# Parallel Implementation of a High Order Implicit Collocation Method for the Heat Equation

Jules Kouatchou*
NASA Goddard Space Flight Center
Code 931
Greenbelt, MD 20771

September 8, 2000

**Abstract**

We combine a high order compact finite difference approximation and collocation techniques to numerically solve the two dimensional heat equation. The resulting method is implicit and can be parallelized with a strategy that allows parallelization across both time and space. We compare the parallel implementation of the new method with a classical implicit method, namely the Crank-Nicolson method, where the parallelization is done across space only. Numerical experiments are carried out on the SGI Origin 2000.

**Key words:** Implicit collocation methods, high-order compact scheme, iterative methods, parallel computers.

**Mathematical Subject Classification:** 65M06, 65N12.

## 1 Introduction

In [6], Jézéquel combined the second-order standard finite difference approximation for the spatial derivative and collocation technique for the time component to numerically solve the one dimensional heat equation. The method, called implicit collocation method (ICM), is unconditionally stable. Its principle is as follows: after discretization in space of the problem, the solution is approximated at each spatial grid point by a polynomial depending on time. The resulting derivation produces a linear system of equations. The order of the method is in space the order of difference approximation and in time the degree of the polynomial [2, 4, 5]. ICM when implemented on parallel computers, allows the parallelization across both time and space. Numerical experiments carried out by Jézéquel on the Cray T3E, show that the proposed ICM algorithm can achieve acceptable efficiency and under some simple assumptions, performs better (computing time) than standard explicit finite difference method [6].

Recently, we applied ICM to the two dimensional heat equation. For the spatial discretization, we used instead a fourth-order compact scheme that was shown to be more

*E-mail: kouatchou@gsfc.nasa.gov. The author is also affiliated with Morgan State University and his research was supported by NASA under the grant No. NAGS-3508.

accurate than the second-order one. Our numerical results, performed on a serial computers, showed again the stability of ICM [7].

Unfortunately, the previous analyses do not include a comparison between ICM and other classical implicit techniques to solve the heat equation. Though in ICM we can compute (utilizing a unique system of equation) the approximated solution at any point in a given time interval, its use introduces an additional dimension (degree of polynomials) to the problem. This increases the size of the system of equations to be solved, creates a lot of memory requirement and a limitation on the number of spatial grid point to be employed. Though ICM offers the parallelization across both time and space, it is not clear that it is more attractive, in terms of accuracy and parallel efficiency, compared to other implicit methods where the parallelization is done across space only.

In this paper, we compare ICM and a well known implicit technique, namely Crank-Nicolson method, in the numerical solution of the two dimensional heat equation when a fourth-order spacial discretization is employed. We use similar algorithms to implement both methods on a shared memory parallel computer. We estimate the memory and computational requirements of each method and determine the the set of conditions for which one method is better than the other one. In our numerical experiments, we present the accuracy of the approximated solution and the parallel efficiency of each method.

An outline of the paper is as follows. Section 2 presents the two different types of discretization. Section 3 discusses some computational considerations needed to implement the two methods. Section 4 describes the algorithms. Numerical experiments appear in Section 5. We formulate some conclusion in Section 6

## 2 Derivation of the System of Equations

We consider the two dimensional heat equation:

$$\frac{\partial u}{\partial t}(x,y,t) = \alpha^2 \left( \frac{\partial^2 u}{\partial x^2}(x,y,t) + \frac{\partial^2 u}{\partial y^2}(x,y,t) \right), \qquad (x,y,t) \in \Omega \times [0,\infty) \qquad (1)$$

where $\Omega = [0,1] \times [0,1]$, and with the initial condition

$$u(x,y,0) = \psi(x,y), \quad (x,y) \in \Omega,$$

and the boundary conditions

$$u(0,y,t) = f_0(y,t), \;\; u(1,y,t) = f_1(y,t), \;\; u(x,0,t) = g_0(x,t), \;\; \text{and} \;\; u(x,1,t) = g_1(x,t) \;\; \texttt{for} \;\; t \geq 0.$$

We assume that $f_0$, $f_1$, $g_0$ and $g_1$ are smooth functions in the variable $t$, i.e., their first derivatives with respect to $t$ exist and are continuous.

