Proceedings of the Fourth Conference on Hypercube Concurrent Computers and Applications, Monterey, CA, March 6-8, 1989, pp. 1227-1232.

A Software Kernel for a Pipeline Model for Sparse Matrix Problems on a Hypercube Applied to Reservoir Simulation

Roger L. Wainwright

Computer Science Department
The University of Tulsa

Amoco Production Company
Tulsa, Oklahoma

ABSTRACT

A Software kernel for a pipeline model was developed and implemented on a hypercube multiprocessor to test various iterative techniques for solving sparse matrices arising from a two dimensional, single phase diffusion equation for a reservoir simulation. The pipeline model works by partitioning the sparse matrix into rows and using the hypercube nodes in a ring topology. The software kernel was developed to implement the pipeline model, and to make it easy for a researcher to test different iterative techniques. All of the hypercube dependent code is hidden from the researcher in the kernel. The researcher need not be aware of communication issues, topologies, distributed memory, or other multiprocessor details. Many iterative techniques fit this pipeline model quite well. The kernel was used to test 10 different iterative methods (or variations) in the reservoir simulation. I found the method yielding the best results was SOR3, a variation of SOR, which over-relaxes three components at a time.

1. INTRODUCTION

Many scientific problems require the solution of linear systems of equations where the coefficient matrix is sparse and very large. The test problems used in this paper arise from a parabolic partial differential equation which is the two dimensional, single phase diffusion equation from petroleum reservoir simulation. The partial differential equation is discretized in space using an irregular block centered grid. If central differences are used to approximate the space derivatives and a backward difference is used to approximate the time derivative, then a five point difference equation is obtained. A similar approximation leads to a nine point discretization. The vast majority of reservoir simulations are two-phase, non-linear, non-symmetric models. In this paper the more basic single phase, linear problem is considered.

Consider Figure 1 which shows a 5 by 5 x-y grid space representing the physical layout of a small reservoir. The pressure, P, at each grid block at any time is given by the equation:

$$aP_{ij-1} + bP_{i-1,j} + cP_{i,j} + dP_{i+1,j} + eP_{i,j+1} = f$$

where the values for a, b, c, d, e, and f are constant over a given time step for grid block i,j. Using the above formula, we see that the pressure in the reservoir at a given grid point is a linear function of the pressures at the surrounding grid points. The linear equations for all 25 grid points in Figure 1 can be arranged as a sparse system of 25 linear equations, Ap = f. In this paper reservoirs of size N x M are considered. This produces sparse linear systems of size NM x NM, where NM represents N times M.

Scott[5] developed a method for the parallel solution of sparse matrices using SOR, which he called iteration pipelining. This is the pipeline model that was implemented in this paper. The method was also developed independently by Patel and Jordan[4]. A variation of this pipeline method is used by Aykanat and Ozguner[1] in their development of the parallel Conjugate Gradient method. This pipeline model is well suited for a distributed memory multiprocessor system such as a hypercube.

2. THE PIPELINE MODEL

The approach I adopted was to develop a generic model capable of solving sparse linear system of equations of various sizes using a variety of different methods. The only restriction is that the methods must fit the pipeline model. Consider again the simple reservoir model in Figure 1. Suppose each row of the grid is assigned to a single processor. This means that each of the five processors is assigned five of the equations to solve in the sparse matrix. In general the problem is to solve a reservoir model on an N x M grid space which corresponds to solving an NM x NM sparse linear system. In Figure 2, I have arbitrarily shown a 16 x 16 grid space partitioned four rows per processor. Since the pressure associated with each grid point is a function of the pressure values of its four neighbors and itself, an iterative solution at each time step would proceed as follows: During the first iteration processor 1 calculates values for rows 1 through 4. Results from row 4 are passed to processor 2 so it can calculate row 5 using the most current row 4 information. Once processor 2 has calculated row 8 it passes the results to processor 3 to begin calculating rows 9 through 12. This pipeline process continues to the last processor where each iteration is concluded. However, once processor 1 has passed the results of row 4 to processor 2, it begins the second iteration, while processor 2 continues to work on the first iteration. Once processor 2 completes row 5 it passes the information back to processor 1 for use in the calculation of row 4 in the next iteration. Thus in this model, a ring of processors is established such that between each neighboring processors the boundary rows are sent back and forth. If p processors are available to process the M rows of the grid then

each processor is responsible for M/p rows. Each processor will send and receive exactly one row from each of its two neighbors per iteration. Only processor p can determine if convergence has occurred.

