MINIMAL SWAPPING FINITE ELEMENT AND FINITE DIFFERENCE IMPLEMENTATIONS

by

Edward Ashford Lee

Memorandum No. UCB/ERL M84/59

27 July 1984

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley, CA 94720

MINIMAL SWAPPING FINITE ELEMENT AND FINITE DIFFERENCE IMPLEMENTATIONS

Edward Ashford Lee

Department of Electrical Engineering and Computer Science
University of California
Berkeley, CA 94720

July 27, 1984

ABSTRACT

This paper presents a general technique for managing memory when implementing certain numerical computations that require more storage than available in the main memory of the computer at hand. The computations to which the technique applies are those which iteratively update variables at a node in a mesh, and the update depends only on the values of variables at neighboring nodes. The Finite Element Method (FEM) (with linear interpolation) and several finite difference schemes, including some implicit methods, fall in this class.

The memory mangement technique reduces the amount of data swapping between main and secondary memory by a factor that is dependent on the dimensionality of the problem and the size of the main memory. The savings can be quite dramatic; for example, given a memory capable of storing 100,000 grid points, a two dimensional FEM problem requires a factor of 60 less swapping using the minimal swapping technique than a standard implementation.

Using the minimal swapping technique, a relationship is derived between the memory size and the I/O bandwidth required to achieve a certain performance for a given problem. This relationship can be used to determine the system requirements for problems of interest.

This research supported by grants from the National Science Foundation (NSF) (ECS-8211071) and Shell Development Corp, Houston, Texas.

1. Introduction

Numerical simulations of certain physical problems, such as the propagation of seismic waves in inhomogeneous media, require a great deal of computer memory. While a sizable research effort is under way to design machines with the processing power necessary to perform the numerical computations in a reasonable amount of time, the perhaps less glamorous memory problem is often neglected. In particular, if the problem requires more than the capacity of the computer main memory, a naive implementation requires the entire problem to be swapped to secondary memory in each time iteration, and the simulation grinds to a near standstill. The communication bandwidth between main memories and even the best secondary memory systems is orders of magnitude short of what is required for a reasonable execution time. Consequently, in practice, even on the most advanced machines, the size of the main memory puts a hard limit on the practical size of a simulation.

A simple modification of certain numerical simulations can greatly reduce the amount of I/O bandwidth required for a problem larger than the size of the computer main memory. The modification is based on the principle that if a set of initial conditions corresponding to part of the problem is loaded into main memory from secondary memory, then the maximum amount of computation should be done before the results are written back to secondary memory. Surprisingly, to my knowledge, this is not usually done.

1.1. A Problem Size Estimate

The simulation of the propagation of seismic waves is one of the more computationally expensive problems around. A reasonable numerical technique to apply to the problem is the Finite Element Method (FEM). This method is taken as an example of a technique in which the update of a state variable at a node in a grid is dependent only on the values of the state variables at the nodes immediately adjacent. What follows is an estimate of the size of the two and three dimensional simulations that geophysicists would ultimately like to run, in order to illustrate that the problem of running simulations that are larger than the main memory of the machine are not merely academic.

The FEM requires about ten nodes per shortest spatial wavelength for accurate simulation when linear interpolation is used (for a detailed analysis of this, see [1]). A useful three-dimensional simulation would cover a surface area of about 50,000 feet squared (about ten miles squared) and a depth of about 25,000 feet (roughly five miles). If c is the velocity of propagation of the waves in the medium, f is the temporal frequency and λ is the wavelength, then

$$\lambda = c / f .$$

Assuming, roughly, that the smallest velocity c of interest is about 5000 ft./sec. (found in water) and the highest frequency of interest is on the order of 50 cycles/sec., the shortest wavelength is about 100 feet. Thus, the smallest spacing of FEM nodes is about 10 feet apart. This implies on the order of 6×10^{10} nodes, assuming that the node spacing is regular.

Depending on the programming details, on the order of ten variables per node must be stored in memory. Thus, as much as 6×10^{11} floating point numbers need to be stored. For 32 bit words, this translates into a memory requirement of 2.4 Terrabytes (2.4×10^{12}).

A problem of this magnitude is not likely to fit in the main memory of any machine in the near future. In fact, it is not likely to fit in the *secondary* memory of any such machine either. Nonetheless, it illustrates that researchers wishing to perform three dimensional simulations will not be satisfied if they are restricted to the main memory of any mortal machine.

Consider, alternatively, a two dimensional model of the same magnitude as the one above. That is, 50,000 feet long by 25,000 feet deep. With a node spacing of ten feet, this means a total of 1.25×10^7 nodes, implying a storage requirement of 1.25×10^8 floating point numbers. For 32 bit numbers, this translates into 500 Megabytes. This is orders of magnitude larger than the main memory of present day machines, but at least it is not inconceivable that with substantial secondary memory, such a simulation might be implementable.

2. One Dimensional FEM

For simplicity, I will begin by illustrating the technique for a one dimensional simulation using the FEM. The technique applies in the same way to finite difference schemes. Although few physical problems are actually one dimensional, extensions to more dimensions will be considered after the one dimensional results are established.

