Yale University
Department of Computer Science

The FFT and Fast Poisson Solvers on Parallel Architectures

S. Lennart Johnsson

YALEU/DCS/TR-582
March 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.

The FFT and Fast Poisson Solvers on Parallel Architectures [1]

S. Lennart Johnsson
Departments of Computer Science
and Electrical Engineering
Yale University
New Haven, CT 06520[2]

Abstract

In this paper we highlight some of the issues that need to be addressed in finding efficient implementations of the conventional radix-2 and radix-4 Fast Fourier Transform algorithm [2] on parallel architectures. We consider the mapping problem for *Ensemble Architectures* [22], and the solution of Poisson's equation using FFT's. We also point out some properties of matrix transposition, which are particularly relevant for Boolean cube configured architectures and related networks. Finally, we comment on FFT in the context of submicron technology and wafer-scale integration.

Mapping of radix-2 and radix-4 FFT to *Ensemble Architectures*

The most important feature of the FFT with respect to parallel, distributed memory, architectures is the data interaction between the elements of the input sequence. The Cooley-Tukey radix-2 or radix-4 FFT factors this dependence into a sequence of $log_2 N$ or $log_4 N$ butterfly computations. The mapping of FFT computations to ensemble architectures, i.e., architectures with a large number of processing elements each with its own local memory, can be formulated as the embedding of the butterfly network in a graph representing the connectivity of the ensemble architecture. Since communication is the most expensive resource in an ensemble architecture, a typical objective in the mapping is to minimize the communication needs by preserving proximity. Encoding one structure into another is an embedding problem. A total of $log_2 N + loglog_2 N$ bits are required to encode the nodes of the butterfly network. Let the address be of the form $(x_{n-1}x_{n-2}\ldots x_0|y_{m-1}\ldots y_0)$, where $n = log_2 N$ and $m = loglog_2 N$. Hence, x encodes the number of items and y the ranks of the butterfly network.

We have derived optimal implementations of the conventional radix-2 and radix-4 FFT algorithms on a variety of distributed memory architectures. In [12,1,11,17] we formally derive various implementations of the FFT on **linear arrays** by using the laws of associativity and distributivity. The implementations are optimal with respect to the storage required, and the number of arithmetic units. The steps can be mechanized, and there exists a direct correspondence between expressions in the formalism and objects in the space-time domain. This work forms the basis for compiling FFT computations on to linear arrays. In the linear array case x is mapped into time and y into space, which results in an array of $logN$ stages operating in time N for a transform of size N. Two FFT's can be pipelined and the total storage requirement is $N-1$.

For the Caltech Tree-Machine project we also investigated the computation of the FFT on complete **binary trees** with data entering and leaving through the root. Hence, input and output are linear in time, and in the limit $N-1$ nodes are used to compute the FFT in time

[1] Abstract of a presentation at the workshop *Fast Fourier Transforms for Vector and Parallel Computers*, The Mathematical Sciences Institute, Cornell University, March 1987.

[2] Currently on leave at Thinking Machines Corp. 245 First Street, Cambridge, MA 02142

N, which is clearly suboptimal.

In the Caltech Cosmic Cube project we studied the mapping of FFT computations on to **Boolean cubes** [15]. The obvious mapping is the complement of the linear array mapping; x is mapped into space and y into time. This mapping can use N processors and operate in time $logN$. A similar mapping was used in [23] for the perfect-shuffle network. Note, that a Boolean cube of N processors can compute one FFT every log_2N time steps, whereas, the butterfly network has a *latency* of log_2N, but it can deliver one FFT every cycle thereafter through pipelining. Pipelining is trivial, since the FFT is a so called *normal* algorithm [24].

The **Cube Connected Cycles** network has been proposed by Preparata and Vuilleman [19]. It can compute an FFT on $Nlog_2N$ points in $2log_N - 1 + loglog_2N$ steps. The CCC network is effectively a butterfly network in which the nodes with the same x address are connected as a loop, i.e., node $(x|y)$ is connected to nodes $(x|(y-1)modm))$ and $(x|(y+1)modm))$ and nodes 0 and log_N are identified. Connections in different ranks of the CCC network are in different dimensions. The CCC FFT first (last) computes log_2N FFT's on N points through pipelining and intercycle communication, followed (preceeded) by $loglog_2N$ steps of intracycle computations for a total of N FFT:s on log_2N points. Note, that if the nodes in a cycle are identified (the cycles collapsed into a point), then the CCC becomes identical to a Boolean cube of N nodes.

