SOLVING ELLIPTIC PROBLEMS ON AN ARRAY PROCESSOR SYSTEM**

Martin H. Schultz

Research Report YALEU/DCS/RR-272
June 1983

** To appear in Elliptic Problem Solvers II. (eds.) G. Birkhoff & A. Schoenstadt. Academic Press, Inc.

SOLVING ELLIPTIC PROBLEMS ON AN ARRAY PROCESSOR SYSTEM*

Martin H. Schultz+

Department of Computer Science
Yale University
New Haven, Connecticut

I. INTRODUCTION

Elliptic problems are not only very important in their own right, but often occur as the compute-intensive inner-most loop of large-scale scientific computations. It is thus incumbant upon us to fully understand the algorithm-software-architecture issues for such problems. In the Proceedings of the 1980 meeting on Elliptic Problem Solvers [4], we gave a hypothetical discussion of the application of array processor systems to solving elliptic problems. While such systems have been very successful in cost-effective signal processing, it is by no means obvious that they would be effective in this application. In this paper we continue our discussion (on a less hypothetical plane) summarizing our progress to date and describing potential future developments.

* *This work was supported in part by ONR Grant #N00014-82-K-0184, NSF Grant #MCS-8106181 and by External Research Grants from Digital Equipment Corp. and Floating Point Systems.*

+ *Research Center for Scientific Computation, Department of Computer Science, Yale University.*

In Section II, we discuss array processor systems and our experience in using one, the Floating Point Systems AP-164, to solve the Poisson equation. We discuss three standard algorithms based on FFTs, their implementation and algorithmic complexity on the AP-164, and numerical experiments. We show that for these algorithms the AP-164 is essentially as fast as published benchmarks for the CDC-7600.

In Section III, we consider future developments for very large three-dimensional problems. In particular, we consider the effect of using secondary storage. We compare three typical systems: (1) the AP-164 with a disk; (2) the Cray-1 with a disk; and (3) the AP-164 with a bulk memory system under development. We show that the AP-164 with bulk memory system is the fastest of the three systems for three dimensional problem with less than one billion unknowns. Furthermore, we show how sufficiently fast FFT devices could be integrated into the system to produce an order of magnitude speed-up. Finally we will briefly discuss the ELI-512, a new array processor being designed at Yale, cf.,[5] for more details

II. ELLIPTIC PROBLEM SOLVERS

It has been observed for several years that Very Long Instruction Word (VLIW) computers and specialized processors are especially cost effective for signal processing. We have been seeking to determine their effectiveness for solving elliptic problems. It has also been observed that the programming effort for these machines is significantly greater in general than for serial computers and we will address that issue.

What are VLIW computers? They are computers with three basic properties:

(1) there is one central control unit issuing a single wide instruction per cycle;

(2) each wide instruction consists of many coupled independent operations; and

(3) each operation requires a small, statically predictable number of cycles to execute (operations may be pipelined). The Floating Point Systems AP-164 is the example of such a machine which we will consider in detail. The ELI-512 is another example.

If we are going the route of specilized processors, one may ask why we bother with VLIW computers at all. Our argument is very simple. Suppose on the one hand that we could build a specialized processor that would do 80% of a large scientific computation with unlimited parallelism, i.e., it would do 80% of the computation in zero time. Then the overall computing time would be reduced by a factor of 5. On the other hand, suppose we could find an average 10-fold parallelism over the whole computation for a VLIW computer. Then we could get a speed-up of a factor of 10 on a VLIW computer which is clearly better. However, combining the best of both approaches would be even better. In fact, we used a specialized processor on 80% of the computation and a VLIW machine on the remaining 20%, we could get a reduction of a factor of 50. In Section III, we explore the use of specialized FFT processors in conjunction with the AP-164 for solving the Poisson equation in three dimensions.

As a proof of feasability, we consider solving the Poisson equation on the AP-164. We made this choice for a number of reasons:

(1) the AP-164 is a commercially available, "off-the-shelf" VLIW computer;

(2) the Poisson equation is the simplest nontrivial elliptic equation and if we can't solve it fast then we can't solve anything fast; and

(3) the Poisson equation occurs in many important applications such as image enhancement, geoacoustics, semiconductor device simulation, fluid mechanics, and mechanical CAD/CAM.