2.1. Standard FEM

Consider a problem to be solved using the FEM with linear interpolation and P nodes in a straight line. Linear interpolation in FEM implies that at each time iteration, the new state at each node is affected only by its nearest neighbors. If T time steps are desired, then the total amount of computation to be done is PT node updates, where a node update will be the basic unit of work considered.

Assume that the main memory capacity is M nodes with $P \gg M$. Then in a naive implementation, the initial conditions for M nodes are loaded into memory, the nodes are updated one time step, and the results written back to secondary memory. Then another M initial conditions are loaded; the process is repeated until all P nodes are updated, at which point the second time step is begun.

The communication workload can be measured using a unit called a swap. One swap is the operation of writing the contents of main memory (M nodes) to secondary memory, and reading back an equivalent amount. The total number of swaps for the standard implementation is

$$S_{std} = PT/M, \tag{1}$$

which can certainly be improved.

On many systems, the programmer is encouraged not to think about the details of memory management. In a virtual memory machine, for example, this standard technique will result by default if the program accesses all elements of the large array or vector containing the node state variables.

The FEM simulation is usually formulated in state variable form as

$$\mathbf{u}(n+1) = \mathbf{A}\mathbf{u}(n) + \mathbf{b}(n) \tag{2}$$

where $\mathbf{u}(n)$ is the state vector at time n (containing the state variables for all the nodes), $\mathbf{A}$ is a sparse matrix containing the physical coefficients, and $\mathbf{b}(n)$ is some forcing term. This formulation, however, conceals the nearest-neighbor nature of the problem that is inherent in the physics. A programmer is likely to completely update the state before proceeding to the next state update, and again, the standard technique results by default.

2.2. Minimal Swapping FEM

To minimize the amount of swapping required to perform a simulation, the program should make maximal use of each swap. Since a node update is dependent only on the nodes immediately next to it, it is possible to do more than just one time step per swap. Starting with the initial conditions for M nodes in memory (at $t=0$), the update can actually only produce $M-2$ new states (at $t=1$) because the nodes neighboring the ones on the end are not in memory. (This effect is negligible for large M and so is not taken into account in computing the communications workload for the standard implementation.) Observe, however, that once $M-2$ new node values have been computed, a new time step can be computed producing $M-4$ new node values at time $t=2$. This process can be repeated, producing a triangle of new node values, as shown in Figure 1. The number of node updates that can be done from one set of M initial conditions is

$$W = \begin{cases} \left(\frac{M}{2} - 1\right)^2 + \frac{M}{2} - 1 & \text{; for } M \text{ even} \\ \left(\frac{M-1}{2}\right)^2 & \text{; for } M \text{ odd} \end{cases} \tag{3}$$

which for large M is approximately $(M/2)^2$. The naive implementation only accomplishes M node updates per swap, so this implies that the number of swaps can, in principle, be reduced by a factor of about $M/4$. For large memories, this can be quite significant.

More work is required to make the updates coincide at the boundaries of the swaps. To accomplish this, divide the problem with P nodes into frames of width M. Assume that the boundary conditions of the problem are time invariant, so only one storage location is needed at each end of the linear mesh to store them. Then if the leftmost frame (the set of M leftmost initial conditions) is loaded, the pattern of node updates that is possible is shown in Figure 2.

Instead of doing all possible updates, consider the following strategy. Update nodes up to time step $t=N$, where $N<M$. The pattern of node updates is shown in Figure 3. Now keep in memory the N node values along the sloping edge, and bring in from bulk memory B more initial conditions, such that

$$M = B + N. \tag{4}$$

Now it is possible to update the nodes in the parallelogram of Figure 4. Repeat the parallelogram updates until the boundary condition at the other end makes possible another rhombus shaped update pattern. Figure 5 shows how the N complete time steps are divided. Once these are complete, the entire process begins again at $t=N$.

Assuming that the problem size P is much greater than the memory size M, most of the update operations will be of the parallelogram type shown in Figure 4. It is easy to see that the number of node updates in the parallelogram is

$(N-1)(B-1)$. Subject to the constraint that $B+N=M$, this is maximized if $N = B$. Therefore, using this strategy, the optimal choice of N and B is

$$N = B = M/2. \tag{5}$$

Neglecting the effect of the rhombuses at the ends of the grid (Figure 5), the total number of swaps for the minimal swapping solution will be

$$S_{\min} = \frac{P}{B}\frac{T}{N} = \frac{4PT}{M^2}. \tag{6}$$

Comparing this with equation (1) it is clear that the amount of swapping has been reduced by a factor of

$$\frac{S_{std}}{S_{\min}} = \frac{M}{4}, \tag{7}$$

which for a large memory is a significant savings.

If the effects of the rhombuses on the ends are considered, the situation is actually improved because the rhombuses represent more work for a single swap than do the parallelograms.

3. Two dimensional Problems

The two dimensional FEM simulation is a much more common problem, and is treated in a similar way in this section.

