Implementations using External Storage for the Crank Nicholson and Extrapolated Backwards Euler Methods for Finite Element Programs

by

D.C. Donnan and R.B. Simpson

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada

Research Report CS-81-07

February 1981

This research was supported in part by a grant from the Natural Science and Engineering Council of Canada, and in part by a contract with Environment Canada.

CHAPTER I

Introduction

It is common in finite element packages to augment the use of addressable memory by storing files of data on disk storage. In this report, we examine the interaction between the use of external disk files and two time stepping algorithms for dynamic finite element calculations. The time stepping algorithms are the commonly used Crank Nicholson time discretization, with its benefits of simplicity and second order accuracy for adequately smooth time evolutions, and the less well known extrapolated backwards Euler methods, which have better numerical stability characteristics for solutions with stiff time evolution, but are somewhat more complicated to implement, and less effective for computations with smooth time evolutions.

A standard technique for using external file storage in the finite element method is the frontal method, see Hinton and Owen [3]. Our discussion is based on an alternative design in which the solution of the global system of equations is done entirely in core by sparse matrix solving codes (SPARSPAK, George, Liu and Ng, [2].) There is a considerable, perhaps surprising, amount of file traffic which appears to be required by these algorithms operating under this design strategy, and this report is intended primarily to serve as an exposition of this fact. As such, it has more of the nature of posing a question, than of resolving one.

In Chapter II, we briefly review the reduction of a parabolic partial differential equation to a system of ordinary differential equations by the spatial discretization provided through the finite element method. In Chapter III, the Crank Nicholson, and extrapolated backwards Euler methods for solving the resulting ordinary differential equations are described. Their performance on a problem with a stiff time history is indicated with an example. In Chapter IV, a model for the data files used by the finite element method, and for the use of addressable memory and disk storage is formulated, which attempts to capture the features of our design that are relevant to the topic of this report. Algorithms for the Crank Nicholson and extrapolated backwards Euler methods are given in terms of this model, and the number of file transfers per time step of each is indicated. A series of computational tests of these algorithms is then described which attempts to ascertain how closely the crude measure of file transfers correlates with the actual running of the implementation.

CHAPTER II

The Finite Element Method for Initial-Boundary Value Problems

The finite element method can be applied to provide approximate solutions to problems from the following general class of parabolic initial-boundary value problems. These problems require the determination of a function $c(x,y,t)$ defined in a space-time cylinder, , specified by a region R in the x-y plane and $t > t_0$ and satisfying:

2.1) equation $\frac{\partial c}{\partial t} = Lc + F$ for $(x,y,t) \in R$

for L a general elliptic operator with possibly time dependent coefficients $D_i(x,y,t)$, $i = 1,2,3,4$, $U(x,y,t)$, $V(x,y,t)$, $Q(x,y,t)$

$$Lc = \frac{\partial}{\partial x}\left(D_1 \frac{\partial c}{\partial x} + D_2 \frac{\partial c}{\partial y}\right) + \frac{\partial}{\partial y}\left(D_3 \frac{\partial c}{\partial y} + D_4 \frac{\partial c}{\partial y}\right) - U \frac{\partial c}{\partial x} - V \frac{\partial c}{\partial y} + Q\, c$$

2.2) Boundary conditions

$$c(x,y,t_0) = g(x,y,t)$$

for $(x,y) \in \partial R$, $t > t_0$

2.3) initial conditions

$$c(x,y,t_0) = c_0(x,y)$$

for $c_0(x,y)$ specified.

A common finite element approach to determining $c(x,y,t)$ 1) - 3) approximates 2.1), 2.2), 2.3) by a system of ordinary differential equations. It can be described as introducing a set of $N + N_b$ basis functions $\phi_k(x,y)$, $k = 1,\ldots,N+N_b$ and approximating c by a linear combination of the ϕ_k with time varying coefficients, $c_k(t)$, $\bar{c}(x,y,t) = \sum_{k=1}^{N+N_b} c_k(t)\, \phi_k(x,y)$.

The last N_b of the coefficients $c_k(t)$ are assumed to be explicitly determined by the boundary conditions, 2.2); they will form an N_b vector denoted $c_b(t)$. All $N + N_b$ are determined at $t - t_0$ by initial conditions 2.3).

The values of the first N of the coefficients, $c_k(t)$, are determined by solving the system of equations for $j = 1$ to N:

2.4) $$\langle \phi_j, \sum_i (dc_i/dt) \phi_i \rangle = \langle \phi_j, \sum_i c_i L\phi_i \rangle + \langle \phi_j, F \rangle$$

where "$\langle , \rangle$" denotes the inner product $\langle f,g \rangle = \iint_R f(x,y,t)\, g(x,y,t)\, dxdy$.

These equations can be rewritten in matrix notation as the system of ordinary differential equations

2.5) $$M \frac{dc}{dt} = Ac + (S + A_b c_b - M_b \frac{dc_b}{dt}) = Ac + f$$

where the traditional finite element designations for these terms are

M - the global mass matrix

A - the global stiffness matrix

S - global load vector.