Let $h = 1/n$ and $\Delta t$ be the uniform spatial and time mesh-widths respectively. We can subdivide the spatial domain and consider the time step as follows:

$$\begin{aligned} x_i = ih, \;\; y_j = jh, \qquad & i,j = 0,1,\ldots,n \\ t_k = k\Delta t, \qquad & k = 0,1,\ldots \end{aligned}$$

For simplicity we write the approximated solution of $u$ and its time derivative at the spatial grid points $(x_i, y_j)$ as:

$$U_{i,j}(t) = u(x_i, y_j, t), \text{ and } U'_{i,j}(t) = \frac{\partial u}{\partial t}(x_i, y_j, t).$$

At any given time $t$, if we use the discretization of the steady state Poisson equation with a fourth-order scheme [3], we can approximate the spatial derivatives of (1). We obtain for any grid point $(x_i, y_j)$ , $i, j = 1, \ldots, n-1$:

$$\begin{aligned} & \frac{1}{2}\left[U'_{i+1,j}(t) + U'_{i,j+1}(t) + U'_{i-1,j}(t) + U'_{i,j-1}(t) + 8U'_{i,j}(t)\right] \\ = \; & \frac{\alpha^2}{h^2}[4\left(U_{i+1,j}(t) + U_{i,j+1}(t) + U_{i-1,j}(t) + U_{i,j-1}(t)\right) \\ & + U_{i+1,j+1}(t) + U_{i-1,j+1}(t) + U_{i-1,j-1}(t) + U_{i+1,j-1}(t) - 20U_{i,j}(t)], \end{aligned} \tag{2}$$

with the conditions

$$\begin{aligned} & U_{i,j}(0) = \psi(x_i, y_j), \\ & U_{0,j}(t) = f_0(y_j, t), \qquad U'_{0,j}(t) = \frac{\partial f_0}{\partial t}(y_j, t), \\ & U_{n,j}(t) = f_1(y_j, t), \qquad U'_{n,j}(t) = \frac{\partial f_1}{\partial t}(y_j, t), \\ & U_{i,0}(t) = g_0(x_i, t), \qquad U'_{i,0}(t) = \frac{\partial g_0}{\partial t}(x_i, t), \\ & U_{i,n}(t) = g_1(x_i, t), \qquad U'_{i,n}(t) = \frac{\partial g_1}{\partial t}(x_i, t). \end{aligned}$$

Eq. 2 is a system of $(n-1)^2$ ordinary differential equations and for any value of $t$, it is fourth-order in space. In the next two sections, we introduce two methods for discretizing the time derivative.

## 2.1 Crank-Nicolson Method

Let $U^k_{i,j}$ be the appoximation of $u(x, y, t)$ at the spatial grid point $(x_i, y_j)$ and the time level $t_k = k\Delta t$. Using (2), if we apply the Crank-Nicolson derivation [1], we get

$$\begin{aligned} & -\rho(U^{k+1}_{i+1,j+1} + U^{k+1}_{i-1,j+1} + U^{k+1}_{i-1,j-1} + U^{k+1}_{i+1,j-1}) \\ & +(1-4\rho)(U^{k+1}_{i+1,j} + U^{k+1}_{i,j+1} + U^{k+1}_{i-1,j} + U^{k+1}_{i,j-1}) + (8+20\rho)U^{k+1}_{i,j} \\ = \; & \rho(U^k_{i+1,j+1} + U^k_{i-1,j+1} + U^k_{i-1,j-1} + U^k_{i+1,j-1}) \\ & +(1+4\rho)(U^k_{i+1,j} + U^k_{i,j+1} + U^k_{i-1,j} + U^k_{i,j-1}) + (8-20\rho)U^k_{i,j} \end{aligned} \tag{3}$$

where $\rho = \frac{\alpha^2}{h^2}\Delta t$. After some simple manipulations, we obtain the linear system of equations

