PARALLEL COMPUTATION OF LARGE-SCALE ELECTROMAGNETIC FIELD DISTRIBUTIONS

Peter S Excell, Adam D Tinniswood* and Kathleen Haigh-Hutchinson
Department of Electronic and Electrical Engineering, University of Bradford, Bradford, UK
*Now with Department of Electrical Engineering, University of Utah, Salt Lake City, USA

ABSTRACT. Some experience of the use of high-frequency electromagnetics software on parallel computers is reported. Types of such computers are reviewed and approaches to the parallelisation of existing serial software are discussed. A practical large-scale problem is presented, involving the modelling in very fine detail of electromagnetic penetration into biological systems. This was tested on state-of-the-art parallel computers and important practical and strategic aspects of the experience derived are discussed. It was found that considerable programmer effort was required to optimise the software to use the computer architecture effectively, but that efficient acceleration of the run-times of typical computational tasks could be achieved, provided that the tasks were large and were partitioned optimally.

1 INTRODUCTION

The computation of electromagnetic field distributions almost invariably involves the repetition of a similar set of computational actions for a large number of data points. For systems of practical interest, this implies a very large computational task. Although it is possible to construct certain problems in a way that enables them to be run on computers of modest power, structures which are large in comparison with the wavelength frequently require the use of the most powerful computers available, usually characterised by the name 'supercomputers'.

In recent years, some of the problems that stretch such supercomputers to the limits of their abilities have been designated as 'grand challenges'. Lists of such grand challenges tend to concentrate on applications in pure science and environmental modelling, such as high-energy physics, quantum chemistry and atmospheric circulation modelling. Computational fluid dynamics (CFD) is also frequently included in the lists, but other applications of industrial interest tend to be given less emphasis. Although it is relevant both to science and to industrial applications, it is frustrating to find that electromagnetics does not appear to have been explicitly mentioned in any well-publicised list of 'grand challenges' and it is very desirable that this situation be remedied.

The main exercise used to test both hardware and software in the present work was the modelling of electromagnetic wave penetration into the human head, using a head image classified from a magnetic-resonance image (MRI), standardised to 1 mm resolution, which gives about 10^7 FDTD nodes [1] (Fig. 1). This is a particularly challenging test due to the discontinuity between air and the very large permittivity of living tissue. The source transmitter was a representation of a generic mobile telephone handset, which was essentially a plastic coated box, in the centre of the top of which was a quarter-wave wire antenna driven at its base with a continuous wave signal. The two frequencies of interest were 900 MHz and 1800 MHz, which are used as the carrier frequencies for most mobile telephone technology in Europe. Although the model of the head has been classified to a resolution of 1 mm it was also re-sampled to resolutions of 2 mm and 2.5 mm to allow the simulation to be run in serial form on a UNIX workstation (Sun Sparc 20 with 128 MBytes of RAM), and to facilitate comparisons of different parallel processing systems. The majority of this work was undertaken as a contribution to a broadly-based European Union project (EUROPORT-2) which aimed to demonstrate good scaleability performance and ease of inter-platform portability for several examples of industry-standard software ported to a range of contemporary parallel computers.

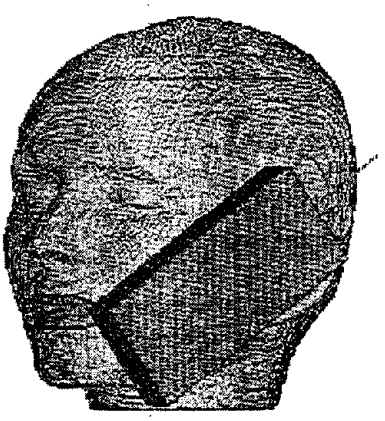

Fig.1 The original (1 mm resolution) head dataset, used as a test piece. Shown here with the generic mobile telephone: the version used in Section 4.2 included a simulated hand.

2 PARALLEL PROCESSING SYSTEMS

The idea of sharing a computational task between many processors in order to achieve higher speed or larger task capacity has been appreciated from the early days of computing. Unfortunately, it often proved impossible to achieve an adequate performance with such systems, except in a few specialised cases, usually using application-specific hardware. The manufacturers of the most powerful hardware thus concentrated on efforts to increase the speed of traditional serial processors. An important hybrid technique that evolved was 'pipelining', in which identical operations were undertaken on several separate data elements in rapid succession.