M_b and A_b are $N \times N_b$ matrices that couple boundary conditions to the approximate solution, and are part of the global mass and stiffness matrices.

CHAPTER III

The Crank Nicholson and Extrapolated Backwards Euler Time Discretization

In the preceeding section we have seen that a spatial discretization of an initial boundary value problem for a general parabolic partial differential equation using a finite element method leads to an initial value problem for a system of ordinary differential equations of the form

3.1) $$M \frac{dc}{dt} = A(t)c + g + \frac{dh}{dt}$$

for (possibly time dependent) matrix $B(t) = M^{-1} A(t)$, and known 'source' functions $g = (S+A_b c_b(t))$ and $dh/dt = - M_b dc_b/dt$.

While this is not an appropriate form for computational purposes, it is a convenient one for the formal discussion of time discretizations. For further convenience of exposition, we shall assume a discrete set of uniformly spaced time levels, $t_n = t_0 + n \Delta t$ and use a subscript on an N vector, W_n, to denote its value at time t_n.

Probably the most commonly used time discretization has been the Crank Nicholson (CN) scheme, which for 3.1), may be written

3.2) $$(M - \Delta t A(t_{n+\frac{1}{2}})/2)c_{n+1} = (M + \Delta t A(t_{n+\frac{1}{2}})/2)c_n + \Delta t(g_{n+1}+g_n)/2 + h_{n+1} - h_n$$

for $t_{n+\frac{1}{2}} = t_n + \Delta t/2$. Doubtless much of the popularity of the CN scheme is due to the combination of its being second order accurate in time and its being relatively simple to implement.

A less common alternative, or parameterized family of alternatives, is the extrapolated backwards Euler (EBE) scheme [5]. For extrapolation parameter, w, the EBE scheme for 3.1) uses two auxiliary storage vectors, a and b, and takes the form

3.3a) $$[M-\Delta t A(t_{n+\frac{1}{2}})/2]a = Mc_n + \Delta t g(t_{n+\frac{1}{2}}) + 2(h(t_{n+\frac{1}{2}}) - h(t_n))$$

b) $$[M-\Delta t A(t_{n+\frac{1}{2}})/2]b = Ma + \Delta t g(t_{n+1}) + 2(h(t_{n+1}) - h(t_{n+\frac{1}{2}})$$

c) $$[M-\Delta t A(t_{n+\frac{1}{2}})]a = Mc_n + \Delta t g(t_{n+1}) + (h(t_{n+1}) - h(t_n))$$

d) $$c_{n+1} = (1+w)b - w\,a$$

In terms of computations per time step, the CN scheme requires one evaluation of A, g and h and solution of one system of equations per step. The EBE schemes require one evaluation of A, two evaluations of g and h, and the solution of three systems of equations, only two of which have distinct matrices of all coefficients.

To see what the EBE schemes have to offer to compensate for involving more than twice as much computation per step, we must look at the stability properties of each method.

3.2 Stability Considerations

To examine the stability properties of these methods, we replace 3.1) by the simple scalar initial value problem

3.4) $dz/dt = \lambda z(t),\ z(t_0) = z_0.$

This problem may be viewed as having a single time scale, or transient, set by λ. The exact solution of (3.4) is

3.5) $z(t_n) = (e^{\lambda\Delta t})^n\ z_0$

The solution of the Crank Nicholson scheme applied to 3.4) is $z_n = E(CN)^n\ z_0$ for the Crank Nicholson 'growth factor".

3.6) $E(CN) = (1+\lambda\Delta t/2)/(1-\lambda\Delta t/2).$

The solution for the EBE scheme can be seen to be $z_n = E_w(EBE)^n\ z_0$ for the EBE growth factor.

3.7) $E_w(EBE) = (1+w)/(1-\Delta t\lambda/2)^2 - w/(1-\Delta t\lambda).$

The second order accuracy in time of the Crank Nicholson scheme is reflected in the fact that

3.8) $E(CN) = e^{\lambda\Delta t} + O(\Delta t^3)$

While the CN method is A-stable (i.e., $|E(CN)| \leq 1$ for RE $\lambda < 0$ and all Δt, [6], page 43), it is well known that it can produce erroneous oscillations in the computed solution of dynamically stiff problems unless the time step Δt is adequately restricted. This behaviour can be anticipated from the fact that $E(CN) \to -1$ as Re $\lambda\, \Delta t \to -\infty$, i.e., the CN method is not stiffly A-stable ([6], page 221). However, the system of ordinary differential equations arising from the Galerkin method for 2.1) is commonly dynamically stiff if the solution to be computed is not smooth in both its spatial and time variations. An example of the weakness of the CN method in dealing with such a problem is given below.

Properties of the Extrapolated Backwards Euler Method

The corresponding properties of the EBE method are probably less well known (see Gourlay and Morris [5]). The 'growth factor' for the EBE methods for $w = 1$ and $w = 1.5$ are shown in Figure 3.1 along with the exact growth factor, $\exp(\lambda\Delta t)$ and $E(CN)$ plotted as functions of $\lambda\Delta t$. Over much of the argument range shown in Figure 3.1, $E_w(EBE)$ is clearly a better approximation to $\exp(\lambda\Delta t)$ than $E(CN)$ is. However, the EBE method is actually only second order accurate for $w = 1$ i.e.