As a specific example, we mention work we are doing with Dr. Ding Lee of NUSC on underwater acoustics. We are investigating the use of a fast Poisson solver as a preconditioner to be used in conjunction with the conjugate gradient method (applied to the preconditioned normal equations), for solving discrete approximations to nonseperable Helmholtz equations. We have been able to prove that this algorithm requires at most $\log n$ iterations to reduce the error in a difference approximation with n grid points in each coordinate by a factor of n^{-2}.

This brings us to the question of how well a VLIW computer like the AP-164 will solve the Poisson equation. In order to put our discussion into context we need to know the following information about the AP-164:

(1) the word length of the AP-164 is 64 bits;

(2) the cycle time is 167 nanoseconds;

(3) 10 instructions can be executed per cycle, including a floating point multiplication and addition;

(4) floating point multiplications and additions are pipelined (requiring 3 and 2 cycles respectively;

(5) a division requires 22 cycles, but vector division requires 6 cycles per component;

(6) the machine can be either microprogrammed in APAL or programmed in FORTRAN; and

(7) there exists a good scientific APAL subroutine library.

The model discrete problem we wish to solve is the standard five point finite difference approximation to the Poisson equation, $-\Delta u=f$, on the unit square, i.e.,

$$(1) \qquad -\Delta_h u_{ik} = f_{ik} \equiv f(x_i,y_k) \quad , \quad 1\le i,k\le n,$$

subject to the boundary conditions

$$(2) \qquad u_{0k} = u_{n+1,k} = u_{i0} = u_{i,n+1} = 0 \quad , \quad 1\le i,k\le n,$$

where $n = 2^t-1$ and $\Delta_h u_{ik} = h^{-2}[-4u_{ik} + u_{i-1,k} + u_{i+1,k} + u_{i,k-1} + u_{i,k+1}]$.

Problem (1)-(2) can be written in matrix form as

$$(3) \qquad M_h \underset{\sim}{u}_h = \underset{\sim}{f}_n$$

where

$$(4) \qquad M_h \equiv h^{-2} \begin{bmatrix} 4 & -1 & & & -1 & 0 \\ -1 & & \ddots & & & -1 \\ & \ddots & & & 0 & \\ & & & & \ddots & \\ -1 & 0 & & & & -1 \\ 0 & -1 & & -1 & & 4 \end{bmatrix}$$

Since array processors are excellent at doing FFTs, we consider three standard FFT based algorithms for solving the discrete Poisson equation (1)-(2), cf., [1],[3],[6],[7],[9] and [10]. The basic idea of all these methods is to extend $\underset{\sim}{u} \equiv \{u_{ik}\}$ and $\underset{\sim}{f} \equiv \{f_{ik}\}$ to real, odd, periodic sequences of period $2(n+1)$ in both dimensions.

If $\underset{\sim}{d} \equiv \{d_{ik}\}$ is the real, doubly periodic sequence of period $2(n+1)$ with

$$d_{0,0} = 4 \ , \ d_{-1,0} = d_{1,0} = d_{0,-1} = d_{0,1} = -1 \ , \text{ and}$$

$$d_{ik} = 0 \quad \text{otherwise,}$$

then the discrete Poisson equation (1)-(2) can be represented as $\underset{\sim}{d} * \underset{\sim}{u} = h^2\underset{\sim}{f}$ or $\hat{\underset{\sim}{d}} \cdot \hat{\underset{\sim}{u}} = h^2\hat{\underset{\sim}{f}}$ or

$$(5) \qquad \hat{u}_{ik} = h^2 \, \hat{d}_{ik}^{-1} \, \hat{f}_{ik} , \qquad 1 \le i,k \le n,$$

where $\hat{z}$ denotes the discrete, double, complex Fourier transform of z. We note that in this example the real Fourier transform or sine transform would suffice, cf.,[2]. We will return to the impact of this observation later. The solution to (1)-(2) is obtained by taking the inverse transform of the normalized vector $\hat{u}$ of (5). We refer to this method as the (double) FOURIER method.

