CONF-8904180--1

PARALLEL SOLUTIONS OF A 2-D PHASE CHANGE PROBLEM ON A HYPERCUBE

H. N. Narang
Computer Science Department
Tuskegee University
Tuskegee, Alabama 36088

CONF-8904180--1

DE89 009619

J. B. Drake
Mathematical Sciences Section
Engineering Physics and Mathematics Division
Oak Ridge National Laboratory
Oak Ridge, Tennessee 37831-8083

DISCLAIMER

* This research was sponsored by the Applied Mathematical Sciences Research Program, Office of Energy Research, U.S. Department of Energy under contract DE-AC05-84OR21400 with the Martin Marietta Energy Systems, Inc.

Parallel Solution of a 2-D Phase Change Problem on a Hypercube

H. N. Narang

Department of Computer Science
Tuskegee University

J. B. Drake

Mathematical Sciences Section
Engineering Physics and Mathematics Division
Oak Ridge National Laboratory

Abstract

This report documents the model, algorithms, and results for a heat transfer problem with phase change simulated numerically on a hypercube parallel computer. The temperature and enthalpy distribution in a rectangular domain with first kind boundary conditions (temperature prescribed) is found as a function of time. The model involves a non-linear partial differential equation with a constitutive relation between enthalpy and temperature. The solution to the system is obtained by finite difference approximation with both explicit and implicit methods. In the implicit case, a successive overrelaxation technique with red, black ordering is implemented. The efficiency of the explicit method is compared with that of the implicit method in a multiprocessor computing environment.

The design, development, and implementation of algorithms targeted INTEL's iPSC/2 hypercube involving up to 64 processors. The computational workload was uniformly divided among processors as far as possible by dividing the finite difference grid into several strips and assigning each strip to a processor.

The results, depicted in terms of tables , show the effect of relaxation parameter, and stepsize on the efficiency of the parallel algorithms. This is contrasted with the sequential algorithms.

1. Introduction

The study of phase change problems has been of special interest to engineers, scientists, and mathematicians for a long time. This interest has been sustained because of the importance of the problems arising in various fields. Examples include manufacturing steel, melting of glaciers in arctic regions and their effect on the global temperature, laser annealing technology and manned space power stations to name a few. The classical formulation of the problem is due to Stefan in 1889 [11]. Stefan's model for ice melting tracked the melting front along with the temperature distribution. Since then many researchers have contributed to the field which is now broadly known as "moving boundary problems". This field includes not only melting but also solidification as well as problems without change of phase, e.g., diffusion of a substance in liquid. Due to the presence of a moving free boundary these problems are essentially non-linear. Analytical solutions are possible in only a few special cases. Carslaw and Jaeger [12] offer several of these.

As the region of interest in which the solution is to be obtained changes with time, numerical methods based on front tracking are complicated by the necessity to move the finite difference grid or to transform the coordinates of the problem. In 1946, Eyres [13], presented a model for this kind of problem which alleviates the need to explicitly track fronts. The heat transfer literature refers to this model as the enthalpy formulation. In this formulation, the movement of the front is embedded in the system consisting of a partial differential equation involving enthalpy and temperature and an auxiliary relationship between them. This formulation has advantages numerically since the problem domain is fixed [Wilson, Solomon, Boggs].

Various researchers, Williams and Wilson [3], have attempted solutions to these problems using super-computers to advantage. With the advent of parallel, distributed processing, the design and implementation of algorithms for solving moving boundary problems in parallel is of interest. The granularity of the parallelism inherent in this type of moving boundary problem is well suited to parallel computation. In addition to the diffusion calculation, the state variables must be updated from the conserved quantity. For example, in a phase change problem, the temperature must be found from the enthalpy at each node of the mesh. This computation of the state variable, temperature, does not vectorize easily as it involves a logical switch. However, this computation may proceed with perfect parallelism on a multiprocessor.

