Parallel Computing 12 (1989) 335–342
North-Holland

Communication efficient matrix multiplication on hypercubes

Jarle BERNTSEN *

Department of Informatics, University of Bergen, Allegt. 55, N-5000 Bergen, Norway

Received May 1988

Abstract. In a recent paper Fox, Otto and Hey consider matrix algorithms for hypercubes. For hypercubes allowing pipelined broadcast of messages they present a communication efficient algorithm. We present in this paper a similar algorithm that uses only nearest neighbour communication. This algorithm will therefore by very communication efficient also on hypercubes not allowing pipelined broadcast. We introduce a new algorithm that reduces the asymptotic communication cost from $2(N^2/P^{1/2})\beta$ to $3(N^2/P^{2/3})\beta$. This is achieved by regarding the hypercube as a set of subcubes and by using the cascade sum algorithm.

Keywords. Matrix multiplication, hypercubes, communication overhead, block algorithms.

1. Introduction

In a recent paper Fox et al. [1] consider matrix multiplication on hypercubes. Their algorithm acts on block matrices, and on hypercubes allowing pipelined broadcast their matrix-matrix multiply will take

$$T = \frac{2N^3}{P}\tau + \frac{2N^2}{\sqrt{P}}t_{\text{comm}} + \sqrt{P}\,(\sqrt{P}-1)\,t_{\text{start}} \tag{1.1}$$

where τ is the time to do a floating-point multiply or add. The matrices to be multiplied both have dimension $N \times N$. P is the number of processors in the hypercube. Their algorithms are developed for the Caltech hypercube. t_{comm} is the time to communicate a single real word and t_{start} is the startup time for the broadcast pipeline per step of the pipe.

In Section 2 we describe the computer model we are dealing with and the rules for the algorithm and the corresponding analysis. We describe an algorithm that uses only neighbour to neighbour communication. The asymptotic communication cost for this algorithm is the same as for the algorithm presented in [1]. In Section 3 we describe a general block matrix algorithm based on the same idea as in Section 2. In Section 4 we describe a technique for reducing the asymptotic communication cost from $2N^2/P^{1/2}$ to $3N^2/P^{2/3}$. This is achieved by regarding the hypercube as a set of subcubes and by using the cascade sum algorithm for adding up contributions to the final product matrix.

* Supported by the Norwegian Research Council for Sciences and Humanities and Statoil, Norway.

2. The computer model, the data decomposition and the algorithm

We are considering multiprocessor systems with the following properties:

(A) P identical processors.
(B) Only distributed memory.
(C) The time to communicate k real words between two processors is $\alpha + \beta k$.
(D) The processors have at least a two-dimensional mesh interconnect with wrap around. The hypercube includes this topology into which it is mapped by a binary Gray code, see [2].
(E) The time to perform a floating-point multiply or add is τ.
(F) Arithmetic and communication cannot overlap.

We will perform the multiplication $C = A * B$ where C, A and B are full $N \times N$ matrices. We will try in this and the next section to develop communication efficient algorithms, and define the rules for the algorithm and the corresponding analysis to be:

(1) No element of C, A or B belongs to more than one processor at the time.
(2) In our analysis of the communication costs we do not take into account any initial loading of A and B.
(3) All submatrices shall after the computation is finished be in the processors where they started.

The matrices are distributed on a $\sqrt{P} \times \sqrt{P}$ mesh of processors. The matrix C is divided in square submatrices C_{ln} holding the elements c_{ik}, with $Nl/\sqrt{P} \leqslant i < N(l+1)/\sqrt{P}$, $Nn/\sqrt{P} \leqslant k < N(n+1)/\sqrt{P}$ and $0 \leqslant l,n < \sqrt{P}$. Each submatrix is given to one processor. A and B are split up in the same way.

