PARALLEL LINEAR EQUATION SOLVERS FOR FINITE ELEMENT COMPUTATIONS

Principal Investigator
James M. Ortega
University of Virginia

Gene Poole, Ph.D., 1986, Research Associate until July 1, 1987
Courtenay Vaughan, Ph.D., 1988
Andrew Cleary, Ph.D., 1988?
Brett Averick, M.S., 1987

James M. Ortega is Charles Henderson Professor and Chairman of Applied Mathematics at the University of Virginia. He is the principal investigator on NASA Grant NAG1-46 through the Computational Structural Mechanics Group and is also principal investigator on the NASA Training Grant NAG1-242.

Eugene Poole received his Ph.D. in Applied Mathematics at the University of Virginia in May 1986 and remained at the University until July 1987, partially supported under grant NAG1-46. He is now employed by Awesome Computing, Inc. and works directly with the CSM Group.

Courtenay Vaughan is a Ph.D. student in Applied Mathematics and expects to finish his degree in Spring 1988. He has been supported by NAG1-242 and also by Oak Ridge National Laboratory in the summers of 1986 and 1987.

Andrew Cleary is a Ph.D. student in Applied Mathematics with expected completion date late in 1988. He has been supported by NAG1-242 and NAG1-46 and has spent the last three summers at NASA-Langley.

Brett Averick will finish a masters degree in Applied Mathematics in Fall 1987 and plans to pursue a Ph.D. He was supported by NAG1-242 while he spent the summer of 1987 at NASA-Langley.

GENERAL OBJECTIVES

Develop numerical algorithms for solution of linear and nonlinear systems of equations on parallel machines.

Apply these algorithms to problems in structural analysis, in particular, the current focus problems.

STATUS

Choleski Banded and Profile Codes on Flex/32 for $Kx = f$

Conjugate Gradient and Preconditioned Conjugate Gradient Codes on Flex/32, Intel iPSC hypercube, and CRAY-2

Flex and CRAY Codes Operating in Conjunction with the CSM Testbed System

Solution for Static Displacements for Panel and Mast Focus Problems.

The overall objective of this research is to develop efficient methods for the solution of linear and nonlinear systems of equations on parallel and supercomputers, and to apply these methods to the solution of problems in structural analysis. Attention has been given so far only to linear equations.

The methods considered for the solution of the stiffness equation $Kx = f$ have been Choleski factorization and the conjugate gradient iteration with SSOR and Incomplete Choleski preconditioning. More detail on these methods will be given on subsequent slides.

These methods have been used to solve for the static displacements for the mast and panel focus problems in conjunction with the CSM testbed system based on NICE/SPAR.

Factorization Methods on Flex/32

Choleski Method

$$K = L\,L^T$$
$$L\,z = f,\quad L^T x = z$$

Choleski Global Memory Code for Banded Storage

Choleski Local Memory Code for Profile Storage

Report, December, 1986

The Choleski method first factors K into LL^T, the product of a lower triangular matrix and its transpose. The solution of the displacement equations $Kx = f$ is then completed by solving the triangular systems of equations $Lz = f$, $L^Tx = z$.

Two versions of this method have been developed for the FLEX/32. The first uses banded storage and utilizes only global memory. The second uses profile (skyline) storage and utilizes the local memories.

Further information on these codes is given in:

A. Cleary, O. Harrar, and J. Ortega, Gaussian Elimination and Choleski Factorization on the FLEX/32. Applied Mathematics Report No. RM-86-13, University of Virginia, December, 1986.

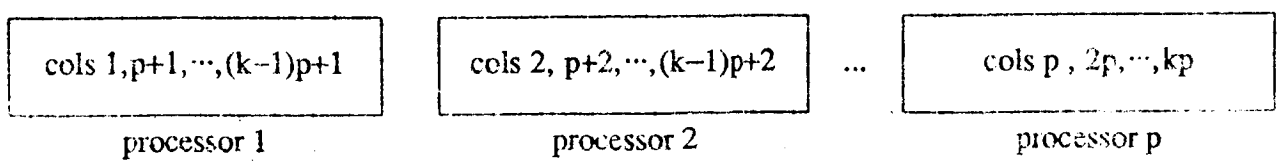
```
for k=1 to N
   l_kk = a_kk^1/2
   for s=k+1 to min(k+β,N)
      l_sk = a_sk / l_kk
   for j=k+1 to min(k+β,N)
      for i=j to min(k+β,N)
         a_ij = a_ij − l_ik l_jk
```

```
for k=1 to N
   cdiv(k)
   for j=k+1 to min(k+ β,N)
      cmod(j,k)
```