In this report we design and implement parallel algorithms for moving boundary problems. A 2-D change of phase problem is used as a model problem and the simulation performed on an Intel iPSC/2 hypercube with a maximum of 64 processors. Both

explicit and implicit algorithms are developed. Results indicate the effect of problem size, relaxation parameter, and step-size on the performance of the parallel algorithms. Performance studies also show the efficiency of parallel solutions compared with sequential solutions for explicit and implicit methods. The front movement with time is shown for an example computation. Finally, some difficulties encountered with implicit SOR convergence, and the development of a strategy to overcome the difficulties are discussed.

2. Model Formulation

The enthalpy formulation of Stefan problems can be written in general, Williams and Wilson [3], as:

$$\rho e_t = div\,(K(T)\,grad\,T) \tag{2.1}$$

$$e(T) = \int_{T_m}^{T} c(z)dz \tag{2.2}$$

where e is enthalpy, T is temperature, $K(T)$ is thermal conductivity, $c(T)$ is thermal heat capacity and ρ is the density of the medium. Many authors have considered this formulation for Stefan problems. For example, see Rose [14], White [15], and Solomon [16].

For the 2-phase, 2-D melting problem with constant heat capacities, the auxiliary relation takes the following form:

$$e = \begin{cases} c_S(T - T_m) & T < T_m \\ [0, H] & T = T_m \\ c_L(T - T_m) + H & T > T_m \end{cases} \tag{2.3}$$

where c_S, c_L are the heat capacities of solid and liquid phases respectively, T_m is the melting temperature, and H is the latent heat.

The inversion of (2.3) can be written as:

$$T = \begin{cases} T_m + e/c_S & e \le 0 \\ T_m & 0 < e < H \\ T_m + (e - H)/c_L & e \ge H \end{cases} \tag{2.4}$$

Although, (2.3) is a multivalued relationship, (2.4) is a single valued function.

As an illustrative example, we will consider the melting of a frozen material, initially solid. A 2-D cross section 1.5×1.5 meters with physical properties given below will be considered. The boundaries of the material are maintained at 10 degrees C. The problem is to predict the distribution of the liquid and solid phases in the region as the material melts and to predict the temperature distribution. We wish to analyze the performance of a multiprocessor in performing calculations of this sort.

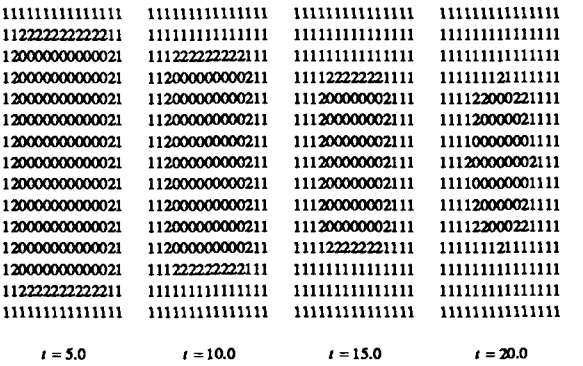

Figure 1. Front Movement for the Example Problem

3. Explicit Method

The enthalpy, temperature equation (2.1) was approximated by finite differences with an explicit method, as follows:

$$\rho \frac{e_{i,j}^{n+1} - e_{i,j}^{n}}{\Delta t} = \frac{q_{i-\frac{1}{2},j}^{n} - q_{i+\frac{1}{2},j}^{n}}{\Delta x} + \frac{q_{i,j-\frac{1}{2}}^{n} - q_{i,j+\frac{1}{2}}^{n}}{\Delta y} \tag{3.1}$$

where $q_{i-\frac{1}{2},j}$, $q_{i+\frac{1}{2},j}$ are heat fluxes on the left and right of the cell centered at (i, j) respectively and $q_{i,j-\frac{1}{2}}$, $q_{i,j+\frac{1}{2}}$ are fluxes at the bottom and top boundaries respectively. The superscript n refers to the time level.

The conductivities $K_{i-\frac{1}{2},j}$, $K_{i+\frac{1}{2},j}$, $K_{i,j-\frac{1}{2}}$, $K_{i,j+\frac{1}{2}}$ are conductivities at left, right, bottom, and top faces of the cell (i, j) respectively and are calculated from:

$$K_{i\pm\frac{1}{2},j} = \frac{K(T_{i,j}) + K(T_{i\pm1,j})}{2} \tag{3.2}$$

$$K_{i,j\pm\frac{1}{2}} = \frac{K(T_{i,j}) + K(T_{i,j\pm1})}{2} \tag{3.3}$$

The equation (3.1) is the update in enthalpy per unit volume in terms of the enthalpy at the previous time step plus the net inflow of the heat in the cell during the time increment Δt.