3. THE SOFTWARE KERNEL

A software tool was delevoped to assist the researcher in developing algorithms for reservoir simulation. A software kernel was developed to implement the pipeline model described in Section 2. The goal was to make it easy for a researcher to test several different iterative techniques without having to know details about the hardware.

The kernel implementing the pipeline model is illustrated in Figures 3 and 4. The cube manager queries the user for the number of processors to be used to solve the problem. Input describing the reservoir model is read from an external file. This information is passed one time to each of the p processors. The cube manager then cycles through a loop capturing and printing results of the simulation from processor p until a termination notice is received.

Figure 3 shows the overall outline of the algorithm for each of the p processors. First, each processor determines its processor number and consequently who its previous and next neighbors are. Next, input describing the reservoir model is received from the cube manager. Each processor receives only that part of the x-y grid it is responsible for. Thus the data is automatically distributed over the number of processors in use. For each new time unit of the simulation, a pipeline of iterations is propagated through the processors. Processor 1 initiates each new iteration and processor p terminates the iterations by checking for convergence. The processors in between simply propagate the iterations. The details of the role of processors 2..p-1 are shown in Figure 4.

The pipeline communication among the various processors is handled by four simple routines: SEND BOTTOM, SEND TOP, RECEIVE BOTTOM, and RECEIVE TOP. Each routine communicates one row of the grid to a neighboring processor to give it the latest information for performing calculations in the next iteration.

All of the hypercube dependent code is hidden from the user. The kernel was developed such that when a new iterative technique is to be tested, the old procedure is removed and the new routine is inserted. The method-dependent routines that implement a particular iterative technique are represented by CALCULATE FIRST ROW, CALCULATE MIDDLE ROWS, and CALCULATE TOP ROW as shown in Figure 4. The researcher supplies a sequential version (simple textbook code) of the iterative technique to be tested and the kernel takes care of the rest.

4. ITERATIVE METHODS TESTED USING THE PIPELINE MODEL

The iterative methods that were tested using this model are described as follows:

(1) GS - Gauss-Seidel Method (SOR with w = 1.0). (2) SOR - The standard Successive Overrelaxation Method where at each step one component is overrelaxed. (3) LSOR - The Line Successive Overrelaxation Method. At each step a single line is overrelaxed. This method is a Block SOR method where at each step a tridiagonal matrix must be solved. (4) SOR2 - The standard Successive Overrelaxation Method where at each step two components are overrelaxed. This requires solving a 2 by 2 linear system at each step. (5) - SOR3 The standard Successive Overrelaxation Method where at each step three components are overrelaxed. This requires solving a 3 by 3 linear system at each step. (6) CONJGRAD - Conjugate Gradient Method with no preconditioning. This method requires global information to be broadcasted to each node after an iteration. I applied this method to the model ignoring the global broadcast to see what would happen. (7) - 1DIM/R One-dimensional r-projection method. (8) - 2DIM/R Two-dimensional r-projection method. (9) - 1DIM/X One-dimensional x-projection method. (10) - 2DIM/X Two-dimensional x-projection method.

The kernel developed for the model made it very easy to test each of these methods by removing one routine and inserting another. Methods 1-5 represent SOR and several variations. These methods generated the best results using the pipeline model. CONJGRAD did not converge. This was expected; the lack of broadcasting the required global information at the end of each iterative step was fatal.

The one and two dimensional r-projection and x-projection methods[6,7] (methods 7-10) were tested. Among the projection methods the two-dimensional x-projection method (with acceleration) performed the best. However this was several times slower than SOR and its variations. Other than slow speed the projection methods work very well using this model. The advantage of the projection methods is that they are very useful for solving non-symmetric systems, ill-conditioned systems, or other types of matrices where the traditional methods of SOR and Conjugate Gradient will not converge.

