N88-23235

ITERATIVE FINITE ELEMENT SOLVER ON TRANSPUTER NETWORKS*

Albert Danial and James Watson
Sparta, Inc.
Huntsville AL 35805

ABSTRACT

Iterative methods have been proven effective in obtaining solutions to large, sparse systems of linear equations such as those generated by finite element and finite difference methods. In addition to being efficient on sequential computers, iterative methods have inherent parallelism that suggests a strong potential for acceleration using parallel processing computer networks. These factors make iterative methods ideal candidates for parallel finite element/finite difference solvers. Here, we describe the parallelism inherent in the Conjugate Gradient method and discuss the initial results of a parallel implementation on a network of twelve transputers.

The high efficiencies obtained (a speed-up of 11.2 was gained with 12 processors) indicate that significant speed-up can be achieved with larger transputer arrays if communication overhead can be kept low. To this effect, we suggest a method of communication that allows large, dynamically reconfigurable transputer arrays to exchange data in $\log_4 N$ steps for N processors.

PRECEDING PAGE BLANK NOT FILMED

*Work done for the Sturctural Dynamics Branch under NASA Contract NAS3-   ; technical monitor: Louis J. Kiraly.

Introduction

As part of a NASA innovative research grant to develop a transputer-based finite element computing engine, researchers at SPARTA have investigated techniques and computing methods which show promise for efficient parallel execution. Here, we discuss recent work done to evaluate a parallel implementation of the Conjugate Gradient (CG) method on a network of transputers. Preconditioning is implied in the context of finite element applications, but is beyond the scope of this presentation. For direct factorization methods, see [George, et al, 1986], and for polynomial preconditioning, see [Allen, 1987].

The CG method has several attributes that make it attractive for solving finite element problems. It is robust even for poorly-conditioned problems, requires less memory than direct methods since there is no matrix fill-in, and works well for widely banded problems such as those created by three dimensional models. In addition, the CG method is well suited to adaptive analysis methods which slightly modify the stiffness matrix after each solution until descretization errors are minimized. In this case, rather than completely resolving the modified system of equations, the CG method can use the most recent solution as an excellent initial guess and will consequently converge quickly. Finally, because of its heavy usage of inherently parallel matrix-vector and vector-vector operations, the CG method shows great potential for efficient concurrent processing.

Here we describe the parallelism inherent to the method, demonstrate why communication determines efficiency, discuss our transputer implementation and show how transputers can be used in massive arrays before communication becomes a problem. Using the fractional summation method described here, we predict that a 1024 transputer network rated at 1.5 gigaflops, could attain a speed-up of 929 and provide a sustained computational rate on the order of 1 gigaflop.

CG Method for Solving Finite Element Problems

- Robust for poorly conditioned problems
- Lower memory requirements than direct methods
- Ideal solution method for adaptive analysis
- Efficient for widely banded, 3-D problems
- Computations are completely parallel

Twelve Transputer Implementation

- Speed-up of 11.2 obtained; higher possible
- Efficiency depends on method of communication

CG Method + Transputers + Link Switcher = Gflop Finite Element Solver

- Dynamically reconfigurable arrays allow efficient communication
- Could attain near-linear speed-up with thousands of processors rated with Gflops of power.

Conjugate Gradient Method

The CG method can be described by 18 single operation steps. These steps, given below, use the following notation:

[A] = stiffness matrix	{x} = displacement vector (the solution)
$\{b\}_o$ = force vector	{p},{r},{s},{t} = work vectors
$\{x\}^o$ = initial guess at a displacement vector	α, β,u,v,w = scalars
	k = iteration counter

```
1.  k            = 0
2.  {p}^o        = [A] {x}^o              matrix-vector multiply
3.  {r}^o        = {b} - {p}^o            vector subtraction
4.  {p}^o        = {r}^o                  vector equivalence
5.  {s}^k        = [A] {p}^k              matrix-vector multiply
6.  u^k          = {r}^k*{r}^k            vector dot product
7.  v^k          = {p}^k*{s}^k            vector dot product
8.  α^k          = u^k/v^k                scalar division
9.  {t}^k        = α^k*{p}^k              vector scaling
10. {x}^{k+1}    = {x}^k + {t}^k          vector addition
11. Stop if ||{t}^k||<tolerance           vector comparison
12. {s}^k        = w^k*{s}^k              vector scaling
13. {r}^{k+1}    = {r}^k - {s}^k          vector subtraction
14. v^{k+1}      = {r}^{k+1}*{r}^{k+1}    vector dot product
15. β^k          = v^{k+1}/u^k            scalar division
16. {p}^k        = β^k*{p}^k              vector scaling
17. {p}^{k+1}    = {r}^{k+1} + {p}^k      vector addition
18. increment k
19. Go to step 5.
```