Once enthalpy has been obtained, the temperature can be determined from the discrete version of the auxiliary equation (2.4).

3.1 System Configuration and Algorithms

The design of a parallel algorithm often begins by identifying independent tasks in a sequential algorithm. For problems involving the solution of partial differential equations the same operations are being repeated at each finite difference grid point and so a spatial assignment of tasks to processors provides a simple divide and conquer strategy for identifying parallelism. For a distributed memory parallel computer, the data must also be divided among the processors. Again the spatial splitting is natural. Since each individual processor has a piece of the grid,

and not the whole grid, usually it must communicate with other processors during the solution of the problem. For example, a piece of the grid requires boundary information before it can be updated in a time marching scheme. Another example requiring communication among the processors arises when computing a norm of the computed solution. Each processor can compute a norm of its piece of the solution, a local norm, but communication is required to find the global norm.

The structure of the computer program reflects the division of tasks between the host and the node processors. A host program must provide for all access to input and output but does little or no computation. The node programs must provide for receipt of initial information from the host and for return of results to the host after completion of the calculation. Message passing generally occurs in the algorithm whenever a processor has a value that is (or will be) needed by other processors. The message passing thus provides a low level of synchronization among the tasks being performed in parallel. The chief constructs for message passing are simply a **send** and a **receive**.

By restricting our attention to a ring topology the grid is simply divided among the processors in strips. The number of grid points in a given strip will determine the load distribution of the computation. In our load distribution strategy, the last processor always gets the remainder of grid rows divided by the number of processors plus the usual load. In some cases it may get much less load than other processors, in other cases it may get much more than others. This will degrade the efficiency of the algorithms.

3.2 Analytical Investigation into the Parallel Efficiency

Let us assume that there are p processors, each with $M \times N$ grid to work on. Let us also assume that each node point requires A operations to update a grid point, and σ time / operation. Let δ be start up time before communicating internal boundary elements (a row). Then if t is the time, we have

$$t_{seq} = MN(A\sigma)$$

$$t_{par} = AMN\frac{\sigma}{p} + 2\delta + 2\tau N$$

(2N elements exchanged by each processor, bottom row and top row)

(σ is start up time for each row communication)

$$S = \text{speed up} = \frac{t_{seq}}{t_{par}}$$

$$= p\left[\frac{AMN\sigma}{AMN\sigma + 2p(\delta + \tau N)}\right]$$

$$= p\left[\frac{A\sigma}{A\sigma + 2p(\frac{\delta}{MN} + \frac{\tau}{M})}\right]$$

when $M \to \infty$

$$S = p$$

when $N \to \infty$

$$S = p\frac{A\sigma}{A\sigma + 2p\frac{\tau}{M}}$$

3.3 Numerical Results and Discussion

The numerical solution to the above problem was obtained with the algorithms described, which were written in C-language on Intel's iPSC/2 hypercube. Results were obtained both for small and large finite difference grids ranging from $M \times N = 15 \times 15$, for the small grids, to 480×480, for the large grids. Timing results were obtained using the iPSC CLOCK function and are accurate to $\pm$ 15 ms. All times are reported in milliseconds.

Conceivably, the larger the grid size, the larger will be the computational time as well as the communication time. How the CPU time varies as the grid size increases is shown in the table below.

TABLE 1.

EXPLICIT METHOD: EFFECT OF GRID SIZE ON CPU TIME

processors = 32

M	N	CPU Time/S/P	CPU Time/S/1	Speed Up
80	80	268.2	2837	10.5
160	160	368.9	11331	30.7
240	240	1314.2	25477	19.4
320	320	1433.6	45277	31.5
400	400	3057.1	70493	23.0
480	480	3205.2	101821	31.7

Since a parallel algorithm should give better performance than a sequential algorithm, some results were obtained for larger grids (480×480) both with the parallel and sequential algorithms (only one processor solving the whole grid). The results are depicted in table the below.