5. RESULTS

SOR and its variations, methods 1-5, had the best results using this model. A fixed-sized problem is largest size problem that can be solved on one processor. In this case, this is a 72 x 72 grid. The problem size is held fixed while the number of processors is increased[2,3]. The difference in CPU times for a fixed-sized problem between the best and worst of the five SOR methods using one node was almost 1700 seconds. The maximum difference using 32 nodes was around 62 seconds. This illustrates an important property of the pipeline model: as the number of processors is increased in a fixed-size problem the ratio of computation time to communication time decreases. In this case using 32 nodes this ratio decreased to the point where communication costs are so dominating that it makes little difference which computational method was used.

Table I depicts the results of SOR3 for scaled problems. Scaled problems are problems where the amount of computation per node is held constant. The total amount of computation is allowed to scale up as the number of processors increase. The largest size problem that all 64 nodes can accommodate is 528 x 528 grid. Each of these scaled problems for a given size cube are in turn considered as fixed sized problems where additional nodes are used. It is important to be able to measure the speedup for scaled problems. The model I used in order to estimate the CPU time on a single processor for scaled problems is similar to the model described in Ref. 2. Given an N x N square grid, the total computation time required using one processor, T_1, is given by

$$T_1 = ciN^2$$

where i is the total number of iterations over the grid and c is the computation time per grid point. The number of iterations is independent of the number of processors used for a given problem, and c is a calculated constant for the particular method used. The value for i is determined by running the problem using multiple processors, then applying the value to one processor. I used the fixed sized problem to determine the value for c for SOR3 and this was used to determine the estimated CPU time for one processor for the scaled problems. These values are indicated by an asterisk in Table I.

Table II depicts the scaled speedup corresponding to the processing times given in Table I. The values of interest are along the main diagonal which correspond to the scaled problems using various number of processors. The speedups range from 1.81 using two processors to a speedup of slightly more than 26 using 64 processors. One of the characteristics of the pipeline model that must be taken into consideration is the ratio of computation time to communication time. Unfortunately, in the pipeline model this ratio decreases as the problem is scaled up. The amount of computation performed per processor is held fixed. However, as the problem is scaled to accommodate for the increase in processors, it necessarily decreases the computation to communication ratio causing a decrease in overall performance per processor. Table II illustrates this point. The efficiency of each problem is given along with the ratio of computation to communication. This ratio is simply the number of rows of the grid assigned to each processor.

6. SUMMARY AND CONCLUSIONS

A Software kernel for a pipeline model was developed and implemented on a hypercube multiprocessor. The kernel was developed to assist researchers in analyzing different iterative techniques for solving sparse matrices arising from a reservoir simulation. The pipeline model proved very effective for many different methods for solving sparse linear systems, and the pipeline model was successfully packaged into a software kernel. The software kernel was responsible for partitioning the problem into subproblems, distributing data, and mapping these subproblems to available nodes of a hypercube. In addition the kernel handled the iteration process pipelined through the hypercube. The kernel also handled administrative details such as incrementing the time steps and testing for convergence. Specifically the software kernel (a) removed the need for the researcher to know any details about the hardware: hypercube communication issues, hypercube topologies, memory limitations, or other multiprocessor details, (b) allowed the researcher to easily test many different methods, or combinations of methods, by removing one subroutine and inserting another without having to reprogram each new method from scratch, and (c) allowed each subroutine to be written in "traditional" sequential code familiar to the researcher. No special code reflecting parallel or vector processing was required by the researcher.

I tested 10 different iterative techniques using the software kernel. The family of SOR methods fit the pipeline model quite well. SOR3, which overrelaxes three components at a time, proved to be the best overall method using the pipeline model. Projection methods were also tested. They fit the pipeline model, , but convergence for this particular reservoir model was too slow. The largest problem that could be solved with present hardware was a 528 x 528 grid. This represents a sparse linear system of size 278,784 x 278,784. Speedups up to 26 were obtained using SOR3 on scaled problems with 64 processors. The pipeline model has the inherent limitation that speedup can never be more than the number of iterations required for convergence of the linear system. Hence speedup does not necessarily increase with the addition of more processors. An additional 64 processors may not add much speedup for the particular linear systems solved here. However, additional processors would certainly allow for larger systems to be solved.

