Yale University
Department of Computer Science

Fast PDE Solvers on Fine and Medium Grain Architectures

S. Lennart Johnsson

YALEU/DCS/TR-583
April 1987

This work has been supported in part by the Office of Naval Research under Contracts N00014-84-K-0043 and N00014-86-K-0564. Approved for public release: distribution is unlimited.

Fast PDE Solvers on Fine and Medium Grain Architectures [1]

S. Lennart Johnsson

Departments of Computer Science
and Electrical Engineering
Yale University
New Haven, CT 06520
and

Thinking Machines Corp.
245 First Street
Cambridge, MA 02142

Abstract

Fast solvers for partial differential equations are often based on tridiagonal system solvers, the Fast Fourier Transform (FFT), and/or a combination thereof. We describe in some detail the implementation of tridiagonal system solvers and the FFT on a massively parallel architecture. The computational complexity of these two methods is compared, and some of the optimization issues that arise in medium scale architectures are discussed. Both odd-even cyclic reduction and the FFT for P equations requires $O(\log_2 P)$ steps and arithmetic time on a massively parallel architecture. The difference in arithmetic time is only a small constant factor. We derive this factor for real and complex systems. Odd-even cyclic reduction and the FFT have similar, but not identical communication topology. For the odd-even cyclic reduction algorithm it is equivalent to a reduction data manipulator network, whereas, that of the FFT is the familiar butterfly network. Communication time is dependent upon the ability of the communication system to support these two communication requirements with the data mapping being used. We derive the number of communications required by the two algorithms both for unlimited and limited buffer sizes, and the total number of element transfers. Measured times for implementations on the Connection Machine are presented.

1 Introduction

Currently, commercially available computers exhibit a wider variation in architecture than ever before. There exists almost every mix of pipelined or vector architectures, parallel architectures, shared bus architectures, and network architectures. With distributed storage for a few different networks, and a range of granularities from very

[1] In *Advances in Computer Methods for Partial Differential Equations - VI*, IMACS, June 1987.

powerful, as in the CRAY family of computers, to very simple processors, as in the Connection Machine [5].

Technology drives architectures toward increased concurrency for high performance, as conventional, fast, technologies are approaching their limits in speed [15,8]. Even if extreme performance is not the objective, technology is currently at a stage where a complete 32-bit microprocessor fits on a single die, and even a floating-point unit. Hence, there is an increasing number of systems available in which a moderate number of microprocessors, with or without floating-point hardware, are assembled into a system. Some are assembled into shared memory systems with a large standard operating system; others use a message passing operating system; and others are synchronous systems with additional data types that allows for the concurrent operation on sets of variables.

An overriding concern in designing fast computers is storage bandwidth. A high bandwidth requires a highly interleaved, or multiported storage system. The Connection Machine [5] illustrates this point in that its storage bandwidth is $32 Gbytes/sec$ at a clock rate of 4 MHz compared to $4 Gbytes/sec$ for the CRAY-2. The Connection Machine has $64k$ storage banks. For most computations, interprocessor communication bandwidth is also important (or intermemory bank bandwidth). However, this bandwidth need for many computations is lower than the need for storage to processor bandwidth in that many computations exhibit locality. For instance, for matrix multiplication with matrices of M elements each, $\sqrt{M}$ arithmetic operations (and local memory references) are performed per intermemory communication. In the case of the FFT, there are $\log_2 M$ local references per remote reference. For tridiagonal systems solvers substructuring can be applied [9,4] such that there are M local storage references and $\log_2 N$ remote references for N processors.

One of the most expensive resources in a computer system is the communication system, whether it be in the form of a fast bus or a switching network. Boolean cube and $\Omega-$network configured ensembles are two of the most commonly proposed (and used) configurations. These networks are more expensive to construct than, for instance, linear and two-dimensional arrays, and binary trees, but they can emulate a large number of communication patterns with little or no slowdown [7].

