Abstract. We propose a few implementations of the Alternating Direction Method for solving parabolic partial differential equations on multiprocessors. A careful complexity analysis of these implementations shows that, contrary to what is generally believed, the method can be made highly efficient on parallel architectures by using pipelining and variations of the classical Gaussian elimination algorithm for solving tridiagonal systems. In [15] we showed that we could obtain linear speedups for moderate numbers of processors in a ring architecture. In this paper we discuss extensions to a large number of processors in a 2–D grid architecture and a hypercube.

Alternating Direction Methods on Multiprocessors

S. Lennart Johnsson, Youcef Saad and Martin H. Schultz

Research Report YALEU/DCS/RR-382
October 1985

This work was supported in part by ONR grants N00014–82–K–0184 and N00014–84–K–0043 and in part by a joint study with IBM/Kingston.

1. Introduction

We propose a few implementations of the Alternating Direction Method for solving parabolic partial differential equations on multiprocessors. A careful complexity analysis of these implementations shows that, contrary to what is generally believed, the method can be made highly efficient on parallel architectures by using pipelining and variations of the classical Gaussian elimination algorithm for solving tridiagonal systems. In [15] we showed that we could obtain linear speedups for moderate numbers of processors in a ring architecture. In this paper we discuss extensions to a large number of processors in a 2–D grid architecture and a hypercube.

2. The Alternating Direction Method and its parallel implementations

We consider the partial differential equation:

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(a(x,y)\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(b(x,y)\frac{\partial u}{\partial y}\right)$$

on the domain $(x, y, t) \in \Omega \times [0, T] \equiv [0, 1] \times [0, 1] \times [0, T]$, with the initial and boundary conditions:

$$u(x,y,0) = u_0(x,y), \quad \forall (x,y) \in \Omega,$$
$$u(\bar{x},\bar{y},t) = g(\bar{x},\bar{y},t), \quad \forall(\bar{x},\bar{y}) \in \partial\Omega, \quad t>0,$$

where $\partial\Omega$ is the boundary of the unit square Ω. After the description and analysis of algorithms for the two-dimensional problem we generalize the results to higher dimensional domains.

A common approach to solve the above problem is the Alternating Direction Method. First the equations are discretized with respect to the space variables x and y using a mesh of n interior points in each direction. The result is the system of $N \equiv n^2$ ordinary differential equations:

$$\frac{d\underline{u}}{dt} = A_x\underline{u} + B_y\underline{u} \tag{2.1}$$

in which the matrices A_x and B_y represent the 3-point central difference approximations to the operators $\frac{\partial}{\partial x}(a(x,y)\frac{\partial}{\partial x})$ and $\frac{\partial}{\partial y}(b(x,y)\frac{\partial}{\partial y})$ respectively.

The Alternating Direction Method algorithm consists in stepping (2.1) forward in time alternately in the x and y directions as follows [21]:

$$(I - \frac{1}{2}\Delta t A_x)u^{i+\frac{1}{2}} = (I + \frac{1}{2}\Delta t B_y)u^i \tag{2.2}$$

$$(I - \frac{1}{2}\Delta t B_y)u^{i+1} = (I + \frac{1}{2}\Delta t A_x)u^{i+\frac{1}{2}}. \tag{2.3}$$

We observe that if the mesh points are ordered by lines in the x direction, then (2.2) constitutes a set of n independent tridiagonal systems whose solution is perfectly parallellizable. However, (2.3) constitutes one tridiagonal system in which the nonzero diagonals are the main diagonal and the $(n + 1)^{st}$ super- and sub-diagonals. It is important to note that this second system can also be recast into a set of n independent tridiagonal systems by *reordering the grid points* by lines, this time in the y direction. This essentially amounts to transposing the matrix representing the solution at the $n \times n$ grid points and is an expensive data permutation operation which is often cited as the main drawback of Alternating Direction Method in regard to its implementation on parallel machines. The other difficulty that has been traditionally associated with parallelizing the

Alternating Direction Method is that the classical algorithms for solving tridiagonal systems are sequential in nature. The arithmetic complexity of the Alternating Direction Method on an idealized parallel architecture without contention for storage or access conflicts has been investigated by Sameh et al. [18]. Various parallel implementations of the Alternate Direction Method have been discussed by Gannon and Van Rosendale [3] Lambiotte [9] and Ortega and Voigt [11].

The main purpose of this paper is to describe some data structures and algorithms that efficiently use some candidate multiprocessor configurations for future parallel computer systems. We limit the discussion to the solution of the two tridiagonal systems (2.2) and (2.3). The communication required for the computation of the right hand side is almost identical to that required for the tridiagonal system, and the arithmetic complexity is approximately half of that for the tridiagonal system. Since the choice of data structure and algorithm is unaffected by the inclusion of the computations for the right hand side we omit them. On a single processor each half step costs $5n^2$ operations for the forward and $3n^2$ operations for the backward sweep. Assuming that s arithmetic operations can be done per second, the time for a half step on a single processor is approximately

$$T_1 = \frac{8n^2}{s}. \tag{2.4}$$

Throughout the paper it is assumed that communication and arithmetic computations cannot take place concurrently. This assumption is essentially made for convenience and simplicity of our analysis and does not affect the qualitative aspects of the conclusion.