The traditional serial processor is categorised as 'single-instruction single data' or SISD and the pipeline processor as 'single-instruction multiple data' or SIMD. Computers employing pipeline processors are usually known as 'vector processors': they have some of the characteristics of parallel processing since the pipelined serial operations are almost indistinguishable from parallel operations. Clearly, independent parallel processors can also function in 'multiple-instruction multiple data' or MIMD mode, but it is rare for different processors to undertake completely heterogeneous tasks as it is extremely difficult to distribute such tasks efficiently between processors. With the increasing power of individual processors it has often become irrelevant to consider tasks at the 'instruction' level since a marginal amount of loss of strict synchronism may mean that processors are not executing similar instructions precisely simultaneously. Taking a broader view, it is more relevant to compare the program segments that each processor is running and hence the abbreviations SPMD and MPMD are now also widely used, where 'P' stands for 'Program'.

By the mid-1980's, it appeared that the vector-processing architecture was approaching the limit of its ability, and hence vector processor manufacturers started to incorporate parallel processing into their computers, but only to a limited degree (typically two to sixteen processors, exemplified by the Cray X-MP and Y-MP computers). Smaller manufacturers had continued to develop parallel computers, many exploiting the Transputer, a microprocessor optimised for parallel applications, but their success was limited, both technically and financially. At least two manufacturers (Thinking Machines and Active Memory Technology) took an extreme path of developing systems with very large numbers (>1000) of low-power processors, but this approach did not achieve widespread success [2]. The 1990's have seen a revival of interest in computers with a modestly large number of processors (typically between 16 and 512 at present) and this type has at last become accepted as the mainstream supercomputer architecture. A re-evaluation of designs with several thousand processors also now appears to be developing, based on low-cost personal computer architecture.

Within the class of parallel computers, a new subdivision emerged, between those having 'shared memory', a single large data store used by all of the processors, and those having 'distributed memory', i.e. standard memory modules attached to each processor. The shared memory approach appeared more natural to program, but it suffered from problems of low speed of access by the processors and complexity in the hardware arrangements. The distributed memory approach enabled a still greater use of standard hardware, but required great care in programming since time could be wasted searching for an item of data in another processor's memory. Alternatively, two versions of the same data could exist in the memory of different processors and there could then be uncertainty over which was the valid version. In practice, the hardware arguments favouring distributed memory have become very strong, and the software techniques required to exploit it are slowly developing towards maturity. Nonetheless, powerful shared-memory machines continue to be made, and their performance is again becoming competitive with distributed-memory machines.

In recent years, another form of distributed-memory parallel processing has appeared as a result of the installation of large networks of UNIX workstations in many companies and universities. These workstations spend much of their time idle, particularly at night, and hence techniques have been devised to take control of unused power in a network and distribute a program across a large number of workstations, the results being fed back to a designated master processor. This form of parallel processing suffers from relatively poor communications, and hence it is important to partition the task in such a way that the communications overhead is minimised. On the other hand, the processing power is available at minimal cost, and hence it has been an attractive option for many organisations [3].

3 PORTING OF LEGACY SOFTWARE TO PARALLEL PLATFORMS

In earlier years, when attempts were made to transfer (or 'port') electromagnetics software from traditional serial machines to multi-processor machines, it was frequently found that some speed-up was achieved when using a few processors in parallel, but when larger numbers were tried the software would not run with the expected speed increase, and it was not unknown for it to run even more slowly than in its serial form on a single processor. The number of processors at which this problem arose could be as small as four [4], and 'Amdahl's Law' was frequently invoked to explain the effect

[5, 6]. This law basically states that there is a limit to the amount of speed-up that is achievable with parallel processing, because real-world software always contains some serial sections which rapidly come to dominate the run time when the parallelisable sections have been accelerated. In practice, the situation is even worse than Amdahl's Law suggests, because parallel sections of the program require extra time for setting up and distribution of the data between the designated processors. When running, the parallel sections can be severely handicapped by communications delays in passing intermediate data between processors, especially if the algorithm is not optimised to minimise communications requirements.

4 PARALLELISATION OF SOFTWARE

Most single-box parallel computers have been sold with accompanying software which attempts to translate existing serial programs in a standard language (usually Fortran) into a form which exploits the parallel architecture efficiently. Similar software has also been provided by vector processor manufacturers, but for both architectures it has been found that the automatic vectorisation or parallelisation routines frequently make poor decisions on the re-working of a serial code, or fail to notice a section of code which is capable of modification. 'Manual' parallelisation or vectorisation has thus usually been used, a highly-skilled programmer making appropriate modifications to the program based on a deep understanding of the architecture of the computer and the operation of the program.