3.9) $$E_1(EBE) = \exp(\lambda\Delta t) + O(\Delta t^3).$$

These figures suggest that we might wish to choose w so as to minimize the error, i.e. chose w to minimize

$$\max_{\lambda\Delta t<0} |E_w(EBE) - \exp(\lambda\Delta t)|.$$

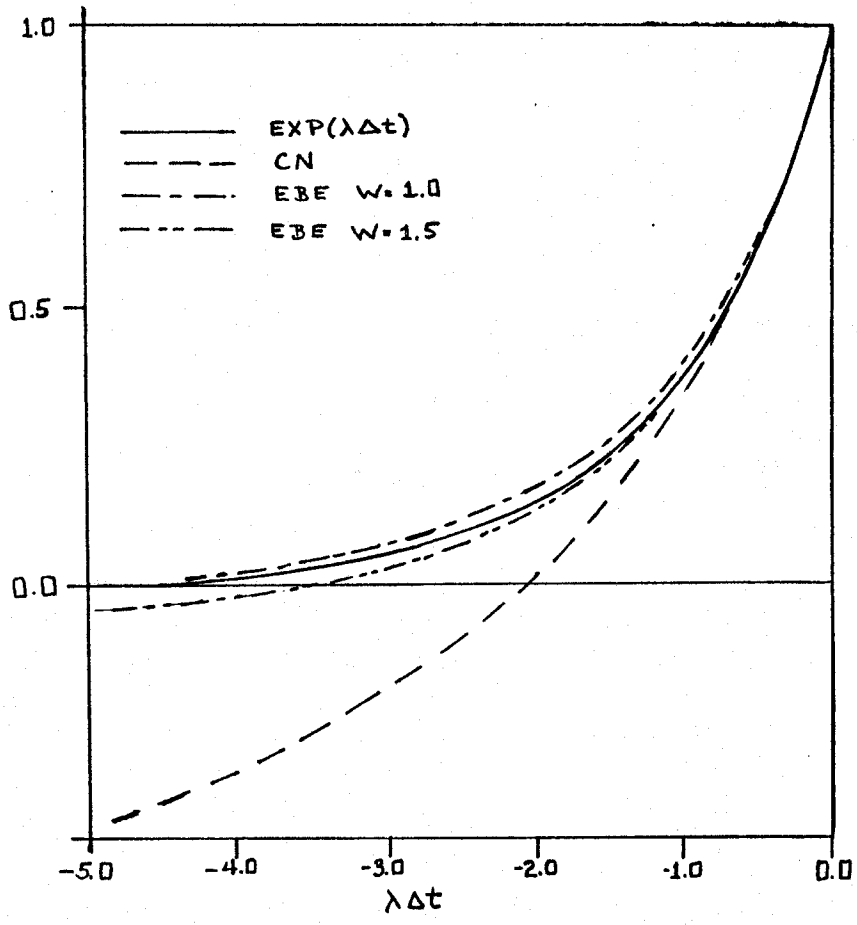

Figure 3.1

Graphs of Approximations to exp(λΔt)

The minimizing value of w is (,to two decimal places,) $w_{opt} = .97$; however the reduction in maximum error obtained by using w_{opt} rather than $w=1$ is so small, that it has not been considered significant, i.e.,

$$\max_{\lambda\Delta t<0} |E_{.97}(EBE) - \exp(\lambda\Delta t)| = .0344$$

compared to

$$\max_{\lambda\Delta t<0} |E_1(EBE) - \exp(\lambda\Delta t)| = .0361$$

It can be seen from 3.7) that the EBE method for $w=1$ is stiffly A-stable, i.e. that $E_1(EBE) \to 0$ as $\lambda\Delta t \to -\infty$. Combining these stability properties with its improved approximation of $\exp(\lambda\Delta t)$ over much of the argument range, we can expect the second order EBE method to give satisfactory computed solutions to dynamically stiff problems for significantly larger time steps than would be possible with the CN method.

To demonstrate the difference in behaviour between the CN method (3.2) and the EBE method (3.3), we solve a test problem which takes the form of a plate being steadily heated by a discontinuous source. The equation to be solved is

3.11) $$\partial u/\partial t = \partial^2 u/\partial x^2 + \partial^2 u/\partial y^2 + S(x)$$

for a solution in the unit square $0 \leq x \leq 1$, $0 \leq y \leq 1$, for $t \geq 0$. The source function is a pulse in the x direction, corresponding to heating in a strip of the plate,

3.12) $$S(x) = \begin{cases} 0 & 0 \leq x < 5/12 \\ 200 & 5/12 \leq x \leq 7/12 \\ 0 & 7/12 < x \leq 1 \end{cases}$$

The solution is initially 0, and is held at zero at the boundary of the square.