3. Alternating Direction Methods on the shared memory model and the broadcast bus model

In this section, we assume that k processors are connected to one large shared memory with total bandwidth B words per second and start up τ. Moreover, we assume that the I/O bandwidth of each processor is b. Thus B/b different processors can simultaneously access the shared memory.

The transfer of data between two processors is done by the first processor writing to the shared memory and then the second processor reading from the shared memory. Consider a method consisting in solving n/k similar tridiagonal systems in each processor for (2.2) then transposing the data and solving (2.3) as n/k tridiagonal systems in each processor (it is assumed that $k \leq n$). The total time required for performing the arithmetic operations of one half step is approximately

$$T_{S,Arith} \approx \frac{n}{k}\frac{8n}{s} = \frac{8n^2}{ks}$$

Performing a transpose of the n x n matrix of the unknown u requires the communication time:

$$\begin{aligned} T_{S,Comm} &\approx 2\frac{k}{B/b}\left(\tau + \frac{n^2}{kb}\right) \\ &\approx 2k\frac{b}{B}\tau + 2\frac{n^2}{B}. \end{aligned}$$

Hence in the shared memory model, it takes a total time of

$$T_S \approx 2k\frac{b}{B}\tau + 2\frac{n^2}{B} + \frac{8n^2}{ks}$$

to perform one half step of the Alternating Direction Method.

Assume now that all k processors are linked to each other via a global bus which allows for broadcasting. The time to perform the arithmetic operations is the same as for the shared memory model.

1	2	3	4
2	3	4	1
3	4	1	2
4	1	2	3

Figure 1: Domain decomposition and assignment of the unit square into 4 processors.

To transpose the matrix, each processor broadcasts in turn its data to all the other processors. This requires a total communication time of

$$T_{B,Comm} \approx k(\tau + \frac{n^2}{kb}) = k\tau + \frac{n^2}{b}$$

Hence the total time

$$T_B \approx k\tau + \frac{n^2}{b} + \frac{8n^2}{ks}$$

for performing one half step of the Alternating Direction Method.

Observe that neither of the above two models achieves a reasonable speed up when k increases because of the limiting term n^2 in the communcation cost. This is the source of the belief that Alternating Direction Methods do not parallelize well.

4. Alternating Direction Methods on a multiprocessor ring

In the ring architecture the k processors are arranged in a ring and each processor can communicate with its two nearest neighbors. It is assumed throughout that the processors are numbered so that processor numbered i is connected to processors numbered $i+1$ and $i-1$ modulo k. In order to efficiently implement the algorithm, we will first assign the grid points such as to minimize data communication costs. The assignment, which is described in Figure 1 for a ring of $k = 4$ processors, will essentially avoid transposing the data at each half step.

This assignment results in each of the tridiagonal systems (2.2), being split into k equal parts, one part per processor. We consider the cost of the half step involving the sweep in the x-direction. Looking at the leftmost column of subsquares in Figure 1, each of the k processors of that column can start performing the forward sweeps simultaneously on the n/k tridiagonal systems that it holds. Thus, the sweep eliminates the variables horizontally from left to right. When the boundary of the first subsquare is reached, each processor sends to its right neighbor in the figure the data that is necessary for that processor to continue its forward sweep on its part of its tridiagonal systems, i.e., the processor numbered j sends to its neighbor numbered $(j+1)$ Mod k data for the $(j-1)\frac{n}{k}+1$ to $j\frac{n}{k}$ tridiagonal systems. The forward sweep can now be pursued on the second column of subsquares, and this process is repeated until the forward sweep is completed on the variables of the last column of subsquares. The backward sweep is accomplished in a similar fashion except that we proceed from right to left. The half step in the y direction is performed similarly, with the forward sweep proceeding from bottom to top and the backward sweep from top to bottom. We observe that no processor is idle at any time. It is clear that the time to perform the arithmetic operations is again of the form:

$$T_{R,Arith} \approx \frac{n}{k}\frac{8n}{s} = \frac{8n^2}{ks}.$$

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Figure 2: Assignment of the subdomains to the 16 processors of a 4 x 4 grid

When we cross an interface in the forward sweep, we must transfer the pivot row plus the corresponding element of the right hand side of each of the n tridiagonal systems in each processor numbered j to the processor numbered $(j+1)$ Mod k. These I/O operations can be done in parallel for each of the processors on the same column. A total of $k-1$ interfaces must be crossed. For the backward sweep, we transfer only one element of the current solution per system. Therefore:

$$T_{R,Comm} = (k-1)\left(\tau + \frac{3n}{kb}\right) + (k-1)\left(\tau + \frac{n}{kb}\right) \approx 2(k-1)\tau + \frac{4n}{b}.$$

Thus, the total execution time comes to

$$T_R(k) = 2(k-1)\tau + \frac{4n}{b} + \frac{8n^2}{ks}.$$

Clearly, the time for solving in the other direction is identical.