The following operations are performed at each iteration:

1 Matrix-vector multiplication
3 Vector dot products
3 Vector scalings
3 Vector additions
1 Vector comparison

Each one of these computations can be performed in parallel.

Parallelism in the Conjugate Gradient Method

The single most time consuming step is the matrix-vector multiply at step 5. Fortunately, it is also the operation most easily performed in parallel: each node receives a horizontal slice of the stiffness matrix and a complete copy of the {p} vector, then independently multiplies the matrix slice with the corresponding terms in {p} to obtain a partial solution for {s}. The vast majority of the remaining steps involve other vector operations, so a first glance might suggest that the algorithm is trivial to complete in parallel since the vectors can be divided up among the processors to be operated on concurrently.

This is only partially true on a local-memory processing network, however, since there are data dependencies between steps that require the processors to exchange data. After each processor computes its segment of {s} at step 5 for example, it can only perform one or two more steps before it needs a complete copy of {s} or a complete sum for v^k to perform the vector scaling at step 9. This type of data dependency (where each processor has a fraction of a value yet requires the sum of all fractions on every processor to continue), the only type encountered in the CG method, is resolved by a process called fractional summation. As its name implies, each processor simultaneously sends, receives and sums individual fractions of the value, preferably in a well-coordinated manner, until each processor has the complete sum. These communication steps can impede performance of a parallel CG solver, and must proceed as quickly as possible. The formula and graph below illustrate the effects of communicate time on speed-up.

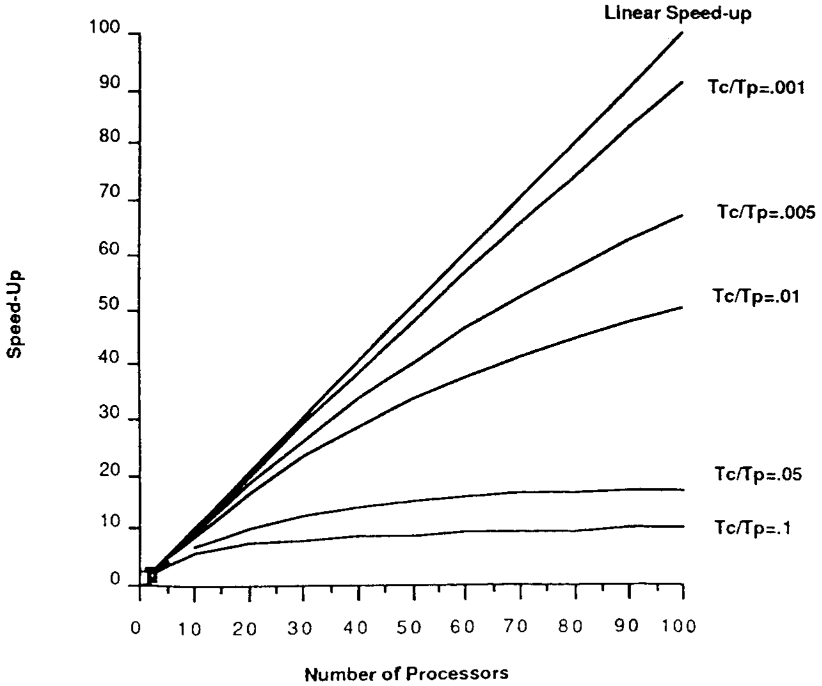
T_c = Time spent communicating

T_p = Time spent executing parallel tasks (all compute time in CG method)