The ratio of computation time to communication time is an important measurement to consider in the pipeline model. This ratio decreases as the scaled problem size increases and has a direct effect on the efficiency of the algorithm. This ratio is a limiting factor in speedup for the pipeline model.

As a comparison the scaled problems were run on an IBM 3090 with 16M of virtual memory. The 528 x 528 and 384 x 384 grid problems were too large to run on the IBM with its existing memory. The largest problem I could run was the 264 x 264 grid. The IBM 3090 executed this problem using 1688 CPU seconds, compared to 2183 CPU seconds using all 64 nodes of the NCUBE hypercube. The IBM costs 25 times that of the hypercube.

The software kernel proved to be a tremendous time saver for testing many different linear system solvers. The pipeline model proved to be an excellent model to use on the hypercube multiprocessor for solving sparse linear systems of equation using iterative techniques. Finally, the hypercube was shown to be cost effective in solving the reservoir simulation. The software kernel greatly increased the rate at which the research was conducted.

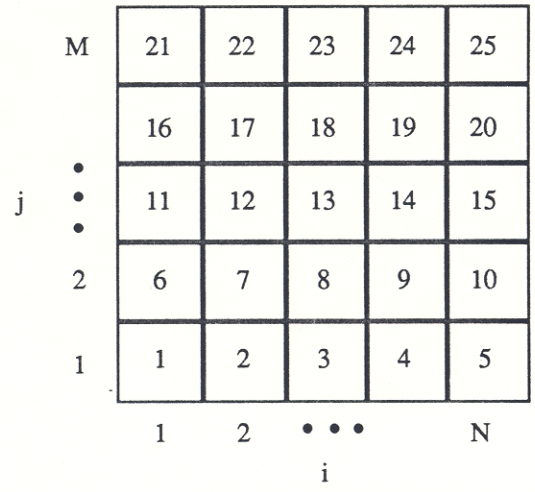

Fig. 1 Example x-y Grid Space for a Reservoir Simulation

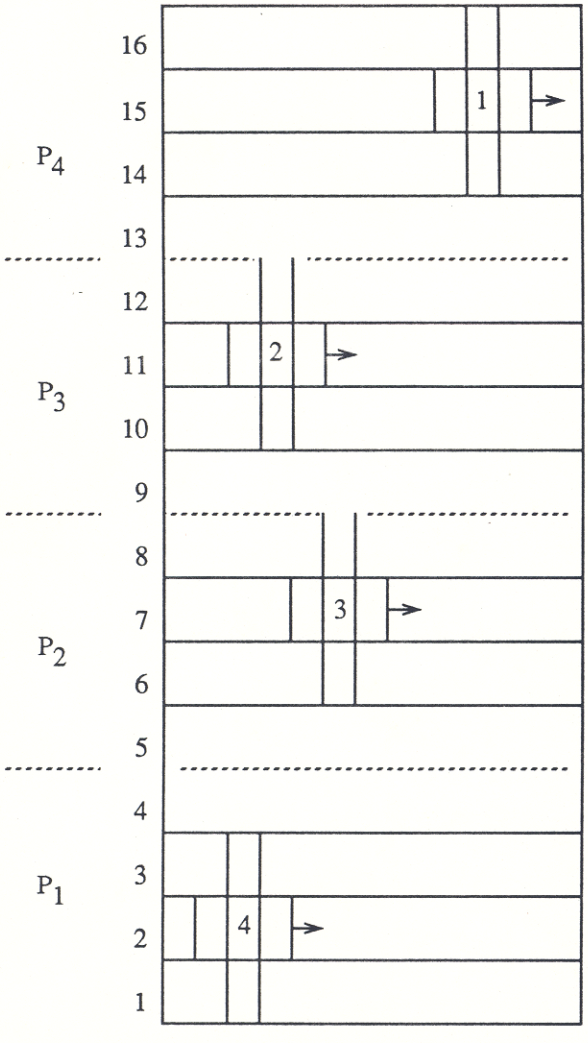

Fig. 2. Pipeline Model Using Four Processors each at a Different Iteration over the x-y Grid Space of a Reservoir