kji Choleski Factorization

```
if (column 1 is assigned to this processor)
   cdiv(1)
   mark column 1 done
for k=1 to N−1
   wait for column k to be done
   if (column k+1 is assigned to this processor)
      cmod(k+1,k)
      cdiv(k+1)
      mark column k+1 done
   for j = k+2 to min(k+β,N)
      if (column j is assigned to this processor)
            cmod(j,k)
```

Parallel Choleski Factorization With Compute-Ahead



Wrapped Interleaved Column Storage

The first figure shows a basic Choleski factorization code using the so-called kji form (L is computed by columns using immediate updates). For simplicity, banded storage with bandwidth β is used in this code. To the right of the first code is shown the same code in a short-hand version: cdiv(k) forms the first column of L and cmod(j,k) modifies the jth column of K using the kth column of L.

The second figure illustrates the code for each processor on the FLEX/32. Synchronization is achieved by marking columns of L "done" when they are computed. At the kth stage, the (k+1)st column in L is first computed and marked done so as to make this column available to other processors as soon as possible; this is the "compute-ahead" since the usual algorithm would compute the (k+1)st column only after the kth stage had been computed.

The third figure shows the storage assignment of the columns of K to the local memories. This interleaving of the columns of K has been observed by several investigators to give good load balancing in a local memory parallel system.