A typical contour plot of the solution after heating has started is shown in Figure 3.2 (for a higher pulse at amplitude than stated in (3.12)). The values of the computed solution obtained by the CN method and the EBE methods are shown in Figure 3.3, for a step size of $\Delta t = .3$ and in Figure 3.4 for $\Delta t = .15$. These computations were done on an 181 element mesh. The mesh was subdivided further into 724 elements, and the results using a time step of size $\Delta t = .3$ are shown in Figure 3.5. The 'ringing' error in the CN solutions is clearly evident, and the smoothness of the EBE results is equally apparent. A comparison of solution values for the 181 element computation is given in Table 3.1, and indicates that the step size $\Delta t = .3$ appears to be giving satisfactory two digit results, although the comparison to the 724 element calculation shows the second digit to be influenced by the spatial discretization.

	181 elements				724 elements	
	$\Delta t = .15$		$\Delta t = .3$		$\Delta t = .15$	
Time	CN	EBE	CN	EBE	CN	EBE
.15	9.51	7.66				
.30	8.88	9.16	12.54	9.16	12.68	9.30
.45	10.13	9.55				
.60	9.32	9.65	8.40	9.64	8.58	9.82
.75	9.96	9.67				
.90	9.45	9.67	10.54	9.68	10.72	9.86