$$BU^{k+1} = FU^k + R \tag{4}$$

where $U^{k+1}$ and $U^k$ are vectors with components $U^{k+1}_{i,j}$ and $U^k_{i,j}$, $i, j = 1, \cdots, n-1$, respectively, $R$ is a $(n-1)^2$ dimensional vector containing the values of $U^{k+1}$ and $U^k$ at the boundary of the domain, and $B$ and $F$ are matrices of order $(n-1)^2$ given by:

$$B = tri[B_{-1}, B_0, B_1]_{n-1}, \qquad F = tri[F_{-1}, F_0, F_1]_{n-1}$$

where

$$\begin{aligned} B_{-1} &= tri[-\rho, 1-4\rho, -\rho]_{n-1}, & F_{-1} &= tri[\rho, 1+4\rho, \rho]_{n-1}, \\ B_0 &= tri[1-4\rho, 8+20\rho, 1-4\rho]_{n-1}, & F_0 &= tri[1+4\rho, 8-20\rho, 1+4\rho]_{n-1}, \\ B_1 &= tri[-\rho, 1-4\rho, -\rho]_{n-1}, & F_1 &= tri[\rho, 1+4\rho, \rho]_{n-1}. \end{aligned}$$

The subscript $n-1$ is the number of rows.

The Crank-Nicolson method (CNM) is unconditionally stable and is of fourth order in space and second order in time.

## 2.2 Implicit Collocation Method

Let $P_{i,j}(t)$ be the polynomial of degree $r$ satisfying the system (2) at the spatial grid point $(x_i, y_j)$ and at times $t_k = k\Delta t$ $(k = 0, \ldots, r-1)$. Then for any $i, j = 1, \ldots, n-1$ and $k = 0, \ldots, r-1$, we have

$$P_{i,j}(t_k) = a_{i,j,r}t_k^r + a_{i,j,r-1}t_k^{r-1} + \cdots + a_{i,j,1}t_k + a_{i,j,0}.$$

The coefficients $a_{i,j,0}$ are determined from the initial condition:

$$a_{i,j,0} = P_{i,j}(0) = U_{i,j}(0) = \psi(x_i, y_j).$$

To solve the system (2) by the collocation method is to determine the coefficients $a_{i,j,1}, \ldots, a_{i,j,r}$, for $i, j = 1, \ldots, n-1$. After some algebraic manipulations (see [6, 7] for details) we obtain the linear system of $r(n-1)^2$ equations

$$AX = S, \tag{5}$$

where $A$ is a block-tridiagonal matrix given by

$$A = tri\left[A_{l-1}, A_l, A_{l+1}\right]_{n-1}.$$

$A_{l-1}$, $A_l$ and $A_{l+1}$ are square matrices (of order $r(n-1)$) defined as

$$\begin{aligned} A_{l-1} &= tri\left[-E, \frac{1}{2}\frac{h^2}{\alpha^2}E' - 4E, -E\right]_{n-1}, \\ A_l &= tri\left[\frac{1}{2}\frac{h^2}{\alpha^2}E' - 4E, 4\frac{h^2}{\alpha^2}E' + 20E, \frac{1}{2}\frac{h^2}{\alpha^2}E' - 4E\right]_{n-1}, \\ A_{l+1} &= tri\left[-E, \frac{1}{2}\frac{h^2}{\alpha^2}E' - 4E, -E\right]_{n-1}. \end{aligned}$$