We consider two computations of interest for fast solvers for partial differential equations. Namely, the solution of multiple tridiagonal systems, and Fast Fourier Transforms. We confine this discussion to two-dimensional lattice problems. The generalization to higher dimensional lattices is straightforward. We consider odd-even cyclic reduction for the concurrent solution of tridiagonal systems of equations [12,10] which for a massively parallel system has the same order of complexity as the FFT, namely $O(\log_2 P)$ for P points. We derive detailed complexity estimates for both methods and describe briefly some of the optimization issues involved in solving the same problems with a significant amount of storage per node.

2 The Connection Machine

The Connection Machine has $64k$ processing elements each with its own storage. The configuration on which the experiments reported here were carried out did not have hardware support for floating-point operations. However, such an option is available. Without the floating-point option the peak arithmetic performance is 130 Megaflop/sec for the IEEE single precision format. The storage bandwidth is 32 Gbytes/sec. A processor chip contains 16 processors with a local interconnection network, and logic for bit-serial pipelined (interchip) routing. The routing logic attempts to resolve contention for communication channels should it occur. If the contention can be resolved, then one minor routing cycle (petit cycle [5]) will suffice; otherwise, a few such cycles may be required. Nearest neighbor communications are conflict free, whereas, random permutations cause conflicts. The interchip communication bandwidth is 24 Gbytes/sec, and the bandwidth for random permutations is $0.3 - 0.4$ Gbytes/sec. For communication on a two-dimensional lattice the bandwidth is 2.5 Gbytes/sec.

In order to maximize utilization of the communication capabilities of the architecture, it is desirable to use data mappings that minimize the number of collisions in the routing system. The Connection Machine processors are interconnected as a Boolean cube of 12 dimensions, and algorithms of the type described in [7], minimize collisions. It deserves to be stressed that in a bit-serial pipelined communication system such as the one on the Connection Machine, the routing time in the absence of collisions is proportional to the length of the message plus the address field plus a few control bits. The routing time is independent of distance, and the architecture can be viewed as a shared memory machine, in that access to the storage of any other processor is the same. Packet switched architectures do not have this property.

3 Tridiagonal Systems

3.1 Background

On a massively parallel architecture the conceptual programming model is one in which there is a sufficient number of processors for a naive identification of objects of the computation with processors. Objects of a computation are often hierarchically organized, conceptually. In lattice problems, there may be a set of different lattices. For each lattice there is the set of nodes and edges, and for each node, and/or edge, there is a set of variables. We consider the solution of partial differential equations on a two-dimensional lattice by tridiagonal system solvers, the FFT, and/or a combination thereof. We associate a processor with a node of the lattice. With an Alternating Direction Method for solving a partial differential equation of the following type

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + V(x,y,t)u \quad \text{in a rectangular region, } R;\ u = u(x,y,t). \tag{1}$$

with $u(x,y,0) = \phi(x,y)$ for $(x,y) \in R$, and $u(x,y,t) = \psi(x,y)$ for $(x,y) \in \partial R$.

the result is

$$\frac{du}{dt} = A_x u + B_y u + f(x,y,t)$$

where A_x is an approximation of $\frac{\partial^2}{\partial x^2}$ that for central differences includes the neighboring points in the x-direction. Similarly, B_y is an approximation of $\frac{\partial^2}{\partial y^2}$ that includes values at neighboring points in the y-direction. For the particular case of constant coefficients the matrices are symmetric Toeplitz, which can be used advantageously. Moreover, with rowwise ordering of the nodes of the lattice, the matrix A_x is block diagonal with the diagonal blocks being tridiagonal matrices. The matrix B_y also has three nonzero diagonals, but with the rowwise ordering, it is not "tridiagonal". However, with a permutation corresponding to a columnwise ordering B_y becomes a tridiagonal matrix (but A_x is no longer tridiagonal).