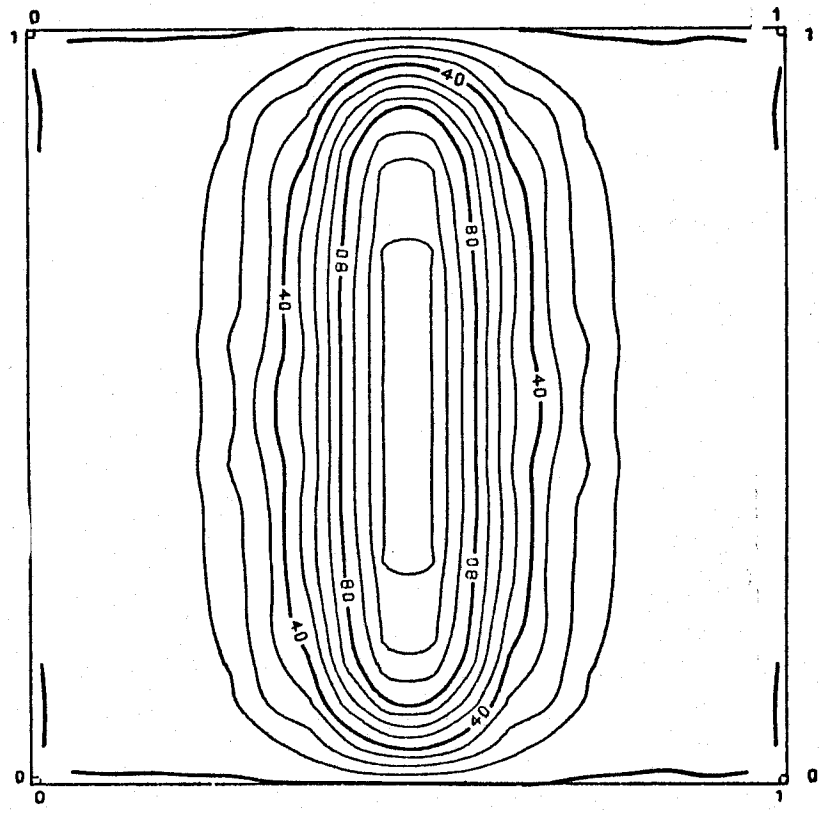

Figure 3.2

A Contour Plot of the Solution of
the Discontinuously Heated Plate Problem

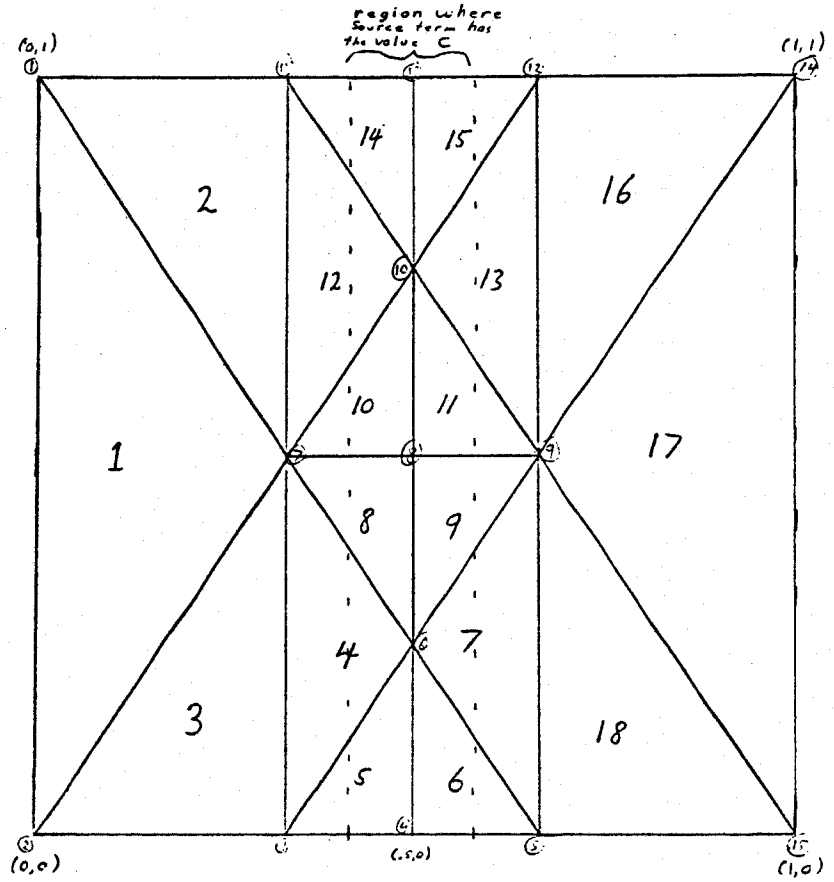

Figure 3.3

Mesh used for the Discontinuously
Heated Plate Problem

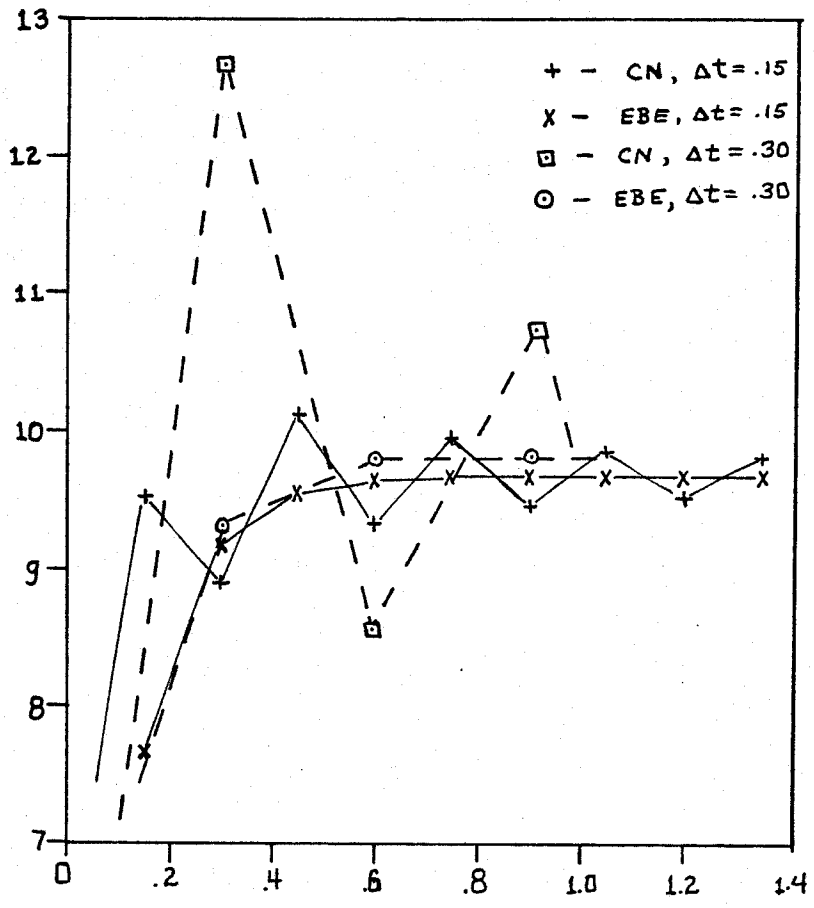

Figure 3.4

Numerical Solution Values at the
Centre of the Discontinuously Heated Plate
(181 elements)