The subscript $n-1$ determines the number of block-rows. $E$ and $E'$ are nonsymmetric matrices of order $r$

$$E = \begin{pmatrix} t_0^r & t_0^{r-1} & \cdots & t_0 \\ t_1^r & t_1^{r-1} & \cdots & t_1 \\ \vdots & \vdots & & \vdots \\ t_{r-1}^r & t_{r-1}^{r-1} & \cdots & t_{r-1} \end{pmatrix}, \qquad E' = \begin{pmatrix} rt_0^{r-1} & (r-1)t_0^{r-2} & \cdots & 2t_0 & 1 \\ rt_1^{r-1} & (r-1)t_1^{r-2} & \cdots & 2t_1 & 1 \\ \vdots & \vdots & & \vdots & \vdots \\ rt_{r-1}^{r-1} & (r-1)t_{r-1}^{r-2} & \cdots & 2t_{r-1} & 1 \end{pmatrix},$$

the vector $X$ has the $r(n-1)^2$ unknowns $a_{i,j,1}, a_{i,j,2}, \ldots, a_{i,j,r}$, for $i,j = 1,\ldots,n-1$, and the right hand side $S$ is explicitly given in [7].

The implicit collocation method (ICM) introduced here is of order four in space and of order $r$ in time. To solve the heat equation by ICM, we first determine the coefficient $a_{i,j,k}$, and then compute the values of the polynomials $P_{i,j}(t)$ at $t = t_k$. Numerical results presented in [7] show that ICM is unconditionally stable and produces high accurate solutions.

## 3 Computational Considerations

To determine the approximated solution of the heat equation using CNM, we need to solve the linear system (4) where $B$ is a matrix of order $(n-1)^2$ and bandwidth $2n+1$. If we use ICM, we first solve the system (5) to obtain the coefficients $a_{i,j,k}$ $(i,j = 1,\ldots,n-1$ and $k = 1,\ldots,r)$. Then the approximated values $U_{i,j}(t_k)$ $(i,j = 1,\ldots,n-1$ and $k = 1,\ldots,r-1)$ are calculated using the polynomials $P_{i,j}(t)$. Here, the matrix A is of order $r(n-1)^2$ and its bandwidth is equal to $(2n+1)r$. The matrix $A$ is $r^2$ times as large as $B$.

For a given $r$, if we were to compare CNM (evaluated $r$ consecutive times) and ICM using the same $\Delta t$, it is obvious that the implementation of CNM would be faster since the decompostion of the matrix $A$ would be by far more computational demanding. The advantage of ICM is that it does not consist of determining the $U_{i,j}(t_k)$ only $(k = 1,\ldots,r-1)$, but also allows us to find the approximated solution at any $t$ in the interval $[t_0, t_{r-1}]$. For a "fair" comparison, we can choose different $\Delta t$ for the two methods as follows.

Let $N_{cnm} = (n-1)^2$ be the number of unknowns in (4) and let $N_{icm} = r(n-1)^2$ the one in (5) where $r$ is the degree of the polynomials. Let $\Delta t_{icm}$ be the time step used for ICM. The solution using ICM, can be determined at any point in the interval $[0,(r-1)\Delta t_{icm}]$. Let $(r-1)m$ be the number of equidistant points where the solution is to be computed in this time interval. To determine the solution at the same points of the interval with CNM, $(r-1)m$ time iterations must be carried out. The corresponding time step is $\Delta t_{cnm} = \Delta t_{icm}/m$.

Given the above assumptions, we want to numerically determine the values of $m$ for which ICM is cheaper than CNM for $r$ and $\Delta t_{icm}$ fixed. In fact, we will compare the two methods over the interval $[0,T]$ (where $T = \eta(r-1)\Delta t_{icm}$). In this case, both methods will be iterated until we reach the approximated solution at $t = T$. We assume ICM and CNM produce solutions of comparable accuracies.

## 4 Description of the Parallel Implementation

With the considerations laid out in the previous section, $\eta$ consective iterations of ICM will be carried out to obtain the solution over the time interval $[0,T]$ $(T = \eta(r-1)\Delta t_{icm})$. At each iteration of ICM, the solution will be approximated at $(r-1)m$ time levels. If we want to determine the solution over the same interval $[0,T]$ using CNM $(\Delta t_{cnm} = \Delta t_{icm}/m)$, CNM will be called $(r-1)\eta m n$ consecutive times.