N = Number of processors

$$\text{Speed-up} = \frac{N}{N \cdot \frac{T_c}{T_p} + 1}$$

Speed-Ups for Various Ratios of Overhead



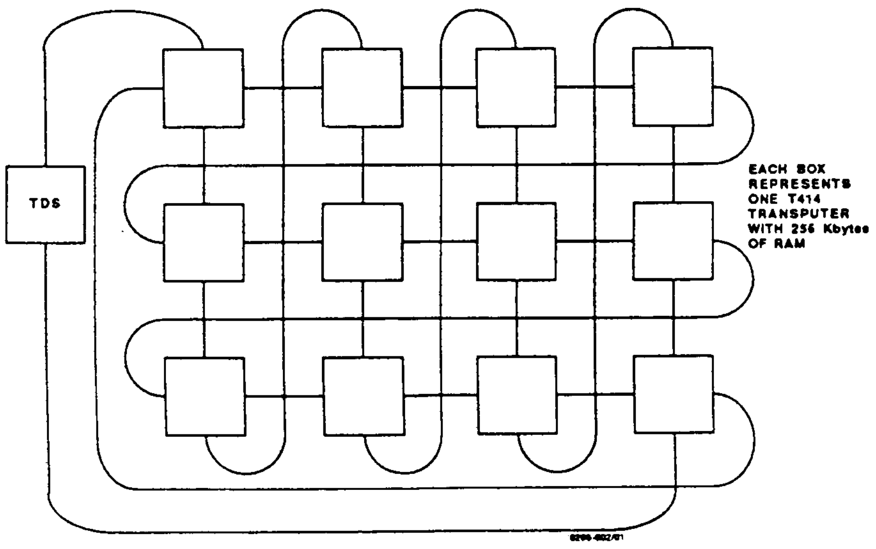
Transputer Network and Test Problem

The parallel processing network we used to implement the CG method consisted of twelve INMOS T414 transputers as shown below. Each transputer has 256 Kbytes of local RAM memory and four links capable of transferring data to other transputers at a rate of 10 Mbits/second.

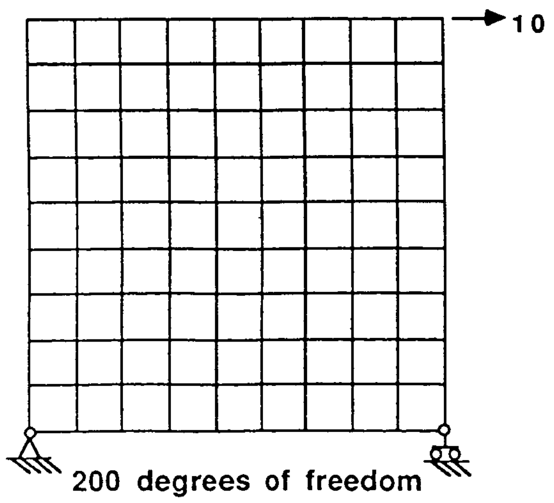
A simple test problem consisted of a 2-D square plate subdivided into 81 isoparametric, four-node elements yielding 200 degrees of freedom. The lower left corner of the plate was pinned, the lower right corner constrained from vertical motion and the top right corner had a horizontal applied load.

Although the stiffness matrix was tightly banded, the implemented CG code carried all matrix and vector operations out in full, as if the matrix were dense.