TABLE 2.

EFFECT OF # OF PROCESSORS ON THE CALCULATION OF A FIXED LARGE GRID (480 × 480)

# processors	CPU Time/S/P	CPU Time/S/1	Speed Up
4	25468.7	101821.2	4.00
8	12744.8	101821.2	7.99
16	6382.4	101821.2	16.03
32	3205.4	101821.2	31.77
64	2098.6	101821.2	48.52

TABLE 3.

EXPLICIT METHOD: EFFECT OF NUMBER OF PROCESSORS ON A LARGE FIXED GRID (320 × 320)

# Processors	CPU Time/S/P	CPU Time/S/1	Speed Up
4	11326.9	45277.1	4.00
8	5671.1	45277.1	7.98
16	2844.5	45277.1	15.92
32	1434.6	45277.1	31.56
64	734.9	45277.1	61.61

Tables 2 and 3 show that efficiences near 100% are achievable when the grid is evenly divisible by the number of processors. Also, for an evenly divisible grid of these sizes the optimal efficiency is with 32 processors.

4. Implicit Method

In this method, the approximation to partial derivatives in the model equations are replaced by forward difference quotients at the advanced time instead of the previous time. Thus equation (2.1) assumes the form:

$$\rho \frac{e_{i,j}^{n+1} - e_{i,j}^{n}}{\Delta t} = \frac{q_{i-\frac{1}{2},j}^{n+\theta} - q_{i+\frac{1}{2},j}^{n+\theta}}{\Delta x} + \frac{q_{i,j-\frac{1}{2}}^{n+\theta} - q_{i,j+\frac{1}{2}}^{n+\theta}}{\Delta y} \quad (4.1)$$

where, θ is the degree of implicitness and lies between 0 and 1, and

$$f^{n+\theta} = \theta f^{n+1} + (1-\theta) f^{n}$$

The quantities $q_{i-\frac{1}{2},j}$, $q_{i+\frac{1}{2},j}$, etc, are heat fluxes as before, where superscripts $n+\theta$ imply that the values of these fluxes are weighted combination of the fluxes at times n, and $n+1$.

For $\theta = 0$, it becomes an explicit method, and at $\theta = 1$ it becomes fully implicit. At $\theta = \frac{1}{2}$, it is well known as Crank-Nicholson scheme. The quantities $q_{i-\frac{1}{2},j}$, $q_{i+\frac{1}{2},j}$, $q_{i,j-\frac{1}{2}}$, $q_{i,j+\frac{1}{2}}$ are defined as in the explicit method with corresponding definitions for $K_{i-\frac{1}{2},j}$, $K_{i+\frac{1}{2},j}$, etc.

With above definitions, the equation (4.1) assumes the form

$$\begin{aligned} e_{i,j}^{n+1} = b_{i,j}^{n} &+ \frac{\theta \Delta t}{\rho \Delta x^2}\Big[K_{i+\frac{1}{2},j}^{n+1} \; (T_{i+1,j}^{n+1} - T_{i,j}^{n+1}) \\ &- K_{i-\frac{1}{2},j}^{n+1} \; (T_{i,j}^{n+1} - T_{i-1,j}^{n+1})\Big] \\ &+ \frac{\theta \Delta t}{\rho \Delta y^2}\Big[K_{i,j+\frac{1}{2}}^{n+1} \; (T_{i,j+1}^{n+1} - T_{i,j}^{n+1}) \\ &- K_{i,j-\frac{1}{2}}^{n+1} \; (T_{i,j}^{n+1} - T_{i,j-1}^{n+1})\Big] \end{aligned} \quad (4.2)$$

where

$$\begin{aligned} b_{i,j}^{n} = e_{i,j}^{n} &+ \frac{(1-\theta)\Delta t}{\rho \Delta x^2}\Big[K_{i+\frac{1}{2},j}^{n} \; (T_{i+1,j}^{n} - T_{i,j}^{n}) \\ &- K_{i-\frac{1}{2},j}^{n} \; (T_{i,j}^{n} - T_{i-1,j}^{n})\Big] \\ &+ \frac{(1-\theta)\Delta t}{\rho \Delta y^2}\Big[K_{i,j+\frac{1}{2}}^{n} \; (T_{i,j+1}^{n} - T_{i,j}^{n}) \\ &- K_{i,j-\frac{1}{2}}^{n} \; (T_{i,j}^{n} - T_{i,j-1}^{n})\Big] \end{aligned} \quad (4.3)$$

In this method, $b_{i,j}^{n}$ assumes the role of $e_{i,j}^{n}$ of the explicit method.