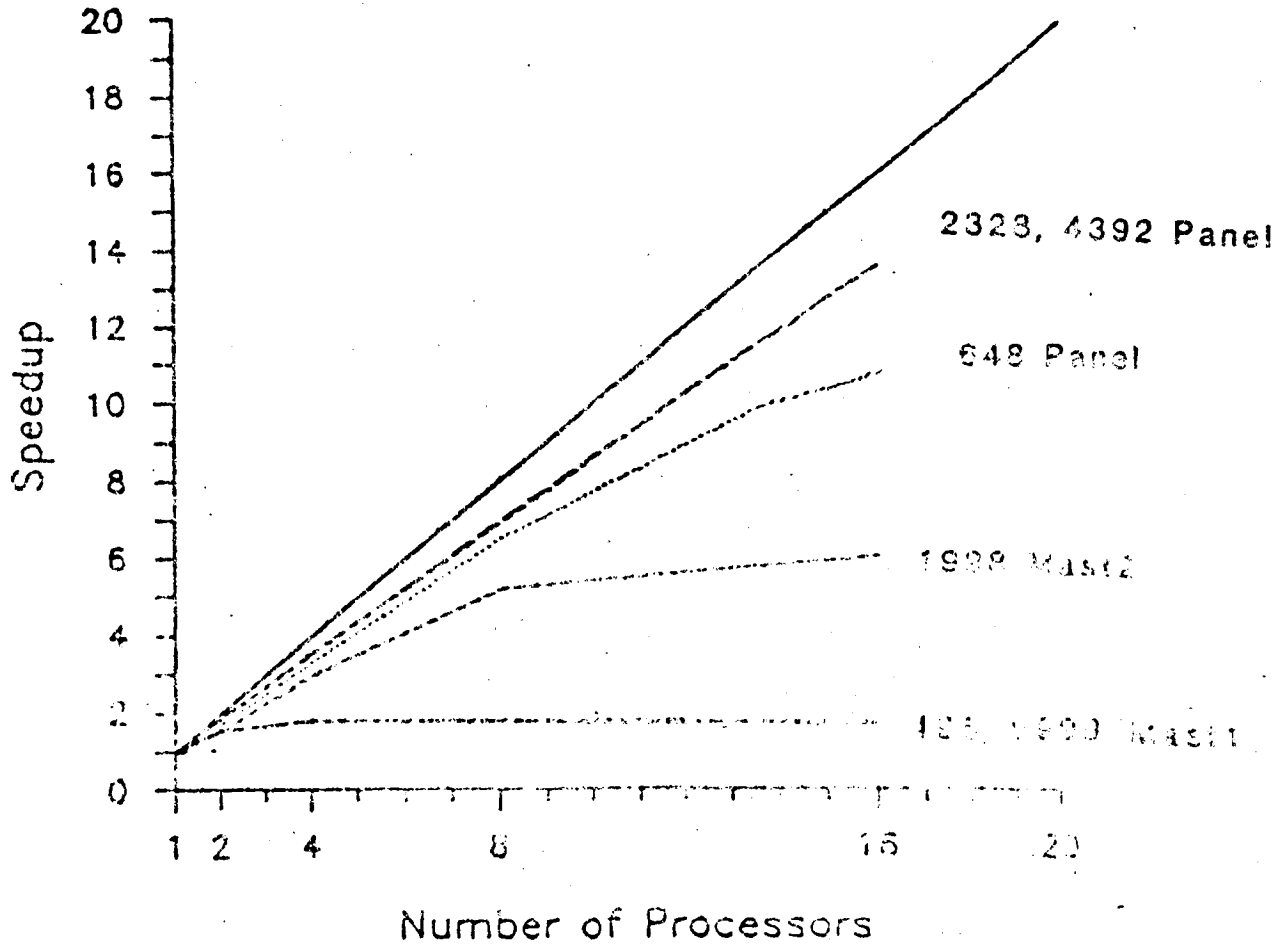
Choleski on the Flex/32

Problem	N	β	Storage	p	Time(secs.)	MFlops	Speed Up*
Panel	648	118	34978	serial	63.9	.0418	
				1	65.0	.0431	.98
				2	33.3	.080	1.91
				4	17.5	.153	3.66
				8	9.70	.276	6.60
				16	6.30	.424	10.14
Panel	2328	240	243126	serial	735	.0439	
				1	744	.0434	.99
				2	375	.0861	1.96
				4	195	.166	3.78
				8	101	.320	7.29
				16	55.5	.582	13.26
Panel	4392	260	501853	serial			
				1			
				2	839	.0858	1.95
				4	432	.167	3.89
				8	223	.323	7.36
				16	120	.600	13.67

* Speed Up for problems too large for single processor are computed using estimated MFlop rate on one processor

This table shows running times for the parallel Choleski code for the panel focus problem on the FLEX/32. Timings are given for 1, 2, 4, 8 and 16 processors. The corresponding speedups and mflop rates are also given. The speed-ups are calculated using a serial code. For comparison, times are also given for the parallel code on a single processor.

For the largest problem, 4392 unknowns, the local memory (4 Mbytes) is too small for the problem to be run on a single processor, and the speedups are computed by using an estimated mflop rate on a single processor.



Parallel Choleski Mast and Panel Results

This figure plots the speedups for the panel focus problems as well as two different mast problems. The first mast problem has a small bandwidth (15) and very poor speedups are obtained. The second mast problem has a bandwidth of about 50 and the speed-ups are better. The panel problems have much larger bandwidths, as given in the table, and the speed-ups are much better. In general, the speedups given by this Choleski code are primarily a function of the bandwidth and not of the number of unknowns. This is illustrated, in particular, for the first mast problem and the smallest panel problem, both of which have almost the same number of unknowns.

Conjugate Gradient Iteration

FLEX/32

Mast and Panel Focus Problems
Incomplete Choleski Preconditioning Using FORCE
SSOR Polynomial Preconditioning

CRAY-2 (Ames)

CG Running on Panel Focus Problem
Preconditioning Later

Intel iPSC Hypercube (Oak Ridge)

CG Running on Panel Focus Problem
Preconditioning Later

The second type of method is iterative, the conjugate gradient method with two different preconditioners: SSOR polynomial and incomplete Choleski factorization. These methods have been used to solve both the panel and mast focus problems on the FLEX/32. The incomplete Choleski codes for the FLEX/32 use the FORCE package developed by H. Jordan. The conjugate gradient method has also been used for the panel focus problem on an Intel iSPC 64-processor hypercube at Oak Ridge National Laboratory, and on the CRAY-2 at NASA-Ames; preconditioning for these codes is under development.