It seems intuitively reasonable to suggest that this situation should not persist in the long term and that fully reliable parallelisation software should eventually become available. This would be given information on the architecture of the machine on which it is running, would then deduce the structure of the serial program that it had been given and thus devise an optimum strategy for partitioning this onto the parallel system. This problem is the subject of several international initiatives which are reviewed below.

Fortran 90 incorporated intrinsic matrix manipulation functions, constituting some of its most significant extensions over Fortran 77. Unfortunately, the efficient execution of such matrix operations is very dependent on the structure and content of the matrices involved and the detailed architecture of the computer in use. To overcome these problems, extensions to Fortran 90 have already been adopted by users of parallel processors and this extended language became known as High Performance Fortran (HPF), which is now strongly influencing the emerging Fortran 95 standard.

In attempts to address the particularly difficult but important problem of efficient partitioning of an algorithm across a network of UNIX workstations, some new approaches were devised. The most important of these were PVM (Parallel Virtual Machine) and MPI (Message-Passing Interface). Although these methods were generally devised for use with asynchronous networks of workstations, they have been adopted equally widely with single-box parallel supercomputers. PVM was initially the most popular method: this loads 'spawned' replicated versions of the main program into each of the designated slave processors, which then proceed to function autonomously [7]. The efficient partitioning of the problem between processors is still under the control of the programmer, although separate graphical performance-indicating tools are available [8] to give an indication of the way in which the program is functioning, the key objectives being 'load balancing' between the processors, and minimisation of the communication activity. MPI was defined later than PVM, but it has now achieved greater popularity. It functions in a similar way to PVM but does not use the organic 'spawning' approach.

Early experience was gained in efforts to vectorise and parallelise standard Method-of-Moments (MoM) codes, with some success [9]. It was found particularly difficult to vectorise and parallelise the critical matrix-solution phase of MoM because of the inherently high degree of interaction between the elements of the very dense interaction matrix that is needed in this method. This means that there is a high communication overhead if the matrix elements are distributed across a parallel processing system. There is also 'data dependency' with both parallel and vector processors, forcing some parts of the process to be handled in a sequential way in order to ensure that the correct version of the input parameters is being used. Filling of the matrix and computation of the final field distributions do not have the same difficulties and these operations can be parallelised relatively easily.

In considering this experience, it became apparent that a method based on a differential-equation formulation, such as the Finite-Difference Time-Domain (FDTD) method, would have an inherent advantage on a distributed (parallel) processing system. This is because the interaction matrix is effectively sparse and the updating of an individual node only requires data from its immediate neighbours, whereas nodes in an integral-equation formulation (e.g. MoM) require data from all of the other nodes in the system. This effect is closely similar to the dichotomy between the competing physical approaches of 'action at a distance' (c.f. integral-equation formulations) and 'fields' (c.f. differential-equation formulations).

A very significant overview of techniques for parallelisation and vectorisation of FDTD algorithms was presented by Gedney and Barnard [10], in particular discussing methods for maximisation of the efficiency of DO-loops, which are the main parallelisable component of programs written in a traditional language.

4.1 Experience with the FDTD Method on a Parallel Computer with Virtual Shared Memory

The KSR-1 was a novel design of parallel processor, produced by the Kendall Square Research Company in the USA. Although the company has now withdrawn from the high-performance computing market, the machine used an interesting architecture which may influence future designs and hence experience gained with it is still relevant. The particular machine used had 64 processors, although the maximum number that could be allocated to a single job was 48; the standard word length was 64 bits. The machine had a unique 'virtual shared memory' architecture, in which sets of processors were connected in hierarchical rings, each having fast communications. The objective was to give an approximation to the behaviour of a shared-memory parallel processing system, whilst using the cheaper, standard, hardware of a distributed memory system. A cache was interposed between each local processor memory and the communications ring to accelerate the accessing of data by other processors.

The Finite-Difference Time-Domain (FDTD) method for electromagnetic field computation was used to test the ability of this computer, the program used being an updated version of THREDE [11]. The test exercise was the modelling of electromagnetic wave penetration into the human head, as discussed in Section 1, using the head image re-sampled to 2.5 mm resolution, which gives about 10^6 FDTD nodes.

