times b_i$ and sends $b_i$ to processor $i-1$ and receives the new part of the vector $b$ form processor $i + 1$. Processor $i = 1$ during the sending stage sends nothing, while processor $i = P$ during the receiving stage receives nothing. At the $k^{th}$ iteration processors $i$, with $i > P - k + 1$ remain idle. In the second phase each processor restores $b_i$, from temporary storage, hence processor $i$ restores $b_i$, and sends it to processor $i+1$, then multiplies $a_{ii-1} \times b_{i-1}$.
during the sending stage sends nothing, while processor $i = 1$ during the receiving stage receives nothing. The algorithm can be expressed as following:

**Phase 1: Multiply the Upper triangular $U$ by $b$**

```plaintext
temp := d
For each PE $i$ do in parallel
   For $j := 0$ to $w_2$
      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 $b$**

```plaintext
For each PE $i$ do in parallel
   begin
      $d := temp$
      For $j := 1$ to $w_2$
         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
```

**Complexity Analysis**

Without any loss of generality we assume $A$ has $K$ non-zero elements, and $N >> w_1 + w_2 + 1$. Under the above assumptions on the time required
to communicate and multiply/add a datum we get:

\[ T_P = \frac{K}{P} \times \tau + (w_1 + w_2 + 1) \times \{t_{\text{start}} + \frac{N}{P}t_{\text{transf}}\} \quad (5.1) \]

Since

\[ T_1 = K \times \tau \quad (5.2) \]

we get a speedup equal to:

\[ S(N, P) = \frac{K \times \tau}{\frac{K}{P} \times \tau + (w_1 + w_2 + 1) \times \{t_{\text{start}} + \frac{N}{P}t_{\text{transf}}\}} \quad (5.3) \]

The space required for each subdomain is: \( O\left(\frac{K}{P} + 3\frac{N}{P}\right) \)

From the equations (5.3) and (3.1) - (3.4), (3.6) we get the speedup for various problem sizes and different configurations. Figure 6, and 7 depict the estimated speedup, and number of iterations for problem sizes \( N = 160, 560, 1200, \) and bandwidths equal to 3, 5, 17, with processors \( P = 2^i \), for \( i = 0, 4, 6, 7, 8, \) and 9.

Application

For example in two dimensions the 5 point star operator for the Poisson equation on a mesh \( N \times N \) will give a band matrix \( A \) whose upper bandwidth is equal to lower bandwidth equal to \( N \). The matrix \( A \) can be viewed as block tridiagonal matrix with blocks in \( \mathbb{R}^{mN \times mN} \), where \( m \in \mathbb{N} - \{0\} \). For given \( N \) and a linear array of \( P \) processors, w.l.o.g assume \( N = \lambda P \), the matrix-vector multiplication of \( A \) times a vector \( x \), can be achieved by applying the algorithm for band matrices. First we partition the matrix into sub-blocks of size \( \lambda N \times \lambda N \), and then we allocate a row of the sub-blocks in each processor, as we did above.

5.2 Band Matrix \( \times \) Band Matrix Multiplication

In this section we investigate the operation of \( C = \alpha C + \beta BC \), where \( A, B \in \mathbb{R}^{N \times N} \), with \( w_1, \delta_1 \) be the upper bandwidths, and \( w_2, \delta_2 \) the lower bandwidths. \( C \in \mathbb{R}^{N \times N} \) with \( w_1 + \delta_1 \) be the upper bandwidth, and \( w_2 + \delta_2 \) the lower bandwidth, and \( \alpha, \beta \) scalers. The interconnection of PEs is a Linear Array. The interconnection of PEs and the distribution of input are shown in Figure 8. PE \( i \) holds the column \( C_i \) of matrix \( C \). In each processor we store one row of elements (in the general case a strip of rows) from matrix \( A \), and one column of elements (in the general case a strip of columns) from
5 PARALLELIZATION OF LEVEL 2, AND 3 BLAS FOR BAND MATRICES

Figure 6: Estimated Speedup of the algorithm for different configurations and bandwidths of the matrix $A \in \mathbb{R}^{64000 \times 64000}$ for the vector $c = c + A \times b$

Figure 7: Iterations required by the algorithm, for the computation of the vector $c = c + A \times b$
Figure 8: The interconnection network and distribution of the input.
matrix $B$.

**Algorithm**

The algorithm consists of two phases as in band-matrix vector multiplication. In the first phase, each PE starts by calculating $c_{ii} = A_i \times B_i$, then each PE $i$ passes $B_i$ to PE $i - 1$. This phase is repeated $w_1 + \delta_1 + 1$ times. In the second phase each PE restores $B_i$, and passes it to PE $i + 1$ and calculates $c_{ij}$.

**Phase 1**

```plaintext
temp := b
For each PE $i$ do in parallel /* each PE contain $a = A_i$, $b = B_i */
    For $j := 0$ to $w_1 + \delta_1$
        if $(i + j \leq N)$ then
            begin
                if ($i = 1$) do nothing
                else Send $b$ to PE $i-1$
                $c(i,i+j) := c(i,i+j) + a \times b$
                if ($i = P$) then do nothing
                else Receive $b$ from PE $i+1$
            endif
        endif
    endfor
endfor
```

**Phase 2**

