Optimisation and large scale computation in integral equation scattering analyses

S J Dodson, S P Walker†, M J Bluck
Mechanical Engineering Department
Imperial College of Science Technology and Medicine
London SW7 2BX

Abstract : **The kinds of difficulties posed by large scattering computations change as larger problems are addressed. Unfavourable cost scalings make the performance of small, core, portions of code dominant, and require that they, and the overall code structure, be optimised for large scale computation. This is discussed in the context of rcs and scattering computations of multi-wavelength bodies using a time domain integral equation treatment. Examples presented include the NASA almond evaluated at 25 wavelengths long, and an assembly of 101 spherical scatterers of ~1/2 wavelength diameter each, occupying a volume of side ~250 wavelengths.**

1. Introduction

Many problems in CEM share the unattractive characteristic of having computational costs which increase sharply with problem (electrical) size[1], and transient scattering and radar cross section computations, on which this paper will concentrate, are a prime example of this. Whilst 'small' problems are thus not computationally difficult, something of a ceiling in electrical size is quickly encountered when tackling only slightly larger ones. Progress requires a judicious combination of frugal algorithm, recasting for large, and in practice massively parallel, computers, and careful coding and optimisation. Once very large problems are tackled, additional difficulties arise in what are otherwise peripheral issues; the areas of mesh generation, and results manipulation and display quickly become major computational tasks in their own right.

In this paper we will describe attempts to reduce the computational cost of integral equation time domain (IETD) analyses[2], using the above approaches, but with particular emphasis on practical techniques we have found effective in optimisation. Additionally, the issues and problems of mesh generation and results display will be considered.

In the remainder of this introduction we will summarise very briefly the algorithms being employed. Section 2 will discuss the various aspects of optimisation which arise in trying to use these on large problems, section 3 considers pre- and post-processing issues, and in section 4 we will give examples of the performance of the codes.

For scattering from a perfectly conducting body, subject to some incident wave, the surface field is given by[3]

$$2\pi \mathbf{H}(\mathbf{r},t) = 4\pi \mathbf{H}_{inc}(\mathbf{r},t) + \int_{\partial\Omega} \left(\mathbf{n}' \times \mathbf{H}(\mathbf{r}',t^*)\right) \times \frac{\hat{\mathbf{R}}}{R^2} + \left(\mathbf{n}' \times \frac{\partial \mathbf{H}}{\partial t}(\mathbf{r}',t^*)\right) \times \frac{\hat{\mathbf{R}}}{cR} ds' \quad (1)$$

where $\mathbf{r}$ and $\mathbf{r}'$ are surface locations, $\mathbf{R} = \mathbf{r}' - \mathbf{r}$, wave speed is c, and time t and retarded time $t^* = t - R/c$. Most of what follows in this paper is independent of the details of the discretisation of (1) employed, but here we will use an implicit

† Correspondence: s.p.walker@ic.ac.uk, +44 (0)171 823 8845 fax, +44 (0) 171 594 7058, http://www.ic.ac.uk/mechanics

curvilinear isoparametric approach. Details are rather tedious[4], but eventually a matrix equation for the field at the next timestep is obtained, with the field expressed as a weighted sum of surface fields over the rest of the body at times up to one transit time (W timesteps) ago. For discretisation with N nodes i and j, at timestep $k+1$, this new field is given by

$$2\pi\mathbf{H}_i^{k+1} - \sum_{j=1}^{N} \alpha_{i,j}^{0}\mathbf{H}_j^{k+1} = \qquad (2)$$

$$4\pi\mathbf{H}_{inc,i}^{k+1} + \sum_{j=1}^{N}\sum_{w=1}^{W} \alpha_{i,j}^{w}\mathbf{H}_j^{k+1-w}$$

The coefficients α (3 by 3 matrices, later reduced to 2 by 2 by application of pec boundary conditions) characterise the influence of components of the historical field at j on components of the new field being sought at i. Note that an explicit treatment is obtained if the timestep is so small that the new fields at nodes adjacent to i do not influence the field at i. Otherwise, (2) represents a sparse matrix equation:

$$\mathbf{A}\cdot\mathbf{h}^{k+1} = \mathbf{c}^{k+1} \qquad (3)$$

where $\mathbf{h}^{k+1}$ is $(\mathbf{H}_1^{k+1},\ldots,\mathbf{H}_N^{k+1})^T$, and the vector $\mathbf{c}$ results from evaluation of the summations on the right of (2).

The main components of the computational work, our primary interest here, are clear from (2) and (3). For a given discretisation, the number of nodes will vary with f^2, so the work of forming the coefficients α will scale with f^4. Cost of formation of each $\mathbf{c}$ thus scales with f^4, and with typically the number of timesteps required being roughly proportional to electrical size, we obtain the usual f^5 cost scaling of IETD approaches. Storage costs of α scale with f^4, and of the field history (nodes x timesteps) with f^3. Fuller discussion of these scaling issues is given by Miller in the paper cited earlier[1].

Illumination with a pulse which is short compared to the body size results in fields which are small over most of the body most of the time. This can be exploited by (a) integrating in (1) over only those portions of the body where the field was significant at the relevant retarded time, and (b) only evaluating the field at locations where it is expected to be found to be significant. With the fraction of the body surface over which significant fields obtain declining roughly with frequency, these two approximations together reduce the scaling of cost with frequency by up to two powers. A fuller description of this, and an investigation of the reduced accuracy and associated cost reductions, has been presented earlier[5,6]. It is on the implementation and optimisation of this modified algorithm that this present paper will concentrate.

2. Optimisations

2.1 Memory and operation tradeoffs

For the conventional IETD the main storage requirement, the coefficients α, scales with f^4. These can be formed once at the beginning of the job and then multiplied by historical values to form the right hand vectors at every timestep. This we will term the 'in-core' method. The size of this matrix is normally ~6 x 2 x 2 x N x N words. For example, for a 1,202 node NASA almond[7] (approximately 4 wavelengths long) the matrix occupies 133Mb if stored (as it is) in single precision. This is sizeable but not impossible on a workstation. However, for say a ~25 wavelength case (such as we will analyse later) the storage required is about 327 Gb; ~ten times the entire core of the largest of supercomputers.

This storage strategy would limit problem sizes on say a 1 Gb workstation to ~3,200 nodes, or an almond ~6.5 wavelengths long. This job would however take only ~1/2 hour to form the matrix, and ~1 hour to perform the timestepping, making the storage clearly the limiting factor.

An alternative approach is to generate the matrix coefficients afresh at every timestep, use them and discard them. The overall scalings are the same, but the costs are increased by a constant factor of rather more than an order of magnitude

(the cost of forming the coefficient, relative to 'using' it in the multiplications of (2)). The memory savings are considerable, with the now dominant history scaling with f^3. For the same 3,200 node almond this requires only ~34 Mb.