We observe that with increasing k the arithmetic time decreases while the communication time increases linearly. The function $T_R(k)$ has a minimum which is achieved for

$$k_{opt} \approx \frac{2n}{\sqrt{s\tau}}$$

and the corresponding optimal time is linear in n:

$$T_{R,opt} \approx 4\left(2\sqrt{\frac{\tau}{s}} + \frac{1}{b}\right) n.$$

Thus by an appropriate choice of algorithm on the ring, with $O(n)$ processors we can achieve an $O(n)$ speed-up contradicting the conventional wisdom.

5. Alternating Direction Method on a square grid of processors

In this section we consider a square two-dimensional grid of processors. We assign the sub-squares defined in the previous section naturally onto the grid, as shown in Figure 2 in which the numbers in each square indicate the processor number to which the subsquare is assigned. It is convenient to define $\kappa \equiv \sqrt{k}$.

This assignment resembles the one of the previous section but each subsquare is now handled by a different processor. If we proceeded as we did for the ring, we would have many processors inactive at all times. In fact at any given time we would have only κ processors working simultaneously; namely all those holding the same column of subsquares. To avoid this we can think of pipelining the computation: the forward sweep is started with only one tridiagonal system in each of the processors in the first column of subsquares (Processors 1, 5, 9 and 13 in the illustration of Figure 2). As soon as these processors are finished with the first n/κ steps of Gaussian elimination of their first tridiagonal systems they start the forward sweep on the κ second tridiagonal systems, while processors to their right in the grid start working on their part of the first tridiagonal system. Thus, after $\kappa - 1$ tridiagonal systems are solved, all processors will be active. It is easy to see that the computation consists of forward sweeps on a total of $(n/\kappa) + \kappa - 1$ tridiagonal systems of size n/κ each: n/κ to complete all the first parts in each of the subsquares of the first column of processors, and then $\kappa - 1$ similar eliminations to complete the parts through the last column. Moreover, these are non-simultaneous eliminations. Each of the partial eliminations is followed by a transfer of the pivot row, i.e. 3 elements. The backward sweep is similar. We find a total time of

$$t_{G0} = (\frac{n}{\kappa} + \kappa - 1)\left[2\tau + \frac{4}{b} + \frac{8n}{\kappa s}\right]$$

For $\kappa^2 = k \leq n$, a nearly linear speed-up over the run time for a single processor is achieved and the optimal time is linear in n. While this is asymptotically equivalent to the result for the ring of processors, it requires an order of magnitude less time for just the data movement. Unfortunately, the above time cannot be less than linear in n, no matter how we choose k, for example if $n < k \leq n^2$ i.e., it is not possible to significantly outperform the ring architecture for implementing the Alternating Direction Method with this algorithm. The reason for this is that we are solving tridiagonal systems with the classical Gaussian elimination algorithm. With Gaussian elimination, each tridiagonal system cannot be solved in time less than $8n/s$, independent of the number of processors used because of its sequential nature.

To exploit the more powerful interconnection features of the grid architecture, we must use a different algorithm. Lambiotte [9] suggested using Gaussian elimination in one direction and odd-even cyclic reduction in the other. This, however will still require a linear time because of the Gaussian elimination in one direction. More recently, Gannon and Van Rosendale [3] have proposed using a scheme based on what the mechanical and structural engineers call problem substructuring or condensation techniques.

There are many substructuring Gaussian elimination algorithms. The one briefly described next is due to Wang [22]. It is a variation of an algorithm first presented by Kuck and Sameh [19, 10]. Advantages of Wang's variant are its low arithmetic complexity with almost no increase in communication complexity and its improved numerical properties [6]. The algorithm can be briefly described as follows. Assume that we want to solve *one* tridiagonal system $Tx = b$ of size n in κ processors, each of which holds n/κ successive rows of the system. The algorithm consists of three phases. In the first phase a variant of Gaussian elimination is performed with almost no interprocessor communication. In the j^{th} processor, $j = \{1, 2, ..., \kappa\}$, a forward elimination is carried out on equations $(j-1)\frac{n}{\kappa} + i$, $i = \{2, 3, ..., \frac{n}{\kappa}\}$, in order to eliminate the sub-diagonal elements within each $\frac{n}{\kappa}$ x $\frac{n}{\kappa}$ diagonal block of T. This requires no interprocessor communication and all the processors compute in parallel. Then, a backward elimination is carried out to eliminate the super-diagonal elements in rows $j\frac{n}{\kappa} - 2$ through $(j-1)\frac{n}{\kappa}$ for $j = \{2, 3, ..., \kappa\}$ and in rows $\frac{n}{\kappa} - 2$ to 1 for $j = 1$. After a small amount of interprocessor computation, all processors compute in parallel. The result of this first phase is illustrated in Figure 3 for a system of size 20 distributed in