Preconditioned Conjugate Gradient Code

Choose $\mathbf{x}^0$. Set $\mathbf{r}^0 = \mathbf{f} - K\mathbf{x}^0$. Solve $M\tilde{\mathbf{r}}^0 = \mathbf{r}^0$. Set $\mathbf{p}^0 = \tilde{\mathbf{r}}^0$.

For $k = 0, 1, \ldots$

one dimensional minimization
$$\begin{cases} \alpha_k = -(\tilde{\mathbf{r}}^k, \mathbf{r}^k)/(\mathbf{p}^k, K\mathbf{p}^k) \\ \mathbf{x}^{k+1} = \mathbf{x}^k - \alpha_k \mathbf{p}^k \end{cases}$$

update residual $\mathbf{r}^{k+1} = \mathbf{r}^k + \alpha_k K\mathbf{p}^k$

Test for convergence

preconditioning Solve $M\tilde{\mathbf{r}}^{k+1} = \mathbf{r}^{k+1}$

new conjugate direction
$$\begin{cases} \beta_k = (\tilde{\mathbf{r}}^{k+1}, \mathbf{r}^{k+1})/(\tilde{\mathbf{r}}^k, \mathbf{r}^k) \\ \mathbf{p}^{k+1} = \tilde{\mathbf{r}}^{k+1} + \beta_k \mathbf{p}^k \end{cases}$$

Note: Key parts are $K\mathbf{p}^k$ and preconditioning

A code is given for the preconditioned conjugate gradient method. (,) is the inner product of two vectors. The first two statements compute the next iterate $\mathbf{x}^{k+1}$ by minimizing the quadratic function $\mathbf{x}^T\mathbf{K}\mathbf{x} - 2\mathbf{x}^T\mathbf{f}$ along the line $\mathbf{x}^k - \alpha\mathbf{p}^k$. The residual at $\mathbf{x}^{k+1}$ is then computed and there is a test for convergence. This test can be based on $(\mathbf{r}^{k+1},\mathbf{r}^{k+1})$, or other tests can be used. The next step carries out the preconditioning by solving a system of equations with coefficient matrix M. Finally, the last two statements compute the next direction vector $\mathbf{p}^{k+1}$.

The potentially time-consuming parts of this process are the matrix-vector multiply $\mathbf{K}\mathbf{p}^k$ and the preconditioning.

Preconditioners

Incomplete Cholesky

$K = LL^T - R$, $M = LL^T$

L has same sparsity as K

$M\tilde{r} = z \Rightarrow Lz = r$, $L^T\tilde{r} = z$
Done each step of iteration

SSOR

An SOR iteration followed by an SOR iteration in reverse direction

m-step SSOR: m SSOR iterations

Polynomial SSOR: combine the m SSOR iterations in optimal way

Note: Multicoloring is used to parallelize SOR

Two preconditioners are used in conjunction with the conjugate gradient iteration. The first is incomplete Choleski factorization, in which the simplest strategy is no-fill: the factor L has a non-zero only if the corresponding position of K is non-zero. The factorization itself is done only once at the beginning of the iteration and the key point is the forward and back substitutions $L\mathbf{z} = \mathbf{r}$ and $L^T \tilde{\mathbf{r}} = \mathbf{z}$, which are done at each iteration. These are achieved by a sparse form of the column sweep algorithm with L stored by interleaved rows.

The second preconditioner is based on the SSOR iteration. This is a combination of an SOR iteration followed by another SOR iteration in the reverse direction. More than one SSOR iteration can be taken, and these SSOR steps can be weighted to give what is called SSOR polynomial preconditioning. Multicoloring of the nodes is used to parallelize the SOR iteration.