Repeated reading of the coefficients from disk, where the provision of tens of Gb is no real problem, is an alternative approach. For a workstation this could be practicable and quicker. However, large problems require large computers, which in practice means parallel computers, and the speedup in input of such machines is generally far below their increase in processing speed, making the recalculation far faster.

Similar arguments cause repeated recalculation of the matrix coefficients to be the best approach for the modified algorithm. This is reinforced by the fact that the approximations in the approach result in much of the matrix never actually being used and hence never calculated.

History storage for the modified algorithm is much reduced, as only the history of 'active' periods is stored. Being a handful of pulse durations worth at any one node, it has a cost scaling with f^2. This is the same scaling, and indeed is lower in amount than, the storage requirements of the mesh data itself. For example, storage requirements of the 25 wavelength almond analysed later using this approach were about 300 Mb. The history alone in the conventional approach would have required 200 Mb; the history in the modified approach was ~50Mb.

2.2 Parallelisation

The whole area of parallelisation, covering domain decomposition, minimising of communications, and load balancing, is large. There has been work on frequency domain integral equation treatments[8,9], but little has been published[10,11] on time domain integral methods. This latter was for the conventional IETD approach. There are significant changes required to exploit massively parallel machines properly using the modified algorithm. A paper detailing these has recently been written[12], and we will summarise the approach here.

Storage of the same data on all processors must be avoided, and this is done by partitioning the mesh by elements, with their associated mesh and history data, over processors. In the modified approach only a subset of 'field' nodes i equation (2) are involved at each timestep, from each of which integration is performed over a subset (different for each i, and differing at each timestep) of 'boundary' nodes j. Each processor is caused to integrate from every relevant i over such boundary nodes j as it is custodian of. However, the nature of the propagation of the pulse is such that both these groups of nodes tend to be spatially fairly contiguous. Given the careful node numbering optimisation schemes built into most CAD packages, they tend also to be fairly contiguous in terms of node number. Deliberate randomisation of node numbering is required to achieve good load balancing, but when this is done very nearly equal work is performed on each processor. With very little interprocessor communication required, overall parallelisation is very effective.

2.3 Profiling

Code development, debugging and so on naturally tends to be performed using small test problems. Whilst different portions of the code may be known in theory to have different cost scalings, fixed overhead costs normally dominate till sizeable problems are tackled. It is only on the largest of jobs that the portions of the code having asymptotically dominant costs become clear. Equally, of course, it is only if the intent is to use the code for such jobs that the effort of such identification and optimisation is warranted. A further discouragement is that the resulting optimisations, to be of most use, are often machine specific, driven say by the particular pipelining and cache arrangements of a particular processor.

The technique of 'profiling' is invaluable in identifying those portions of a code where most effort is being expended. It is often the case that

a tiny fraction, by lines of code, is found to account for the vast majority of the execution time. Sometimes, an optimised library routine may be available, for a 'generic' activity such as matrix-vector multiplication, and this can be substituted. Alternatively, manual optimisation can bring significant gains.

We would comment in passing that in our experience such optimisation should be confined strictly to those portions of the code which are found to be truly costly. The kinds of optimisation we will describe shortly almost invariably make the code more complicated, and harder to read and understand. Introduction of temporary scalar variables, loop unrollings and so on, need copious in-code documentation to be comprehensible even to the developer himself only a short while later.

In the conventional IETD, run 'in core' it was suggested above that repeated formation of the right hand side of (2) was the dominant cost. This is confirmed by the profile of figure 1. The routine performing this (pecformrhs) accounts for over 90% of the cpu cycles. The second most expensive, amatmult, is part of the iterative solver used in repeated solution of the sparse matrix equation in (2). This profile was run for a 1,202 node, 4 wavelength body. For the small test cases on which the code was originally developed things are markedly different.

What matters changes greatly when we move to the modified algorithm, with repeated matrix coefficient calculation, but only for node-element pairs corresponding both to significant expected field values and significant historical (retarded) fields. Figure 2 shows the profile of this case, for the same 1,202 node, 4 wavelength test body. We see that the majority of the time is spent in a routine 'ipecnsq9_': the routine which integrates over a non-self element, on a pec body, discretised with a 9 noded quadrilateral element. What was dominant, the right hand side formation (routine p3formrhs), is now an order of magnitude smaller in relative importance.

Forms of profile like these are very attractive from the point of view of optimisation, with the main computational cost so localised, such that minor changes can provide significant gains. The routine ipecnsq9 for example, comprises only some ~200 lines of code, some 10 of which in turn account for the vast majority of ipecnsq9 time, in a package of ~14,000 lines.

2.4 Coding modifications

Modern optimising compilers themselves perform a variety of optimisations to make the code run faster. The level of optimisation which results in the best performance varies from program to program, and changing this compilation option can often result in run-time improvements. However, if the algorithm is poorly organised, any amount of optimisation is unlikely to result in efficient code.

The profile of the modified code shows that the majority of the time is spent in the non-self element integration routine. This routine integrates from a field node over an element by performing the following summations:

$$\alpha_i^m = \sum_{q_1}\sum_{q_2}\sum_{\beta}\sum_{\alpha} S_\alpha(\xi_{q_1},\eta_{q_2}) \left[\frac{T_\beta(\tau)}{R^3}+\frac{\dot{T}_\beta(\tau)}{\delta t R^2}\right][A']\left|\mathbf{J}(\xi_{q_1},\eta_{q_2})\right|\omega_{q_1}\omega_{q_2} \tag{4}$$

These summations are over respectively the 9 spatial shape functions S , the three temporal shape functions T, and the two sets of Gaussian integration locations for the two dimensional surface integral.

One obvious and classical optimisation performed by compilers is to move expressions which are 'loop invariant' outside of the loop:

(a) Original Code

```
do I = 1, N
   do J= 1, M
      A(J,I) = A(J,I) + B(I) * C + D
* E
   end do
end do
```

(b) Optimised Version

```
temp = D * E
do I = 1, N
   temp1 = temp + B(I) * C
   do J = 1, M
      A(J,I) = A(J,I) + temp1
   end do
end do
```

For example, in (4) all the terms (except δt) in the above expression are dependent on Gauss point locations q_1 and q_2. Only S_α is dependent on α and only T_β and $\dot{T}_\beta$ on β. Therefore, we can rewrite (1) in a slightly different form as

$$\alpha_i^m = \sum_{q_1}\sum_{q_2}[A']\left|\mathbf{J}(\xi_{q_1},\eta_{q_2})\right|\omega_{q_1}\omega_{q_2}\frac{1}{R^2} \qquad (5)$$

$$\sum_\beta\left[\frac{T_\beta(\tau)}{R}+\frac{\dot{T}_\beta(\tau)}{\delta t}\right]\sum_\alpha S_\alpha(\xi_{q_1},\eta_{q_2})$$

(with 1/R itself actually evaluated outside the summation, but this becomes cumbersome to show algebraically). For simple expressions most compilers will perform this optimisation automatically, and (we at least) commonly write code to reflect the most natural form of the algebra, assuming that the compiler will optimise later. However, for more complicated expressions such as (4) we have found it is much more effective to separate the terms by hand.