One ADM step for this equation consists of two half steps,

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

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

Each half step consists in solving a number of independent tridiagonal systems along rows or columns. In a massively parallel architecture it is natural to assign a node of the lattice to each processor, and to use a parallel algorithm such as odd-even cyclic reduction, or parallel cyclic reduction [6], for solving the tridiagonal systems. We have implemented both methods on the Connection Machine, and a medium grain architecture [11]. In the following, we assume that there are $P = 2^p - 1$ equations in each tridiagonal system, and that there are N processors in the system.

3.2 A lattice point per processor

With one node of the lattice per processor odd-even cyclic reduction [2,1] carried out in the straightforward way results in $17(p-2) + 14$ arithmetic operations, which can

be reduced to $16(p-2)+17$ operations by performing the scaling (division for the backsubstitution) between the forward and backward solves. A further reduction at the expense of slightly increased communication is achieved by distributing the elimination operations for one equation between two processors. This can be done by sending the subdiagonal element to be eliminated to the processor holding the equation used for the elimination, and sending the equation used for the elimination of a superdiagonal element to the processor holding that equation. Computation of elimination ratios and the multiplications for the super- and subdiagonal elements to be eliminated are performed concurrently. This requires 4 sequential arithmetic steps. The additions for the diagonal element and the right hand sides, requires an additional 4 arithmetic operations, if made in the processor holding the row on which elimination is performed. In backsubstitution, the two multiplications can be performed simultaneously, but the additions are serial. In total, $11(p-2)+17$ arithmetic steps are required. At the expense of additional communications this number can be reduced further to $9(p-2)+17$ operations by distributing the four additions in the reduction phase among 2 processors. All these estimates are for the case with real coefficients. Parallel cyclic reduction requires $12p$ arithmetic operations in sequence, and does not offer any advantage with respect to arithmetic for the case of P equations and $N = P$ processors.

If the tridiagonal systems are complex then the number of real arithmetic operations for odd-even cyclic reduction are reduced from $82(p-2)+84$ to $47(p-2)+61$ by the same technique as described above. This again can be reduced to $43(p-2)+57$ with additional communication. Parallel cyclic reduction requires $66p$ real operations. Hence, even though parallel cyclic reduction only requires p steps instead of $2(p-1)$ steps, it does require 40% more arithmetic operations in the complex case. In the symmetric Toeplitz case odd-even cyclic reduction can be implemented to yield $23(p-1)+const$ arithmetic operations, and parallel cyclic reduction will require $23p$ arithmetic operations.

Hence, in the event of a massively parallel architecture, odd-even cyclic reduction can be implemented to yield an arithmetic complexity that is lower in most instances. It is never higher than that of parallel cyclic reduction. However, the number of communication steps is higher than for parallel cyclic reduction, but the communication volume is lower. For the naive implementation of odd-even cyclic reduction a total of $4(2^p-p)$ interprocessor communications are needed in $2(p-1)$ steps. For parallel cyclic reduction $p2^{p+1}$ interprocessor communications are needed in p steps. The parallel cyclic reduction algorithm is much more communication intensive, and is faster than the odd-even cyclic reduction algorithm only if there are few conflicts in the communication system. The communication topology of odd-even cyclic reduction and parallel cyclic reduction are illustrated in Figures 1 and 2, respectively.

In the implementations of odd-even cyclic reduction with reduced arithmetic requirements there are two communications per reduction step. One step contains four floating-point numbers, and the other three numbers. In back substitution, there are two communications each containing one floating-point number. This communication

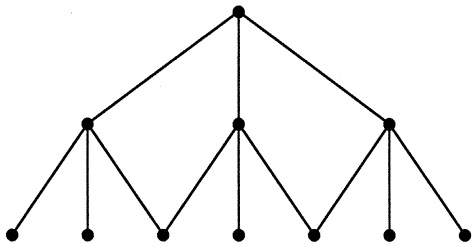

Figure 1: The communication topology of odd-even cyclic reduction (reduction data manipulator network).

