The Journal of the Iowa Academy of Science: JIAS

Volume 101 | Number Article 11

1994

Parallel Algorithms for Solving Partial Differential Equations

Stacy Pschenica
Iowa State University

Jour. Iowa Acad. Sci. 101(2)70-72, 1994

Parallel Algorithms for Solving Partial Differential Equations+

STACY PSCHENICA

Department of Aerospace Engineering, Iowa State University

This paper describes the use of a parallel computer system in applying a finite difference method to solve various types of partial differential equations. A sequential implementation of this method was made parallel by essentially spatially decomposing the problem domain into pieces that could be separately analyzed and assigning each piece to a processor in a large, multi-processor system. Although this approach is attractive in theory, it suffers in practice because of inter-processor interactions. Thus, particular emphasis was placed on developing efficient methods for the sharing of information among the processors.

Initially, a simple two-dimensional board game, life, was implemented and used to develop inter-processor communication techniques. These techniques were then applied to parallel solutions of the one-dimensional wave equation and of the two-dimensional Laplace's equation. Different initial conditions were used to illustrate the feasibility of this approach.
INDEX DESCRIPTORS: Parallel computer systems, wave equation

Partial differential equations play a key role in many fields of science and engineering. Because of the complexity involved in solving partial differential equations, it is often desired to use computers to calculate their solutions numerically. One common approach for solving such equations is the finite difference method which iterates toward a solution given some initial conditions. This approach lends itself particularly well to computer solution because of the repetitive nature of the solution algorithm.

Historically, finite difference methods have been implemented on large, high-performance computer systems and solution methods on such systems are well known. Recently, however, the emergence of massively parallel computer systems has suggested an alternative computational approach. The basic idea is to divide the solution space into pieces, with each piece assigned to a particular processor in the parallel system. Thus, rather than using one large processor to solve a given partial differential equation, several small processors are used. This parallel processing approach can in some cases be faster, cheaper, and easier to enhance than a single processor solution.

Unfortunately, parallel processing systems have not always realized their full potential. In theory, if a problem can be properly decomposed, a K-processor solution will execute approximately K times as fast as a single processor solution. Ideal performance with parallel systems is unattainable due to difficulty in decomposing the problem and communication activities generally required among processors.

This project focuses on the use of parallel computer systems in solving certain partial differential equations arising in computational fluid dynamics. The hypothesis presented here is that partial differential equations can be effectively solved on parallel computer systems and that solutions on such systems can be competitive with solutions on large, high-performance systems. Two types of partial differential equations, the one-dimensional wave equation and the two-dimensional Laplace's equation, were chosen for this study. The finite difference method was used with the problem decomposed spatially over the domain of interest. This is a natural method of problem decomposition and isolates the "cost" of the parallel approach to that of inter-processor communication.

A method of assigning problem parts to processors was developed and implemented for a mesh-connected parallel computer system. Inter-processor interactions were specified and tested for a two-dimensional board game called life. The rules of life and the inter-processor activities for life behave very much like finite difference solutions to partial differential equations as far as incrementing the current values to the next time step. Thus, the methods developed for modeling the game of life will carry over directly to the solution of partial differential equations. That carryover is demonstrated using the wave equation and Laplace's equation.

SOLUTION APPROACH

This game of life works with an array (much like a checkerboard) consisting of mostly zeros and a small pattern of ones. The ones are known as "counters". Each counter has eight neighbors. Four are adjacent diagonally and four are adjacent orthogonally. The "genetic rules" for life are given by John Conway, a mathematician at Gonville and Caius College of the University of Cambridge, and are as follows:

Survivals. Every counter with two or three neighboring counters survives for the next generation.

Deaths. Each counter with four or more neighbors dies (is removed) from overpopulation. Every counter with one neighbor or none dies from isolation.

Births. Each empty cell adjacent to exactly three neighbors — no more, no fewer — is a birth cell. A counter is placed in this cell at the next move.

It is important to realize that all births and deaths take place simultaneously — all of the cells are evaluated at the same instant. It is not until the next iteration that they are changed.

The game of life was originally programmed for a sequential computer. From this sequential program a parallel program was created. The behavior of this parallel program is illustrated below. For simplicity, assume sixteen processors (numbered 0-15) are being used. Processor 0 is used as the main communicator and it breaks down each processor's share as indicated in Figure 1.

Processor 0 initially holds the entire original array, but to reduce computation time it gives 1/16 of the original array to each of the processors to evaluate. In order for each cell to have direct access to its full eight neighbors, each processor receives an extra row of numbers around the outside of its own sub-matrix. These extra numbers are composed of the adjacent rows or columns from the neighboring processors. This is illustrated in Figure 2.

The algorithm for interprocessor interactions used in the aforementioned method is as follows:

— processor 0 holds the original array

— processor 0 then sends each processor its share along with the extra boundary rows

— each processor copies its share into a small temporary array

— the processors check each cell's neighbors in their share arrays

+This work was made possible in part by funds from Hewlett Packard, NSF grant nos. USE-8951656 and USE-9053807, and by the Scalable Computing Laboratory which is funded by Iowa State University and the Ames Laboratory, U.S. DOE, Contract No. W-7405-ENG-82.