The equations (4.1), (4.2), (4.3) make the system implicit in that one has to solve the whole system simultaneously. With the natural ordering, the coefficient matrix assumes a banded diagonal structure. The solution of such a non-linear system can be found by quasi-linearizing the system at every time step with coefficients at the old time step and then applying some direct method. Or one can solve the system iteratively by assuming a starting solution and iterating until convergence is obtained.

We shall solve the system by a well known iterative method **"successive over relaxation"** commonly known as SOR. In this method, one starts with an initial approximation to the solution (usually initial conditions) and then obtains updated solutions by successively making weighted residuals go to zero. The relaxation parameter, ω, the weighting factor for residuals is normally chosen to lie between 1, and 2, for faster convergence to the solution. One may refer to Young [6] for details of this method.

To apply this method, we write equation (4.2) in the following fashion:

Let $\tau_{i,j}^{p}$ denote the p th approximation to $T_{i,j}^{n+1}$, and $\eta_{i,j}^{p}$ as the p th approximation to $e_{i,j}^{n+1}$. Then if we define

$$\begin{aligned} z_{i,j}^{p} = b_{i,j}^{n} &+ \frac{\theta \Delta t}{\rho \Delta x^2}(K_{i+\frac{1}{2},j}^{p} \, \tau_{i+1,j}^{p} + K_{i-\frac{1}{2},j}^{p+1} \, \tau_{i-1,j}^{p+1}) \\ &+ \frac{\theta \Delta t}{\rho \Delta y^2}(K_{i,j+\frac{1}{2}}^{p} \; \tau_{i,j+1}^{p} + K_{i,j-\frac{1}{2}}^{p+1} \; \tau_{i,j-1}^{p+1}) \end{aligned} \quad (4.4)$$

and

$$c^p_{i,j} = \frac{\theta \Delta t}{\rho \Delta x^2}(K^p_{i+\frac{1}{2},j} + K^{p+1}_{i-\frac{1}{2},j}) + \frac{\theta \Delta t}{\rho \Delta y^2}(K^p_{i,j+\frac{1}{2}} + K^{p+1}_{i,j-\frac{1}{2}}) \quad (4.5)$$

equation (4.2) assumes the iterative form

$$\eta^{p+1}_{i,j} = z^p_{i,j} - c^p_{i,j}\,\tau^{p+1}_{i,j} \quad (4.6)$$

As (4.6) is the iterative form of the finite difference approximation to enthalpy, and temperature equation (4.1) or (4.2), the auxiliary equation corresponding to (2.4) of explicit method assumes the following form:

$$\tau^{p+1}_{i,j} = \begin{cases} \dfrac{c_S T_m + z^p_{i,j}}{c^p_{i,j} + c_S} & z^p_{i,j} - c^p_{i,j} \cdot T_m \le 0 \\ T_m & 0 < z^p_{i,j} - c^p_{i,j} T_m < H \\ \dfrac{c_L T_m - z^p_{i,j} - H}{c^p_{i,j} + c_L} & z^p_{i,j} - c_{i,j}\, T_m \ge H \end{cases} \quad (4.7)$$

The role of e in explicit solution is played here by the quantity $z^p_{i,j} - c^p_{i,j}\,\tau^p_{i,j}$.

This happens because in the iterative process of updating the node values, these nodes are encountered first and thus attain newer values.