Another and often more important reason to breakdown each term into its dependencies is to determine the loop ordering. This is relevant in optimising memory references. For multi-dimensional arrays in FORTRAN, iterating the first subscript in a loop gives fastest memory access, with unit stride (sequential assessing of adjacent memory locations) optimising benefit gained from the cache. Therefore, we order the dimensions of our arrays according to the loop ordering in the routine which dominates the run-time. This means if an array requires (Gauss point, β,α) subscripts we then order the subscripts $A(\alpha,\beta,q_1,q_2)$ if optimisation requires the loops to be ordered as shown in (2). It is a farsighted developer who can foresee the details of loop ordering of dominant-cost routines at the early stage at which data structures are specified, and such optimisations can then require considerable retrospective book-keeping code modifications. This provides another strong incentive to confine them to truly costly code portions.

RISC workstations, such as the DEC alpha, can perform multiple instructions per clock cycle and can pipeline operations. This can allow further optimisations to be made. A classic example is the unrolling of loops to allow calculations from different iterations to be executed together.

These optimisations are well known and can be applied to a number of codes across platforms. Any tuning of a program will have to involve a combination of optimisations. For example, unrolling of loops, use of temporary scalars and so on. These slight changes to the code can result in the compiler interpreting the code differently. Some optimisations will be general, while others will work best on certain platforms. Clearly, there is a trade-off between run-time gained in optimising a code and time spent performing the optimisations. However, knowledge of the concepts of compiler optimisations can help in producing fast code and if the majority of the run-time is spent in a few lines of code, optimisation can be quick and effective.

2.5 Results of optimisations

In this section we present the results obtained by the optimisations discussed above. All

optimisations were made on the non-self integration routine ipecnsq9. Relative timings for an entire run using the modified algorithm are shown in Table 1.

TABLE 1
Relative CPU time for two different NASA Almond meshes with different code optimisations.

Code Options (see key for code descriptions)	1,202 Nodes	5,282 Nodes
1. Original code	2.536	3.661
2. Unit stride in inner loop array	1.585	2.070
3. Optimum compilation level	1.526	2.034
4. Loop invariant calculations removed	1.136	1.130
5. Inner loop unrolled	1.000	1.000

1. Original ipecnsq9 compiled with the default optimisation level (-O4).
2. As 1, with the subscript ordering of the array which holds the integration weights modified for unit stride in the inner loop of the integration routine.
3. As 2, with a compilation level which should provide fast run-time (-O5 -fast)
4. As 3, with loop invariant calculations of β and α terms manually removed from the inner loop.
5. As 4, with inner loop (3 x 3 matrix scalar multiply) unrolled manually

This table shows clearly the benefits of progressive code optimisations; approaching a factor of four on the larger problem. In all cases only a few lines of the routine where altered. The most beneficial of all the optimisations was modification of the array in the inner loop to use unit stride lengths. The benefits of the optimisation increase with problem size, as an increasing fraction of total cost is incurred in the subject routine.

3. Pre- and post processing

Mesh generation is performed using commercially available CAD packages, primarily MacNeal Schwendler's Patran, and SDRC's Ideas. In both cases, the prime use of the CAD suite is for mechanical design, but the capabilities they offer are well suited to electromagnetic modelling. The families of elements we employ, such as 8 and 9 noded quadratic quadrilaterals, and 6 noded quadratic triangles, are supported as a matter of course. Facilities such as automatic mapped and paved mesh generation, mesh seeding, and local mesh refinement are of considerable use. Display of results is done in much the same way, with (say) the time dependent vector surface field fed back into the CAD package.

However, much as some optimisation issues only arise on large problems, various additional difficulties arise in the pre- and post-processing for large problems.

Some of the surface meshes we have generated approach 100,000 nodes in size. Since such CAD packages are primarily designed to generate three dimensional meshes for finite element stress analysis, such mesh sizes are well within their capability. They nonetheless require quite powerful workstations to manipulate and display them.

With quadratic modelling in both time and space, a rational (normalised) timestep would be equal to the largest nodal separation, and this is the criterion we would generally employ on modest sizes of problem. However, it became clear that on large bodies, it was common to generate (and indeed difficult not to generate) meshes with a very small number of elements rather larger than say 1/10 of a wavelength, or whatever nominal target the analyst had intended. The automated timestep selection was modified to examine the profile of the nodal separations, and to select a timestep corresponding to typically the 95th percentile nodal separation. Whilst a handful of percent of elements with perhaps double the intended

nodal separations will not degrade the entire calculation to any significant extent, a timestep based on this would do so.

Sheer size of the results is also a problem. Using the conventional IETD algorithm for a 25 wavelength almond, for the ~3 transits necessary, generates an (ASCII) surface field history comprising some 1.3 Gb. Manipulation of files of this size is cumbersome, to say the least, and getting a CAD package to read, manipulate and display it is of itself a major computational task. One major practical by-product of the modified algorithm is that the results files, containing as they do the history only for the relatively brief active periods, are much smaller. The same 25 wavelength almond, for example, analysed thus generates a much more manageable result file of only 74 Mb.

Similar difficulties arise in calculation of the rcs. To do this efficiently requires that the surface result be stored in core. For the modified algorithm this is straightforward. For the conventional approach in the 25 wavelength case rcs calculation using the 1.3 Gb (ASCII) file would require about 1 Gb of memory.

4. Example results

4.1 NASA almonds

We will show some example results and timings for analysis of multi-wavelength NASA almonds.

Meshes comprising from 2,962 to 41,266 nodes were employed, each suitable for analysing (down to) a particular pulse width, and the extraction via fourier transform of results up to some particular frequency. Head or rear-on illumination was employed, but symmetry was not exploited.

In figures 3(a) and 3(b) are shown VV rcs calculations over a wide frequency range, with the almond varying from 1 to ~25 wavelengths long. The frequency range over which each mesh was employed is indicated on the figures. Also indicated on the figure are results computed elsewhere[7,13,14], and measured results[7], extracted by manual measurement from the published graphs. (This latter is practically a difficult process, and there is additional uncertainty introduced by this.) As is seen, the modified algorithm generally computes results in good agreement with the experiments and other computations, with reasonable consistency between meshes.