3.1. Standard FEM

Assuming a memory size of M, a naive program loads a $B{\times}D$ area of initial conditions, computes one time update, and writes the result back to secondary memory, where $M = BD$. The total number of swaps for an standard implementation of a problem with P nodes and T time steps is

$$S_{std} = \frac{PT}{BD} = \frac{PT}{M} \tag{8}$$

Again, this can be improved.

3.2. Minimal Swapping FEM

The technique for maximizing the amount of work done per swap is similar to the one dimensional case. Figure 6 illustrates the analogous division of N time steps. Again assuming that the problem size P is much larger than the memory size M, most of the node updates take place in the parallelepipeds of Figure 7. To maximize the number of node updates performed for each of these typical swaps, we clearly need to maximize the number of nodes in this parallelepiped, subject to the constraint that

$$M = D(B-1) + B(N-1) + N(D-1) \tag{9}$$

which is the total number of nodes required to start the computation. The number of node updates in the parallelepiped is

$$W = (B-1)(D-1)(N-1) \tag{10}$$

because we do not count the initial nodes as node updates. This function can be maximized subject to its constraint using Lagrange multipliers, and the result is

$$B = N = D, \tag{11}$$

and the optimal dimensions are

$$D(B-1) = B(N-1) = N(D-1) = M/3. \tag{12}$$

When combined with equation (11) this becomes a quadratic that can be solved yielding

$$B = N = D = \frac{1 + (1 + 4M/3)^{1/2}}{2}, \tag{13}$$

which is approximately $(M/3)^{1/2}$ for large M. This result is intuitive, because the volume of a parallelepiped is maximized subject to a surface area constraint by making it a cube.

The total amount of swapping required to run a simulation with P nodes for T time steps using the minimal swapping technique is

$$S_{\min} = \frac{PT}{(D-1)^3} \cong \frac{PT}{(M/3)^{3/2}}. \tag{14}$$

So the advantage gained by using the minimal swapping technique is a factor of

$$\frac{S_{std}}{S_{\min}} \cong \frac{M^{\frac{1}{2}}}{3^{3/2}} \qquad (15)$$

reduction in the number of swaps. For a memory capable of storing 100,000 nodes (about a 4 Megabyte memory) this is a reduction in the required swapping by a factor of about 60, which is again significant.

Note that the improvement is dependent only on the size of the memory, not on the size of the problem, as long as the problem is much bigger than the memory. However, as in the one dimensional case, if the problem is not much bigger than the memory, then the edge shapes in Figure 6 become significant, and the situation is actually improved. The advantage of the minimal swapping technique will be greater.

4. Three Dimensional Problems

All of the arguments presented for the lower dimensional cases apply as well to three dimensional problems, but visualizing the problem is considerably more difficult because the solution volume that must be carved up is four dimensional. Nonetheless, let us try.

In the two dimensional problem, the parallelepiped of Figure 7 represents a transition in time from the square at its base (the initial conditions) to the square at its crest, where two side faces of the parallelepiped were previously computed and given as side conditions. These side conditions are used as needed as time progresses. Similarly, the equivalent to the parallelepiped for the three dimensional problem is the progression in time from the cube in Figure 8a (the initial conditions) to the cube in 8b, where the dashed cube shows the relative position of the original cube; side conditions are used along the way but are not drawn because they are used as the computation progresses through the fourth dimension, time. Figure 9 shows the shape of the volume through which the simulation progresses; at the beginning of a frame computation, this shape is filled by the initial conditions and the side conditions. This volume represents the memory required to run the simulation through the appropriate hyper-parallelepiped. The optimal dimensions of the hyper-parallelepiped are, as before, all equal. It can be shown then, that to run an $N \times N \times N \times N$ hyper-parallelepiped, the memory volume of Figure 9 is $4N^3$ nodes, so that

$$N = (M/4)^{1/3} \tag{16}$$

and the amount of work per swap is

$$N^4 = (M/4)^{4/3} \tag{17}$$

and thus

$$S_{\min} = \frac{4^{4/3} PT}{M^{4/3}}. \tag{18}$$

The savings of the minimal swapping method is a factor of

$$\frac{S_{std}}{S_{\min}} = \frac{M^{1/3}}{4^{4/3}} \tag{19}$$

using the fact that for the three dimensional problem S_{std} is the same as for the two and one dimensional problems, and is given in (1) and (3). For $M = 100{,}000$ this represents a reduction in the amount of swapping by a factor of about 7.3. The savings is clearly much more modest for three dimensional problems.