In the above iteration process, $\tau^{p+1}_{i,j}$ is in fact the Gauss-Seidel value. We shall call this updated value $\tilde{\tau}^{p+1}_{i,j}$. A successive overrelaxation method finds the updated value by taking a linear combination of the old value and the Gauss-Seidel updated value. Let us call the SOR updated value $\hat{\tau}^{p+1}_{i,j}$. Thus

$$\hat{\tau}^{p+1}_{i,j} = (1-\omega)\,\tau^p_{i,j} + \omega\,\tilde{\tau}^{p+1}_{i,j} \quad (4.8)$$

Elliot and Ockendon [8], suggest taking an SOR update for points inside the grid and Gauss-Seidel update near the phase (moving) front. This they suggest to avoid oscillations about T_m. We take this advice in our algorithm, i.e., we take

$$\tau^{p+1}_{i,j} = \begin{cases} \tilde{\tau}^{p+1}_{i,j} & \text{if } (\tau^p_{i,j} - T_m)(\hat{\tau}^{p+1}_{i,j} - T_m) \le 0 \\ \hat{\tau}^{p+1}_{i,j} & \text{otherwise} \end{cases} \quad (4.9)$$

4.1 Parallel (Implicit) SOR Algorithm

The SOR algorithm with the natural ordering of the equations is inherently sequential. For parallel implementation of the SOR algorithm, one needs to resort to the "red, black" ordering technique. Nodes are labeled alternately "red" and "black". In this scheme, "red" nodes will always have "black" neighbors, and "black" nodes will have "red" neighbors. A SOR sweep through the system is first made by updating red nodes, and then another sweep is made by updating black nodes to complete the processes. The details of this scheme can be found in Ortega [1], Evans [2].

4.2 Analytical Analysis of Speed-Up for the Implicit Algorithm

As in the case of the explicit algorithm, the speed up S, and efficiency E are given below based on operation counts. It may be remarked that the major difference comes from the fact that there will be more communication involved in the implicit method. The added communication comes from exchanges of boundary values for each red and black update. One may think it might involve more calculations per grid point, but if one remembers the fact that we are taking larger time steps (several times that of explicit time step) the advantages or disadvantages are not clear.

Ignoring the difference in the number of calculations needed to update a grid point the speed up and efficiency can be given by the following expressions:

$$S = \frac{A\,MN\,\sigma}{A\,MN\,\sigma/p + 4\delta + 4\tau N}$$

$$= p\left(\frac{A\,\sigma}{A\,\sigma + p\left(\frac{4\delta}{MN} + \frac{4\tau}{M}\right)}\right)$$

$$E = S/p = \frac{A\,\sigma}{A\,\sigma + p\left(\frac{4\delta}{MN} + \frac{4\tau}{M}\right)}$$

Again as $M \to \infty$, $E \approx 1$

and as $N \to \infty$ $E \approx \dfrac{A\,\sigma}{A\,\sigma + \dfrac{4\tau p}{M}} < 1$

4.3 Numerical Results and Discussion

The numerical results were obtained by simulation of the same problem as in the explicit case. The extra information needed to be supplied for the implicit SOR is the following:

convergence parameters:

$\omega = 1.6$
$\theta = 0.5$ (Crank–Nicholson)
$\varepsilon = 0.001$
$tmult = 10$

The initial, boundary conditions, and grid parameters as well as material properties etc, remained unchanged. Below are results for temperature calculations, front movements, and performance studies.

The table below shows the effect of the number of processors on the speed up for a fixed problem size.

TABLE 4.

SOR - IMPLICIT: EFFECT OF # OF PROCESSORS

ON SPEED UP OF A LARGE GRID

(320×320)

$\omega = 1.3$

# processors	CPU time/S/P	CPU time/S/1	speed up
4	372778	1486304	3.99
8	187022	1486304	7.95
16	94030	1486304	15.81
32	48090	1486304	30.99
64	26106	1486304	56.94

5. Performance Comparison Between Explicit and Implicit Algorithms

To compare the advantages of the parallel explicit with the parallel implicit algorithm, some results were obtained for large grid sizes. The table below shows a CPU time comparison for each grid size for a given time step. The explicit times given in the tables below are the execution times for a single explicit time step. Since the implicit method takes a larger time step, normalization is required for a comparison. The implicit times represent the execution time required to simulate one explicit time step.

TABLE 5.