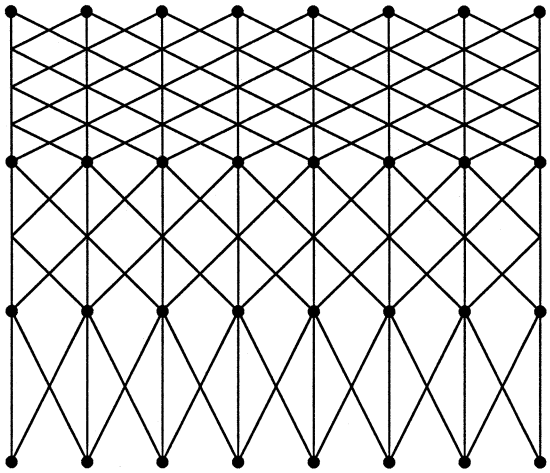

Figure 2: The communication topology of parallel cyclic reduction (data manipulator network).

Algorithm	# of Arithmetic op's	# of Comm.	# of Element transfers
Naive CR, real	$16(p-2)+17$	$4(p-1)$	$10(p-2)+6$
Split CR, version 1, real	$11(p-2)+17$	$4(p-1)$	$9(p-2)+6$
Split CR, version 2, real	$9(p-2)+17$	$6(p-1)$	$11(p-2)+6$
Parallel CR, real	$12(p-2)+17$	$2p$	$8(p-1)+3$
Naive CR, complex	$82(p-2)+84$	$4(p-1)$	$20(p-2)+12$
Split CR, version 1, complex	$47(p-2)+61$	$4(p-1)$	$18(p-2)+12$
Split CR, version 2, complex	$43(p-2)+57$	$6(p-1)$	$22(p-2)+12$
Parallel CR, complex	$66p$	$2p$	$16(p-1)+6$

Table 1: Complexity Estimates of Cyclic Reduction on Massively Parallel Architectures.

complexity results in $11(p-2)+17$ arithmetic operations in the real case. This number can be reduced to $10(p-2)+17$ operations by a few local moves and to $9(p-2)+17$ operations at the expense of two additional sends of single floating-point numbers. In the complex case the same communications are made, except every floating-point number is replaced by two numbers.

In Table 1 the stated number of communications is counted as the number of receives of the processor(s) with the maximum number of such operations.

We have implemented both odd-even cyclic reduction and parallel cyclic reduction on the Connection Machine. The processor utilization is discussed above. In order to minimize communication time, it is important to effectively emulate the communication topology of the algorithms on the network interconnecting the processing elements. It is well known that a path can be embedded in a Boolean cube preserving adjacency by using a Gray code, such as a binary-reflected Gray code [14,12,7]. Furthermore, if G_i is the binary-reflected Gray code of i, then $Hamming(G_i, G_{i+2^j}) = 2, \quad j > 0$ [9]. Moreover, the paths between i and $i+2^j$ for all i, j in the reduction data manipulator network (Figure 1) can be made edge disjoint [9]. This property is particularly important in a bit-serial pipelined communication system, since it means that there exist routing algorithms that yield collision-free communication for odd-even cyclic reduction. The Hamming distance between adjacent nodes at the various levels of the data manipulator network is of particular interest in packet switched networks, since the communication time is proportional to the number of edges to be routed, in the absence of conflicts (queues).

The Connection Machine processor chips are interconnected as a Boolean cube, with 16 processors with local storage per chip. With one equation per processor we let the two lowest order bits of the equation index be mapped on to the same chip (four for a one-dimensional partitioning of the lattice), and the remaining $p-2$ bits be mapped by a binary-reflected Gray code encoding on to a subcube of 2^{p-2} chips. After two reduction

steps there is only one active processor per system of equations per chip, i.e., there are four active processors per chip, but they represent equations for different tridiagonal systems. For subsequent reduction steps the number of active chips is reduced by a factor of two for each step. In parallel cyclic reduction, every processor is active for every reduction step. The number of off-chip communications (sends or receives) is 8 as long as the chip is active in odd-even cyclic reduction, but for parallel cyclic reduction it increases from 8 to 32 *exchanges* (sends *and* receives) during the first three reduction steps, and remains constant for the remainder of the algorithm.