For $n$, $r$, $\Delta t_{icm}$, $\eta$ and $m$ given, we can define the algorithms for both methods as follows:

**ICM's Algorithm**

1. Define the matrix $A$
2. Decompose the matrix $A$ to obtain the matrix $M \approx A^{-1}$

3. For $l = 1 \rightarrow \eta$, do:
4. Define the right hand side $S$
5. Determine the coefficients $a_{i,j,k}$: $MAX = MS$
6. Find $U_{i,j}(t_k)$ at $(r-1)m$ consecutive time steps by computing $P_{i,j}(t_k)$
7. End do

**CNM's Algorithm**

1. Define the matrix $B$
2. Decompose the matrix $B$ to obtain the matrix $Q$ ($\approx B^{-1}$)
3. For $l = 1 \rightarrow m(r-1)\eta$, do:
4. Define the right hand side $S = FU^{l-1} + R$
5. Find the solution: $QBU^l = QS$
7. End do

We present two strategies for implementing the two algorithms. We focus on ICM's Algorithm with the assumption that the same techniques will be utilized for the one of CNM.

**Strategy 1:**
To solve the linear system of equations, we used the decomposition algorithm for inverting asymmetric and indefinite matrices proposed by Luo [8]. Here, the inverse of the matrix $A$ is explicitly computed, i.e., $M = A^{-1}$ (Step 2). It follows that Step 5 ($X = MS$) is just a matrix vector multiplication. Given the polynomial coefficients $a_{i,j,k}$ (obtained in Step 5), Step 6 computes $P_{i,j}(t_k)$ using Horner's algorithm.

**Strategy 2:**
In this case, a sparse matrix $M \approx A^{-1}$ is explicitly computed and used as a preconditioner for the sparse matrix $A$ (Step 2). A method based on Luo's algorithm is employed by dropping (in the calculations of $M$) all the entries that are less than (in absolute value) a prescribed tolerance $\tau > 0$ [9]. To carry out Step 5 ($MAX = MS$), the General Minimal Residual (GMRES) algorithm is used as the iterative accelerator. Finally Step 6 is the same as in the previous strategy.
To avoid the storage of zero elements, the nonzero entries of the matrix $A$ are stored by rows with the compressed row format.

In Strategy 1, no data compression is used. In the case of of ICM, it limits the number of spatial grid points we can choose. The decomposition of matrix $A$ (ICM) will cost at least $r^3$ times more than the one of $B$ (CNM).

The fact to obtain the solution at $(r-1)m$ consecutive time steps during one iteration of the ICM's algorithm shows that with ICM, the parallelization is achieved across both time and space.

In [7], we observed that the choice of a small value of $r$ (say $r = 3$ or $r = 4$) can be appropriate to obtain accurate solutions in a short amount of time. Such a choice limits the size of the matrix $A$ which decomposition is the most demanding computation and the most difficult to parallelize. Since Steps 1 and 2 are done once, more time will be spent on right hand side updates (Step 4), matrix-vector multiplications or iterative process (Step 5) and basic loop calculations (Step 6) until the desired time step is reached. All these three steps

can easily be implemented on parallel computers and the efficiency of the algorithm will be improved. In the numerical experiments, we consider $r = 3$.

# 5 Numerical Experiments

The numerical experiments were conducted on a SGI Origin 2000 with 24 processors each running at 250 MHz and 512 MB memory. The programs were coded in Fortran 90 programming language in double precision with 64-bit arithmetic. The parallel implementation of the two algorithms was achieved by introducing parallel directives (thread commands) in all the major loops.

For all our simulations, we consider Eq. 1 with the conditions

$$\alpha = \frac{1}{\pi}, \quad u(x, y, 0) = \sin \pi x + \sin \pi y,$$
$$f_0(y, t) = f_1(y, t) = e^{-t} \sin \pi y, \quad g_0(x, t) = g_1(x, t) = e^{-t} \sin \pi x.$$

The exact solution is given by $u(x, y, t) = (\sin \pi x + \sin \pi y)e^{-t}$.

To simplify our analysis, we take for all the experiments $n = 32$, $r = 3$, $\Delta t_{icm} = 0.01$, and $T = 10.0$.