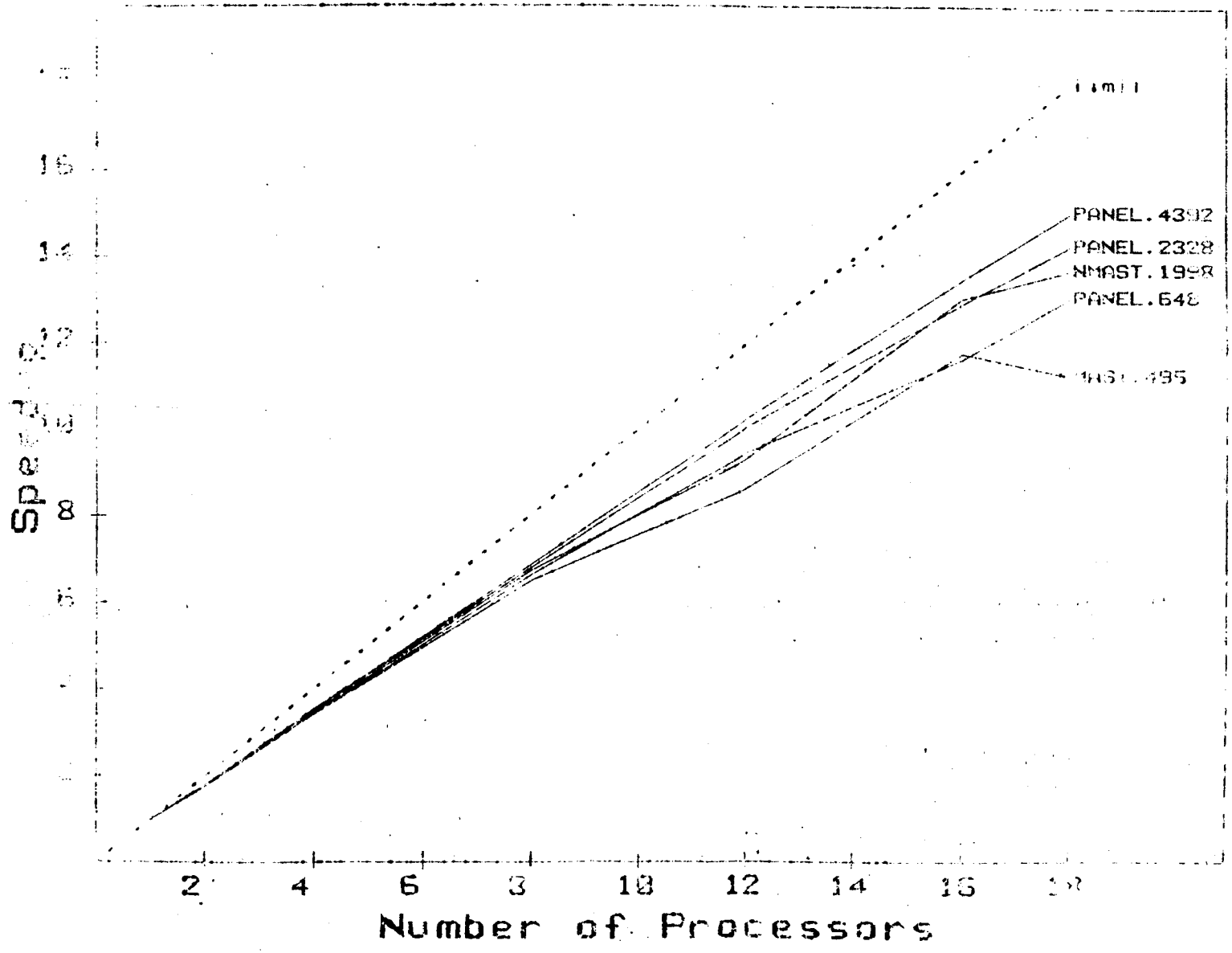
SSOR Preconditioned Conjugate Gradient

Time (seconds)

	MAST.495		NMAST.1998		PANEL.648		PANEL.2328		PANEL.4392	
	CG	PCG	CG	PCG	CG	PCG	CG	PCG	CG	PCG
p =	445	205	390	159	299	126	1152	422	1238	487
1	144.2	147.5	683.5	629.6	154.8	149.4	2522	2066	5416	4743
2	82.2	82.8	379.1	371.7	87.9	84.5	1407	1138	3016	2626
4	42.1	43.3	197.1	211.3	45.2	45.1	717	576	1530	1315
8	22.2	23.4	101.6	110.5	23.4	25.1	372	304	789	690
12	16.7	16.8	73.3	74.7	16.3	19.8	251	215	528	468
16	12.3	13.5	52.3	58.6	13.3	17.8	195	173	401	370
18	12.8	12.3	49.9	46.8	11.8	15.2	177	157	360	322