```plaintext
b := temp
For Each PE $i$ in parallel do
    For $j := 1$ to $w_2 + \delta_2$
        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 $i-1$
                $c(i, i-j) := c(i,i-j) + a \times b$
            endif
        endif
    endfor
endfor
```

**Complexity Analysis**
To calculate the speedup, assume that $K_1, K_2$ denote the number of non-zero elements for the matrices $A, B$ correspondingly, and $N > bw_1 + bw_2$, where $bw_1 = w_1 + w_2 + 1$, and $bw_2 = d_1 + d_2 + 1$. To calculate $T_p$, we have to perform $bw$ iteration, each iteration consists of sending and receiving a strip of $\frac{N}{P}$ vectors each with $\min(bw_1, bw_2)$ elements, and multiply $\frac{N}{P}$ vectors by $\frac{N}{P}$ vectors. Under the above assumptions on the time required to communicate and multiply/add a datum we get:

$$ T_p = \min\left(\frac{(K_1 \delta, K_2 w)}{P}\right) + \left\{t_{\text{start}} + \frac{N}{P} t_{\text{transf}} \min(w, \delta)\right\} \quad (5.4) $$

where $bw = w + \delta$

Since

$$ T_1 = \min(K_1 \delta, K_2 w) \tau \quad (5.5) $$

we get speedup equal to

$$ S(N, P) = \frac{\min(K_1 \delta, K_2 w) \tau}{\min(K_1 \delta, K_2 w) \tau + \left\{t_{\text{start}} + \frac{N}{P} t_{\text{transf}} \min(w, \delta)\right\}} \quad (5.6) $$

From the equations (5.6) and (3.1) - (3.4), (3.8) we get the speedup for various problem sizes and different configurations. Figure 9 depicts the estimated speedup for problem sizes $N = 160, 560, 1200$, and bandwidths equal to $3, 5, 17$, with processors $P = 2^i$, for $i = 0, 4, 6, 7, 8, \text{and } 9$.

5.3 Performance on NCUBE-6400

By fixing the size of the matrix for each processor the resulting performance curves should ideally show constant MFlop as a function of the matrix size and constant time (in Seconds) as a function of Number of Processors, [Gust 88]. Tables 3, and 4 realize the scalability of the algorithm for band matrices with upper bandwidth equal to lower bandwidth equal to $8, 16, 32$ and $64$, the time (in Seconds) does not vary considerably little with the hypercube dimension.

References

Figure 9: Estimated speedup for $N = 160, 560, 1200, \text{ and } P = 1, 16, 64, 128, 256, 512$ processors.

Table 3: Measured Total Elapsed time (in Seconds) for block tridiagonal matrices, each block is of size $n \times n$, where $n = 8, 16, 32, 64$

<table>
<thead>
<tr>
<th>matrix size / node</th>
<th>4 nodes</th>
<th>8 nodes</th>
<th>16 nodes</th>
<th>32 nodes</th>
<th>64 nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 x 24</td>
<td>3.9E-3</td>
<td>4.1E-3</td>
<td>4.1E-3</td>
<td>4.2E-3</td>
<td>4.1E-3</td>
</tr>
<tr>
<td>16 x 48</td>
<td>1.18E-2</td>
<td>1.24E-2</td>
<td>1.27E-2</td>
<td>1.28E-2</td>
<td>1.2E-2</td>
</tr>
<tr>
<td>64 x 192</td>
<td>0.1680</td>
<td>0.1756</td>
<td>0.1794</td>
<td>0.1813</td>
<td>0.1822</td>
</tr>
</tbody>
</table>

Table 4: Measured Mflops for block tridiagonal matrices, each block is of size $n \times n$, where $n = 8, 16, 32, 64$

<table>
<thead>
<tr>
<th>matrix size / node</th>
<th>4 nodes</th>
<th>8 nodes</th>
<th>16 nodes</th>
<th>32 nodes</th>
<th>64 nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 x 24</td>
<td>0.239</td>
<td>0.494</td>
<td>1.004</td>
<td>2.0265</td>
<td>4.122</td>
</tr>
<tr>
<td>16 x 48</td>
<td>0.309</td>
<td>0.635</td>
<td>1.284</td>
<td>2.589</td>
<td>5.198</td>
</tr>
<tr>
<td>32 x 96</td>
<td>0.334</td>
<td>0.688</td>
<td>1.393</td>
<td>2.802</td>
<td>5.620</td>
</tr>
<tr>
<td>64 x 192</td>
<td>0.343</td>
<td>0.704</td>
<td>1.425</td>
<td>2.868</td>
<td>5.752</td>
</tr>
</tbody>
</table>
5 PARALLELIZATION OF LEVEL 2, AND 3 BLAS FOR BAND MATRICES


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