5. Implicit Solution Techniques

The minimal swapping technique can also be applied to implicit techniques in a fairly straightforward way. Consider, for example, the solution technique given by

$$\mathbf{A}\mathbf{u}(n+1) = \mathbf{B}\mathbf{u}(n). \tag{20}$$

Assume that the **A** and **B** matrices are sparse in just the way so that an element of the $\mathbf{u}(n+1)$ vector depends only on nearest neighbors in the $\mathbf{u}(n+1)$ and $\mathbf{u}(n)$ vectors. (Note that I mean "nearest neighbors" in a physical sense; they need not be adjacent in the vector). This implicit method can sometimes be approximately solved using the Jacobi method. Write the **A** matrix as a sum of an upper triangular matrix, a diagonal matrix, and a lower triangular matrix, so that

$$D(U+I+L)u(n+1) = Bu(n) \tag{21}$$

where

$$A = D(U+I+L) \tag{22}$$

and thus

$$u(n+1) = D^{-1}Bu(n) - (U+L)u(n+1). \tag{23}$$

The value of $u(n+1)$ can be approximated by defining

$$u_0(n+1) = u(n) \tag{24}$$

and

$$u_i(n+1) = D^{-1}Bu(n) - (U+L)u_{i-1}(n+1) \tag{25}$$

where $i = 1,2,...,I$. If the method converges, then I is selected to achieve the desired accuracy. Except for the term $Bu(n)$ this looks just like the iteration in time of an explicit method, but with I times as many iterations. The presence of the $Bu(n)$ term merely means a memory penalty that must be paid, because this term is only updated every I iterations, but otherwise the iterations of the above equation can be implemented using the minimal swapping technique.

6. A Simplified Programming Method

Because of the peculiar shapes involved in the minimal swapping technique, its implementation is not completely straightforward. Where the necessary programming effort is not justified, a simple technique with suboptimal efficiency can be employed satisfactorily.

Since the end results of a run of the parallelogram, parallelepiped, or hyper-parallelepiped are node values that are valid at different points in time, it seems reasonable to store along with the state of each node the time at which that state is valid. Then, when initial conditions are loaded, the parallelepiped of Figure 7 can be "grown" like a crystal starting in the lower, forward, right hand corner and performing a node update whenever possible. This simple technique requires examining all nodes to see if they can be updated, so the number of nodes to be operated on will be

$$MN = \frac{M^{3/2}}{3^{1/2}} \tag{26}$$

rather than

$$(B-1)(D-1)(N-1) \cong \frac{M^{3/2}}{3^{3/2}} \tag{27}$$

Using this algorithm, the minimal swapping technique involves about three times as many node operations as the standard technique. In addition, each node update is slightly more complicated because of the tests that must be performed to determine whether the update can be done. This will be somewhat offset by the fact that for two thirds of the node operations no update will be performed; but in any case, it is hard to imagine that it would complicate the node operations by more than a factor of two or so. Six times the computational complexity is a small price to pay for 1/60 times the communications with secondary memory, when such communications dominate the execution time of the program.

More elegant implementation techniques immediately come to mind. For example, one could load the side conditions from secondary memory as they are needed, and write node values to secondary memory when the nodes can no longer be updated. This will reduce the amount of memory needed to achieve a given reduction in the swapping rate, but the precise amount will depend on the computer architecture. It will also reduce the workload, because every time a node is encountered which cannot be updated beyond a given time step, it is banished from memory, and the program will not attempt to update it on future time steps. Many other machine dependent embellishments are possible.

There is one potential difficulty that arises from the use of the minimal swapping technique. Since at any given time the states at different nodes are valid for different iteration numbers, examining the entire grid at a given iteration number (grid snapshot) is difficult. The solution, if snapshots of the grid are required at random times, seems to be to monitor the time variable corresponding to each node, and when it reaches the iteration number desired, output the state. This will add some code to the inner loop. An alternative is to restrict snapshots to the times when they are immediately available in memory.

7. Designing or Selecting a Suitable Computer System

Since the amount of swapping done when the minimal swapping algorithm is used is dependent on the size of the memory, a relationship can be established between the memory size and the I/O bandwidth that is required to achieve a certain performance. Specifically, the two parameters may be traded off to achieve a computer system optimized for these numerical problems. To illustrate the tradeoff, I will consider only the two dimensional problem.

Recall that P is the total number of nodes in the problem and T is the total number of time steps to be computed. Recall also that M is the total number of nodes that fit in main memory and that S_{std} is the number of swaps required to solve the problem using the standard method, and $S_{\min}$ is the number required using the minimal swapping method. Define C as the communication bandwidth available between main and secondary memory (in bytes/sec) and T_C as the acceptable total amount of time for the system to spend on communication between these memories for the execution of the entire simulation (in seconds). Further assume that 10 floating point numbers per node must be stored, and that 4 bytes are required per floating point number. Then the bandwidth required for the standard implementation can be computed as

$$C = \frac{4\times10\times MS_{std}}{T_C} = 40\,\frac{PT}{T_C} \tag{28}$$

using equation (8).

Let us try some typical numbers. Assume that it is acceptable for our computer system to devote one hour to communication so $T_C = 3600$ seconds.

Assume also that we wish to run 5000 time steps, so $T = 5000$. The two dimensional problem from section 1.1 had $P = 1.25\times10^7$ nodes. Equation (28) yields

$$C = 694 \text{ Megabytes} / \text{sec}$$

which is not a trivial requirement.