It is also possible to view the discrete Poisson equations as being block tridiagonal, i.e., if we partition the solution vector $\underset{\sim}{u}$ as

$$\underset{\sim}{u} \equiv \begin{bmatrix} \underset{\sim}{u}_1 \\ \underset{\sim}{u}_2 \\ \vdots \\ \underset{\sim}{u}_n \end{bmatrix} , \qquad \text{where } \underset{\sim}{u}_i \equiv \begin{bmatrix} u_{i1} \\ u_{i2} \\ \vdots \\ u_{in} \end{bmatrix} , \qquad \text{then}$$

$$M_n = \begin{bmatrix} A & I & & 0 \\ I & A & I & \\ & \ddots & \ddots & \ddots \\ & & \ddots & I \\ 0 & & I & A \end{bmatrix} , \qquad \text{where} \qquad A \qquad \begin{bmatrix} -4 & 1 & & 0 \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ 0 & & 1 & -4 \end{bmatrix}$$

has the eigenvalues $\lambda_k = -4 + 2\cos\left(\frac{k\pi}{n+1}\right)$ and

eigenvectors $\underset{\sim}{V}_k \equiv \left[V_{kj}\right] \equiv \left[\frac{1}{\sqrt{n+1}} \sin\left(\frac{jk\pi}{n+1}\right)\right]$.

If $\underset{\sim}{u}_i = \Sigma \, \hat{u}_{ij} \underset{\sim}{v}_j$, then

$$A\underset{\sim}{u}_1 + \underset{\sim}{u}_2 = \underset{\sim}{y}_1,$$

$$\underset{\sim}{u}_{i-1} + A\underset{\sim}{u}_i + \underset{\sim}{u}_{i+1} = \underset{\sim}{y}_i , \qquad 2 \le i \le n-1,$$

$$\underset{\sim}{u}_{n-1} + A\underset{\sim}{u}_n = \underset{\sim}{y}_n .$$

or for each component $1 \le k \le n$

$$\lambda_k \hat{u}_{1,k} + \hat{u}_{2,k} = \hat{y}_{1,k}$$

$$\hat{u}_{i-1,k} + \lambda_k \hat{u}_{i,k} + \hat{u}_{i+1,k} = \hat{y}_{i,k} \quad , \quad 2 \le i \le n-1, \tag{6}$$

$$\hat{u}_{n-1,k} + \lambda_k \hat{u}_{n,k} = \hat{y}_{n,k} \ ,$$

which is an easily solved tridiagonal system. Once we have solved for all the transformed vectors $\{\hat{u}_i\}$, we obtain the solution by doing n one dimensional inverse transforms. We call this the FOURIER/TRID method.

The third method which we call the ODD/EVEN method consists of first doing an odd/even reduction, cf.,[3] and [9], to eliminate all the unknowns along vertical lines with even index and then applying the FOURIER/TRID method to the remaining unknowns. Table I summarizes the asymptotic complexity of these three methods.

TABLE I

	FOURIER	FOURIER/TRID	ODD/EVEN
# transforms of length 2n+1*	4n	2n	n
# tridiagonal systems of order n**	0	n	n/2
# tridiagonal systems of order n/2**	0	0	n

* requires 0(nlogn) work

** requires 0(n) work

It is clear that on a serial computer that the ODD/EVEN method is asymptotically fastest and the FOURIER/TRID method is asymptotically second fastest. On a vector machine, the hierarchy is not so clear. It would seem that if we could do FFTs sufficiently fast it wouldn't pay to trade them off for tridiagonal solves which are basically serial. In addition, there are a variety of transforms which may be employed. As shown in [2] , the ratio of work for the transforms is given in Table II.

TABLE II

complex FFT	1
real FFT*	1/2
sine transform*	1/4

* requires low order pre- and post- processing.

On a serial machine, it is clear that we would choose the sine transform. On a VLIW machine, the choice may not be so obvious because an optimal microcoded implementation can be so much faster than a compiled one. The AP-164 has very efficient complex and real FFTs in the APAL subroutine library while it does not have a comparable sine transform at present. A sine transform on the AP-164 must be done by doing the pre- and post- processing [2] in FORTRAN and calling the APAL real transform for the inner-loop.

Vectorization is not the only important factor in algorithm selection for a machine like the AP-164 which has

a very slow division (especially scalar) operation. In the FOURIER/TRID and ODD/EVEN methods we must solve linear systems with tridiagonal matrices of the form

$$B \equiv \begin{bmatrix} \lambda & 1 & & 0 \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ 0 & & 1 & \lambda \end{bmatrix} \quad \text{where} \quad |\lambda| > 2.$$

It would seem that this would involve quite a few divisions.