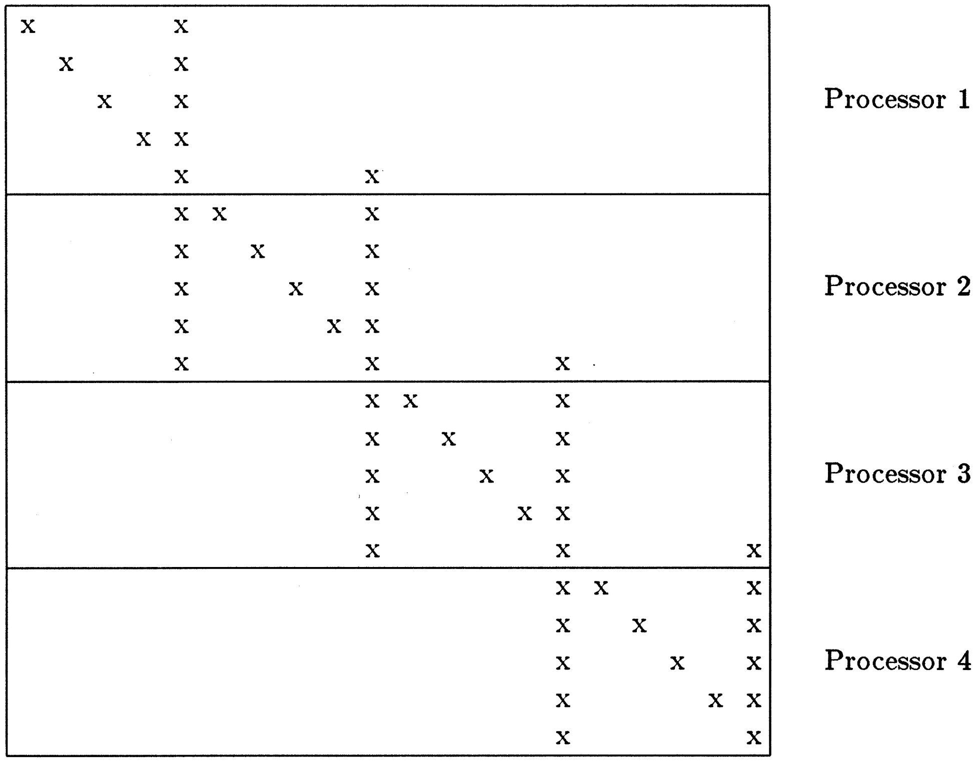

Figure 3: A substructuring Gaussian elimination algorithm on four processors.

4 processors. Communication of three nonzero matrix elements and one right hand side element, is required between adjacent processors, leading to a total cost of

$$t_1 = (\frac{n}{\kappa} - 1)\frac{12}{s} + \tau + \frac{4}{b},$$

for the first phase.

We observe that the unknowns $j\frac{n}{\kappa}, \quad j = \{1, 2, ..., \kappa\}$, satisfy an independent tridiagonal system of κ equations distributed with one equation per processor. Clearly, the second phase will consist in solving for these unknowns. If the processors form a ring or linear array, the total cost, using a regular Gaussian elimination algorithm, is roughly

$$t_2 \approx (\kappa - 1)(2\tau + \frac{4}{b} + \frac{8}{s})$$

In the third and final phase, each processor j solves for the other variables by subtracting multiples of the fill-in columns. This operation involves no communication, except for the transfer of the variables $j\frac{n}{\kappa}$ already computed in phase 2 from processors j to processors $j+1$ for $j = \{1, 2, ..., \kappa - 1\}$. Therefore, the cost of the third phase on a linear array or ring is approximately

$$t_3 \approx \frac{5n}{\kappa s} + \tau + \frac{1}{b}.$$

Going back to implementing the Alternating Direction Method on our grid of processors, we observe that each row of processors can be considered as a linear array of $\kappa \equiv \sqrt{k}$ processors.

In each row of processors we will solve n/κ tridiagonal systems by the substructuring Gaussian elimination. In the first pass of the Alternating Direction Method all the rows of the processor grid will work simultaneously, with no communication between the processors of different rows. In the second pass of the Alternating Direction Method all processors of the same column of the processor grid will work simultaneously with no communication between different columns.

Implementing the first phase is straighforward and its cost is

$$t_1' = \frac{n}{\kappa}.\frac{12n}{\kappa s} + \tau + \frac{n}{\kappa}.\frac{4}{b} = \frac{12n^2}{ks} + \tau + \frac{4n}{\kappa b}.$$

Implementing the second phase requires a little more care. The problem is that we have n/κ tridiagonal systems each of size κ and with one row per processor. Just as was pointed out in our first implementation of the Alternating Direction Method for the grid described in the very beginning of this section, a naive implementation consisting in sweeping in the horizontal direction for all variables of the first column of subsquares before forward-sweeping in the second column of subsquares and so on, would be inefficient because only one processor of each row will be active at any time. In fact, this naive approach would require solving n/κ tridiagonal systems of size κ each and therefore the time would again be nearly linear in n. Once again the remedy is to use pipelining. It is interesting that the seemingly minor modification of pipelining leads to a completely different complexity analyšis and a different conclusion.