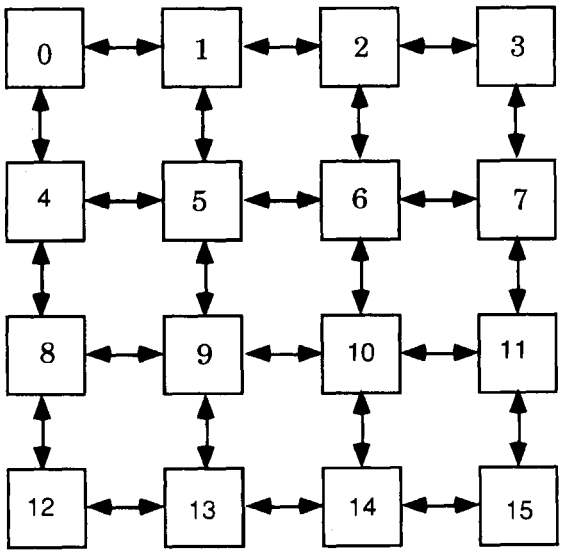

Fig. 1. Layout of 16 processors for evaluating a matrix.

then record the necessary changes in their temporary arrays
— each processor copies its temporary array (which holds all the changes) into a final array — the final array contains only the processor's share and not the extra boundary rows
— each processor sends its evaluated array back to processor 0
— processor 0 reassembles the large array, prints it, and sends out the shares for the next iteration

Using a sequential program as the basis for a parallel program is not necessarily the best approach. Although the work load for evaluating the matrix is divided among sixteen processors, the communication can only go through processor 0. This leaves processor 0 with an excess amount of work while the other processors are waiting for their assignments.

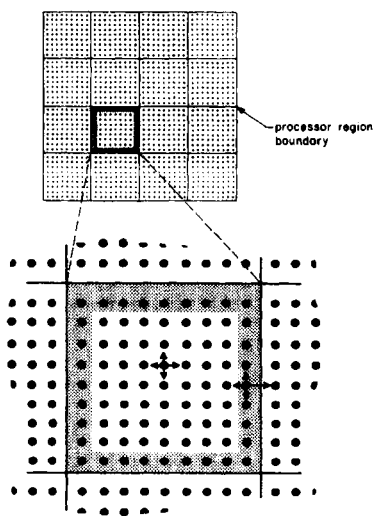

Fig. 2. Breakdown of matrix to each processor, showing the shared rows and columns.

This leads to a new method. This method proposes defining the problem then going about writing a parallel program with numerous processors in mind. A new program was written concerning the game of life. This method looks at the layout of processors, say the same sixteen, and determines each processor's four neighboring processors. These neighbors would need the processor's outer rows and columns for the updating of their own arrays.

For example, in the processor layout in Figure 1, processor 5's neighbors are north = 1, south = 9, east = 6, west = 4. Processor 1 would receive 5's first row, 9 would receive 5's last row and 4 and 6 would receive the left-most and right-most columns respectively. This greatly reduces the communication costs since the processors can immediately get their messages and proceed with their tasks.

The algorithm for this method is as follows:

— processor 0 holds the original array
— processor 0 sends each processor its share along with the extra boundary rows
— each processor copies its share into a small temporary array
— the processors check each cell's neighbors in their share arrays then record the necessary changes in their temporary arrays
— each processor copies its temporary array, which holds all the changes, into a final array — the final array contains only the share and not the extra boundary rows
— each processor sends its final evaluated array to processor 0, which reassembles and prints the large array
— starting with this next iteration, the processors identify their neighbors
— each processor sends its neighbors the respective rows or columns to satisfy the boundary conditions
— the processors evaluate their shares as before
— processor 0 receives copies of each processor's share, reassembles the large array, and prints
— the iterations continue with the processors communicating amongst themselves

PARTIAL DIFFERENTIAL EQUATIONS

The one-dimensional wave equation is given by:

$$\frac{1}{c^2} * \frac{\partial^2 \psi}{\partial t^2} - \frac{\partial^2 \psi}{\partial x^2} = 0$$

where $\varphi = \varphi(x,t)$, x = position, t = time

One approximation of this partial differential equation is given by the finite difference equation:

$$\psi_i(t+\Delta t) = 2\psi_i(t) - \psi_i(t-\Delta t) + \tau^2[\psi_{i-1}(t) - 2\psi_i(t) + \psi_{i+1}(t)]$$

$$\text{where} \quad \tau = \frac{c\Delta t}{\Delta x}$$

Note that this finite difference equation very much resembles a set of genetic rules similar to those used in the game of life. In this case, since the equation is one-dimensional, the solution is stored as a column of numbers (as opposed to a two-dimensional array). Each element in the column represents the value of φ at a particular point in time, say t. To start the evaluation process, two initial columns are defined according to the given initial conditions. (These two initial columns give the values for φ at time 0 and time $0+\Delta t$.) The second column is where the finite difference equation is applied while the first column is used as a last iteration comparison. The program uses these two columns to determine values for φ at time $2\Delta t$. These values are stored in a third column. When the program finishes computing the values for the third column it then repeats the process.