In order to compute C the following computations must be performed:

$$C_{ln} = \sum_m A_{lm} * B_{mn}. \tag{2.1}$$

We will distribute the matrices initially in such a way that one of these submatrix multiplications may be performed in each processor. We will let the submatrices C_{ln} stay in the processors where they are placed initially during the computation. After the product $A_{lm} * B_{mn}$ is performed for some m, the processor holding these two submatrices has used all the information it need to update C_{ln}. Before we can perform the next of the multiplications in (2.1), we therefore have to send off the used A and B submatrices, and receive two new ones that may contribute to C_{ln}. By distributing the submatrices initially as described for a 4×4

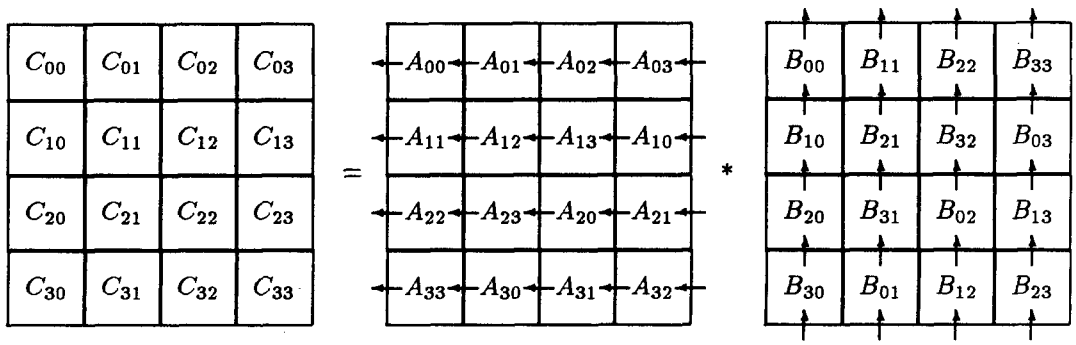

Fig. 1. The decomposition and flow of submatrices.

mesh of processors in Fig. 1, we are able to do the necessary sending with only neighbour to neighbour communication. The algorithm proceeds as follows:

for $m = 0$ **to** $\sqrt{P} - 1$ **do**
Multiply the residing submatrices of A and B and add the product to the residing part of C.
Send the submatrix of A to the west.
Receive a new submatrix of A from the east.
Send the submatrix of B to the north.
Receive a new submatrix of B from the south.

The analysis of this algorithm is very simple. We have $2\sqrt{P}$ neighbour to neighbour communications of matrices having $(N/\sqrt{P})^2$ elements, and $\sqrt{P}$ multiplications of matrices of this size.

$$T = \sqrt{P}\left(2\left(\frac{N}{\sqrt{P}}\right)^3\tau + 2\left(\alpha + \left(\frac{N}{\sqrt{P}}\right)^2\beta\right)\right)$$
$$= \frac{2N^3}{P}\tau + 2\sqrt{P}\,\alpha + \frac{2N^2}{\sqrt{P}}\beta. \tag{2.2}$$

3. A general block matrix algorithm for the hypercube

We now divide the matrices C, A and B in submatrices of size $2^k \times 2^k$ as indicated by Fig. 1. We will use a DIM dimensional hypercube which is mapped on to a 2-dimensional mesh $2^{n_1} \times 2^{n_2}$ with $n_1 + n_2 = DIM$ by using a binary Gray code, see [2], to multiply A and B. Each processor will be given $2^{k-n_1} \times 2^{k-n_2}$ submatrices of A, B and B for $k \geqslant n_1$ and $k \geqslant n_2$. The algorithm proceeds as follows:

for $m = 0$ **to** $2^k - 1$ **do**
Multiply the residing submatrices of A and B and add the product to the residing parts of C.
Replace the submatrices of A with its neighbour to the east.
Replace the submatrices of B with its neighbour to the south.