An argument similar to the one of the beginning of this section shows that in order to complete the forward sweep we need a total of $n/\kappa + \kappa - 1$ successive steps each consisting of one Gaussian elimination of one row of a tridiagonal system plus a transfer of the pivot row. Clearly, the backward sweep can by pipelined likewise. The total time for this second phase is approximately

$$t_2' = (\frac{n}{\kappa} + \kappa - 1)\left(2\tau + \frac{4}{b} + \frac{8}{s}\right).$$

Finally, the third phase is fully parallelizable and requires a time of

$$t_3' \approx \frac{5n^2}{ks} + \tau + \frac{n}{\kappa}.\frac{1}{b}.$$

The total as a function of k comes to

$$T_G(k) = \frac{17n^2}{ks} + (\frac{n}{\kappa} + \kappa - 1)\left(2\tau + \frac{4}{b} + \frac{8}{s}\right) + 2\tau + \frac{5n}{\kappa b}.$$

The important point is that it is now possible to obtain an execution time of order less than $O(n)$. In fact, since $T_G(k)$ is of the form

$$T_G(k) = \alpha\frac{n^2}{k} + \beta\frac{n}{\sqrt{k}} + \gamma\sqrt{k} + Constant,$$

its *approximate* minimum is achieved for

$$k_* = \left(2\frac{\alpha}{\gamma}n^2\right)^{2/3}$$

and has the value

$$T_{G,*} \approx \frac{3}{2}\left(2\gamma^2\alpha\right)^{1/3} n^{2/3}.$$

It is easy to show by employing an argument similar to one used in [14] that $T_{G,*}$ is close to the actual minimum of $T_G(k)$ when n tends to infinity.

6. Alternating Direction Methods on Hypercube Architectures

There are currently several existing loosely coupled multiprocessors, labeled under the name hypercube, which are based on the binary n-cube networks [1, 17, 16, 20]. An m-dimensional hypercube, or $m-$cube, consists of 2^m nodes that are numbered by m-bit binary numbers, from 0 to $2^m - 1$ and interconnected so that there is a link between two processors if and only if their binary representation differs by one and only one bit. For the case $m = 3$, the 8 nodes can be represented as the vertices of a three dimensional cube. One of the main advantages of hypercubes is that many of the classical topologies such as two-dimensional or three-dimensional meshes [8, 2, 17, 20, 7] can be imbedded preserving proximity in them. An immediate consequence of this is that the algorithms presented for the ring and the grid architectures can be immediately adapted to the hypercube. Thus one step of the Alternating Direction Method can be run on a hypercube in the same time as on a mesh configured multiprocessor with the same number of processors, i.e., in time $O(n^{2/3})$ for a sufficiently large dimension. All we need is to imbed the grid into the hypercube and then use the substructuring Gaussian elimination of the previous section. For a detailed description on how to map a grid into a hypercube, see for example [17]. An important natural question is whether or not there is another algorithm which achieves a better asymptotic time because of the more powerful topological properties of the hypercube. Once again the answer is yes.

A natural candidate for solving tridiagonal systems on multiprocessors is the cyclic reduction algorithm [4, 5]. Using the same mapping as for the two-dimensional grid, i.e., partitioning the grid into squares of size $\frac{n}{\kappa} \times \frac{n}{\kappa}$ and assigning the square in position (i,j) to processor (i,j) of a two-dimensional mesh of $\kappa \times \kappa$ processors embedded in the hypercube by the proximity preserving embedding described above, it is clear that each of the solve phases in the Alternating Direction Method amounts to solving in each row or column of the conceptual processor mesh n/κ independant tridiagonal systems each of which is split into κ equal parts.

It is important to note that when using the cyclic reduction algorithm the distance between the rows of the tridiagonal system will increase if the grid points are not carefully assigned to the nodes. With a mesh embedding based on a binary-reflected Gray code proximity is preserved throughout the reduction process [5]. For *one* tridiagonal system of 2^m equations, equation i is mapped into the node whose binary label is g_i, where $g_0, g_1, \ldots g_{2^m-1}$ is the m-bit binary-reflected Gray code [12]. It can be easily shown that g_i and $g_{(i+2^j) mod\ 2^m}$ differ in exactly two bits, for all $j>0$. This property means that, with this Gray code assignment, the distance between consecutive rows in the first step of cyclic reduction is one and is exactly two in the subsequent steps. If the tridiagonal system is of size n larger than $\kappa \equiv 2^m$, then substructuring is used with substructures of $\frac{n}{\kappa}$ equations each and substructure j, $j = \{0, 1, ..., \kappa - 1\}$, assigned to processor g_j. The solution process is similar to the substructured Gaussian elimination described previously, except that cyclic reduction is used for phase 2. The times for phases 1 and 3 are the same as before. The time for phase 2 is [5]

$$t_2 = \left(\frac{17}{s} + 4\tau + \frac{10}{b}\right)(log_2\kappa - 1) - 2\tau - \frac{5}{b} + \frac{1}{s}.$$