In converting the program for parallel operation, the computer manufacturer's automatic parallelisation tool was first tried, but this only gave significant benefit on the most trivial DO-loops (this experience was also found with similar tools on other computers). The virtual shared memory was also found to be a potential source of problems, much like virtual memory on earlier serial systems, in which inefficient page swapping could slow down program execution dramatically if the task was not correctly partitioned. Data was normally swapped between processors in 'subpages' and it was found that maximum efficiency was achieved in the FDTD algorithm when the field data arrays were each arranged to start on a subpage boundary. It was concluded that, for satisfactory performance, memory must be managed explicitly by the programmer and internal automatic procedures cannot be relied upon. Similarly, it was found that the automatic procedure frequently decided to use fewer than the maximum number of processors for many sections of parallelised code: explicit control of the number of processors thus also appears essential. Explicit 'affinity directives' were thus used to instruct the operating system to partition the arrays in a pre-determined pattern, and to use an identical pattern for all of the arrays: this enabled remote data to be located rapidly, rather than time being wasted in a search. It was necessary to use explicit correlation of the data partitions and the number of processors, otherwise the default algorithm in the operating system would tend to make non-optimum, uncorrelated, decisions on these parameters. To avoid re-compilation when the number of processors was changed, this number was configured as a system variable, to be set outside the program.

Extracting optimum performance from a machine of this type invariably involves a number of minor code modifications, based on experience and a detailed understanding of the operation of the hardware and software. In this case, a problem was found in that the existing code, optimised for a vector processor, used separate loops to iterate the FDTD time-marching equations in the x, y and z planes. When this was altered to a single loop to update all components at once for each cell, the amount of data loading and storing required was reduced by a factor of three (this is discussed more fully in [10]). Loading and storing is a slow process on this computer, whereas it is fast on a vector processor. This modification gave a speed improvement by a factor greater than ten. It was found that the computer was particularly fast with multiply operations but slow with division, load, store and trigonometric functions. To circumvent this problem, the number of divisions was reduced by converting commonly-used denominators into reciprocals which could then be used with the multiply function. Similarly, commonly-used trigonometric function results were stored to avoid fresh function calls.

4.1.1 Scaleability

'Scaleability' is an important objective of parallel computers and software. A hardware-software combination has perfect scaleability if the program runs N times faster when the number of processors in use is increased by a factor N. In practice this never happens, but a viable system can be expected to approach at least about 50% of this ideal performance. Popular measures of parallelism performance are 'speed-up' and 'parallelism efficiency', which are defined thus:

Speed-up $$S = \frac{T_1}{T_n} \qquad (1)$$

$$\text{Parallelism efficiency } \eta = \frac{T_1}{nT_n} \times 100\% = \frac{S}{n} \times 100\% \quad (2)$$

where T_1 is the execution time on one processor and T_n is the execution time on n processors.

The product of the number of processors and the observed run time can also be used to give an indication of the parallelism efficiency when T_1 is not known. This product is clearly equal to T_1 when the efficiency is 100% and will be greater if the efficiency falls.

As a test of the scaleability of the modified THREDE program, the 2.5 mm resolution dataset was run for 250 iterations (not enough for convergence, but adequate for this test), on 10, 20 and 30 KSR-1 processors. Results are shown in Table 1: near-linear scaleability was found from 10 to 20 processors, but a substantial loss of efficiency occurred with more, probably due to an excessive communications burden resulting from non-optimum partitioning of the data matrix. The scaleability and speed-up should strictly be calculated using the run-time with just one processor as the baseline. However, the task was too large to run on one processor, as is very often the case. The result is that scaleability has to be assessed from the results with a larger number of processors (ten in this case), under the assumption that near-perfect performance is achieved up to this number. An alternative performance measure that has been used in some circumstances is 'scaled speed-up' [12], in which the size of the computational task is scaled in proportion to the number of processors. This gives results that appear better, but it was impractical to scale the task studied here down to the level that could be run on one processor.

Table 1. Scaleability tests with KSR-1 computer (One-dimensional partition; 100×100×100 nodes; 250 iterations)

No. of Processors	Run time (mins)	Processors × Time (mins)	Speed-up (see note)	Efficiency (see note)
10	20	200	10	100%
20	10	200	20	100%
30	15	450	13.3	44%

Note: task too large for one processor, hence speed-up and efficiency calculated by assuming they are ideal with 10 processors.