Figures 4 (a), (b) and (c) show harmonic surface fields, extracted via fourier transform, at different frequencies. As the frequency increases, the fields on the downstream, shadowed portion of the body are falling noticeably, as the regime where optical methods become applicable is approached. *[These are colour figures; please visit www.me.ic.ac.uk/mechanics/CWP/CWP_publications.html or www.emclab.umr.edu/aces/acesjrnl.htmlto view]*

With the CAD packages employed it is straightforward to construct and mesh a 'cutting plane', for the display of near field results. This is done in figure 5, for a fourier transform at a frequency corresponding to the almond being 25 wavelengths long. Near field evaluation was itself a sizeable computation, with 60,000 locations defined in the cutting plane, from each of which integration over the ~41,000 node body had to be performed, taking some 24 hours on a Dec alpha workstation. The near field results show clearly the rearwards shedding of the field, accounting for the low backscattered rcs.

[This is a colour figure; please visit www.me.ic.ac.uk/mechanics/CWP/CWP_publications.html or www.emclab.umr.edu/aces/acesjrnl.htmlto view]

As an example of timings, the 25 wavelength case took the equivalent of 2 1/4 hours on a 512 processor Cray T3D. Table 2 shows normalised timings for the various almond cases. Times are characterised by numbers of integrations (as seen earlier, by far the dominant cost), to eliminate inter-machine differences, and the table gives times relative the that for the 2962 case. The

size in wavelengths associated with each mesh is the largest which can be extracted with consistent criteria regarding discretisation and timestep.

The 'scaling' shown is the power of frequency relating the time for the size in question to the next size smaller. Normal IETD cost scaling is with frequency to the fifth power. As is seen, in all cases the scaling is below 4, and it is falling as larger cases are considered. A knowledge of how many integrations were performed allows the cost relative to the conventional case to be accurately assessed; taking the 25 wavelength case as an example, the factor by which costs are lower is about 50.

TABLE 2. Relative timing for various NASA Almonds (meshes / lengths in wavelengths).

Nodes	w/l	Tip on		Tail on	
		Time	Scaling	Time	Scaling
2,962	6.9	1.0		1.0	
5,282	9.2	3.1	3.85	2.9	3.73
10,866	13.2	11.9	3.77	11.8	3.85
21,522	18.5	38.7	3.46	42.7	3.77
41,266	25.6	111.1	3.24	124.9	3.30

4.2 Multiple bodies

One advantage of integral methods is the absence of any need to discretise the free space between different scatterers. A simple example is a random assembly of 10 small spheres, each 2 units in diameter, occupying a circumscribing box of side ~100 units. These were illuminated with a pulse making the 'box' and individual spheres respectively about 25 and 1/2 wavelengths in size in terms of the extractable frequency.

Figure 6 shows conventional and modified IETD results for the surface field at an arbitrarily selected node on one sphere. Obviously there is no analytical result to compare this with, but the modified and conventional approaches are both seen to predict very similar fields. As the results indicate, any one sphere is essentially quiescent much of the time, awaiting the arrival of some reflected wave from elsewhere. The major saving of the modified approach in this case, which is by a factor of about 32 here, arises because no nugatory calculations are performed during this wait.

Figure 7 shows a larger version of a similar test problem, with now 100 spheres, and the box now about 250 wavelengths in side, with a total of ~10,000 nodes. The modified algorithm needed ~70 Mb of memory, and took ~48 hours on a workstation. This size of problem could not be run by us with the conventional IETD approach. The field history alone would have required about 1 Gb of memory, and the run would have taken some 720 times longer. The surface field at the three locations indicated in figure 7 is shown in figure 8.

5. Conclusions

Efficient algorithms, coupled with use of large parallel machines, and appropriate optimisation of code to suit large problems and large machines, have been shown together to make significant reductions in the computational cost of large scattering problems. This has been demonstrated by analysis of much larger bodies than has been reported previously using time domain integral methods.

References

1. Miller, E.K. A selective survey of computational electromagnetics. *IEEE Transactions on Antennas and Propagation* 36:pp1281-1305, (1988).

2. Gomez Martin, R., Salinas, A. and Rubio Bretones, A. Time-Domain Integral Equation

Methods For Transient Analysis. *IEEE Antennas and Propagation Magazine* 34(3):pp15-22, (1992).

3. Poggio, A.J. and Miller, E.K. Integral Equation Methods of Three-Dimensional Scattering Problems. In: *Computer Techniques for Electromagnetics*, edited by Mittra, R. Oxford: Pergamon Press, 1973, p. 159-265.

4. Bluck, M.J. and Walker, S.P. Time Domain BIE Analysis of Large Three Dimensional Electromagnetic Scattering Problems. *IEEE Transactions on Antennas and Propagation* 45:pp894-901, (1997).

5. Walker, S.P. Scattering analysis via time domain integral equations; methods to reduce the scaling of costs with frequency. *IEEE Antennas and Propagation Magazine* 39:pp13-20, (1997).

6. Dodson, S.J., Walker, S.P. and Bluck, M.J. Costs and cost scalings in time domain integral equation analysis of electromagnetic scattering. *IEEE Antennas and Propagation Magazine* (submitted):(1997).

7. Woo, A.C., Wang, H.T.G. and Schuh, M.J. Benchmark radar targets for the validation of computational electromagnetics programs. *IEEE Antennas and Propagation Magazine* 35:pp84-89, (1993).

8. Davidson, D.B. Parallel matrix solvers for moment method codes for MIMD computers. *Applied Computational Electromagnetics Society Journal* 8:pp144-175, (1993).

9. Cwik, T. Parallel decomposition methods for the solution of electromagnetic scattering problems. *Electromagnetics* 12:pp343-357, (1992).

10. Walker, S.P. and Leung, C.Y. Parallel computation of integral equation methods for three dimensional transient wave propagation. *Communications in Numerical Methods in Engineering* 11:pp515-524, (1995).

11. Walker, S.P. and Leung, C.Y. Parallel computation of time domain integral equation analyses of electromagnetic scattering and rcs. *IEEE Transactions on Antennas and Propagation* 45:pp614-619, (1997).

12. Dodson, S.J., Walker, S.P. and Bluck, M.J. Parallelisation issues for high speed time domain integral equation analysis. *Parallel Computing* submitted:(1997).

13. Volakis, J.L. Carlos-3D; A general purpose three dimensional method of moments scattering code. *IEEE Antennas and Propagation Magazine* 35:(1993).

14. Miller, E.M, Andersh, D.J and Terzouli, A.J.Jr Target facetisation level and the effect on Xpatch predictions. *9th Annual Rev Appl Comp Electromagnetics* pp610-617, (1993).