Consider now the problem of employing this technique in the Alternating Direction Method. An important observation is that each column (or row) of processors is a subcube of the hypercube, on which work can be done independently. Thus, one can consider using the above mapping based on Gray codes on each of the subcubes and the κ different subcubes will solve in parallel a set of n/κ tridiagonal systems each of size n and spread in κ processors, n/κ equations per processor. With $\frac{n}{\kappa}$ independant tridiagonal systems to be solved in each subcube in phase 2 the following estimate is derived assuming no pipelining of communication actions:

$$t_2' = \left[4\tau + \left(\frac{10}{b} + \frac{17}{s}\right)\frac{n}{\kappa}\right](log_2\kappa - 1) - 2\tau - \frac{5n}{b\kappa} + \frac{n}{s\kappa}.$$

If the two successive communications in a reduction step can be pipelined, then the term $\frac{10}{b}$ can be reduced to $\frac{5}{b}$.

By comparing the time for phase 2 using cyclic reduction on a hypercube to that using Gaussian elimination we conclude that if the communication start-up time is high with respect to the elemental transfer time $1/b$ and with respect to the arithmetic processing time $1/s$, then cyclic reduction is preferrable to Gaussian elimination. If start-ups can be ignored then Gaussian elimination is superior when n is large relative to κ while cyclic reduction is superior when n is small. More precisely, let us assume that $\tau = 0$, and define $\alpha \equiv b/s$, the ratio of communication to arithmetic speeds. Then, by comparing the expressions for t_2' for Gaussian elimination and cyclic reduction respectively and by neglecting the lower order terms, we find that the break-even point is realized when n is approximately

$$n(\alpha) = \frac{\kappa^2}{\frac{10+17\alpha}{4+8\alpha}log_2\kappa - \frac{19+24\alpha}{4+8\alpha}}. \tag{6.1}$$

What is interesting is that the two fractional coefficients in the denominator of the above expression lie in the intervals $[2.125, 2.5]$ and $[3, 4.75]$ respectively for all values of α between zero and infinity. Therefore, for the sake of simplicity we replace the above expression of $n(\alpha)$ by

$$n(\alpha) \approx \frac{\kappa^2}{2log_2\kappa - 4} = \frac{k}{log_2\frac{k}{16}}.$$

Thus, under the above assumptions, when $\kappa = 16, (k = 256)$ cyclic reduction is faster for all values of n not exceeding $n \approx 64$. When $\kappa = 64, (k = 4096)$ the break-even occurs at $n \approx 512$ while for $\kappa = 1024 (k = 2^{20} = 1{,}048{,}576)$ it takes place at $n \approx 65{,}536$. Of course, if communication start-ups were to be included then the competitiveness of cyclic reduction is further improved.

It should be noticed that with the above simplistic implementation of cyclic reduction the reduction process for all equations converge to the same processor and this leads to a bottleneck. The implementation can be improved by making the reduction process for different equations converge to different processors. That the reduction process can be made to converge to an arbitrary processor is easily seen if, conceptually, a tridiagonal system is extended with the appropriate number of leading, or trailing equations with a corresponding 0 solution (0 off-diagonal elements and right hand side). For κ tridiagonal systems solved in a subcube of κ nodes the load balancing is perfect, and each node performs the same number of arithmetic operations in the reduction of all κ equations as is required for the reduction of a single equation. Hence, for this improved implementation of cyclic reduction we arrive at the following estimate for the time of one step of the Alternating Direction Method

$$\hat{t}_2 = 2(2log_2\kappa - 1)\tau + \left(2.\frac{5\kappa - log_2\kappa}{b} + \frac{17\kappa - 18log_2\kappa + 2}{s}\right)\lceil\frac{n}{\kappa^2}\rceil.$$

The number of start-ups is the same as in the naive implementation of the cyclic reduction method, but the communication and arithmetic bandwidth requirements are reduced. This improved implementation of cyclic reduction is of lower complexity than Gaussian elimination if $n \leq \kappa^2$, approximately, ignoring start-ups. The reason for the advantage of Gaussian elimination over cyclic reduction for $\frac{n}{\kappa} > \kappa$ is due to its lower arithmetic complexity. The solution of a tridiagonal system by cyclic reduction requires approximately twice the number of arithmetic operations of Gaussian elimination, and the sequential nature of Gaussian elimination is outweighed by its arithmetic efficiency for sufficiently many equations per node. The improved implementation of cyclic reduction is competitive for $n \leq 64$ for $\kappa = 8$, and $n \leq 1024$ for $\kappa = 32$.