TWELVE TRANSPUTER NETWORK
ARRANGED IN A DOUBLE-RING GRID


TDS
EACH BOX REPRESENTS ONE T414 TRANSPUTER WITH 256 Kbytes OF RAM

TEST PROBLEM



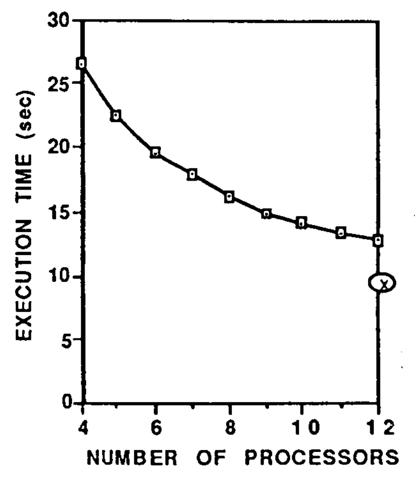
Results

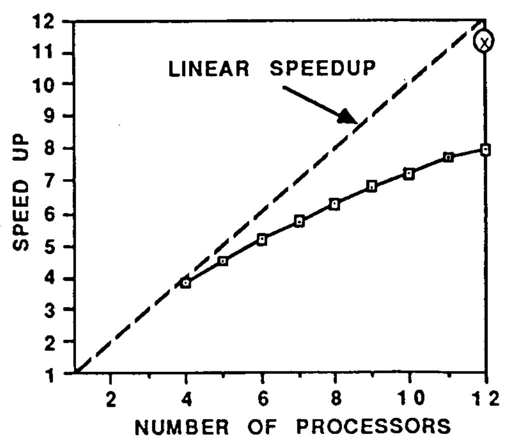
Three versions of the CG method were implemented: a fully sequential version to provide a reference for performance, a parallel version written with Adnet (a high-level communications environment) and a second parallel version using direct, hardcoded communications that sent messages around a ring. The programs all stopped after 73 iterations, when changes to the displacement vector were less than a tolerance of 0.00001. The execution times are tabulated and shown graphically below.

Method:	Sequential	Parallel with Adnet Sec. (speed up)	Parallel without Adnet Sec. (speed up)
Number of Processors			
1	102.2		
4		26.61 (3.84)	
5		22.53 (4.54)	
6		19.60 (5.21)	
7		17.91 (5.71)	
8		16.22 (6.30)	
9		14.98 (6.82)	
10		14.14 (7.22)	
11		13.29 (7.69)	
12		12.87 (7.94)	9.13 (11.2)

Despite the impressive speed-up obtained, a timing analysis of the data exchanges showed that still higher speed-ups are possible. When done independently, the data exchanges around the processor ring take less than one-thousandth of the time calculations require, indicating that efficiencies of 0.989, or a speed-up of 11.87, should be possible on the network of twelve transputers. Further analyses of our implementation are being conducted to pinpoint the causes for the sub-optimum run times.

EXECUTION TIME AND SPEED UP VERSUS NUMBER OF PROCESSORS FOR THE TEST PROBLEM





Ⓧ - DIRECT LOW LEVEL COMMUNICATIONS

□ - COMMUNICATION HANDLED BY A CONVENIENT MESSAGE PASSING SUBROUTINE

Fractional Summation on a Large, Dynamically Reconfigurable Network

If every processor in a network were directly connected to all other processors, fractional summation would be trivial -- each node would simply send its fraction out on every out-link and collect fractions from other nodes from its in-links. Few parallel processors, however, have more than 10 links, so direct connection schemes can only be used on small networks. Networks of indirectly connected processors perform fractional summation in a series of transmit and receive steps and can spend considerable amounts of time communicating. Large networks require more communication steps than small networks, making high speed-ups increasingly difficult to obtain. The table below lists the number of communication steps required to perform a fractional summation on several types of network topologies.

N = Number of processors

S = Number of steps required for fractional summation

Topology	S	Number of steps required if N = 1024
Ring	$N - 1$	1023
Double-ring grid	$2\sqrt{N}$	64
Shuffled exchange [Allen, 1987]	$2\log_2 N$	20
Hypercube	$\log_2 N$	10
Dynamically reconfigurable transputer array	$\log_4 N$	5