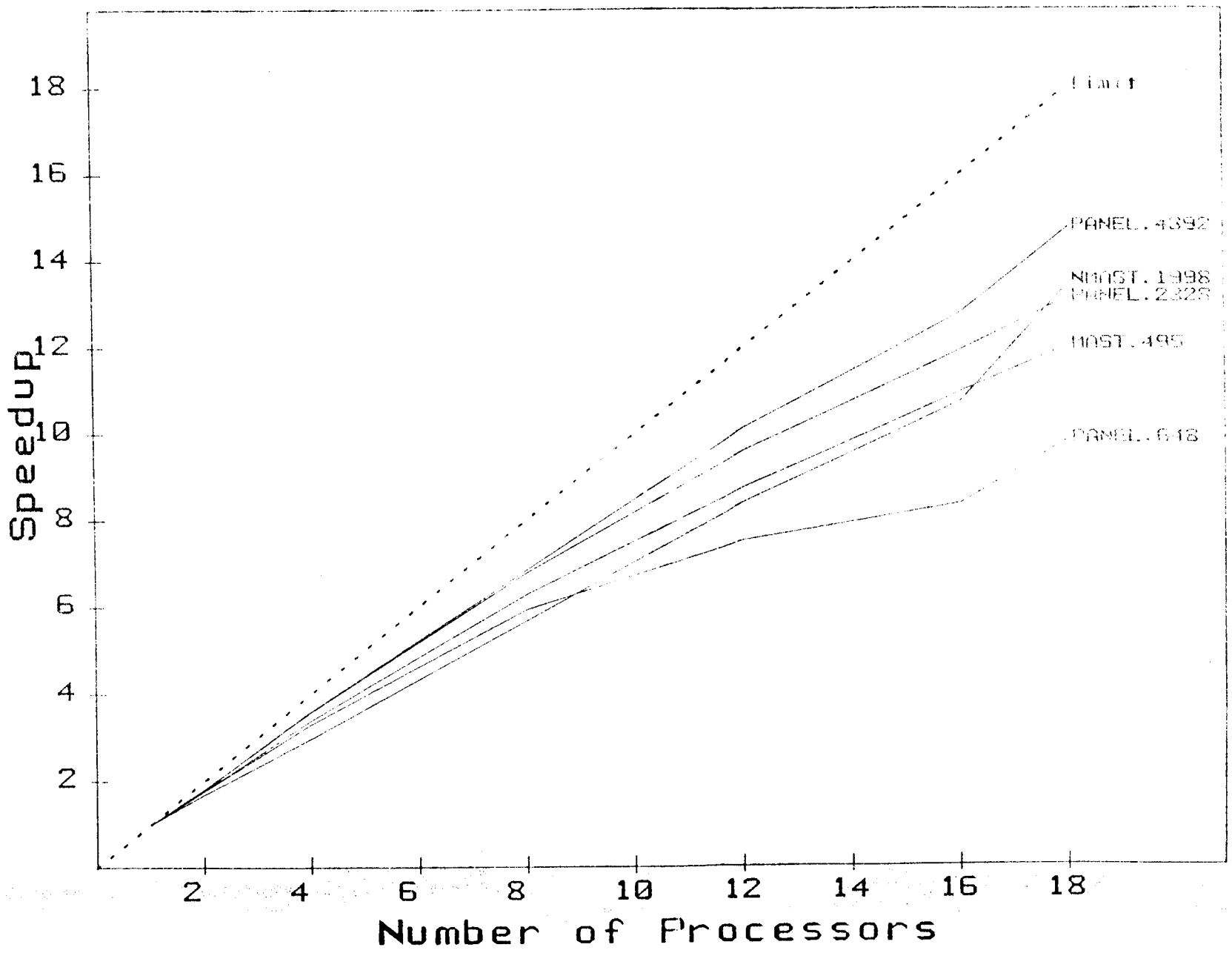
This table gives the number of iterations and the running times on the Flex/32 for the conjugate gradient and SSOR preconditioned conjugate gradient codes. The mast and panel focus problems are the same as used for the Choleski factorization. The convergence criterion used was $(\mathbf{r}^{k+1},\mathbf{r}^{k+1}) \leq 10^{-6}$ which gives about four decimal places of accuracy in the solution. A later chart will show the dependence of the number of iterations, and the times, on the convergence criterion.