Following [6] and [8] we can factor

$$B = \begin{bmatrix} 1 & & 0 \\ \ell_1 & \ddots & \\ & \ddots & 1 \\ 0 & \ell_{n-1} & \end{bmatrix} \begin{bmatrix} u_1 & 1 & 0 \\ & \ddots & 1 \\ 0 & & u_n \end{bmatrix} \equiv LU$$

where $u_i^{-1} = \ell_i$, $1 \le i \le n-1$. Moreover, $\ell_i \to \ell$ as $i \to \infty$, which means we can avoid most of the divisions and storage in calculating and using the LU-factorization. If we are in the regime where we are solving several Poisson equations, then we can view the factorization as preprocessing and we need do only the forward and backward substitution. Moreover, since $u_i^{-1} = \ell_i$, we need store only L and forward and backward substitution can be carried out without any divisions.

There are a number of important vectorization "tricks" or techniques which can be employed on these problems, cf. [1], [6], and [10]:

(1) The normalization in the FOURIER method can be vectorized. In fact, if we are solving at least two Poisson equations, it pays to form the reciprocals of the normalization constants by means of a vector divide and then to do a normalization by means of a vector multiplication.

(2) In the FOURIER/TRID and ODD/EVEN methods, the collection of tridiagonal systems can be solved in parallel. Thus on the AP-164, the time required to solve the tridiagonal systems is about twice the time required to do the

normalization assuming preprocessing of the normalization constants and it probably requres less time assuming no preprocessing.

(3) The reduction step in the ODD/EVEN method can be done in parallel.

(4) The pre- and post- processing necessary to compute all the sine transforms, given the existence of a real FFT, can be vectorized. Thus on the AP-164, we should not have to pay too large a penalty for not having a hand-optimized APAL sine transform code.

These points have some implications for our algorithm selection. Points (2) and (3) imply that the FOURIER method is very unlikely to be the most efficient on the AP-164 no matter how fast we can do an FFT. Point (4) implies that we should use the vectorized sine transform whenever possible even if it isn't implemented entirely in APAL.

We have run a large number of experiments for these methods on the AP-164. The results are too voluminous to reproduce here and we refer to [6]. We do reproduce one interesting graph (Figure I) showing the predicted running times of these algorithms based on a sine transform hand-coded in APAL. We approximated the running time of such a transform by taking the product of the running time of the APAL real FFT and the quotient of the running time of the FORTRAN sine transform divided by the running time of the FORTRAN real FFT. The curve labeled D is the FOURIER method without preprocessing or vectorized normalization, the curve labeled T is the FOURIER/TRID method using the fact that the entries in the factors converge but without preprocessing or vectorization, and the curve labeled R is the ODD/EVEN method without preprocessing or vectorization. Curves 1 and 2 from [10] are for algorithms similar to those in T and R except implemented on a CDC-7600.

Since we didn't exhaust our possible improvements in the benchmarks of Figure I, it seems quite clear that even if we were slightly optimistic on the running time of our sine transform, Poisson solvers on a AP-164 are competetive with comparable solvers on a CDC-7600. This proves our contention that we can effectively solve nontrivial elliptic problems on a VLIW computer.