After 2^k steps of the algorithm C is computed. In each step $2^{k-n_1} + 2^{k-n_2}$ borderline submatrices have to be communicated between neighbouring processors. In each submatrix there are $(N/2^k)^2$ words. In order to reduce the number of startup times all submatrices to be sent from one processor to another may be sent in one package. The total time will then be $(P = 2^{DIM})$

$$T = 2^k\left((2^{k-n_1} * 2^{k-n_2})2\left(\frac{N}{2^k}\right)^3\tau + 2\alpha + (2^{k-n_1} + 2^{k-n_2})\left(\left(\frac{N}{2^k}\right)^2\beta\right)\right)$$
$$= \frac{2N^3}{P}\tau + 2 * 2^k\alpha + 2^k(2^{k-n_1} + 2^{k-n_2})\left(\frac{N}{2^k}\right)^2\beta. \tag{3.1}$$

We will try to select n_1, n_2 and k in order to reduce the communication cost. The term in β is independent of k and we find the minimum of this term when $1/2^{n_1} + 1/2^{n_2}$ achieves its minimum. On even cubes (for even values of DIM) this is achieved for $n_1 = n_2 = \frac{1}{2}DIM$, and on odd cubes (for odd values of DIM) this is achieved for $n_1 = \frac{1}{2}DIM$ and $n_2 = \frac{1}{2}(DIM + 1)$.

In order to select k we look at the term in α. If we select for even cubes $k = n_1 = n_2 = \frac{1}{2}DIM$, this term will take its minimum, and on odd cubes we may select $k = n_1 + 1 = n_2 = \frac{1}{2}(DIM + 1)$. With these choices of n_1, n_2 and k the total time to compute C on even cubes is given by (2.2), and the total time on odd cubes will be

$$T = \frac{2N^3}{P}\tau + 2\sqrt{2P}\,\alpha + \frac{3N^2}{\sqrt{2P}}\beta. \tag{3.2}$$

Compared with the even cubes we notice that the number of startup times is increased, but the asymptotic communication cost defined by the term in β is on the other hand reduced. The factor 2 in (2.2) is replaced by $3/\sqrt{2}$.

We may also easily compute, as they do in [1], the minimum communication cost if we slice our matrices only row or column wise, that is we choose n_1 or n_2 equal to 0. The minimum will then be achieved for $k = n_i = DIM$, $i = 1$ or 2, and the corresponding total time is

$$T = \frac{2N^3}{P}\tau + P\alpha + N^2\beta. \tag{3.3}$$

When we compare expressions (2.2) and (3.3), we notice that when we go from treating the hypercube as a one-dimensional ring to treating it as a 2-D mesh, the asymptotic communication time is reduced by a factor $2/\sqrt{P}$. The question may now be raised: Can we reduce the asymptotic communication time even further by exploiting more of the dimensions available in a hypercube? If we look at expression (2.1), we see that when we let a given C_{ln} stay in one processor during the computation, only two new submatrices of A and B must be replaced in each step of the algorithm. It suffices to use two directions in the hypercube to achieve this. We therefore claim that expressions (3.1) and (3.3) represent the minimum cost for performing a matrix multiplication on a hypercube satisfying (A), (B), (C), (E) and (F) of Section 2 with the algorithm defined by (2.1), and with the rules (1), (2) and (3) given in the same section. We are aware that we could have decided to let the submatrices of A or B stay in the same processors, but then the two other submatrices in each processor must have been replaced in each step and the communication costs would have been the same.

4. Reducing the communication cost for matrix multiplication on hypercubes

The conclusion of Section 3 that it will not help to exploit more than two of the dimensions available in a hypercube when performing matrix multiplication, is connected to the rules (1), (2) and (3) defined in Section 2. In this section we therefore try to reduce the communication cost by allowing the same submatrices of C to be stored in more than one processor, and we will study to which extent we then may exploit more of the connectivity available in a hypercube.