Note that the run times for the preconditioned method are barely better than the conjugate gradient method and even worse in a few cases. But for the conjugate gradient code, the coefficient matrix has been scaled so that its diagonal elements are one. This is a simple form of preconditioning and without it the conjugate gradient method did not converge within n iterations, where n is the size of the matrix. Nevertheless, it remains an open question whether the SSOR preconditioning can be improved enough to be useful.



This chart plots the speed-ups of the conjugate gradient code. The speed-ups are quite satisfactory, about 15 on 18 processors for the largest problem size and somewhat less for the smaller problems. Note that the small bandwidth of the mast problems has essentially no effect on the conjugate gradient method, as opposed to Choleski factorization.

Speedup PCG Flex



This chart shows the speed-ups of the SSOR polynomial preconditioned conjugate gradient method. These speed-ups are a little worse than for the conjugate gradient method but still satisfactory.

Comparison of Choleski and Preconditioned Conjugate Gradient (Seconds)			
Panel	p	Choleski	PCG
648	1	64	150
	16	6.3	18
2328	1	744	2066
	16	56	173
4392	2	839	2626
	16	110	370

This chart compares the run times in seconds on the FLEX/32 of the Choleski factorization and preconditioned conjugate gradient methods. Times are given for three sizes of the panel focus problem and for 1 and 16 processors (except for the largest problem which Choleski could not run on a single processor).

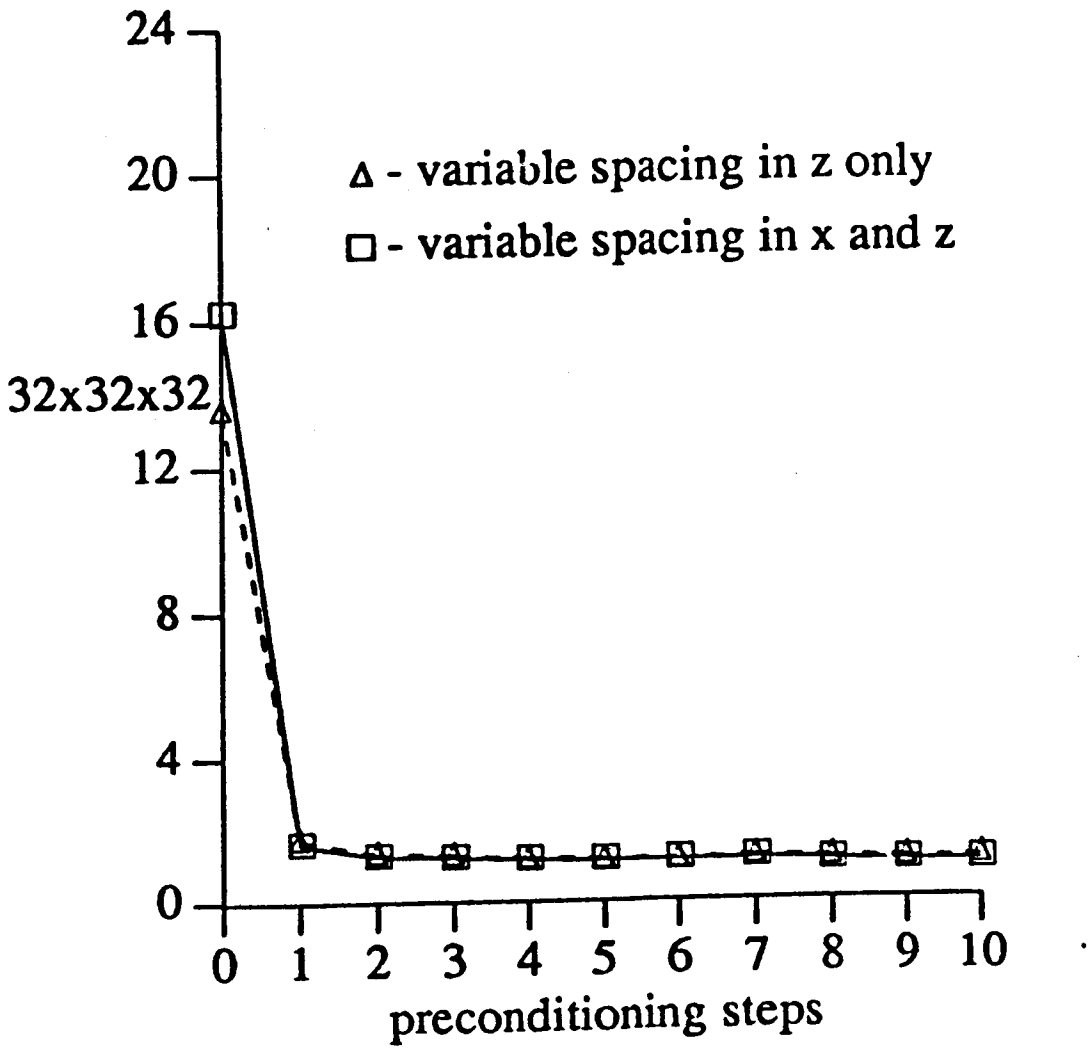
The Choleski method has a clear advantage on this problem.