Implementation Experience

We have implemented FFT's on the Alliant FX/8, the Intel iPSC, and the Connection Machine. The Alliant FX/8 is a shared memory machine with 8 processors, each having its own instruction stream. Processors also have vector features with a maximum vector length of 32. The Intel iPSC is a Boolean cube configured ensemble architecture in which each processor has its own storage and instruction stream. It is designed for up to 128 nodes, which can be provided with vector features. Synchronization and data interchange take place through message passing. The Connection Machine is a massively parallel architecture with up to $64k$ nodes sharing a common instruction stream. The $64k$ processor configuration consists of $4k$ chips interconnected as a Boolean cube with 16 processors per chip. Each processor has its own storage. There is an interconnection network on chip for the 16 processors making the Connection Machine resemble a CCC network, but the Connection Machine has a higher connectivity per chip than a cycle.

With more data points than processors several items have to be identified with the same processor. We refer to the *identification*, or *aggregation*, on the high order bits as *cyclic* mapping since data items $x(i)$ and $x(j)$ are identified if $i \equiv jmodP$ for P processors. Identification on the low order bits is refered to as *consecutive* mapping. Consecutive elements in the linear ordering are mapped into the same processor [6]. Either scheme for identification can be used for the FFT, however, in the context of fast Poisson solvers the consecutive scheme results in lower communication complexity [8].

For the **Alliant FX/8**, the loop interchange order suggested by Petersen [18] offered a siginifcant speed-up over the constant loop ordering. (The Petersen algorithm guarantees a vector length of size N/2 or greater). Loop unrolling helps increase the performance, as well as paying attention to some idiosyncrasy in the compiler in computing array indices. A performance improvement of approximately 50% is the result of using these techniques.

For the **Intel iPSC**, and early versions of its operating system with significant overhead in communication, a maximum size hypercube is not always optimal. We will discuss this issue in the context of fast Poisson solvers.

For the **Connection Machine**, the scheme outlined for the CCC can be used. An alternative is to make use of the router, which provides bit-serial, pipelined communication. We note that

in the CCC, a FFT is computed by performing (cyclic) shifts in the cycles such that after $log_2 N$ shifts an FFT on N points is computed. If the inter-cycle dimensions are labeled $0, 1, \ldots, n$ as encountered for one of the FFT's of size N, then the dimensions for the other $log_2 N - 1$ FFT's of size N are encountered in cyclicly rotated orders. Hence, with the proper data allocation (cube skewing) all $log_2 N$ FFT's can be performed concurrently, instead of pipelined. However, a data reallocation is needed for the intracycle FFT's. This data permutation can be performed by the router, and is an inexpensive operation on the Connection Machine [7].

Fast Poisson Solvers

Fast solvers for Poisson's equation in two dimensions use the FFT in two dimensions; or the FFT may be used in one dimension and a tridiagonal system solver in the other dimension.

We first note that in a **massively parallel** architecture with N^2 grid points and N^2 processors a full complex FFT can be performed with $5log_2 N$ arithmetic operations [15,7], giving a total of $10log_2 N$ operations for an FFT and an inverse FFT. A tridiagonal solver on N points can be carried out with at most $11log_2 N$ arithmetic operations [9,10]. Indeed, the number of arithmetic operations can be reduced to $9log_2 N$ at the expense of additional communication. This further reduction does not pay off on the Connection Machine. The sequential arithmetic complexity of the FFT is approximately $5Nlog_2 N$, and that of a tridiagonal system solver $8N$. Hence, in a massively parallel system the time for arithmetic is approximately the same for the FFT and the tridiagonal system solver. Communication complexity is the distinguishing feature. For odd-even cyclic reduction the communication requirement is less than for the FFT, which in turn is less than for Paralell Cyclic Reduction [5,10].

With **medium scale parallelism**, such as is the case for the Intel iPSC several tridiagonal systems are allocated to a subcube. The systems can be solved by a matrix transpose followed by local Gaussian elimination and another transpose operation of the result. Gaussian elimination requires about half the number of arithmetic operations of odd-even cyclic reduction. Another alternative is to use substructuring followed by a transpose operation, local solve, a transpose, and local backsubstitution. Substructuring reduces the data volume being transposed at the expense of increased arithmetic complexity. The transpose can be avoided by using cyclic reduction, which for load balancing purposes is carried out as *Balanced Cyclic reduction* [9]. Finally, the substructuring and balanced cyclic reduction scheme can be combined with the transpose and local elimination scheme to yield a hybrid scheme. These alternatives have been implemented and are analyzed in [21,13].