For different values of $m$, we report the elapsed times obtained with both ICM and CNM when 4 processors are employed. The results appear in Figure 1 and Figure 2 for Strategy 1 and Strategy 2 (with the dropping parameter $\tau = 10^{-5}$) respectively. We observe that for $m > 80$, ICM with Strategy 1 is more cost effective. If Strategy 2 is instead used, we need only $m > 10$ to have ICM better than CNM.
In addition, for ICM, Strategy 1 requires more time to reach the target time step. In fact, the factorization of the matrix $A$ takes most of the computing time. We did not take advantage of the bandwidth of the matrix $A$. If we did, we would have reduced the factorization time at the expense of the parallel efficiency [9].

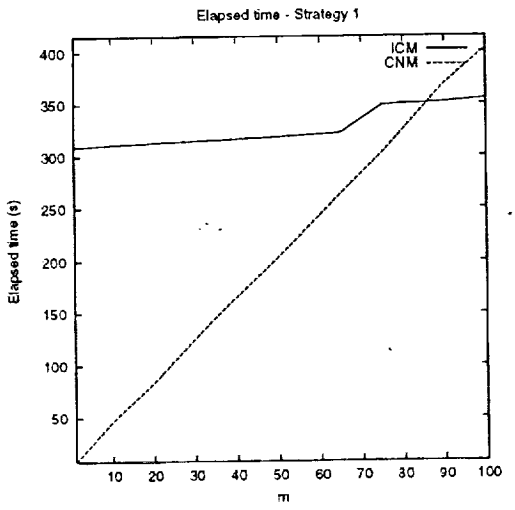


Figure 1: Strategy 1: elapsed time as function of $m$

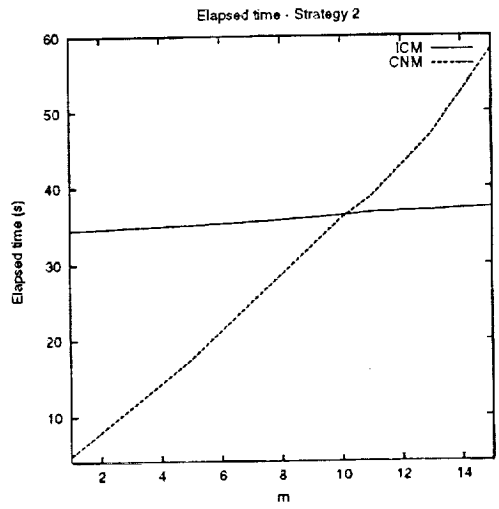


Figure 2: Strategy 2: elapsed time as function of $m$.

For $m = 15$, we present the speedup as function of the number of processors when ICM and CNM are both implemented with Strategy 1 and Strategy 2.

From Figure 3, we note that when Strategy 1 is used, CNM scales very well across the processors whereas ICM is less efficient. If we examine the computing time required by each component of the algorithm, we see that for CNM, at most 7% of the total time is spent on the matrix decomposition. In ICM, the same decomposition takes at least 80% of the time. The Luo's matrix decomposition has a moderate efficiency. This explains why ICM does not scale that well.

On the other hand, if we focus on Strategy 2, the matrix factorization (done sequentially) takes a small percentage of the time (less than 13%) for both ICM and CNM (due in part for the data compression). However, ICM displays a better efficiency (Figure 4). This is due to the fact that ICM requires a larger problem size and therefore a better load balancing is achieved on the remaining part of the code (at least 87% of the total time) that is easy to parallelize.

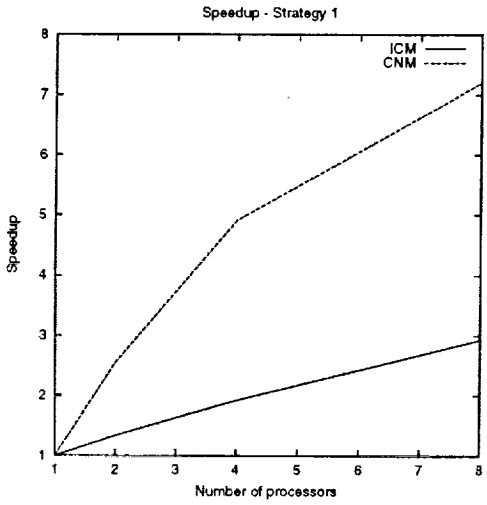


Figure 3: Strategy 1: speedup as function of the number of processors for $m = 15$.