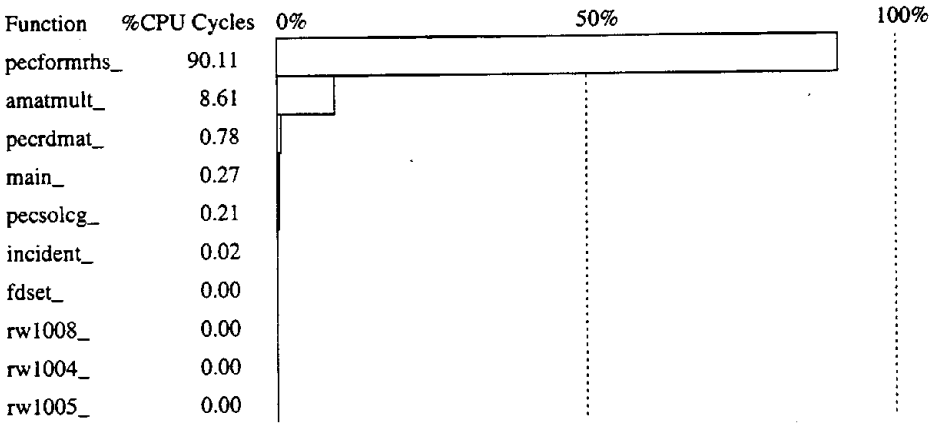


Figure 1

Profile for the conventional algorithm, 1202 node problem, matrix stored in-core. The bar chart shows the percentage of the CPU cycles spent in different subroutines. The dominant routine 'pecformrhs_' multiplies the matrix coefficients by historical field values to form the right hand side vector. The routine 'amatmult_' performs the iterative solution of the sparse left hand side [A] matrix.

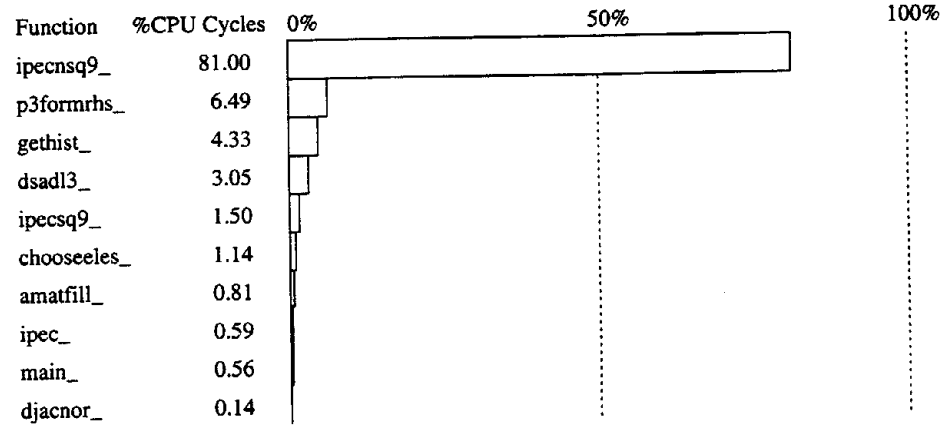