Finally, we note that in the context of Poisson's equation some of the equations are strongly diagonally dominant. Hence, some of the systems are diagonalized already during the substructuring phase. Others are diagonalized after a few steps of cyclic reduction [16].

Matrix Transpose

An integral part of many computations is the matrix transpose operation which has been disclosed above. The transposition of a matrix can be performed on butterfly networks as described in [3,23]. In [6] we show that the paths of elements from different nodes are edge disjoint, and in [4] we give an optimum transpose algorithm for matrices partitioned in both dimensions. Optimality of the exchange algorithm is proved in [14].

In conclusion we notice that for course grained systems, i.e., for systems with a relatively large amount of storage per node, meshes are more effective than Boolean cubes or butterfly networks themselves, to accomodate the data movement required by butterfly type algorithms like the FFT [20].

References

[1] Danny Cohen and S. Lennart Johnsson. Mathematical approach to computational networks. In *Proc. IEEE International Conference on Computer Design: VLSI in Computers*, pages 642 –646, IEEE Computer Society, 1983.

[2] Jim C. Cooley, P.A.W. Tukey, and P.D. Welch. The fast fourier transform algorithm: progamming considerations in the calcualtion of the sine, cosine and laplace transforms. *J. Sound Vibrations*, 12(3):315–337, 1970.

[3] J.O. Eklundh. A fast computer method for matrix transposing. *IEEE Trans. Computers*, C-21(7):801–803, 1972.

[4] Ching-Tien Ho and S. Lennart Johnsson. *Matrix Transposition on Boolean n-cube Configured Ensemble Architectures.* Technical Report YALEU/DCS/RR-494, Yale University, Dept. of Computer Science, September 1986.

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

[6] 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).

[7] S. Lennart Johnsson. *The Fast Fourier Transform, Sine and Cosine Transforms on the Connection Machine.* Technical Report , Thinking Machines Corp., In preparation 1987.

[8] 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.

[9] 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).

[10] S. Lennart Johnsson. *Solving tridiagonal systems on the Connection Machine.* Technical Report , Thinking Machines Corp., In preparation 1987.

[11] S. Lennart Johnsson and Danny Cohen. An algebraic description of array implementations of fft algorithms. In *20th Allerton Conference on Communication, Control, and Computing*, Electrical Engineering, University of Illinois, Urbana/Champaign, 1982.

[12] S. Lennart Johnsson and Danny Cohen. *Mathematical Approach to Computational Networks for the Discrete Fourier Transform.* Technical Report, Department of Computer Science, Yale University, 1984.

[13] 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.

[14] S. Lennart Johnsson and Ching-Tien Ho. *Spanning Graphs for Optimum Broadcasting and Personalized Communication in Hypercubes.* Technical Report YALEU/DCS/RR-500, Yale University, Dept. of Computer Science, November 1986. To appear in IEEE Trans. Computers.

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

[16] S. Lennart Johnsson and Faisal Saied. *Convergence of substructuring in Poisson's equation.* Technical Report , Yale University, In preparation 1987.

[17] S. Lennart Johnsson, Uri Weiser, Danny Cohen, and Al Davis. Towards a formal treatment of vlsi arrays. In *Proceedings of the Second Caltech Conference on VLSI*, pages 375 – 398, Caltech Computer Science Department, January 1981.

[18] W.P. Petersen. Vector fortran for numerical problems on cray-1. *Communications of the ACM*, 26(11):1008–1021, November 1983.

[19] Franco P. Preparata and J.E. Vuillemin. The cube connected cycles: a versatile network for parallel computation. In *Proc. Twentieth Annual IEEE Symposium on Foundations of Computer Science*, pages 140–147, 1979.

[20] Abhiram Ranade and S. Lennart Johnsson. The communication efficiency of meshes, boolean cubes, and cube connected cycles for wafer scale integration. In *Int. Conf. on Parallel Processing*, pages 479–482, IEEE Computer Society, 1987.

[21] Faisal Saied, Ching-Tien Ho, S. Lennart Johnsson, and Martin H. Schultz. Solving schroedinger's equation on the intel ipsc by the alternating direction method. In *Hypercube Multiprocessors 1987*, pages 680–691, SIAM, September 1986. Tech. report YALEU/DCS/RR-502, January 1987.

[22] Charles L. Seitz. Ensemble architectures for vlsi – a survey and taxonomy. In P. Penfield Jr., editor, *1982 Conf on Advanced Research in VLSI*, pages 130 – 135, Artech House, January 1982.

[23] Harold S. Stone. Parallel processing with the perfect shuffle. *IEEE Trans. Computers*, C-20:153–161, 1971.

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