Example of Fractional Summation on a Dynamically Reconfigurable Network

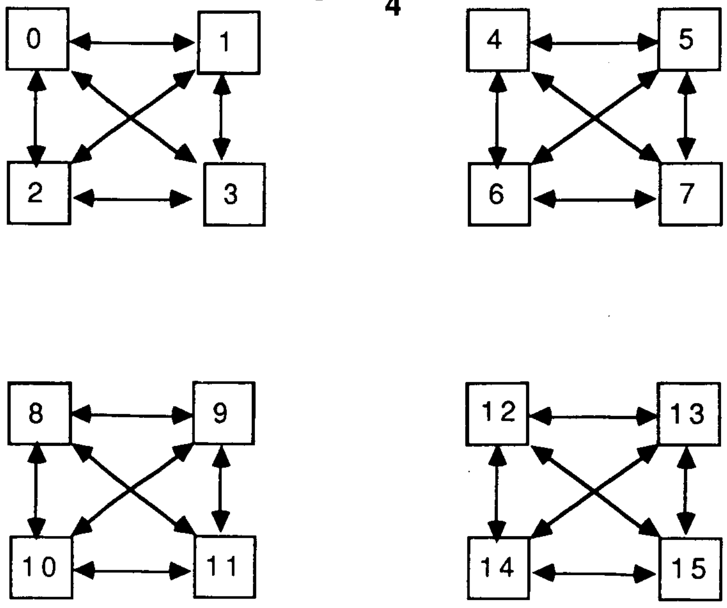
Although transputers have only four links each, programmable link switches such as the INMOS C004 and the Unisys Switch Slice allow programs to change the network configuration during execution. Configuration changes can be made in one microsecond and can take place while the processors are busy computing, so negligible overhead is incurred. These link switches are extremely powerful devices and make possible several advanced types of network data distribution, one of which is fractional summation on a dynamically reconfigurable network. The basic idea behind this kind of fractional summation is to group together small islands of directly connected processors, allow them to exchange values, then reshuffle the processor connections so that each processor is relinked with a completely different set of processors. In this manner, the number of communication steps required will be reduced to $\log_{(1+L)} N$ where L is the number of links each processor uses to exchange data.

The example below illustrates how a network of 16 transputers connected to a programmable link switch can perform a fractional summation in two steps.

STEP 1

Network configuration: Fully connected sets of four processors. Set J contains the processors whose ID's satisfy the integer division equation

$$J = \frac{ID}{4}$$



The sums on each processor will then be:

Node 0: 0+1+2+3
Node 1: 0+1+2+3
Node 2: 0+1+2+3
Node 3: 0+1+2+3
Node 4: 4+5+6+7
Node 5: 4+5+6+7
Node 6: 4+5+6+7
Node 7: 4+5+6+7
Node 8: 8+9+10+11
Node 9: 8+9+10+11
Node 10: 8+9+10+11
Node 11: 8+9+10+11
Node 12: 12+13+14+15
Node 13: 12+13+14+15
Node 14: 12+13+14+15
Node 15: 12+13+14+15

Example of Fractional Summation on a Dynamically Reconfigurable Network (Continued)

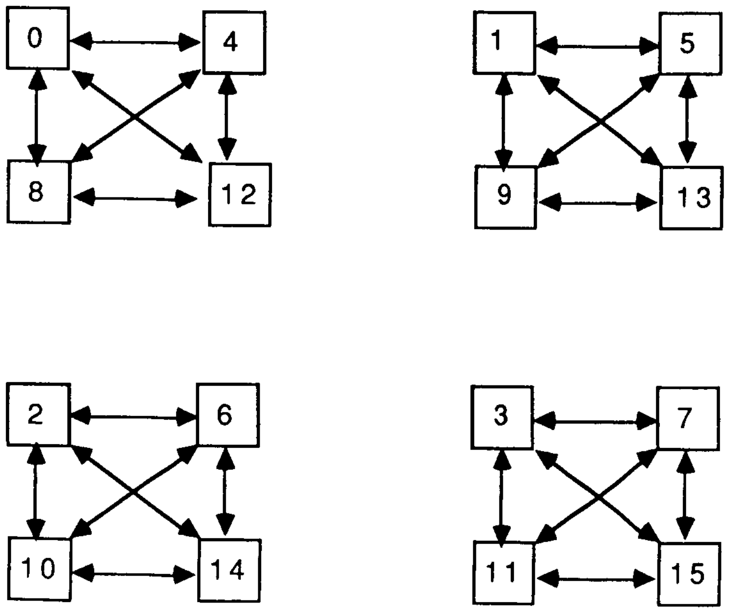
Here, only three of the four links on each transputer are being used (L=3). A free link is reserved on each node to allow the node to send control information to the link switches, or to some master transputer which controls the network configuration. If a timing scheme is used to control link switchings, all four links can be used (L=4) and the number of communication steps will be reduced to $\log_5 N$.