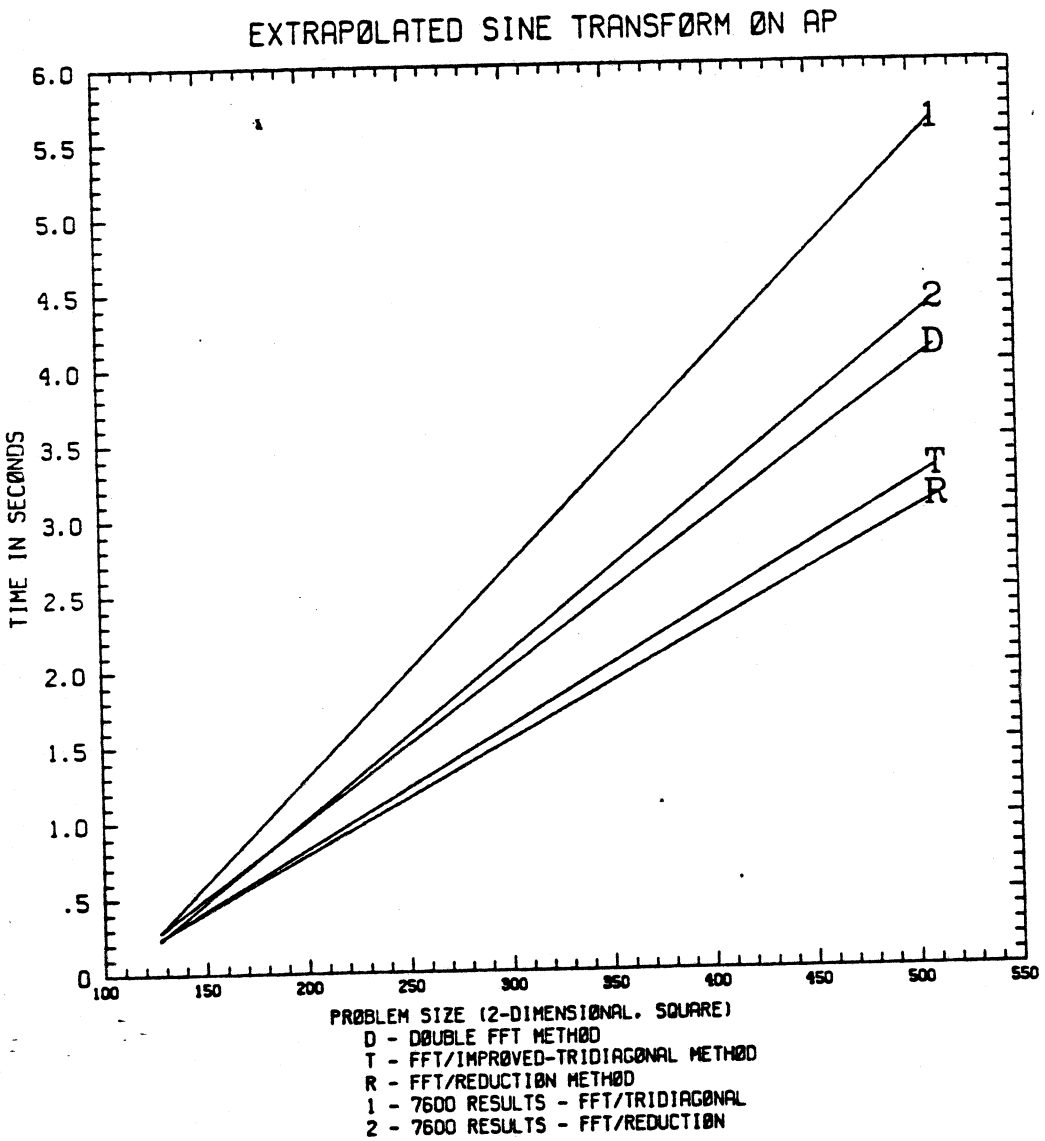

FIGURE I

III. VERY LARGE PROBLEMS

We now switch our attention to solving very large three dimensional Poisson problems. To get a handle on things we consider a simple out-of-core method based on the FOURIER method in three dimensions. Our goal is to analyze the impact of different architectures on the running time of a fixed algorithm (as distinct from Section II in which we fixed the architecture and varied the algorithm).

In this section, we assume that the source term, i.e., the right-hand side of the equation, is initially stored as a 3-dimensional block array is secondary storage (so far unspecified)

OUT-OF-CORE ALGORITHM:

```
FOR EACH DIMENSION:
     BRING EVERY BLOCK ROW INTO MAIN MEMORY
     DO 1-DIMENSIONAL FFTs
     STORE TRANSFORMS BACK IN SECONDARY STORAGE

FOR EACH DIMENSION:
     BRING EVERY TRANSFORMED BLOCK ROW INTO MAIN MEMORY
     IF DIMENSION=1:
          BRING NORMALIZATION CONSTANTS INTO MAIN
          MEMORY and NORMALIZE
     DO 1-D INVERSE FFTs
     STORE ANSWER BACK IN SECONDARY STORAGE
```

The required I/O for the source, solution, and normalization constants is $13n^3$ words. We consider the following three systems:

(1) a AP-164 with a disk having a transfer rate of 1.2 megabytes/sec;

(2) a Cray-1 with a disk having a transfer rate of 5 megabytes/sec; and

(3) a AP-164 with a bulk memory having a transfer rate of 48 megabytes/sec.

To simplify the analysis and favor the disks as much as possible we ignore latency. As an aside we mention that disk latency is on the order of milliseconds, while bulk memory latency is on the order of microseconds.