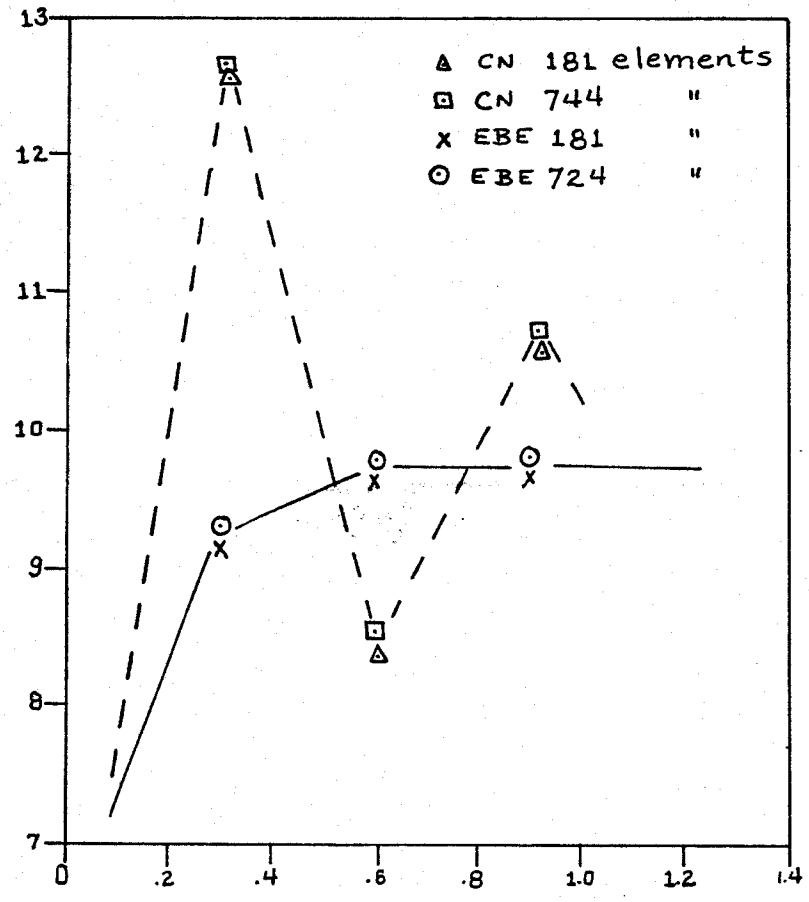

Figure 3.5

Numerical Solution Values at the Centre of
the Discontinuously Heated Plate
(181 and 724 elements)

CHAPTER IV

Implementation using External Store

We now turn to a discussion of the implementation of these methods into algorithms using external (disk) storage, paying particular attention to data transfers between addressable and external store. Basically, our discussion forms a model for the implementations carried out for the finite element package FEMPAK, [1]; developed on an experimental basis at the University of Waterloo. Three classes of data are distinguished in the algorithms:

I) - mesh and problem data

II) - local stiffness matrices and load vectors

III) - the data for the global system

The basic flow of data in a time stepping algorithm is shown in Figure 4.1.

We assume that the implementations require random access to data I and data III, but only sequential access to data II. A common technique for extending the problem size that can be handled through the use of external store is referred to as the frontal method in which data I is resident in addressable memory, data II exists temporarily in addressable memory and possibly is also stored externally, and data III is organized so as to be stored in a combination of addressable and external store.

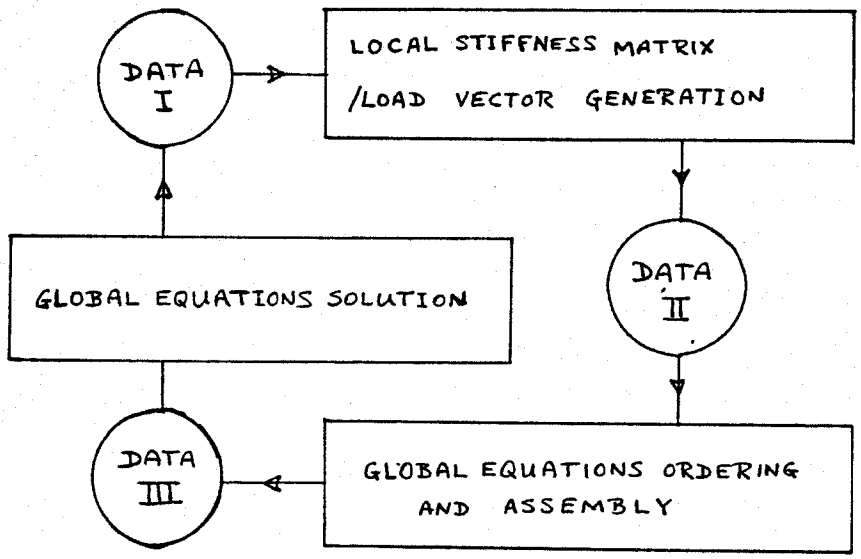

Figure 4.1

Flow of Data in Time Stepping Algorithms

In our case, we wish to perform the computations with the global system of equations entirely in core using available sparse matrix software (SPARSPAK, [2]). We assume then, that for reasons of problem size and modularity, that addressable memory can be used for data I or data III in a mutually exclusive manner. Copies of data I and data III are maintained in external store, the dynamic part of which must be updated periodically. When an algorithm requires data I to be in addressable memory, we will refer to the algorithm as being in state I, and similarly for the algorithm being in state III.

Some of the data is unchanged during the cycling of the time stepping algorithm. Such data will be referred to as static; e.g. the meshes being considered do not change with time. Data that may change with the cycles of the time stepping algorithm will be called dynamic. The designation of data as static or dynamic depends, of course, on the problem class being considered, as well as the method being used. We shall consider two problem classes:

a) fully dynamic equations i.e. the coefficients of the partial differential equation may change in time (either explicitly or due to a nonlinearity)

b) constant coefficients i.e. the coefficients of the partial differential equation may be spatially variable but not time varying.

One objective is to have time stepping algorithms for the fully dynamic case which are easily modified for the constant coefficient case so as to avoid recomputing and rewriting data to external storage that is static in the latter case. For this purpose, we break the three data classes introduced above into files that are described in Table 4.1 and which are the external files of our implementation constant.

One of the virtues of the Crank Nicholson method is its simplicity; this makes it a useful introduction to our high level description of algorithms including file transfers to external storage. All such transfers are indicated either by 'read from' or 'write to'. The phrase 'switch to state I' is intended for the orientation of the reader and does not indicate any resource utilization beyond what is explicitly indicated. The global matrix sparsity information is available from the local load vector file, hence this file is used for data for the global equation ordering step.