Comparison of Explicit and Implicit Schemes for 20×15 Grid

processors = 4

$\omega = 1.3$

tmult	Explicit-CPU time/S/P	Implicit-CPU time/S/P
1	40	6302.7
5	40	1260.5
10	40	864.6
15	40	605.5
20	40	500.2
25	40	382.1
30	40	332.9
35	40	297.3
40	40	303.1

TABLE 6.

Comparison of Explicit and Implicit Schemes for 320×320 Grid

tmult = 20

$\omega = 1.3$

# processors	Explicit-CPU time/S/P	Implicit-CPU time/S/P
4	11326	186387
8	5671	9351
16	2844	4701
32	1434	2404
64	735	1305

The table shows that the implicit scheme is somewhat inferior to the explicit scheme in terms of CPU-time used. It may be remarked that this is primarily so because the red-black ordering in this scheme entails more information exchange at the internal boundaries than does the explicit scheme.

6. Conclusions

We have presented parallel algorithms, explicit and implicit SOR for a change of phase problem in two space dimensions. It seems that the explicit algorithm does better. This may be due to the fact that the implicit SOR algorithm involves red, black ordering which necessitates more exchanges of data among the strips. Since communication in distributed processing is still much more time consuming than computation, this makes the implicit SOR algorithm less attractive. As the communication rates improve this may change the picture. Therefore for larger grid sizes, implicit SOR is either worse or comparable to the explicit method in terms of CPU time consumed.

Our results show excellent speed up for the parallel algorithms on large grids. Large grid calculations may be infeasible on sequential computers due to memory limitations. A 480×480 grid was unable to execute on one node on the Intel's hypercube due to lack of memory.

Acknowledgements

The authors are thankful to George Wilson, Ray Flanery, Al Geist, Tom Dunigan, Chuck Romine, and Esmond Ng for many useful discussions.

References

1. James Ortega and Robert Voigt, "Solution to Partial Differential Equations on Vector and Parallel Computers", *SIAM Review* 1-240 (June 1985).
2. D. J. Evans, "Parallel SOR Iterative Methods", *Parallel Computers* **1**, 3-18 (1984).
3. M. A. Williams and D. G. Wilson, "IMPSOR, A Fully Vectorized Fortran Code for Three Dimensional Moving Boundary Value Problems with Dirichlet or Neumann Boundary Conditions", Report ORNL-6393, Oak Ridge National Laboratory, Oak Ridge, Tennessee, August 1987.
4. M. A. Williams, "Iterative Solution of a Non-Linear System Arising in Phase Change Problems", Report ORNL-6398, Oak Ridge National Laboratory, Oak Ridge, Tennessee, August 1987.
5. D. G. Wilson and R. E. Flanery, "Modeling Cyclic Melting and Refreezing in a Hollow Metal Canister", Report ORNL-6497, Oak Ridge National Laboratory, Oak Ridge, Tennessee, September 1988.
6. D. M. Young, "Iterative Solution of Large Linear Systems", Academic Press, 1971.
7. D. G. Wilson, A. D. Solomon, and P. Boggs, "Moving Boundary Problems", Academic Press, 1978.
8. C. M. Elliot and J. R. Ockendon, "Weak and Variational Methods for Moving Boundary Problems", Pitman Advanced Publishing Program, Boston, 1981.
9. Intel Corporation, "IPSC Program Development Guide", October 1986.
10. Intel Corporation, "IPSC/2 Users Guide", March 1988.
11. J. Stefan, "Uber einize Probleme der Theorie der Warmeleitung", *S. B. Wein. Akad. Mat. Natur.* 173-484 (1889).
12. Carslaw and Jaegar, "Heat conduction in Solids", Oxford University Press, London, 1959.
13. N. R. Eyres, et al, "The Calculation of Variable Heat Flow in Solids", *Philosophical Transactions of the Royal Society of London, Series A,* **240**, 1-57 (1946).
14. M. A. Rose, "Method for Calculating Solutions of Parabolic Equations", ed. J. K. Reid, Academic Press, New York, 1971.
15. R. E. White, "An Enthalpy Formulation of the Stefan Problem", *SIAM J. Num. Analysis* **19**, 1129-1157 (1982).
16. A. D. Solomon, "Some Remarks on Stefan Problem", *Math. Comp.* **20**, 347-360 (1976).