The I/O time in microsecond required for this algorithm on systems (1)-(3) is given by the following estimates: (1) $87n^3$; (2) $21n^3$; and (3) $2.2n^3$. The compute time on the AP-164 required for this algorithm is asymptotically equal to the time to do n^3 multiplications and $6n^2$ 1-dimensional FFTs which is approximately $2n^3\log n$ microseconds.

Comparing these numbers, we can reach several conclusions:

(1) the relative times needed for I/O on systems (1)-(3) are approximately 40:10:1 ;

(2) if $n \leq 2^{11}$, disk I/O on the Cray-1 takes more time than I/O and computation on the AP-164 with bulk memory; and

(3) a AP-164 with a disk takes about $1+(44/\log n)$ times longer than a AP-164 with a bulk memory to solve this problem. If $n=2^7$, this works out to about 7 times longer.

It is interesting to note that this algorithm is one of the most favorable for systems for slow I/O. Most iterative methods for solving elliptic problems involve work proportional to the amount of data (and not a higher power of the amount of data) for every sweep through the data. For these algorithms, data movement is as important asymptotically as floating point computation. In this sense, solving elliptic

problems is quite different than combinatorial computing for which the number of operations often grows superlinearly (or even expoentially) with the amount of data.

The time and memory requirements for solving the 3-dimensional Poisson equation for $n=2^7, 2^8$, and 2^{10} by the FOURIER OUT-OF-CORE method on the AP-164 with bulk memory are given in Table III.

Table III

n	time (secs)	memory (megabytes)
2^7	28	16
2^8	256	128
2^{10}	20,000	8,000

Clearly $n=2^7$ is viable on this system, $n=2^8$ is marginal because of the large amount of memory required, and $n=2^{10}$ is too large.

What can be done to reduce this time? Even though the AP-164 does FFTs very rapidly, it can be significantly improved by the addition of special hardware. Moreover, if that hardware is interfaced to the bulk memory, we may be able to get data to and from it sufficiently fast.

Consider the situation shown in Figure II.

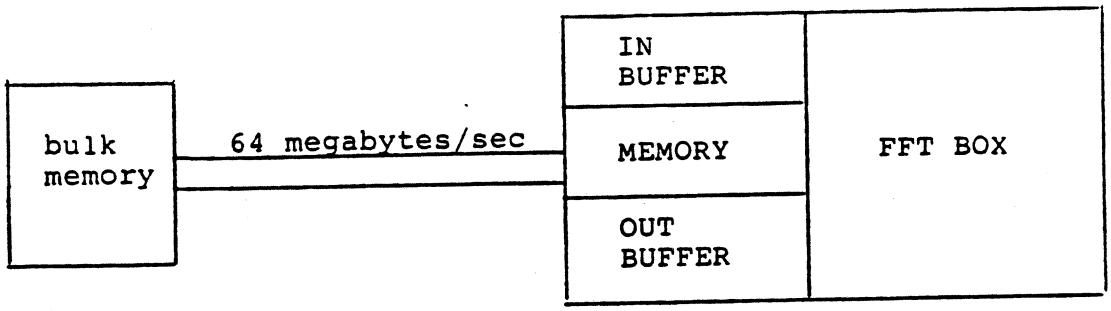

FIGURE II

To simplify our analysis, we make the assumption that all the 1-dimensional FFTs on the data in the memory of the FFT-box can be completed in the time, T, necessary to fill the in-buffer and empty the out-buffer. Thus the computation in the FFT-box is totally overlapped with I/O. If we have an nxnxn grid and a kxkxk block size and a block row fits into each buffer, then

$$T = k^2 n/4 \cdot 2^{20} \quad \text{secs},$$

which implies that the flops rate of the FFT box must be at least

$$(k^2 \cdot 2 \cdot 2n \log n)/(k^2 n/4 \cdot 2^{20}) \approx 2^{24} \log n$$

or

16 logn megaflops. Thus for $n=2^7$, the FFT box must achieve a computational rate of 112 megaflops, which is clearly achievable with today's technology.