One way of doing this is the following: We split A by columns and B by rows in 2^k parts, see Fig. 2. Each $A^{(i)}$ matrix will now be $N \times (N/2^k)$, and each $B^{(i)}$ will be $(N/2^k) \times N$. We will compute C by using the expression

$$C = \sum_{i=0}^{2^k-1} C^{(i)} = \sum_{i=0}^{2^k-1} A^{(i)}B^{(i)}. \tag{4.1}$$

We split the DIM dimensional cube in 2^k subcubes, see [2], and let each subcube do one of the submatrix multiplications in (4.1). These submatrix multiplications we will perform with the algorithm already described in Section 2. The submatrices to be multiplied within a subcube will, however, now be rectangular. The submatrices of A will be $(N/2^{(DIM-k)/2}) \times$

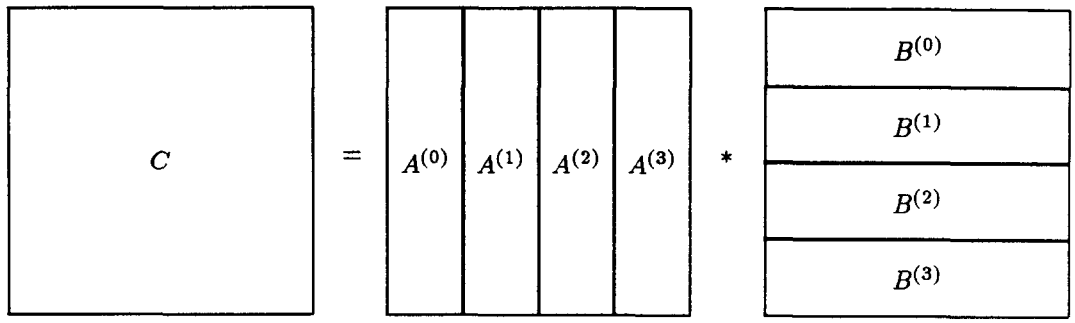

Fig. 2. The decomposition of A and B on to subcubes.

$(N/2^{(DIM+k)/2})$ and the submatrices of B will be $(N/2^{(DIM+k)/2}) \times (N/2^{(DIM-k)/2})$ for $(DIM-k)$ even. For $(DIM-k)$ odd we apply the algorithm described in Section 3 with the choice of submatrix splitting resulting in the expression (3.2). The submatrices of A will then be $(N/2^{(DIM-k+1)/2}) \times (N/2^{(DIM+k+1)/2})$ and B will be $(N/2^{(DIM+k+1)/2}) \times (N/2^{(DIM-k+1)/2})$ for $(DIM-k)$ odd. Within each subcube we number the subblocks with lower indices, and the following computations will be performed:

$$C_{ln}^{(i)} = \sum_m A_{lm}^{(i)} B_{mn}^{(i)}. \tag{4.2}$$

The communication time will be for $(DIM-k)$ even

$$T_{c_1} = 2^{(DIM-k)/2}\left(2\left(\alpha + \frac{N^2}{P}\beta\right)\right) \tag{4.3}$$

and for $(DIM-k)$ odd

$$T_{c_1} = 2^{(DIM-k+1)/2}\left(2\alpha + \frac{3}{2}\frac{N^2}{P}\beta\right). \tag{4.4}$$

After performing the multiplications in (4.2), we will perform the additions by communicating the C submatrices between the subcubes and by using the cascade sum algorithm described in Appendix A. After the process is completed each processor in even cubes will have one submatrix of the final C of size $(N/\sqrt{P}) \times (N/\sqrt{P})$. In odd cubes each processor will have two submatrices of the final C, each of size $(N/\sqrt{2P}) \times (N/\sqrt{2P})$. The time for exchanging the C submatrices between the subcubes will be, see Appendix A,

$$T_{c_2} = k\alpha + (2^k - 1)\frac{N^2}{P}\beta. \tag{4.5}$$

The time for arithmetic is unchanged and the total time for computing C will be

$$T = \frac{2N^3}{P}\tau + T_{c_1} + T_{c_2}. \tag{4.6}$$

For $DIM-k$ even the minimum of this expression is achieved for $k = \frac{1}{3}DIM$.