With one connection between any pair of chips, the communication time for parallel cyclic reduction is expected to be greater than odd-even cyclic reduction for the reduction phase. Our measurements confirm the predicted ratio between the communication time for parallel cyclic reduction and odd-even cyclic reduction. The difference in communication time between the two methods is increasing with the problem size. The communication time for parallel cyclic reduction is greater than that of odd-even cyclic reduction, except for very small systems.

3.3 Multiple lattice points per processor

In the event that the machine size is smaller than the problem size, several nodes of the lattice must be mapped into the same processor. Two standard techniques being used are *cyclic* and *consecutive* partitioning [7], which effectively means that identification of nodes of the lattice with processors is made on the high, or low, order bits. In some computations the identification scheme is irrelevant with respect to performance. However, in others one may be significantly better than the other. In LU-decomposition of dense matrices, cyclic mapping yields higher processor utilization than consecutive mapping. For banded systems the difference in performance is in the range 1.5 - 2 [3]. However, for the solution of tridiagonal systems consecutive mapping is preferable, since substructuring can then be performed and no communication is necessary except for the boundary equations [10].

As the granularity, or the number of nodes of the lattice per processor increases, various trade-offs arise. Simplicity of the above approach is lost. In [11], we study the various trade-offs for a small to medium scale parallel processor. The methods considered are data transposition followed by Gaussian elimination; TGET (possibly preceded by a substructuring phase); *Balanced Cyclic Reduction* [10] (preceded by a substructuring phase); and a combination of the two methods. The advantage of not performing substructuring for the TGET algorithm is that the substructuring phase requires 17 arithmetic operations per equation, but Gaussian elimination only 8, a savings of 9 arithmetic operations per equation. The drawback is that all the data are subject to transposition, not only one equation per system of equations per processor. Balanced Cyclic Reduction is similar to parallel cyclic reduction, in that initially its communication topology is that of a data manipulator network. However, each system

is solved by odd-even cyclic reduction. The data manipulator network topology arises by performing elimination on odd equations for half of the systems, and on even equations for the other half of the systems of equations, recursively. If there are fewer systems than equations per system, then the first several steps correspond to the lower levels of the data manipulator network, and the upper levels to a reduction data manipulator network.

If there are few nodes of the lattice per processor, then a Balanced Cyclic Reduction yields better performance than the TGET, or the substructured TGET algorithm. As the granularity increases the fact that Gaussian elimination (without substructuring) requires approximately half of the number of arithmetic operations of cyclic reduction, changes the preference away from Balanced Cyclic Reduction. Note, however, that substructured elimination requires the same number of operations as odd-even cyclic reduction. The number of communication steps in the TGET approach is $4\log_2\sqrt{N}$ with exchanges counted as two communications, and the set of processors divided into $\sqrt{N}$ subsets of $\sqrt{N}$ processors each. The minimum number of communication steps in any implementation of the Balanced Cyclic Reduction algorithm we know of is $8\log_2\sqrt{N}$ [11]. However, if the data volume is large, i.e., $P \gg \sqrt{N}$, then in systems with limited packet sizes the number of communications for the TGET algorithm may still be higher than for Balanced Cyclic Reduction, since the data volume is reduced by a factor of 2 in every step of the latter, but remains constant in the TGET algorithm. Hence, with a maximum packet size of B bytes the number of communications in the TGET algorithm is $2(\lceil\frac{16P^2}{2NB}\rceil + \lceil\frac{4P^2}{2NB}\rceil)\log_2\sqrt{N}$ for real systems on a $P \times P$ lattice mapped consecutively on to a $\sqrt{N} \times \sqrt{N}$ processor grid. These expressions assume no substructuring. Substructuring reduces the data volume by a factor of $\frac{P}{\sqrt{N}}$. Details of the derivation can be found in [11]. For the Balanced Cyclic Reduction algorithm the approximate number of communications is $4\sum_{i=1}^{\log_2\sqrt{N}}(\lceil\frac{16P}{2^iB\sqrt{N}}\rceil + \lceil\frac{4P}{2^iB\sqrt{N}}\rceil)$. The total number of element transfers in a non-pipelined transpose algorithm is $2\log_2\sqrt{N}(\frac{16P^2}{2N} + \frac{4P^2}{2N})$. In a pipelined routing system the $\log_2\sqrt{N}$ factor becomes additive [7]. The number of element transfers in the Balanced Cyclic Reduction algorithm is $4\sum_{i=1}^{\log_2\sqrt{N}}(\lceil\frac{16P}{2^i\sqrt{N}}\rceil + \lceil\frac{4P}{2^i\sqrt{N}}\rceil)$. These estimates are all for real systems and are summarized in Table 2. For the estimates in the table it is also assumed that $\frac{P}{\sqrt{N}}$ is a multiple of $\sqrt{N}$ for simplicity in deriving estimates.