Adding the complexities for all three phases $(t_1', \hat{t}_2, t_3')$ of one step of the Alternating Direction Method yields

$$T_C(k) = \frac{17n^2}{ks} + 2(2log_2\kappa - 1)\tau + \left(2.\frac{5\kappa - log_2\kappa}{b} + \frac{17\kappa - 18log_2\kappa + 2}{s}\right)\lceil\frac{n}{\kappa^2}\rceil + 2\tau + \frac{n}{\kappa}\frac{5}{b}.$$

The form of $T_C(k)$ is

$$T_C(k) = \alpha\frac{n^2}{k} + \beta\frac{n}{\sqrt{k}} + \gamma\frac{n}{k}log_2k + Constant,$$

and its minimum is achieved for $\kappa = n$. For this value of κ

$$T_C \approx \left(4\tau + \frac{10}{b} + \frac{17}{s}\right)log_2n.$$

7. Higher Dimensional Problems

For the solution of higher dimensional problems we note that there exist no proximity preserving embedding of a m-dimensional grid in an n-dimensional grid for $m > n$ [13]. Hence, the order of the computational complexity for solving a two-dimensional problem on a linear array cannot be reduced below $O(n)$ by extending the linear array to more than $O(n)$ nodes. Solving a m-dimensional problem with n^m interior grid points, $m \geq 2$, on a linear array requires a time of order $O(n^{m-1})$.

Similarly, for $m \geq 3$ and a two-dimensional array of processors, the minimum time is of order $O(n^{m-2})$ and the corresponding number of processors is n^2. In the case of a three-dimensional problem one simple embedding in a two-dimensional array of processors is obtained by identifying all points in one dimension and using the two-dimensional grid embedding described previously. Two of the three computational steps of the Alternating Direction Method for each time step are performed as in the two-dimensional grid case but repeated n times (for the third dimension). The third step is local to each processor and consists in solving $\left(\frac{n}{\kappa}\right)^2$ tridiagonal systems of n equations each. The time for this step is

$$T_G^3 = \frac{8n}{s}\left(\frac{n}{\kappa}\right)^2$$

and the time for each of the other two steps is

$$T_G^1 = T_G^2 = \frac{17n}{s}\left(\frac{n}{\kappa}\right)^2 + (n\frac{n}{\kappa} + \kappa - 1)\left(2\tau + \frac{4}{b} + \frac{8}{s}\right) + 2\tau + \frac{5n}{b}\frac{n}{\kappa}$$

assuming that the reduced system is solved by Gaussian elimination. It is clear that for this embedding $\kappa = n$ minimizes $T_G^i, \quad i = \{1,2,3\}$.

Higher dimensional meshes can be embedded in sufficiently large hypercubes preserving proximity by subdividing the cube into m subcubes for a m-dimensional grid. As for the two-dimensional grid the m-dimensional grid is partitioned into regions and the regions embedded in the hypercube by encoding their indices in a binary-reflected Gray code. With a m-dimensional grid partitioned uniformly into κ partitions in each dimension, the grid having n interior points in each dimension, and the subcubes being of size κ, the complexity estimate for each of the m steps of the Alternating Direction Method is

$$T_C^m = \frac{17}{s}\left(\frac{n}{\kappa}\right)^m + 2(2log_2\kappa - 1)\tau + \left(2.\frac{5\kappa - log_2\kappa}{b} + \frac{17\kappa - 18log_2\kappa + 2}{s}\right)\lceil\frac{1}{\kappa}\left(\frac{n}{\kappa}\right)^{m-1}\rceil + \\ + 2\tau + \frac{5}{b}\left(\frac{n}{\kappa}\right)^{m-1}$$

using cyclic reduction for the solution of the reduced system. The minimum is the same as in the two-dimensional grid case, namely

$$T_C^m \approx \left(4\tau + 2\frac{5}{b} + \frac{17}{s}\right) log_2 n.$$

8. Conclusion

In this paper we have proposed several implementations of the Alternating Direction method on multiprocessors. Estimates of the time to execute one half step of the algorithm lead us to the following conclusions.