The finite difference equation can be looked at in terms of rows and columns:

$$\psi[j][n+1] = 2\psi[j][n] - \psi[j][n-1] + \tau^2(\psi[j+1][n] - 2\psi[j][n] + \psi[j-1][n])$$

The program is simply comparing the values in neighboring cells of the column and the present and past iterations. Procedures for the solution of the wave equation were drawn from two different methods. The first was to find a sequentially-based parallel algorithm; the second was to come up with a strictly parallel algorithm.

The sequentially-based parallel algorithm for the wave equation is as follows:

— processor 0 holds original two columns
— processor 0 gives each processor its own share of the second column and the necessary boundary numbers
— each processor copies its share into a temporary array
— each processor looks at its share and makes the necessary changes in this temporary array
— processor 0 receives each processor's share, reassembles, then prints
— iterations continue

The following is a strictly parallel algorithm:

— processor 0 holds original two columns
— processor 0 gives each processor its share of the second column and the necessary boundary numbers
— each processor copies its share into a temporary array
— each processor looks at its share and makes the necessary changes in this temporary array
— processor 0 receives a copy of each processor's share, reassembles, and prints
— starting with this next iteration each processor recognizes its neighbor
— each processor sends its first and last numbers to its adjacent neighbors, respectively
— each processor evaluates its share as before
— processor 0 receives a copy again to print
— the iterations continue as the processors communicate amongst themselves

The two-dimensional Laplace's equation is given by:

$$\nabla^2\phi = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = 0$$

$$\text{where } \phi = \phi(x,y)$$

The iterative technique adopted for the finite difference equation is Jacobi iteration. The Jacobi update method for the next iteration is written as:

$$\phi_i^{(k)} = \frac{1}{4}\left[\phi_{(i-x)}^{(k-1)} + \phi_{(i-y)}^{(k-1)} + \phi_{(i+x)}^{(k-1)} + \phi_{(i+y)}^{(k-1)}\right]$$

where the subscript represents position and the superscript represents time.

Because Laplace's equation is two-dimensional, its solution is stored as a matrix. Before any evaluation takes place an original matrix is filled with numbers according to the given boundary conditions. The update for each cell in its next time iteration is simply the average of its four orthogonal neighbors. To track the communications initially between a cell's neighbors, the game of life was implemented.

The finite difference equation for Laplace's equation is basically just another set of genetic rules for the game of life. Instead of comparing eight neighbors for the next iteration, the program only has to take the average of four neighbors. The game of life modeled the communication procedures between the processors for use in solving two-dimensional partial differential equations.

RESULTS

The solution for the one-dimensional wave equation was obtained through running a parallel program written in the C programming language on a 64 node nCUBE 2 computer system. The program run time was tested on various numbers of processors and compared to the time taken by using only one processor. These data are recorded in Table 1.

Table 1. Sequential Algorithm Made Parallel

Number of Processors	Total Time	Speedup
1	23.11484	-----
2	11.584766	1.995
4	5.833079	3.963
8	2.944459	7.850
16	1.500456	15.405
32	.778641	29.686
64	.418033	55.294

Ideally, an N-processor system should yield a program speedup of N. In practice, communication costs will lower the speedup. The speedups obtained (e.g. 55.3 for 64 processors) were very encouraging.

Note now that the above comparison was against the time taken by one processor to run a parallel algorithm program. To get a true idea of speedup, the parallel times should be compared to a strictly sequential program. A sequential program for the one-dimensional wave equation was implemented and the run time was recorded. The speedup of the parallel version versus the sequential version is noted in Table 2. The speedups in this case were even better than before.

Table 2. Speedup of Parallel Algorithm

Number of Processors	Speedup
2	2.182
4	4.333
8	8.584
16	16.845
32	32.460
64	60.461

Experiments are currently being tried to test the sensitivity of our solution to the array size. Intuitively, our solution should be more efficient the larger the array. Preliminary results support this, but more tests are planned.

CONCLUSIONS

The game of life was implemented in a parallel version of the C programming language on a 64-node nCUBE 2 computer system. This implementation was then used with only slight modification to provide a parallel solution to both the one-dimensional wave equation and the two-dimensional Laplace's equation. Although performance data are not yet complete, preliminary results are very encouraging. The speedups obtained for the wave equation solution suggest that communication costs are not necessarily going to be a big obstacle in solving many partial differential equations numerically on parallel computer systems. The implementations will be further tested with a broad range of boundary conditions and performance data obtained as a part of future work. Results are pending for the solution to Laplace's equation.