Figure 2

Profile for the modified approach, 1202 node problem, generating matrix coefficients as and when needed. The first five routines (ipecnsq9_ to ipecsq9_) form the matrix coefficients and multiply them by historical field values. The dominant routine, ipecnsq9_, forms the matrix coefficients for a non-self node / element pair. The optimisations shown in table 1 were performed on only this routine.

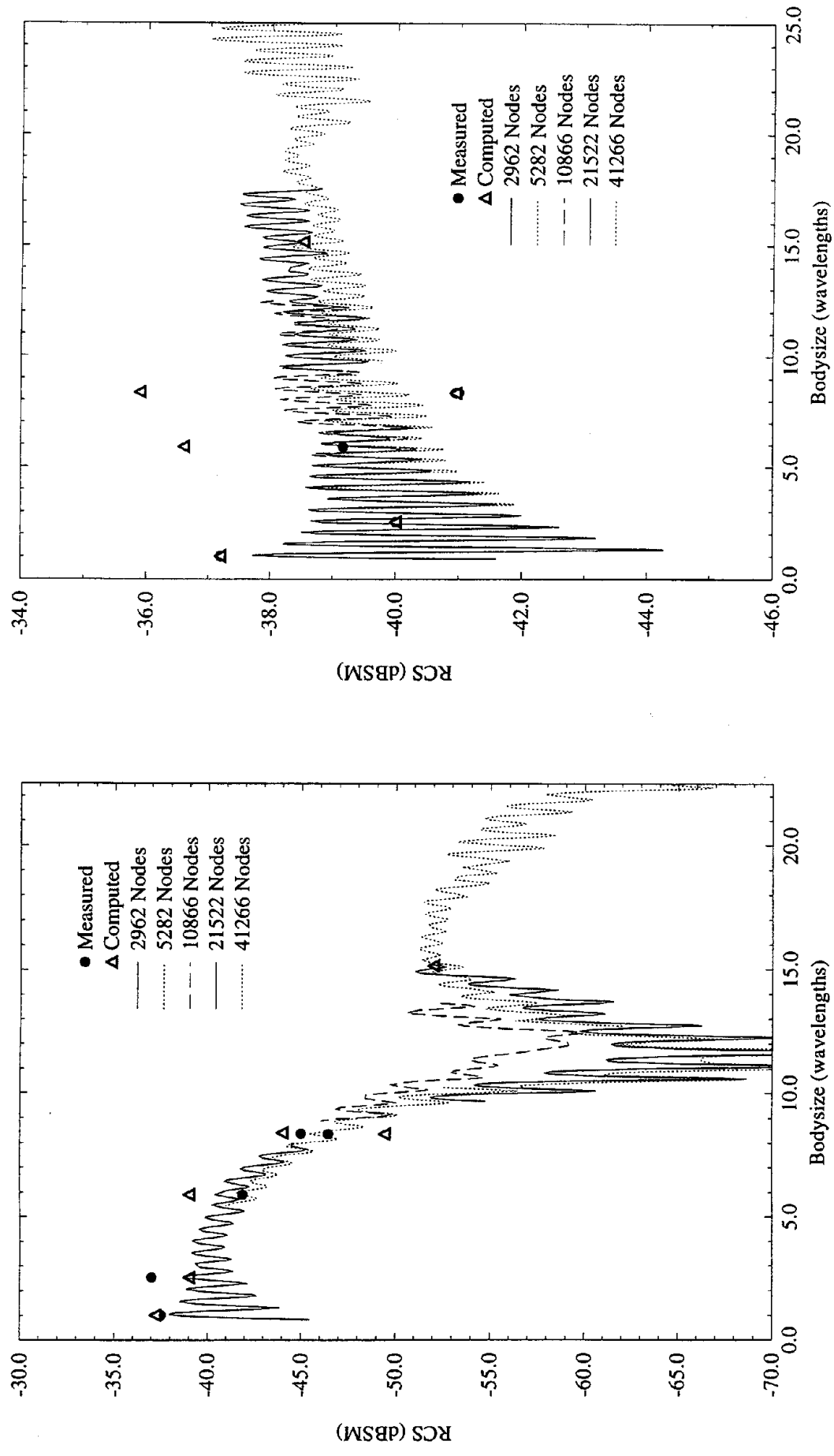
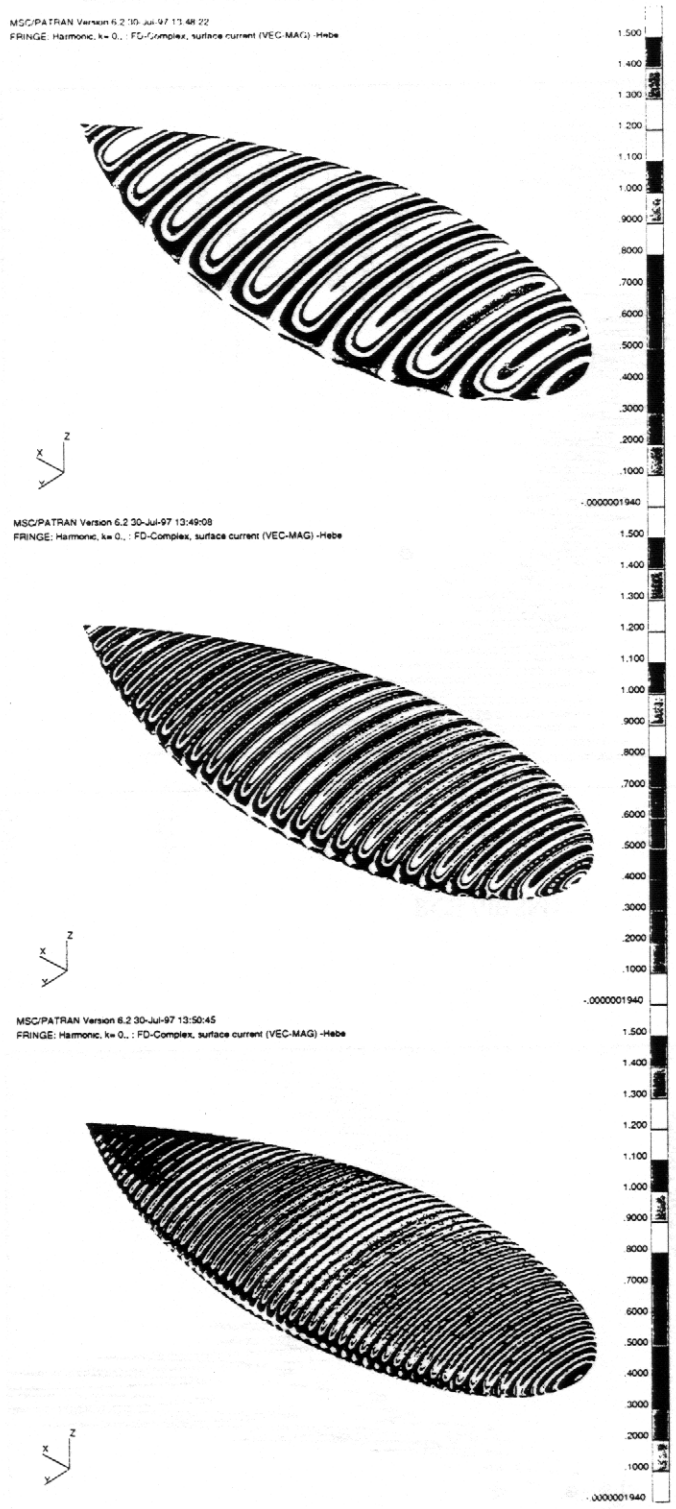