Since we need to do $6 \cdot (\frac{n}{k})^2$ block transforms, each requiring time T, the total time for the transforms is $6 \cdot (\frac{n}{k})^2 \cdot \frac{k^2 n}{4\ 2^{20}} \approx .75n^3 \cdot 10^{-6}$ seconds. For the normalization, we need:

(1) to transfer both the normalization constants and the data to the AP-164 which takes $2n^3$ cycles;

(2) to carry out the normalization which requires $3n^3$ cycles; and

(3) to output the normalized data back to the bulk memory which takes n^3 cycles.

This analysis implies that the normalization requires $6n^3$ cycles or $n^3 10^{-6}$ seconds. Thus the total computation would require $1.75\ n^3\ \ 10^{-6}$ seconds. We note that while an FFT box saves us a logn factor in the run time of the FFT portion of the algorithm, a normalization box (attached to the bulk memory) would save us at most a factor of two in the run time of the normalization portion of the algorithm.

With multiple bulk memory boxes, it is possible to interleave I/O. For example, with 4 memory boxes we could get a 4-fold interleave which would speed I/O by a factor of 4. If we increased the computing speed of the FFT box accordingly, the transforms could be done 4 times faster. However, if we are using the AP-164 for the normalization, it probably isn't worth the effort.

Of course, we could build a normalization box with a 4-fold interface. In fact, at 256 megabytes/second, we could input two words and output one word approximately every 80 nanoseconds. In order to be I/O bound such a box would have to be able to do a 64 bit floating point multiply every 80 nanoseconds. These new timings are summarized in Table IV.

n	AP-164 and bulk memory	AP-164, bulk memory, and FFT box
2^7	28	2.8
2^8	256	22
2^{10}	20,000	1,333

TABLE IV

So far, we have talked only about the Poisson solver. Clearly we want something faster for the rest of the code. We are developing a replacement for the AP-164 which push the VLIW architecture to its limit, cf., [5]. We call it the ELI Enormously Long Instruction)-512. The 512 comes from the instruction word length. The goals of the ELI project are:

(1) to run compiled general purpose scientific code written in FORTRAN;

(2) yield high speed at low cost without microcoding; and

(3) to first build an "optimizing" compiler called BULLDOG which will help determine how much parallelism to build into ELI.

Very roughly speaking, the machine could have a cycle time 4 times faster (if built out of ECL) than the AP-164 and 8 times as much parallelism. Thus it is potentially 32 times as powerful.

Of course, ELI has its limitations. We want to make sure that everyone understands that the ELI computer:

(1) is an attached processor;

(2) is optimized for running only compute bound scientific code; and

(3) will perform poorly for "dynamic" code.

ACKNOWLEDGMENTS

We acknowledge the assistance of our colleagues at Yale, especially Peter Geiger and Susan O'Donnell who aided with the research described in Section II.

REFERENCES

[1] B.L. Buzbee. A Fast Poisson solver amenable to parallel computation. IEEE Transactions on Computers C-22:793-796 (1973).

[2] J.W. Cooley, P.A.W. Lewis and P.D. Welch. The Fast Fourier Transform Algorithm: Programming considerations in the calculation of Sine, Cosine and Laplace transforms. J. Sound Vib 12:315-337 (1970).

[3] F.W. Dorr. The direct solution of the discrete Poisson equation on a rectangle. Siam Rev 12:248-263 (1970).

[4] S.C. Eisenstat and M.H. Schultz. On some trends in elliptic problem solvers. In "Elliptic Problem Solvers" (Martin H. Schultz, ed.) pp. 99-114. Academic Press, New York, (1981).

[5] J.A. Fisher. Very long instruction word architectures and the ELI-512. Technical Report 253, Department of Computer Science, Yale University, 1982.

[6] P. Geiger, S. O'Donnell, M.H. Schultz. Some fast Poisson solvers on an array processor. In preparation.

[7] R.W. Hockney. A fast direct solution of Poisson's equation using Fourier analysis. Journal of the ACM 12: 95-113 (1965).

[8] M.A. Malcolm and J.Palmer. A fast method for solving a class of tridiagonal linear systems. Communications of the ACM 17:14-17 (1974).

[9] P.N. Swartztrauber. The methods of cyclic reduction, Fourier analysis and the FACR algorithm for the discrete solution of Poisson's equation on a rectangle. SIAM Rev 19:490-501 (1977).

[10] P.N. Swartztrauber. Vectorizing the FFTs. In "Parallel Computations" (Garry Rodrigue, ed.) pp. 51-83. Academic Press, New York, (1982).