$$T\left(\frac{DIM}{3}\right) = \frac{2N^3}{P}\tau + \left(2P^{1/3} + \frac{DIM}{3}\right)\alpha + \left(2\frac{N^2}{P^{2/3}} + (P^{1/3}-1)\frac{N^2}{P}\right)\beta \tag{4.7}$$

From (4.7) we see that we have reduced the asymptotic communication cost from $2N^2/P^{1/2}$ to approximately $3N^2/P^{2/3}$. This means that we will be able to exploit higher dimensional hypercubes more efficiently by using more than two of the dimensions present in a cube. We will on the other hand have to store $2N^2/P + 2^k N^2/P$ matrix elements. For $k = \frac{1}{3}DIM$ this

equals $2N^2/P + N^2/P^{2/3}$. For $DIM - k$ odd the asymptotic communication cost may be reduced to the same order of size $(3.12N^2/P^{2/3})$.

One technique that one could consider in order to reduce the communication costs even further, is to apply the idea above once more when computing the products $A^{(i)}B^{(i)}$ in (4.1). However, when using the cascade sum algorithm to add up contributions to each $C^{(i)}$ the subblocks will now be much larger and therefore this will not help.

To summarise we have reduced the communication cost by

(1) Using more of the dimensions in a hypercube
(2) Regarding the hypercube as a set of subcubes.
(3) Using the cascade sum algorithm for adding up contributions to the final product matrix.
(4) Balancing the communication cost for the A and B submatrices with the communication costs for the C submatrices.
(5) Using a two level block matrix algorithm.
(6) Using temporarily more storage.

5. Conclusions

In the first part of this paper we have presented an algorithm for multiplying matrices on hypercubes that may be regarded as a variant of the algorithm introduced in [1]. Our algorithm uses only neighbour to neighbour communication and the asymptotic communication cost will therefore be $2N^2/P^{1/2}$ also for hypercubes that do not allow pipelined broadcast.

The major contribution lies, however, in Section 4 where we by regarding the hypercube as a set of subcubes and by adding up the contributions to the final product matrix with the cascade sum algorithm, reduce the communication cost to $3N^2/P^{2/3}$. In order to achieve this we must apply a two-level block matrix algorithm and we must be able to store temporarily $2N^2/P + N^2/P^{2/3}$ matrix elements in each processor.

Appendix A. The cascade sum algorithm

We will outline in more detail the algorithm described in Section 4. Each processor in a hypercube has a specific binary representation $(i_1, \ldots, i_{DIM})$. We may now from the DIM dimensional cube form 2^k subcubes of dimension 2^{DIM-k} by splitting the cube on k bits, for instance the first k bits, $k \leqslant DIM$, see [2]. We will illustrate the algorithm for DIM and k even. The processors of each subcube we will number (i, j), $0 \leqslant i < 2^{k/2}$, $0 \leqslant j < 2^{k/2}$. Processor (i, j) will compute contributions to submatrices $C_{l_i n_j}$. Each $C_{l_i n_j}$ will now be $(N/\sqrt{P}) \times (N/\sqrt{P})$ as in Section 2. 2^k submatrices $C_{l_i n_j}$ form one submatrix C_{ln} of Section 4.

Initially we let processors (i, j) compute contributions to submatrices $C_{l_i n_j}$, $2^{k/2} i \leqslant l_i < 2^{k/2}(i+1)$, $2^{k/2} j \leqslant n_j < 2^{k/2}(j+1)$. After all submatrix multiplications are performed and all local contributions to each $C_{l_i n_j}$ are formed, we will add up all contributions with the cascade sum algorithm.

Each subcube has binary representation $(i_1, \ldots, i_k)$. Corresponding processors of the different subcubes compute contributions to the same C submatrices. Below we describe an algorithm for adding up the contributions.

Algorithm cascade sum.

Processor (i, j) holds the submatrices $C_{l_i n_j}$, $low_l \leqslant l \leqslant up_l$, $low_n \leqslant n < up_n$, and resides in a subcube with binary representation $(i_1, \ldots, i_k)$.