On the other hand, the communication bandwidth required using the minimal swapping method is

$$C = \frac{4\times10\times MS_{\min}}{T_C} = \frac{40\times3^{3/2}\,PT}{T_C M^{1/2}} \tag{29}$$

using equation (14). For M=100,000 nodes, using the same numbers as above, this becomes

$$C = 11.4 \text{ Megabytes} / \text{ second}$$

which is much more manageable.

An interesting use of equation (29) is to solve for M as a function of the system bandwidth C.

$$M = \left[\frac{40\times3^{3/2}\,PT}{CT_C}\right]^2 \tag{30}$$

With the numbers assumed above, this is plotted in Figure 10. This figure indicates that to solve the two dimensional problem in a matter of hours, assuming that the arithmetic hardware is designed for performance comparable to the memory management, then a communication bandwidth of 10 to 20 Megabytes/second implies a memory size of one to five Megabytes. The required memory size grows very fast as the bandwidth goes below about five Megabytes/second. Of course, these figures are quite sensitive to parameters such as the size of the problem and the acceptable time devoted to communications, but they are useful to show the orders of magnitude of the numbers involved.

8. Conclusions

I have presented a general technique which can achieve significant reductions in the amount of memory bandwidth required to run certain large numerical simulations when the problem size is larger than the main memory size of the computer. The reduction in bandwidth depends on the dimensionality of the problem, with greater advantage for smaller dimensions, and on the size of the main memory. Thus, a tradeoff between memory size and I/O bandwidth is established; this can be used to design or select computer systems optimized for solving the kinds of numerical problems considered.

The technique applies to all numerical methods where iterative updates of a value at a node are made dependent on the values at the neighboring nodes. The technique also applies to some implicit methods with similar nearest-neighbor dependencies.

9. Acknowledgements

Special thanks to Rhonda Righter for valuable help in simplifying and formalizing the technique. Ted Baker observed that the technique could be applied to implicit solution techniques. I had additional valuable discussions with Bill Moorhead, of the Shell Development Company, Prof. Frank Morrison, Prof. William Kahan, Prof. Dave Messerschmitt, Prof. Beresford Parlett, and Niklas Nordstrom. With the exception of Bill Moorhead, all are from U. C. Berkeley.

References

1. Smith, W. D., *A Finite Element Study of the Effects of Structural Irregularities on Body Wave Propagation*, PhD Thesis in Geophysics Department, U. C. Berkeley 1975.

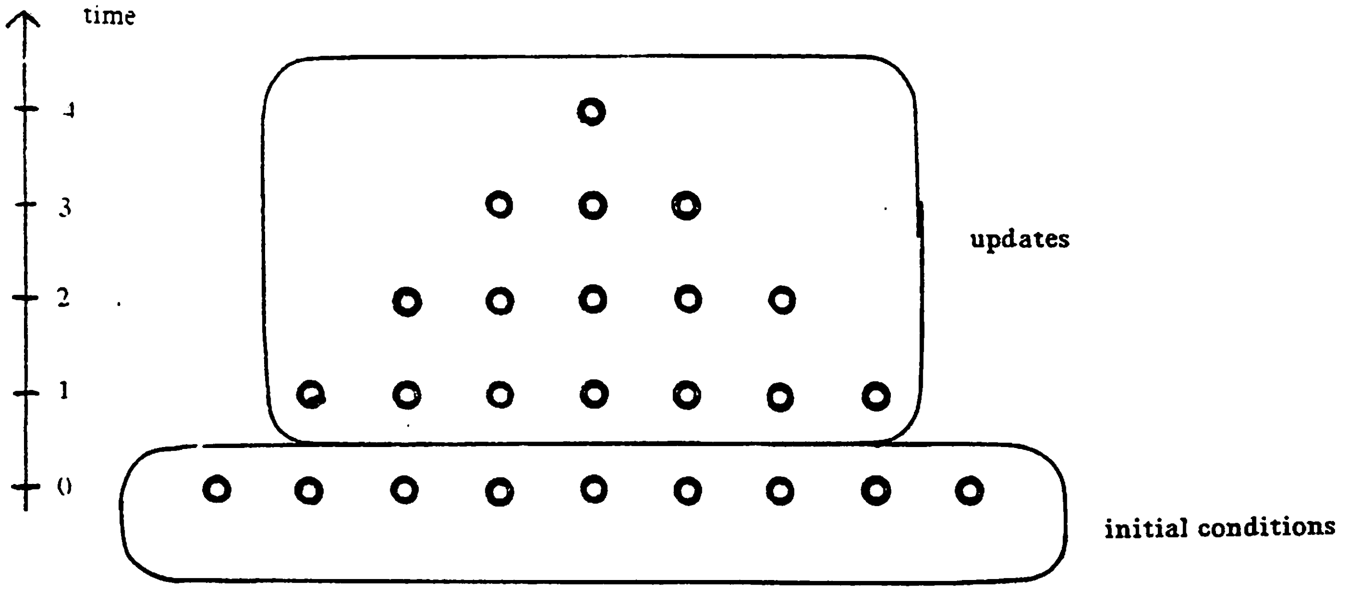

Figure 1: The triangular pattern of node updates that can be made from M=9 initial conditions with no further information