- The shared memory model and broadcast bus model do not allow for highly efficient implementations of the Alternating Direction Method.
- If $k \leq n$ then the ring architecture is sufficient to obtain an asymptotically optimal speed up. The optimal time is of order $O(n^{m-1})$ for a m-dimensional grid of $O(n^m)$ interior points. The optimum is attained for n processors.
- In order to reduce the computational complexity further it is necesary to use an array of processors of a dimensionality closer to that of the problem. For a two-dimensional grid a two-dimensional array of processors suffices to achieve a complexity less than $O(n)$. By using a substructuring technique and solving the reduced system by either Gaussian elimination or cyclic reduction a minimum complexity of order $O(n^{2/3})$ is obtained. The number of processors for which the minimum is attained is $O(n^{4/3})$. The method which has the lowest absolute complexity is dependent on the ratio of the communication and arithmetic bandwidths and the ratio $\frac{n}{\kappa}$ [5]. For a m-dimensional grid, $m \geq 3$, the minimum time for a two-dimensional array of processors is of order $O(n^{m-2})$ and the corresponding number of processors $O(n^2)$.
- The hypercube architecture makes it possible to exploit very large scale parallelism in the Alternating Direction Method. One step of ADI on an an m-dimensional problem can be performed in time proportional to $\alpha log_2 n$ by using an n^m-node hypercube. This time is of the order of the lower bound.
- For $k < n$, i.e., when the number of processors is less than the number of grid points in one dimension, the lower arithmetic complexity of Gaussian elimination compared to cyclic reduction may outweigh its inherent sequential nature, since the degree of parallelism is relatively low. For a two-dimensional problem a substructuring algorithm based entirely on Gaussian elimination may be of a lower complexity than combining local Gaussian elimination with "global" cyclic reduction if $\frac{n}{\kappa} > \kappa$. Consequently, a ring or a grid architecture may yield a complexity

comparable to that of a hypercube architecture. For higher dimensional problems the hypercube architecture always yields a lower asymptotic complexity than a ring or a two-dimensional array architecture. The break-even point between Gaussian elimination and cyclic reduction occurs approximately at $n \approx \kappa^{m/(m-1)}$.

References

[1] L. N. Bhuyan, D.P. Agrawal, *Generalized Hypercube and Hyperbus structures for a computer network,* IEEE Trans. Comp., C-33 (1984), pp. 323–333.

[2] T.F. Chan, Y. Saad, *Multigrid Algorithms on the Hypercube multiprocessor,* Technical Report 368, Computer Science Dept., Yale University, 1985.

[3] D. Gannon, J. van Rosendale, *On the Impact of Communication Complexity in the Design of Parallel Algorithms,* Technical Report 84-41, ICASE, 1984.

[4] R.W. Hockney, C.R. Jesshope, *Parallel Computers,* Adam Hilger Ltd, Bristol, England, 1981.

[5] S.L. Johnsson, *Odd-Even Cyclic Reduction on Ensemble Architectures.,* Technical Report YALEU/DCS/RR-339, Computer Science Dept., Yale University, 1984. To appear in SIAM J. Sci. Stat. Comp.

[6] ———, *Solving Narrow Banded Systems on Ensemble Architectures,* ACM, TOMS, 11/3 (1985).

[7] ———, *Communication Efficient Basic Linear Algebra Computations on Hypercube Architectures,* Technical Report YALEU/CSD/RR-361, Dept. of Computer Science, Yale University, September 1985.

[8] ———, *Data Permutations and Basic Linear Algebra Computations on Ensemble Architectures,* Technical Report YALEU/CSD/RR-367, Yale University, Dept. of Computer Science, February 1985.

[9] J.J. Lambiotte, *The solution of linear systems of equations on a vector computer,* Ph.D. Thesis, University of Virginia, 1975.

[10] D. Lawrie, A.H. Sameh, *The Computation and Communication Complexity of a Parallel Banded Linear System Solver,* ACM-TOMS, 10/2 (1984), pp. 185–195.

[11] J.M. Ortega, R.G. Voigt, *Solution of partial differential equations on vector and parallel computers,* Technical Report 85-1, ICASE, NASA Langley Research Center, 1985.

[12] E.M. Reingold, J. Nievergelt, N. Deo, *Combinatorial Algorithms,* Prentice Hall, New-York, 1977.

[13] A.L. Rosenberg, *Preserving Proximity in Arrays,* SIAM J. Computing, 4/4 (1975), pp. 443–460.

[14] Y. Saad, M.H. Schultz, *Direct parallel methods for solving banded linear systems,* Technical Report YALEU/DCS/RR-387, Computer Science Dept., Yale University, 1985.

[15] ———, *Alternating Direction Methods on Multiprocessors : An Extended Abstract.,* Technical Report YALEU/DCS/RR-381, Computer Science Dept., Yale University, 1985.

[16] ———, *Data Communication in Hypercubes,* Technical Report , Computer Science Dept., Yale University, 1985. In preparation.

[17] ———, *Topological properties of hypercubes,* Technical Report YALEU/DCS/RR-389, Computer Science Dept., Yale University, 1985.

[18] A.H. Sameh, S. Chen, D Kuck, *Parallel Poisson and Biharmonic Solvers,* Computing, 17 (1976), pp. 219–230.

[19] A.H. Sameh, D.J. Kuck, *On Stable Parallel Linear System Solvers,* JACM, 25 (1978), pp. 81–91.

[20] C.L. Seitz, *The Cosmic Cube,* CACM, 28 (1985), pp. 22–33.

[21] R.S. Varga, *Matrix Iterative Analysis,* Prentice Hall, Englewood Cliffs, New Jersey, 1962.

[22] H.H. Wang, *A parallel method for tridiagonal equations,* ACM Trans. Math. Soft., 7 (1981), pp. 170–183.