Data Class	File Name	Dynamic Case	Constant Coefficient Case
Ia	mesh description	static	static
Ib	solution	dynamic	dynamic
IIa	local stiffness matrices	dynamic	static
IIb	local load vectors	dynamic	dynamic
III a)	equation ordering data	static	static
III b)	global matrix factors	dynamic*	static
		*not explicitly stored in external memory.	

TABLE 4.1

Basic files for time stepping algorithms

The Crank Nicholson Method (fully dynamic case)

1) Initialize the mesh, and starting solution (in state I)

- write to mesh description file (Ia)
- write to solution file (Ib)

FOR TIME STEPS = FIRST TO LAST

2) Form local stiffness matrices, local load vectors (in state I)

- write to local stiffness matrix file (IIa)
- write to local load vector file (IIb)

IF TIME STEP = FIRST

THEN

3) Order global equations and set up sparse matrix data structures (in state III)

- read from local load vector file (IIb)
- write to global equation ordering data file (IIIa)

ELSE

4) Switch to state III

- read global equation ordering data file (IIIa)

5) Assemble global matrix, load vector, solve for C_{n+1} (see 1.B) (in state III)

- read from local stiffness matrix file (IIa)
- read from local load vector file (IIb)
- write to solution file (Ib)

6) Switch to state I

- read mesh description file (Ia)
- read solution file (Ib)

We shall assume that a gross characterization of an algorithm's performance can be obtained by counting the number of file transfers required to perform N time steps. For the fully dynamic case of the Crank Nicholson method, as described, we have

4.2) CN(FD) = 4 + 8N file transfers

The modifications necessary for the constant coefficient case are indicated in the following description.

The Crank Nicholson Method (constant coefficient case)

1) Initialize the mesh and starting solution (in state I)
   - write to Ia, Ib
2) From local stiffness matrices (in state I)
   - write to IIa
3) Order global equations and set up sparse matrix data structures (in state III)
   - read from IIa
   - write to IIIa
4) Assemble global matrix, factor it and store the factors (in state III)
   - write to IIIb
5) Switch to state I
   - read from Ia, Ib

FOR TIME STEP = FIRST TO LAST

6) Form local load vectors (in state I)
   - write to IIb

7) Switch to state III

   - read from IIIa, IIIb

8) Assemble global load vector and solve for C_{n+1} (in state III)

   - read from IIb

   - write to I

9) Switch to state I

   - read from Ia, Ib

For the constant coefficient case then, the number of file transfers is

4.3) $CN(CC) = 8 + 7N$

In the extrapolated backwards Euler method, considerably more file traffic per step is generated, due to having to form and solve several equations per step that have some interdependencies. This gives rise to multiple versions of some of the categories of data; e.g. for IIa), the local stiffness matrices, there will be two files of these, one contributing to $M - \Delta t A(t_{n+\frac{1}{2}})$, the other to $M - \Delta t A(t_{n+\frac{1}{2}})/2$. It might be expected that one could gain some storage advantage from the algebraic similarity between these two matrices. However, our initial experiments with storing one of these matrices only, and making explicit calculations of the modification needed for the other proved more costly than generating two independent files of local stiffness matrices. Hence we take advantage of the similarity only during the computation of local stiffness matrices, not to effect any storage solving. Similarly, independent copies of the local load vectors, solutions, and global matrix Cholesky factors are maintained.

The Extrapolated Backwards Euler Method (fully dynamic case)

1) Initialize the mesh and starting solution (in state I)

   - write to Ia

FOR TIME STEP = FIRST TO LAST

   2) Form local stiffness matrices, and local load vectors for 3.3a) and 3.3c)

      - write to IIa (2 files), IIb (2 files)

   IF TIME STEP = FIRST

      THEN

      3) Order global equations and set up sparse matrix data structures (in state III)

         - read from IIb

         - write to IIIa

      ELSE

      4) Switch to state III

         - read IIIa

   5) Assemble global matrix, load vector for 3.3a), solve for a and store matrix factors

      - read IIa, IIb

      - write Ib, IIIb

   6) Switch to state I

      - read Ia, Ib

   7) Form local load vectors for 3.3b)

      - write to IIb

   8) Switch to state III; read ordering data plus matrix factors

      - read IIIa), IIIb)

9) Assemble load vector for 3.3b; and solve for b
   - read from IIb
   - write to Ib

10) Assemble global matrix, load vector for 3.3c, and solve for a
   - read IIa, IIb
   - write Ib (2nd file)

11) Switch to state I, and perform extrapolation 3.3d)
   - read Ia, Ib (2 files)

The fully dynamic extrapolated backwards Euler method thus requires

4.4) EBE(FD) = 12 + 22N file transfers

It should be noted that in this algorithm, a copy of the current solution, c_n, is never stored in the external file for data Ib. If this were desired, it could ge accomplished by an extra write to Ib at step 1, and at step II, which would result in 13 + 23N file transfers for N time steps.