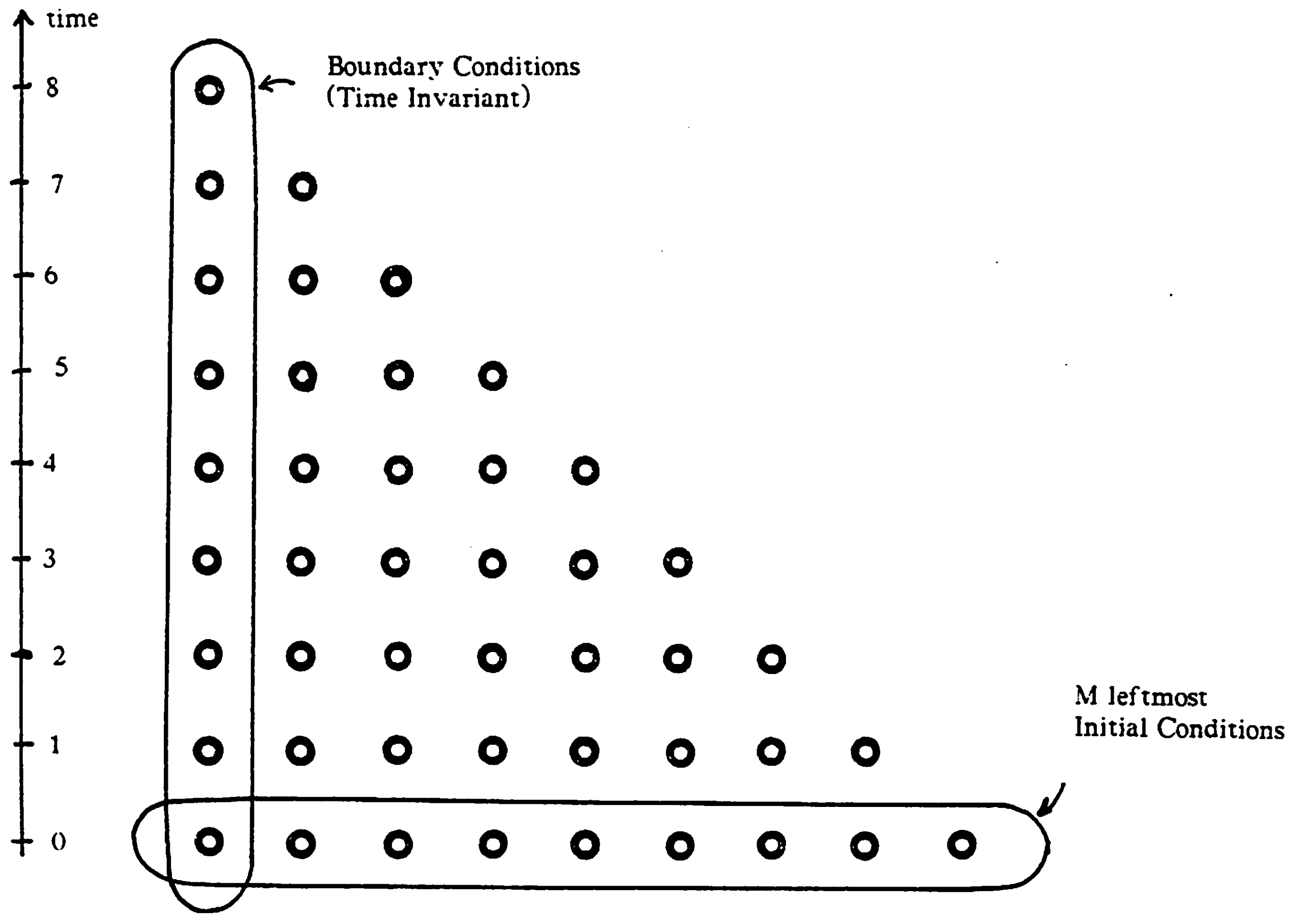

Figure 2: The triangular pattern of node updates that can be made in the leftmost frame when time invariant boundary conditions are known.

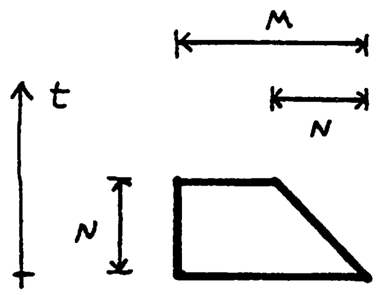

Figure 3: Truncated Version of the triangle in Figure 2.

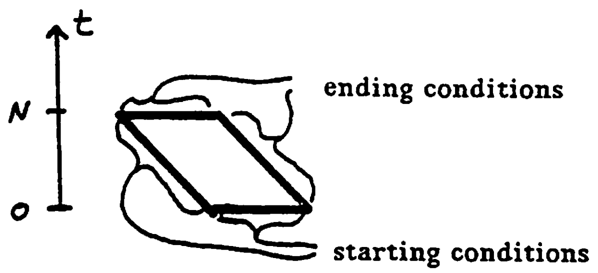

Figure 4: Parallelepiped representing the block of computations that will dominate.