4 The Fast Fourier Transform

For some problems the use of the FFT and its inverse is an alternative solution technique to tridiagonal system solvers, like in the solution of Poisson's equation. In the massively parallel case with one lattice point per processor the 10 real operations per complex butterfly can be carried out in 5 arithmetic steps. Hence, a complex FFT requires $5p + const$ operations for $P = 2^p$ points. An FFT followed by an inverse FFT requires

Algorithm	# of arithmetic op's	# of Comm.	# of Element transfers
TGET	$\frac{P^2}{N}$	$2(\lceil\frac{16P^2}{2NB}\rceil + \lceil\frac{4P^2}{2NB}\rceil)\log_2\sqrt{N}\rceil$	$2\log_2\sqrt{N}(\frac{16P^2}{2N} + \frac{4P^2}{2N})$
TGET w. substr.	$17\frac{P^2}{N} + 8\frac{P}{\sqrt{N}}$	$2(\lceil\frac{16P}{2B\sqrt{N}}\rceil + \lceil\frac{4P}{2B\sqrt{N}}\rceil)\log_2\sqrt{N}\rceil$	$2\log_2\sqrt{N}(\frac{16P}{2\sqrt{N}} + \frac{4P}{2\sqrt{N}})$
BCR	$17\frac{P^2}{N} + 16\frac{P}{\sqrt{N}}$	$4\sum_{i=1}^{\log_2\sqrt{N}}(\lceil\frac{16P}{2^i\sqrt{N}}\rceil + \lceil\frac{4P}{2^i\sqrt{N}}\rceil)$	$4\sum_{i=1}^{\log_2\sqrt{N}}(\lceil\frac{16P}{2^i\sqrt{N}}\rceil + \lceil\frac{4P}{2^i\sqrt{N}}\rceil)$

Table 2: Estimated complexities for the solution of tridiagonal systems with multiple equations per processor.

Algorithm	# of Arithmetic op's	# of Comm.	# Element transfers
FFT & IFFT, real	$10p - const$	$4p$	$8p - const$
Split CR, v. 1, real	$11(p-2)+17$	$4(p-1)$	$9(p-1)$
FFT & IFFT, compl.	$10p$	$4p$	$8p$
Split CR, v 1, compl.	$47(p-2)+17$	$4(p-1)$	$18(p-1)$

Table 3: Comparing the complexities of FFT and tridiagonal solvers, one lattice node per processor.

approximately the same number of arithmetic operations as the tridiagonal solver for real data, but a factor of 2.3 to 4.7 fewer arithmetic operations in the complex case. Note, that in the massively parallel case the difference is a small constant factor in favor of the FFT, whereas in the sequential case the FFT requires approximately $5pP$ operations yielding a total of $10pP$ operations for an FFT and its inverse. A Gaussian elimination tridiagonal solver requires at most $8P$ operations.