4.1.2 Partitioning of the Problem Space

A diagrammatic representation of the way in which the problem space was partitioned on the KSR-1 is shown in Fig. 2. The problem space was divided into a number of slices, each to be computed on a separate processor (for simplicity, only five slices are shown). This is a one-dimensional partition, but two- and three-dimensional variants are also possible (see below). It is important to give each processor approximately the same workload, as the computational effort is almost identical in each node of the FDTD calculation. An unbalance in workload leads to wasted processing time on idle processors, thus reducing efficiency. Since the 2.5 mm dataset had array dimensions of about 100 in each direction, thirty processors would each only be handling three or four slices of the dataset, and a proportionately increased number of data transfers would be required to access the field data from adjacent processors. Even if the amount of serial code in a program were to be zero, it is thus evident that a point would always be reached where a problem of a given size would start to run more slowly as the number of processors was increased, since communications activity would start to dominate over computation. This is true of any parallel processing architecture, not just the rather unusual case of the KSR-1.

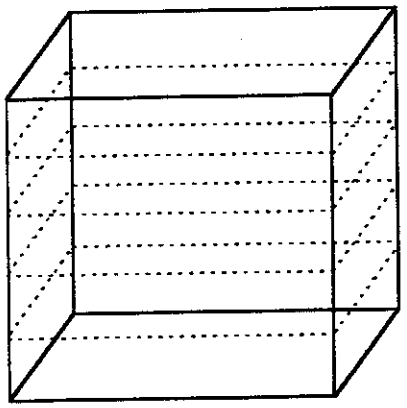

Figure 2: A one-dimensional partitioning of the FDTD problem space into sub-volumes, each to be handled by a separate processor.

A three-dimensional partition is illustrated in Fig. 3: here the problem space is shown partitioned for execution on a hypothetical eight-processor computer. Each of the eight processors is assigned an equal part of the problem in such a way that inter-processor communications are minimised.

Treating these representations of a three-dimensional data array as fictitious three-dimensional solids, it can be said that the 'surface area' of any sub-volume is proportional to the sum of the amount of inter-processor communication it will have to undertake, plus the amount of absorbing boundary condition (ABC) treatment it will need. The latter only applies to the outer surfaces of sub-volumes at the boundaries, but this type of surface will be in a majority for regularly partitioned cubic volumes having

fewer than 12 sub-volumes. The 'volume' of any sub-volume is proportional to the amount of internal calculation it will have to undertake, using data stored in its own memory. ABC treatments may be expected to be a little slower than standard FDTD calculations, but communications are almost invariably very significantly slower. Thus the optimum strategy should be to aim for partition sub-volumes that have a maximised ratio of volume to surface area.

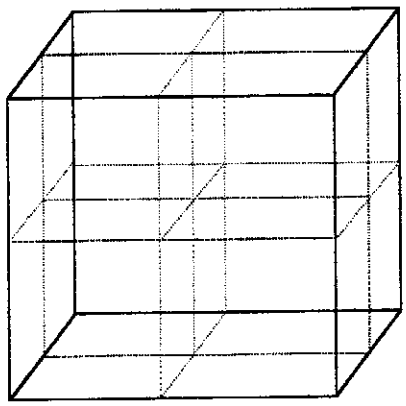

Figure 3: A three-dimensional partition of the FDTD problem space.

Using the one-dimensional partition, the total surface area of all of the sub-volumes, measured in data elements, is:

$$A_1 = 2n^2(p+2) \qquad (3)$$

where n is the number of data elements in each dimension of the problem space and p is the number of sub-volumes (i.e. processors). The factor 2 is included because each internal surface is seen twice, once from each side. The problem space is assumed to be a regular cube for simplicity: the same general conclusions will be valid for cuboidal volumes except when they are very long and thin.

The total *communicating* surface area of all of the sub-volumes (i.e. excluding external surfaces requiring ABC treatment) is:

$$A_{C1} = 2n^2(p-1) \qquad (4)$$

However, for the three-dimensional partition the equivalent total area is:

$$A_3 = 6n^2\sqrt[3]{p} \qquad (5)$$

And the communicating area is:

$$A_{C3} = 6n^2(\sqrt[3]{p} - 1) \qquad (6)$$

since there are $\sqrt[3]{p} - 1$ internal planes in each of three co-ordinate directions, each seen from two sides and each with area n^2.

Taking, for example, the perfect cubes p = 8, 27 and 1000, these equations give the results shown in Table 2.