Conjugate Gradient for PANEL.648

ε vs. error $(\mathbf{r}^{k+1}, \mathbf{r}^{k+1}) \leq \varepsilon$ maximum answer 0.216670		
ε	maximum error	iterations
10^{-6}	0.000056	299
10^{-5}	0.000697	268
10^{-4}	0.001966	230
10^{-3}	0.003683	188

The running times for an iterative method depend critically upon the convergence criterion. This table shows the results of varying the parameter ε in the convergence test $(\mathbf{r}^{k+1}, \mathbf{r}^{k+1}) \leq \varepsilon$ for the conjugate gradient method.

The approximate conjugate gradient solution is compared with the solution produced by NICE/SPAR to obtain the maximum error for different values of ε. As ε increases the number of iterations required to satisfy the convergence test decreases, as expected. With $\varepsilon = 10^{-3}$ the results are still accurate to a few units in the third decimal place.



SSOR Preconditioned Conjugate Gradient for $\nabla (a \nabla u) = f$

Time on Cyber 205

The chart shows what can be achieved by preconditioning. The problem is a three-dimensioned Poisson-type equation of the form $\nabla (a \nabla u) = f$, discretized by finite differences. For a $32 \times 32 \times 32$ cube, one step of SSOR preconditioning reduces the time by almost a factor of 10. For reasons which are not yet clear, this problem is very suitable for SSOR preconditioning whereas the focus problems are not.

Conjugate Gradient on CRAY-2

Panel 648

Conjugate Gradient: .26 seconds

NICE/SPAR: 2.2 seconds

Panel 2328

Conjugate Gradient: 3.5 seconds

NICE/SPAR: 18.9 seconds

Notes: NICE/SPAR Times Using Minimum Degree Ordering

Conjugate Gradient Convergence: $(\mathbf{r}^{k+1}, \mathbf{r}^{k+1}) \leq 10^{-6}$

This chart gives running times on a single processor of the CRAY-2 at NASA-Ames for two sizes of the panel problem. For comparison, times are also given for the solution phase of NICE/SPAR. The NICE/SPAR program uses the minimum degree ordering of the unknowns, which is favorable for the algorithms in NICE/SPAR. Note, however, that NICE/SPAR has not been optimized for the CRAY-2.

Summary and Conclusions

Choleski factorization gives good speedups if bandwidth sufficiently large but poor for small bandwidth.

Conjugate gradient gives good speedups independent of bandwidth.

Choleski factorization almost three times faster than preconditioned conjugate gradient on panel focus problem. But this depends on convergence test used for conjugate gradient.