Figure 3

Backscatter versus bodysize for the 9.936" NASA almond. (a) represents propagation in the -x (tip-on) direction, and (b) propagation in the +x (end-on) direction. E polarisation is vertical (+z) in both cases. Frequency ranges for some meshes are shortened to improve the clarity of the graph.



Figures 4(a), (b) and (c)

Surface $|\mathbf{H}|$ extracted at (a) 5.67GHz, (b) 14.9GHz and (c) 30.4GHz, on the NASA almond. Propagation is $\pm x$ (broad end on), **E** vertical. Electrical lengths 4.77, 12.5 and 25.7 wavelengths respectively.

[These are colour figures; please visit www.me.ic.ac.uk/mechanics/CWP/CWP_publications.html or www.emclab.umr.edu/aces/acesjrnl.htmlto view]

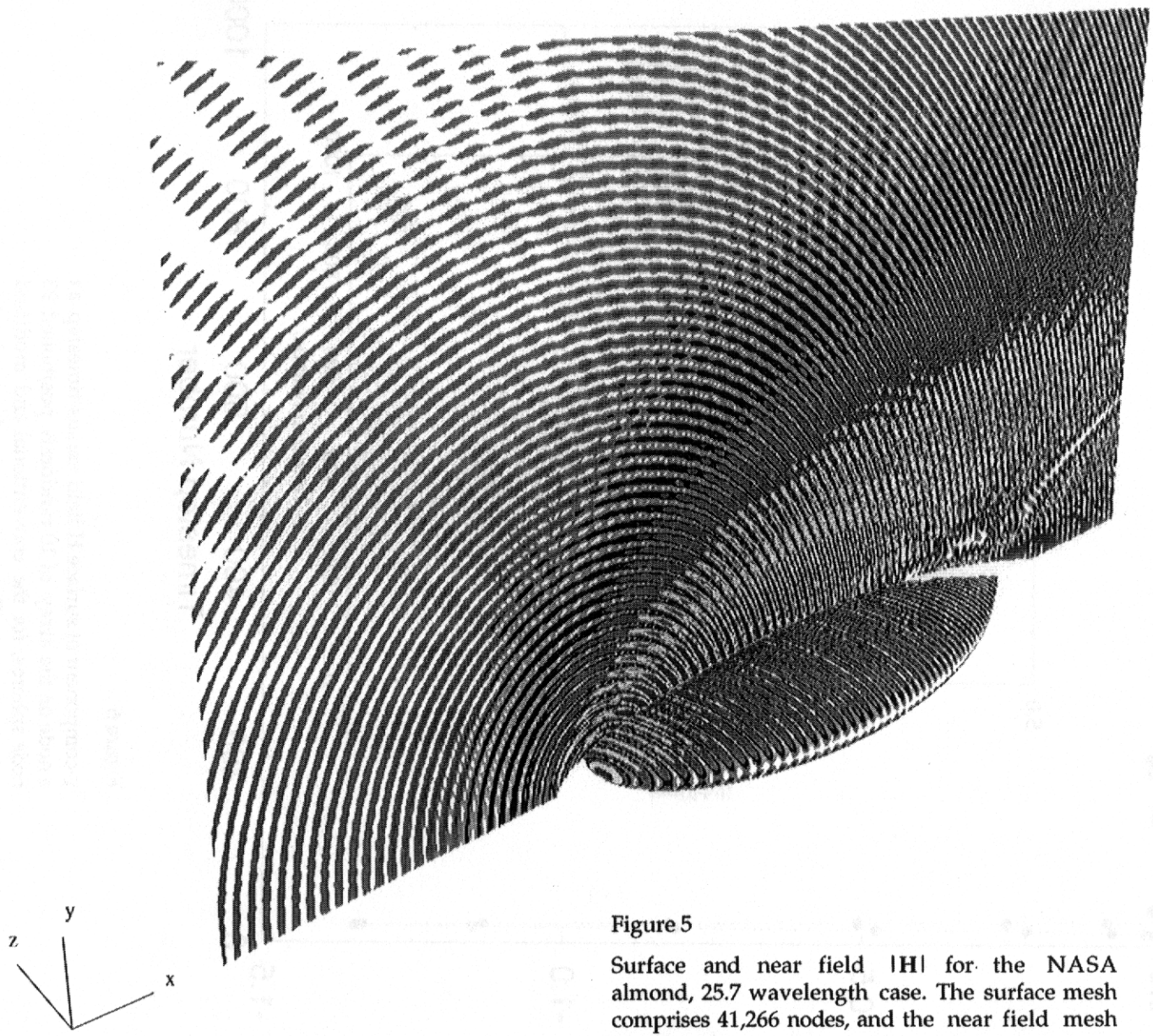


Figure 5

Surface and near field |**H**| for the NASA almond, 25.7 wavelength case. The surface mesh comprises 41,266 nodes, and the near field mesh 82,033 nodes.

[This is a colour figure; please visit www.me.ic.ac.uk/mechanics/CWP/CWP_publications.html or www.emclab.umr.edu/aces/acesjrnl.htmlto view]

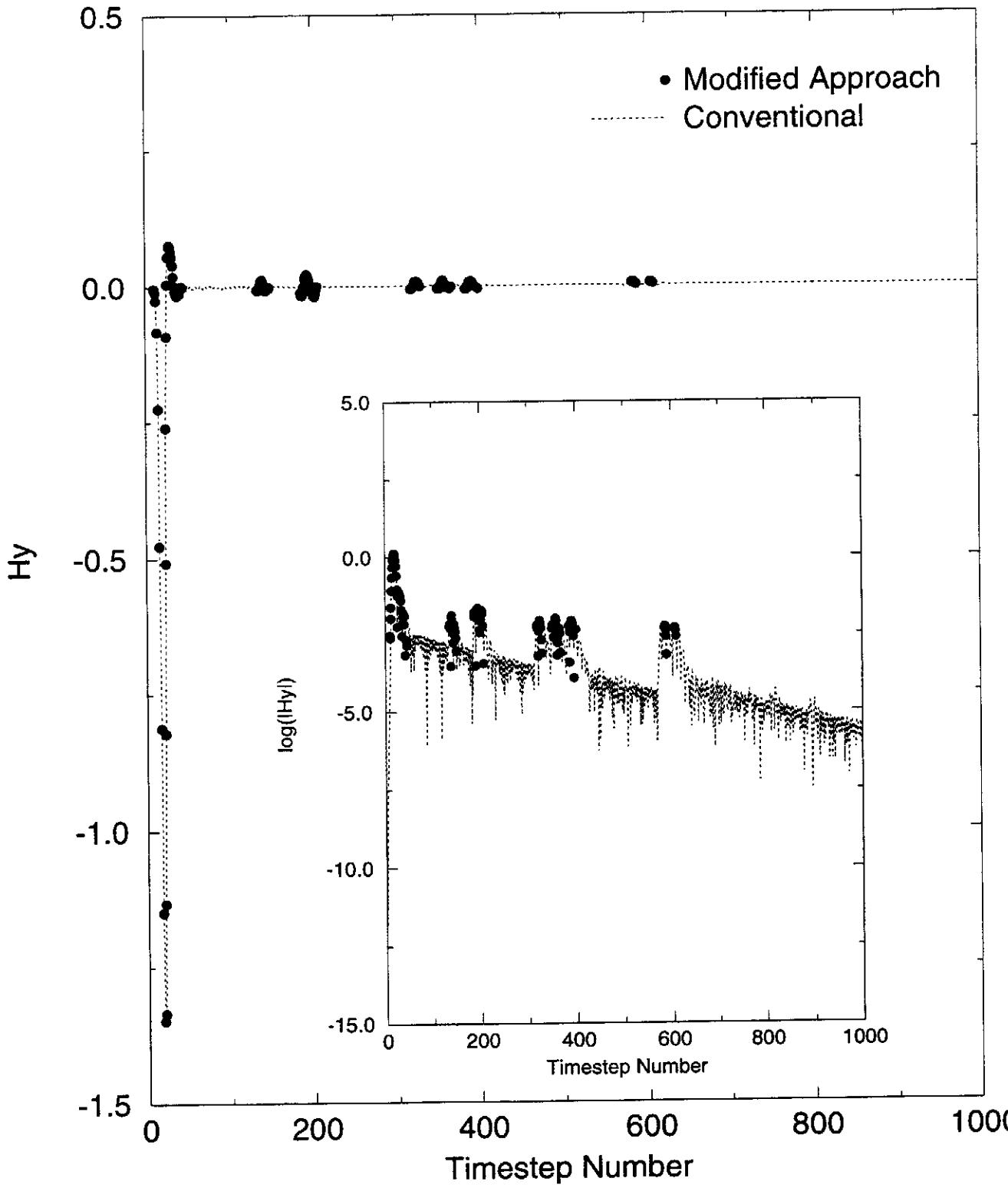


Figure 6

y-component of surface **H** field versus timestep at a node on the array of 10 randomly positioned 98 node spheres, for the conventional and modified approaches. The inset shows this logarithmically.