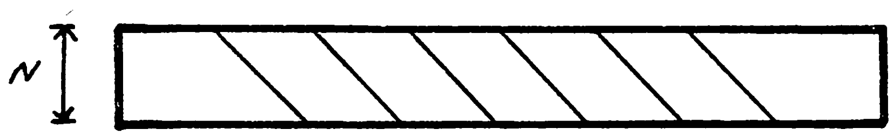

Figure 5: Division of an N time step block.

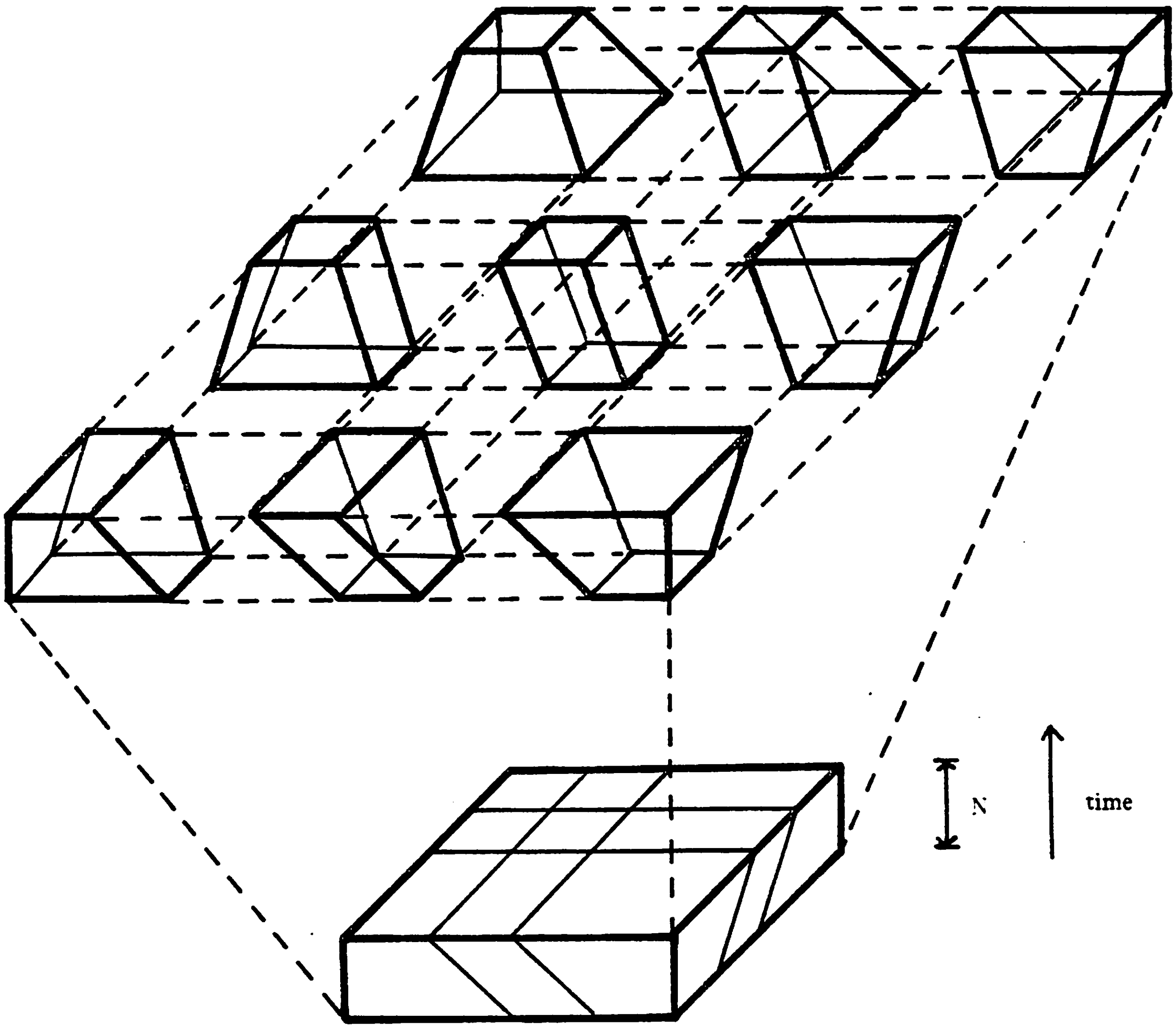

Figure 6: Breakdown of N time steps of the overlapped simulation for minimum swapping.

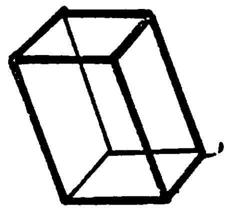

Figure 7: The parallelepiped that dominates the computation when the main memory is much smaller than the problem size.