The communication required by the FFT is that of a butterfly network, and that required by the odd-even cyclic reduction solver that of a reduction data manipulator network. The total amount of communication for odd-even cyclic reduction is less than for the FFT, but depending on the underlying network it still need not be faster than for the FFT. For instance, a Boolean cube configured network can emulate a normal [16] butterfly network computation with no slowdown, but a data manipulator network, or a reduction data manipulator network cannot be emulated entirely with nearest neighbor communications. Table 3 summarizes the complexities of an FFT and an inverse FFT for the real and complex case with one node of the lattice per processor. Table 4 gives the same data for the case of multiple nodes of the lattice per processor. The tables also includes estimates for tridiagonal system solvers for comparison.

On the Connection Machine, two stages of the butterfly computations are performed on chip (four in the one-dimensional case). Note, that a two dimensional FFT can be performed with the same data structures as a one dimensional FFT, but with different

Algorithm	# of Arithmetic op's	# of Comm.	# element transfers
FFT & IFFT, real	$10p - const$	$4p$	$8p - const$
BCR, real	$17\frac{P^2}{N} + 11(p-2) + 17$	$4\sum_{i=1}^{\log_2 \sqrt{N}}(\lceil \frac{16P}{2^i B\sqrt{N}} \rceil + \lceil \frac{4P}{2^i B\sqrt{N}} \rceil)$	$4\sum_{i=1}^{\log_2 \sqrt{N}}(\lceil \frac{16P}{2^i \sqrt{N}} \rceil + \lceil \frac{4P}{2^i \sqrt{N}} \rceil)$
FFT & IFFT, c.	$10\frac{P^2}{N}p$	$4\frac{P^2}{NB}(\frac{\log_2 P}{\log_2 \frac{P}{\sqrt{N}}} - 1)$	$8\frac{P^2}{NB}(\frac{\log_2 P}{\log_2 \frac{P}{\sqrt{N}}} - 1)$
BCR, c.	$84\frac{P^2}{N} + 47(p-2) + 17$	$4\sum_{i=1}^{\log_2 \sqrt{N}}(\lceil \frac{32P}{2^i B\sqrt{N}} \rceil + \lceil \frac{8P}{2^i B\sqrt{N}} \rceil)$	$4\sum_{i=1}^{\log_2 \sqrt{N}}(\lceil \frac{32P}{2^i \sqrt{N}} \rceil + \lceil \frac{8P}{2^i \sqrt{N}} \rceil)$

Table 4: Comparing the complexities of FFT and tridiagonal solvers on a massively parallel architecture.

twiddle factors. However, we assume the same lattice embedding as for the tridiagonal solvers discussed perviously. The butterfly stages requiring off-chip communication can be made more effective by either pipelining the different butterfly computations for a stage of the FFT, or by rotating the different butterflies of a stage, cube *skewing* [13]. The intent of both techniques is to avoid communication contention, since there is only one communication link between any pair of chips. Yet another possibility is to perform a permutation after an on-chip FFT, such that after the permutation a new same size on-chip FFT is performed.

On the Connection Machine our odd-even cyclic reduction solver for $P = \sqrt{N}$ and real coefficients is approximately 25% faster than our FFT plus IFFT, which are based on cube skewing. Communication time in the tridiagonal system solver is about 9% of the total time, and about 25% in our FFT. In the complex case our FFT plus IFFT is 3.5 times faster than our complex tridiagonal solver.

5 Summary

If there is a sufficient number of processors to map a lattice point to a distinct processor, then the arithmetic complexity of odd-even cyclic reduction can be made less than that of parallel cyclic reduction. Furthermore, the communication complexity of odd-even cyclic reduction is always lower. With multiple lattice point per processor the fact that parallel cyclic reduction is an algorithm of sequential arithmetic complexity of order $O(P \log_2 P)$ compared to $O(P)$ for odd-even cyclic reduction makes it clearly inferior. We conclude that unless communication overhead dominates the performance odd-even cyclic reduction is the method of choice.