If the global stiffness matrix does not change for subsequent time steps, some reduction in the file traffic can be made through moving matrix operations forward outside the time stepping loop, as was done for the Crank Nicholson method. The following algorithm describes this case.

Algorithm for Extrapolated Backward Euler (Constant Coefficient Case)

1) Initialize the mesh & starting solution (in state I)
   - write to Ia
2) Form local stiffness matrices (half & full time steps) (in state I)
   - write to IIa (2 files)
3) Order global equations & set up sparse matrix data structures
   - read from IIa
   - write to IIIa
4) Assemble global matrices - factor them and store the factors (in State III)
   - write to IIIb (2 files)
5) Switch to state I
   - read from Ia
   - read from Ib

FOR TIME STEP = FIRST TO LAST

6) Form local load vectors for 3.3a, 3.3c (in state I)
   - write to IIb (2 files)
7) Switch to state III
   - read from IIIa, IIIb
8) Assemble global load vector for 3.3a & solve for a of 3.3a (state III)
   - read from IIa
   - write to Ib
9) Switch to State I
   - read from Ia, Ib

10) Form local load vectors for (3.36)

- write to IIb

11) Switch to state III and read ordering data and factors:

- read from IIIa, IIIb

12) Assemble global load vector for 3.3b and solve b of 3.3b

- read from IIb

- write to Ib

13) Assemble load vector 3.3c and solve for a

- read from IIb

- write to Ib (2nd file)

14) Switch to state I and perform extrapolation 3.3d

- read Ia, Ib (2 files)

The number of file transfers in N time steps of this constant coefficient extrapolated backwards Euler method algorithm is

4.5) EBE(CC) = 9 + 17N.

We summarize these file transfer formulae (4.2, 4.3, 4.4, 4.5) in Table 4.1.

Method	CN(CC)	CN(FD)	EBE(CC)	EBE(FD)
No. of file transfers	8 + 7N	4 + 8N	9 + 17N	2 + 22N

Table 4.1

Summary of Number of File Transfers
for N Time Steps

We might speculate that the file transfer formulae of Table 3.2 would give a rough guide to the (relative) execution times to be expected from these algorithms. This might be expected partly because the file transfers take a significant part of the execution time, and partly because other activities are highly correlated with them, particularly the activity of solving the linear equations.

The four algorithms have been implemented as FORTRAN modules and integrated into FEMPAK, a finite element package developed at the University of Waterloo. The running times of these implementations for a test problem are reported in Table 4.2. The test problem used was the discontinuously heated plate problem discussed in Chapter II, with the mesh shown in Figure 3.3 and a subdivision factor of 4 (288 elements).

One basic question of interest is to what extend the simple count of file transfers represents the amount of work done by the algorithms. In Table 4.3, we list the execution time per file transfer for the four algorithms, for several numbers of steps.

N	CN(cc)	CN(FD)	EBE(cc)	EBE(FD)
1	1.66	1.06	3.03	2.99
2	2.39	2.12	4.82	6.18
3	3.19	3.18	6.62	9.33
4	3.93	4.24	8.4	12.51
5	4.70	5.45	10.21	15.71
6	5.50	6.56	12.0	18.93
7	6.26	7.62	13.94	22.3
8	7.12	8.67	15.75	25.65
9	7.85	9.79	17.53	29.03
10	8.58	10.93	19.33	32.42

Table 4.2

Execution Times (seconds) for Four Algorithms
on Test Problem

N	CN(cc)	CN(FD)	EBE(cc)	EBE(FD)
1	.110	.088	.117	.124
4	.109	.118	.109	.139
7	.110	.127	.109	.143
10	.110	.130	.108	.146

Table 4.4

Execution Time per File Transfer for N steps

It is, of course, not surprising that the execution time file transfer ratio is roughly constant for each algorithm, (after the start up at N = 1), since both are roughly constant for each time step. It is more interesting, however to note that between algorithms, the execution time file transfer ratio varies by only a factor of about 1.4, at least for this example problem. It suggests then, that there is some validity to comparing these algorithms simply on the basis of file transfers.

REFERENCES

[1] J.A. George and R.B. Simpson, "The Waterloo Finite Element Package, User Guide", Appendix, Final Report, Project Number 606-03-03, Waterloo Research Institute, March, 1979.

[2] A. George, J. Liu and E. Ng, "User Guide for SPARSPAK: Waterloo Sparse Linear Equations Package", Research Report, CS-78-30, Department of Computer Science, University of Waterloo.

[3] E. Hinton and D.R.J. Owen, "Finite Element Programming", Academic Press, 1977.

[4] J.D. Lawson and D.A. Swayne, "A Simple Efficient Algorithm for the Solution of Heat Conduction Problems", Proc. 6th Manitoba Conference on Numerical Mathematics, University of Manitoba, 1976.

[5] A.R. Gourlay and J.Ll. Morris, "The Extrapolation of First Order Methods for Parabolic Partial Differential Equations, II", SIAM J. Num. Anal., Vol. 17, 1980, pp. 641-655.

[6] C.W. Gear, "Numerical Initial Value Problems in Ordinary Differential Equations", Prentice-Hall, 1971.