Initially
$low_l = 2^{k/2}i$,
$up_l = 2^{k/2}(i+1)$,
$low_n = 2^{k/2}j$,
$up_n = 2^{k/2}(j+1)$,
$rows = 2^{k/2}$ and
$cols = 2^{k/2}$.
for $s = 1$ **to** k **do**
  if s is odd **then**
    $cols = cols/2$
    if $i_s = 0$ **then**
      Send off submatrices $C_{l_i n_j}$, $low_l \leqslant l_i < up_l$, $low_n + cols \leqslant n_j < up_n$, to subcubes with $i_s = 1$.
      Receive submatrices $C_{l_i n_j}$, $low_l \leqslant l_i < up_l$, $low_n \leqslant n_j < low_n + cols$, from subcubes with $i_s = 1$.
      Add the received submatrices to the corresponding already residing submatrices.
      $up_n = up_n - cols$
    elseif $i_s = 1$ **then**
      Send off submatrices $C_{l_i n_j}$, $low_l \leqslant l_i < up_l$, $low_n \leqslant n_j < low_n + cols$, to subcubes with $i_s = 0$.
      Receive submatrices $C_{l_i n_j}$, $low_l \leqslant l_i < up_l$, $low_n + cols \leqslant n_j < up_n$, from subcubes with $i_s = 0$.
      Add the received submatrices to the corresponding already residing submatrices.
      $low_n = low_n + cols$
    endif
  elseif s is even **then**
    $rows = rows/2$
    if $i_s = 0$ **then**
      Send off submatrices $C_{l_i n_j}$, $low_l + rows \leqslant l < up_l$, $low_n \leqslant n < up_n$, to subcubes with $i_s = 1$.
      Receive submatrices $C_{l_i n_j}$, $low_l \leqslant l_i < low_l + rows$, $low_n \leqslant n_j < up_n$, from subcubes with $i_s = 1$.
      Add the received submatrices to the corresponding already residing submatrices.
      $up_l = up_l - rows$
    elseif $i_s = 1$ **then**
      Send off submatrices $C_{l_i n_j}$, $low_l \leqslant l_i < low_l + rows$, $low_n \leqslant n_j < up_n$, to subcubes with $i_s = 0$.
      Receive submatrices $C_{l_i n_j}$, $low_l + rows \leqslant l_i < up_l$, $low_n \leqslant n_j < up_n$, from subcubes with $i_s = 0$.
      Add the received submatrices to the corresponding already residing submatrices.
      $low_l = low_l + rows$
    endif
  endif
end for
end cascade sum.

After the above algorithm is completed on all processors, each processor will have the final value of one and only one submatrix of C. Processor (i, j) in subcube $(i_1, \ldots, i_k)$ will have the submatrix $C_{l_i n_j}$ where $l_i = 2^{k/2}i + \sum_{r=2(2)k} i_r 2^{(k/2-r/2)}$ and $n_j = 2^{k/2}j + \sum_{r=1(2)k} i_r 2^{(k/2-(r+1)/2)}$.

In the first step of this algorithm we send 2^{k-1} submatrices of size N^2/P. The number of submatrices that we send in the following steps, is reduced by a factor 2 in each step. The total communication cost will therefore be

$$T_{\text{cascadesum}} = k\alpha + \sum_{i=1}^{k} 2^{i-1} \frac{N^2}{P} \beta = k\alpha + (2^k - 1) \frac{N^2}{P} \beta. \tag{A.1}$$

The above algorithm is developed for *DIM* and k even. Similar algorithms will work also for other permutations of odd/even k and odd/even *DIM*, and the expression (A.1) holds also for all such permutations.

References

[1] G.C. Fox, S.W. Otto and A.J.G. Hey, Matrix algorithms on a hypercube I: Matrix multiplication, *Parallel Comput.* **4** (1987) 17–31.

[2] Y. Saad and M.H. Schultz, Topological properties of the hypercube, Technical Report YALEU/DCS/RR-389, Department of Computer Science, Yale University, 1985.