1. WHOAMI Processor determines which processor it is and its neighbors.
2. RECEIVE INPUT data (one time) from the cube manager describing the reservoir model. The processor determines which rows of the x-y grid it is responsible for.
3. SEND BOTTOM row of data to the previous neighbor (except processor 1).
4. TIME LOOP

   IF processor 1:

   Code for processor 1

   IF processor 2..p-1:

   Code for processors 2..p-1
   (See Figure 4 for details)

   IF processor p:

   Code for processor p

   UNTIL end of time

Fig. 3. Pipeline Model for Each Processor 1..p

1. RECEIVE BOTTOM row of data from the previous neighbor. This includes iterative information propagating through the pipeline.
2. IF this is a termination message THEN SEND termination message to the next neighbor and STOP.
3. IF this is the start of a new time unit THEN alter the rows to the new linear system.
4. CALCULATE FIRST ROW
5. SEND BOTTOM row of data to previous neighbor.
6. CALCULATE MIDDLE ROWS.
7. RECEIVE TOP row of data from next neighbor.
8. CALCULATE TOP ROW.
9. SEND TOP row of data to next neighbor.

Fig. 4. Pipeline Model for Each Processors 2..p-1

Table I.

CPU Results for Scaled Problems Using SOR3
All times are in Seconds

	GRID SIZE OF THE PROBLEM					
Nodes Used	96x96	132x132	192x192	264x264	384x384	528x528
1	3369.4*	8590.0*	25189.2*	49567.8*	104251.4*	197100.3*
2	1865.3	---	---	---	---	---
4	1021.0	2462.7	---	---	---	---
8	576.0	1359.5	3274.3	---	---	---
16	353.6	828.9	1874.6	4110.3	---	---
32	243.3	646.5	1194.5	2753.5	5355.7	---
64	---	540.1	847.2	2183.6	3606.4	7572.2

* Estimated CPU Times using $T_1 = ciN^2$

Table II.

Scaled Problems Using SOR3

Nodes Used	Size Grid	Scaled Speedup	Efficiency	Computation to Commun. Ratio
2	96x96	1.81	90.5 %	48/1
4	132x132	3.49	87.3	33/1
8	192x192	6.59	82.4	24/1
16	264x264	12.06	75.4	16.5/1
32	384x384	19.47	60.8	12/1
64	528x528	26.03	40.7	8.25/1

ACKNOWLEDGEMENTS

The work reported in this paper was supported by Amoco Production Company Research Center, Tulsa, Oklahoma. Special thanks to Dr. Stuart L. Scott for his invaluable advice, consultations, and thorough reading of the manuscript, which resulted in numerous improvements and clarifications.

REFERENCES

1. Aykanat, C. and Ozguner, F., "Large Grain Parallel Conjugate Gradient Algorithms on a Hypercube Multiprocessor", Proceedings of the 1987 International Conference on Parallel Processing, pp. 641-644, August, 1987.
2. Denning, P. J., "The Science of Computing - Speeding up Parallel Processing", American Scientist, vol. 76, pp. 347-349, July-August, 1988.
3. Gustafson, J. L., Monty, G. R., Brenner, R. E., "Development of Parallel Methods for a 1024-Processor Hypercube", SIAM Journal on Scientific and Statistical Computing, vol 9, no. 4, July, 1988.
4. Patel, N. R. and Jordan, H. F., "A Parallelized Point Successive Over-Relaxation Method on a Multiprocessor," Parallel Computing, pp. 207-222, 1984.
5. Scott, S. L. "Computer Design to Optimize Matrix Operations and Solution of a Two-Dimensional Single Phase Reservoir Simulator, M.S. Thesis, The University of Tulsa, 1985.
6. Wainwright, R. L., "Acceleration Techniques for a class of x-projection Methods for Solving Systems of Linear Equations", Computers and Mathematics with Applications, vol. 5, no. 1, 1979.
7. Wainwright, R. L., "Four Dimensional x-Projection Method (with acceleration techniques) for Solving Systems of Linear Equations", Computers and Mathematics with Applications, vol. 8, no. 4, 1882.
8. Wainwright, R. L., "Deriving Parallel Computations from Functional Specifications: A Seismic Example on a Hypercube", International Journal of Parallel Programming, vol. 16, no. 3, June, 1987, (appeared May, 1988).