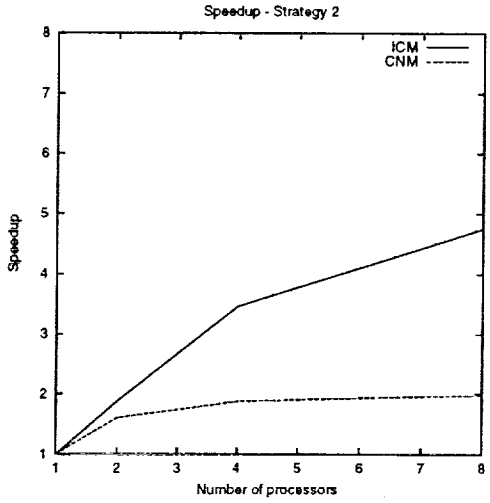


Figure 4: Strategy 2: speedup as function of the number of processors for $m = 15$.

Finally, we observe that the approximate solutions were slightly more accurate with CNM for Strategy 1 and with ICM for Strategy 2. The maximum errors obtained were not significantly different to draw definitive conclusions.

## 6 Conclusions

We have compared the Crank-Nicolson method (CNM) and the implicit collocation method (ICM) in the numerical solution of the two dimensional heat equation. In both methods, the spatial derivatives are discretized with a high order finite difference scheme and the resulting approximations give linear systems of equations. We presented two parallel strategies to implement ICM and CNM on the SGI Origin 2000. Our numerical experiments showed that under some simple assumptions, ICM can require less elapsed time than CNM and can be

more efficient. The larger number of unknowns in ICM can allow a better load balancing but unfortunately does not allow us to solve large size problems whereas CNM does.

In our codes, we used parallel directives on all the major Do-loops. The matrix decompositions used here were either done sequentially or did not display good efficiency. In future work, we plan to implement CNM with message passing directive and to explore parallel matrix decomposition algorithms.

**Acknowledgment:** I am grateful to Jun Zhang and Fabienne Jézéquel for their useful advice. I would like to also thank Milton Halem and Nancy Palm for their support.

## References

[1] J. Crank and P. Nicolson, A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type, *Proceedings of the Cambridge Philosophical Society*, Vol. 43, No. 50, p. 50-67 (1947).

[2] J. Douglas Jr. and T. Dupont, *Collocation Methods for Parabolic Equations in a Single Space Variable*, Lecture Notes in Mathematics, Vol. 385, Springer, Berlin, 1974.

[3] M.M. Gupta, A fourth-order Poisson solver, *J. Comp. Phys.*, **55**, p. 166-172 (1984).

[4] E. Hairer, S.P. Norsett and G. Wanner, *Solving Ordinary Differential Equations I: Non-stiff Problems*, Springer Series in Computational Mathematics, Vol. 8, Springer, Berlin, 1987.

[5] B.L. Hulme, One-step piecewise polynomial Galerkin methods for initial values problems, *Math. Comp.*, **26**, p. 415-426 (1972).

[6] F. Jézéquel, A validated parallel across time and space solution of the heat transfer equation, *Applied Numerical Mathematics*, **31**, p. 65-79 (1999).

[7] J. Kouatchou, Finite differences and collocation methods for the solution of the two dimensional heat equation, *Num. Meth. for PDEs*, to appear.

[8] J. Luo, A new class of decomposition for inverting asymmetric and indefinite matrices, *Computers Math. Applic.*, **24** (4), p. 95-104 (1993).

[9] J. Zhang, A sparse approximate inverse preconditioner for parallel preconditioning of general sparse matrices, *Int. J. Com. Math.*, to appear.