In a massively or data-level parallel architecture, a real tridiagonal system solver based on odd-even cyclic reduction requires approximately the same number of arithmetic operations as an FFT followed by an inverse FFT. The arithmetic complexity of our implementations of a real tridiagonal solver for $P = 2^p - 1$ equations on P processors

is $9(p-2)+17$ to $11(p-2)+17$, where the implementation with higher complexity requires less communications. A leading arithmetic complexity term for the real FFT followed by an IFFT is $10p$. The communication requirements for the FFT are more extensive than those of odd-even cyclic reduction. Choosing an algorithm with respect to performance depends on the ability of the communication system to support the communication needs of the two algorithms. Differences in overhead are due to different control structures. For complex systems the leading term for the FFT does not change, but that of the odd-even cyclic reduction solver increases to $47(p-2)+61$. Arithmetic complexity of an FFT followed by an IFFT is lower by a constant factor. In the sequential case the FFT plus IFFT requires $5p$ operations per point for real data, compared to 8 for Gaussian elimination. For complex systems the comparison is $10p$ for FFT & IFFT and 44 for Gaussian elimination.

We conclude that in a highly concurrent architecture the computation of an FFT *and* its inverse, can be faster than the solution of tridiagonal systems by odd-even cyclic reduction, which has a complexity of minimal order for bounded degree networks. Measured times for the implementations of the various algorithms on the Connection Machine confirm the predictions.

References

[1] Billy L. Buzbee. A fast poisson solver amenable to parallel computation. *IEEE Trans. Computers*, C-22:793–796, 1973.

[2] Billy L. Buzbee, Gene H. Golub, and C W. Nielson. On direct methods for solving poisson's equations. *SIAM J. Numer. Anal.*, 7(4):627–656, December 1970.

[3] Jack Dongarra and S. Lennart Johnsson. Solving banded systems on a parallel processor. *Parallel Computing*, 5(1&2):219–246, 1987. (ANL/MCS-TM-85, November 1986).

[4] Dennis Gannon and John Van Rosendale. On the impact of communication complexity in the design of parallel numerical algorithms. *IEEE Trans. Computers*, C-33(12):1180–1194, December 1984.

[5] W. Daniel Hillis. *The Connection Machine.* MIT Press, 1985.

[6] Roger W. Hockney and C.R. Jesshope. *Parallel Computers.* Adam Hilger, 1981.

[7] S. Lennart Johnsson. Communication efficient basic linear algebra computations on hypercube architectures. *Journal of Parallel and Distributed Computing*, 4(2):133–172, April 1987. (Report YALEU/DCS/RR-361, January 1985).

[8] S. Lennart Johnsson. *Future High Performance Computation: The Megaflop per dollar alternative.* Technical Report YALEU/DCS/RR-360, Department of Computer Science, Yale University, January 1985.

[9] S. Lennart Johnsson. *Odd-Even Cyclic Reduction on Ensemble Architectures and the Solution Tridiagonal Systems of Equations.* Technical Report YALE/DCS/RR-339, Department of Computer Science, Yale University, October 1984.

[10] S. Lennart Johnsson. Solving tridiagonal systems on ensemble architectures. *SIAM J. Sci. Stat. Comp.*, 8(3):354–392, May 1987. (Report YALEU/DCS/RR-436, November 1985).

[11] S. Lennart Johnsson and Ching-Tien Ho. *Multiple tridiagonal systems, the Alternating Direction Method, and Boolean cube configured multiprocessors.* Technical Report YALEU/DCS/RR-532, Yale University, June 1987.

[12] S. Lennart Johnsson and Peggy Li. *Solutionset for AMA/CS 146.* Technical Report 5085:DF:83, California Institute of Technology, May 1983.

[13] Abhiram Ranade. *Bitonic Sort and Related Algorithms On the Connection Machine.* Technical Report YALEU/CSD/RR-, Yale University, Dept. of Computer Science, In preparation 1986.

[14] Edward M. Reingold, Jurg Nievergelt, and Narsingh Deo. *Combinatorial Algorithms.* Prentice Hall, 1977.

[15] Charles L. Seitz. Concurrent VLSI architectures. *IEEE Trans. Comp.*, C-33(12):1247–1265, 1984.

[16] Jeffrey D. Ullman. *Computational Aspects of VLSI.* Computer Sciences Press, 1984.