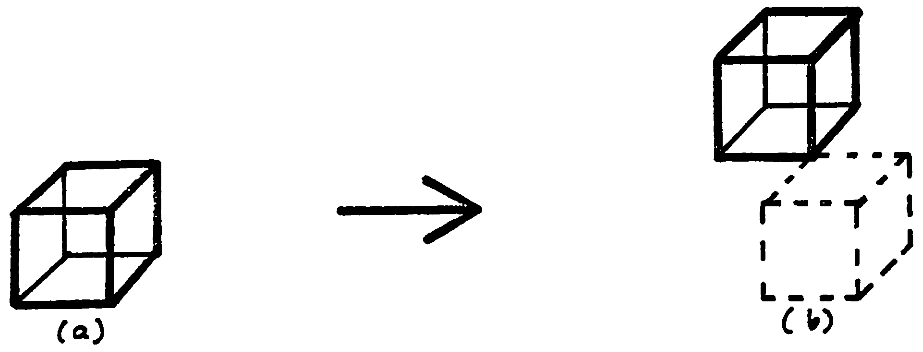

Figure 8: The progression in time representing the hyper-parallelepiped that dominates the computation for a three dimensional simulation.

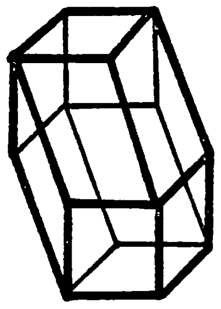

Figure 9: The volume of memory required to compute the hyper-parallelepiped of Figure 8.

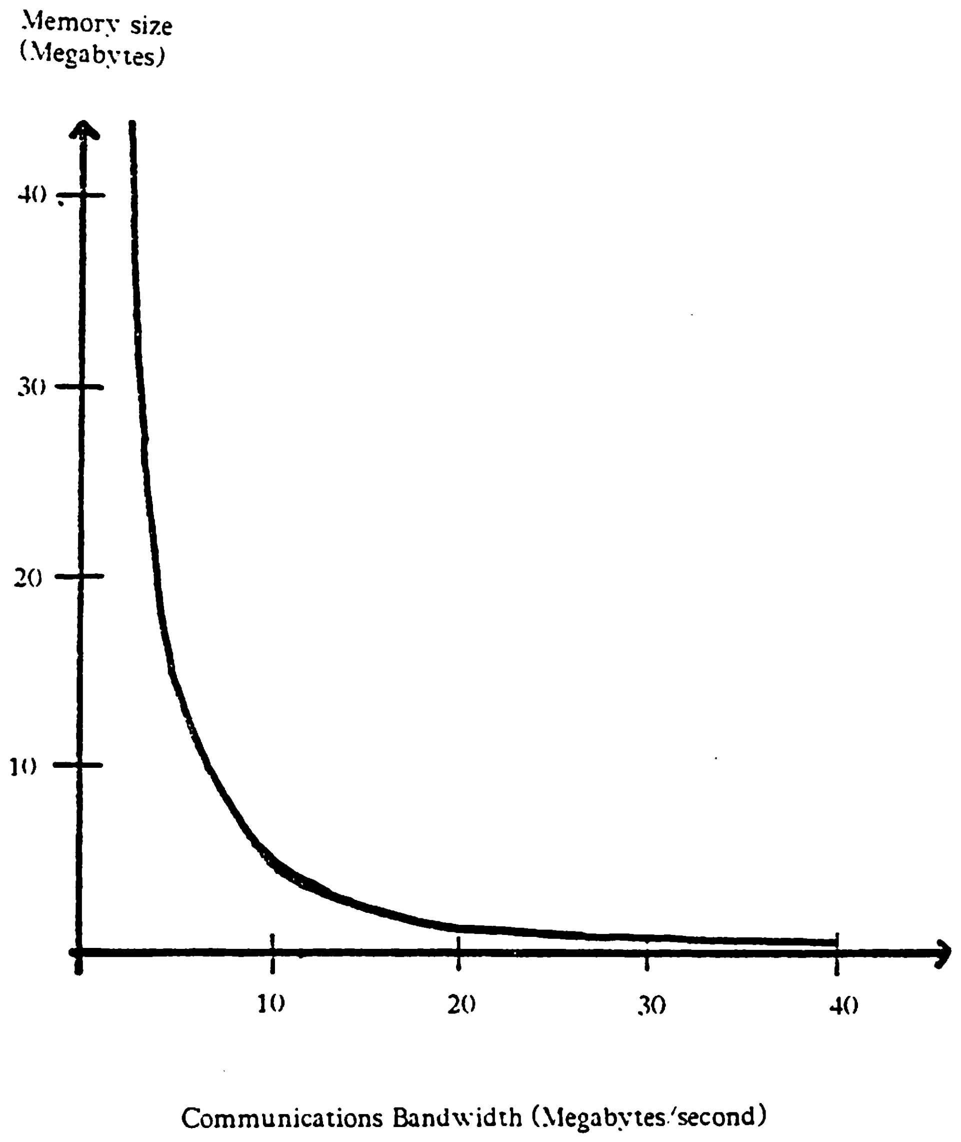

Figure 10: Tradeoff between memory size and communications bandwidth of a system devoting one hour to communication while solving a two dimensional problem with 12.5 million nodes and 5000 time steps. The minimal swapping technique is assumed.