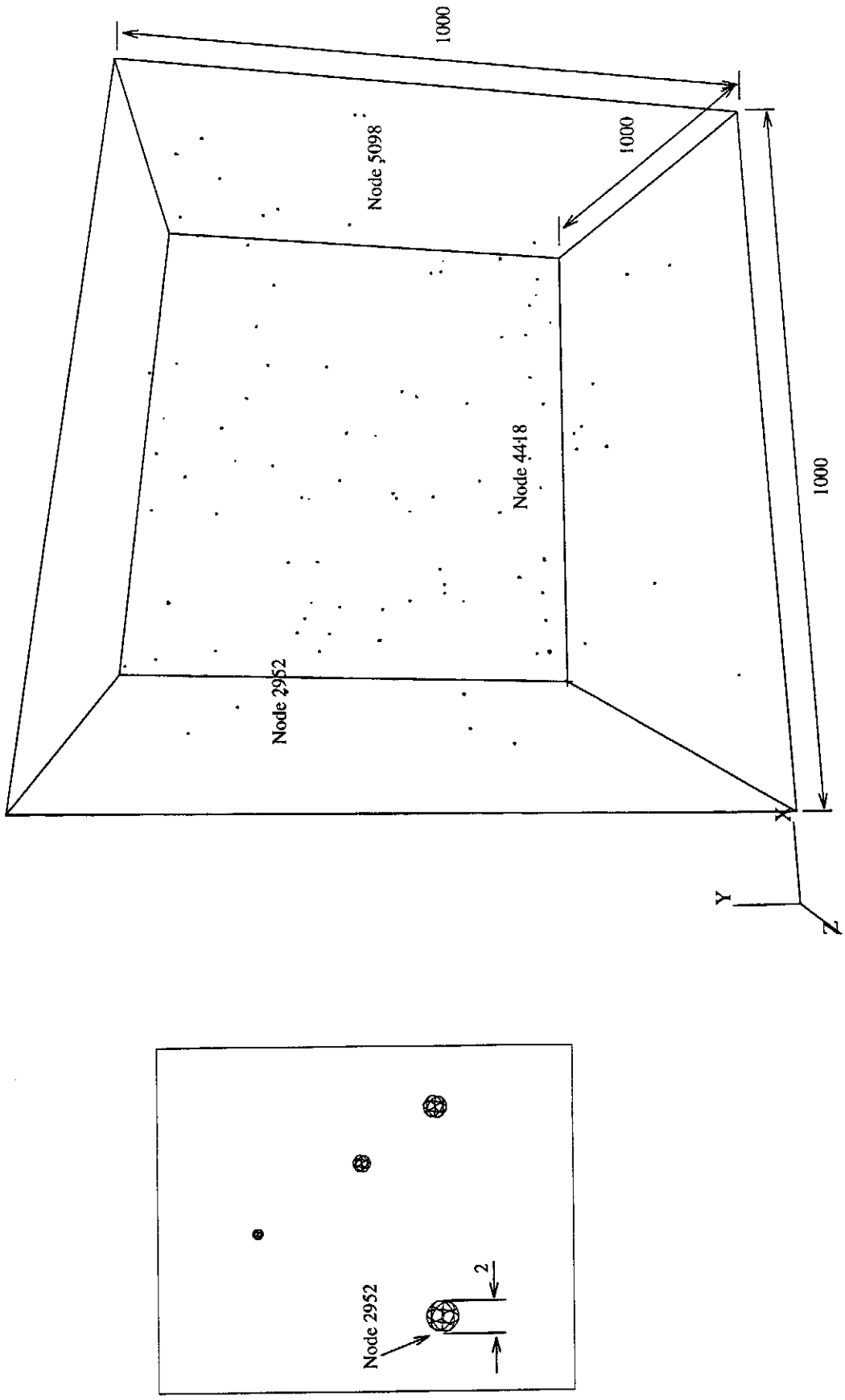


Figure 7

The array of 101 randomly positioned 98 node spheres. The box drawn around the spheres is for illustrative purposes only. The inset shows a close-up of a part of the array.

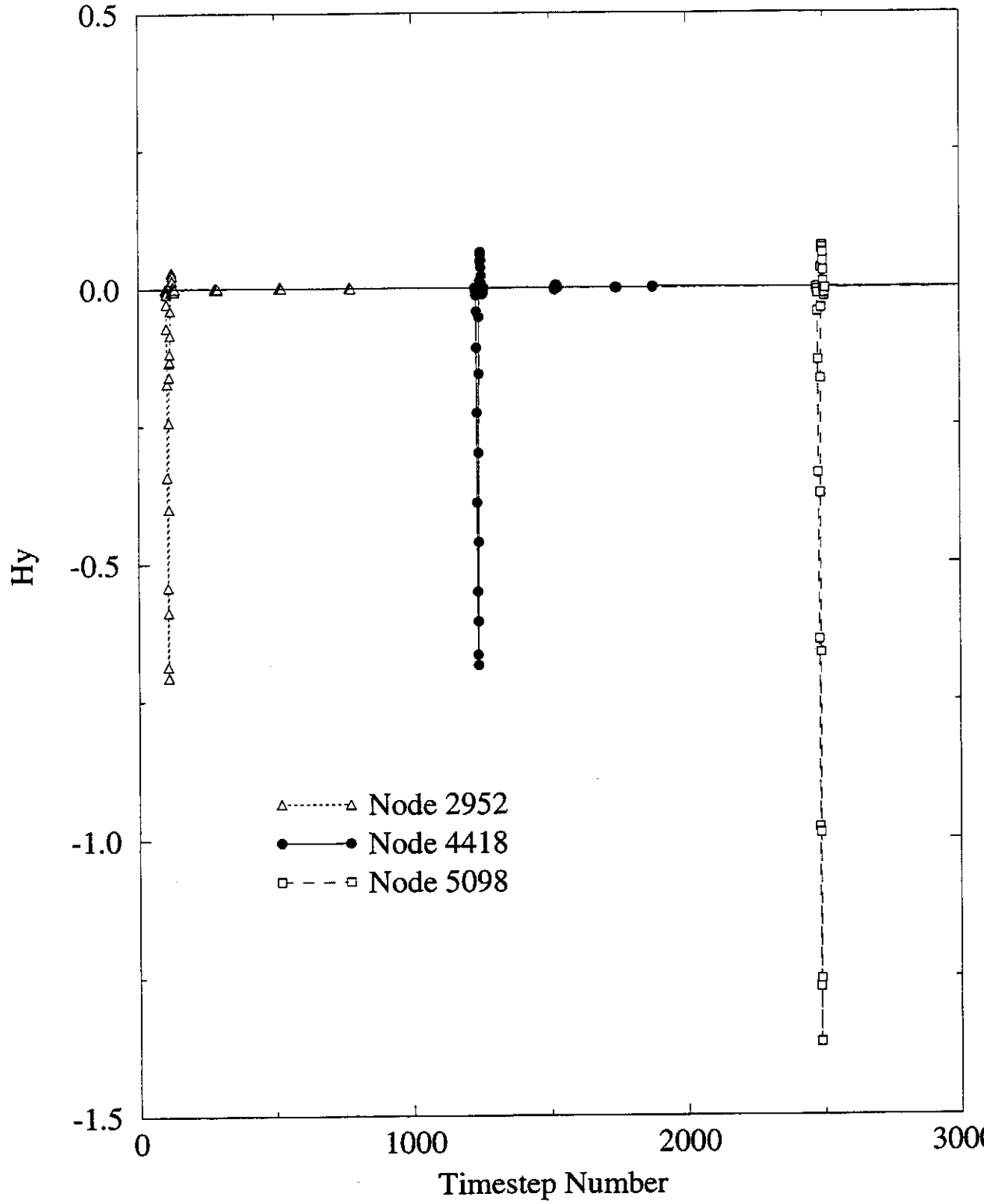


Figure 8

y-component of surface **H** field versus timestep at 3 nodes on the array of 101 randomly positioned spheres shown in figure 7. Propagation is in +x, and **E** polarisation is +z. The nodes' locations are indicated in figure 7.