The methods and equipment described here can be used to assemble a massive CG solver capable of obtaining three orders of magnitude of speed-up, and sustaining on the order of one gigaflop of double precision computations. An array of 1024 T800 transputers with 1 Mbyte of RAM connected by 196 programmable link switches, should be able to run a CG algorithm with an overhead fraction (Tc/Tp) between 0.0001 and 0.001.

These overhead fractions correspond to a speed-up range from 506 to 929.

STEP 2

Network configuration: Fully connected sets of four processors. Set J contains the processors [J, J + 4, J + 2(4), J + 3(4)]



After Step 2, all of the processors will have the complete sum:

Node 0: 0+1+2+3+4+5+6+7+8+
9+10+11+12+13+14+15
Node 1: 0+1+2+3+4+5+6+7+8+
9+10+11+12+13+14+15
Node 2: 0+1+2+3+4+5+6+7+8+
9+10+11+12+13+14+15
Node 3: 0+1+2+3+4+5+6+7+8+
9+10+11+12+13+14+15
Node 4: 0+1+2+3+4+5+6+7+8+
9+10+11+12+13+14+15
•
•
•
Node 15: 0+1+2+3+4+5+6+7+8+
9+10+11+12+13+14+15

Conclusion

The parallel CG method has all the attributes of an ideal finite element solver: its computations are completely parallel enabling many processors to obtain large speed-ups; its iterative nature makes it the solution method of choice for adaptive analysis, where small refinements to the stiffness matrix only require few additional computations to obtain a new solution; and finally, the matrices can be stored in compact form since the method does not fill them in as direct methods do.

The only overhead incurred in the parallel CG method is the communication time it takes to resolve data dependencies. It was demonstrated that even inefficient ring communication schemes could attain high speed-ups - our code ran 11.2 times faster on 12 transputers than it did on one. Speed-up for the CG method is inversely proportional to the time spent communicating during fractional summation, so large networks must have efficient methods of exchanging data in order to maintain high speed-ups.

Programmable link switches, devices that permit connections between transputers to be made through software control, can be used in large transputer networks to distribute data faster than any other local-memory MIMD architecture. This permits larger networks to operate at a given communicate-to-compute ratio. This permits large networks of transputers to operate at the same overhead levels as much smaller, hardwired networks. Fractional summation on a dynamically reconfigurable network was shown to require only $\log_4 N$ communication steps - half the number a hypercube of the same size needs. The resulting reduction in communication overhead should enable more than one thousand transputers to run parallel CG code with an efficiency above 90%. At the current price of a 1 Mbyte T800 transputer rated at 1.5 megaflops, a 1 gigaflop finite element solver could be built for less than $1,000.000.

CG Method Excellent for Parallel Finite Element Solvers

- Computations are completely parallel
- Natural solution technique for adaptive analysis
- Can solve larger problems than direct methods in the same amount of RAM

Results for 12 Transputer Implementation

- Speed-up of 11.2 obtained; many improvements possible
- Demonstrated that efficiency depends on fraction of communication time to compute time

Dynamically Reconfigurable Transputer Arrays

- Reduce communication overhead
- Permit thousand-processor networks to function efficiently
- Could make possible a Gflop finite element machine for less than $1,000,000

References

1. Allen, R., 1987, "Matrix/Vector Multiplication and the Conjugate Gradient Algorithm on Transputers," presented at the Occam Users Group Meeting, September 29, 1987, Chicago, IL.

2. George, A. et al., 1986, "Sparse Cholesky Factorization on a Local Memory Multiprocessor," Oak Ridge National Laboratory TM-9962, Oak Ridge, TN.