Table 2: Typical total areas of sub-volume surfaces, from Eqns (3) to (6)

p	$\sqrt[3]{p}$	A_1	A_{C1}	A_3	A_{C3}
8	2	$20n^2$	$14n^2$	$12n^2$	$6n^2$
27	3	$58n^2$	$52n^2$	$18n^2$	$12n^2$
1000	10	$2004n^2$	$1998n^2$	$60n^2$	$54n^2$

Since the volumes of the sub-volumes are constant in each row of this table (volume = n^3/p), these results show that, whether or not ABC treatments are considered to have a significant effect on the size of the task, the ratio of 'surface area' to 'volume' for each processor will be much smaller for three-dimensional 'cubic' partitions than for one-dimensional 'slice' partitions.

This shows that the 'slice' partition used in the KSR-1 experiments was non-optimal, and it explains the poor efficiency obtained with 30 processors (it was not possible to repeat the experiment with cubic partition as support for the computer was terminated following the untimely withdrawal of the manufacturer from the parallel computer business). The KSR 'virtual shared memory' architecture is not being perpetuated at present, although it has some similarities with that used in another current manufacturer's products. Nonetheless, experience gained with such experimental architectures is always instructive in assessing the viability of future architecture proposals.

It should be noted that the above calculations only give an indication of the relative amount of time required for communication of data when comparing the same problem run on different topologies (an estimate of communication time relative to computation time is given in [13]). The basic fact remains, however, that a calculation that involves data communicated between processors will be slower than an equivalent one that does not. It therefore appears that the optimum strategy will always be to minimise the ratio of communication to internal processing tasks, per processor.

4.2 Experience with Parallelised FDTD using PVM.

In 1994-95 the European Union's ESPRIT III action initiated

a programme entitled EUROPORT, the object of which was to demonstrate portable and scaleable parallelisation of industry-standard programs, using automated tools as far as possible. 'Portable' means that the program will operate, without substantial modification, on a range of platforms, including parallel processors from various manufacturers, and workstation clusters. One of the projects was PEPSE (Parallel Electromagnetics Programming Support Environment) [14], which was concerned with parallelisation of a standard FDTD program and a linked graphical input-output program.

The FDTD program chosen for parallelisation was EMA3D, which is similar to, and has evolved from, THREDE [11]. The parallelisation was undertaken using PVM [7], due to its widespread acceptance in the parallel-processing community at the start of the project. This operates by running a supervisory 'C' program which calls appropriate PVM library routines. These 'spawn' copies of the main task program (which may be in Fortran, but called from a C program) onto designated processors. The partitioning of the computational task is controlled by passing variables to the replicated programs, chosen to control the portion of the data that is to be associated with each processor.

The main time-marching FDTD equations and the ABC treatments were fully parallelised to ensure efficient execution of the code. However, the thin wire algorithm used for the handset antenna [15] was parallelised in a degenerate manner. This means that the thin wire computation was carried out on the root processor (the one assigned for supervisory code and serial parts of the program), and the results of this distributed to the relevant processors. Many of the problems used as test cases for the parallel FDTD code make use of thin wire sections in the code and could be slowed down by this. It is not, however, a computationally significant part of the modelling of these problems and thus it does not greatly affect the efficiency of the parallel code.

A small scaleability test, involving the modelling of simple dipole radiation, was undertaken with the FDTD software on two Meiko CS-2 parallel computers: the size of the problem space was $120\times240\times50$ ($=1.44\times10^6$) nodes, run for 500 iterations. Results are shown in Table 3: the 'topology' indicates the way in which the x, y and z indices of the data matrix were partitioned between processors.

The generally constant value of the product of the number of processors and the run time indicates that the scaleability is near-perfect under these conditions, except with only three processors, where the poor result obtained is probably due to the necessarily poorly-matched topology, in which there has to be a one-dimensional partition with one portion of the task requiring twice as much communications traffic as the other two. The comparative results with different topologies for eight processors are interesting, as the differences are small, although the (nominally) optimum topology ($2\times2\times2$) actually gives the poorest result, albeit by a negligibly small margin. It is more interesting to note that the least optimal partition ($1\times8\times1$) shows no degradation of efficiency in this case, indicating that processor workload imbalance is negligible in this case, and that the 'slice' partitions are not so thin as to be dominated by communications. These particular results may have been influenced by the relatively 'long and thin' problem space considered. Amdahl's law suggests that the scaleability will be degraded with a large number of processors, and this effect is starting to appear with sixteen processors, but not to a great degree.

Table 3. Scaleability tests with Meiko CS-2 computers

p	f_c. MHz	Run time mins	Top-ology	p×time (mins)	Speed-up†	Efficiency †
3	50	65:33	1×3×1	196.7	1.5	49%
4	50	24:00	2×2×1	96	4	100%
6	50	16:57	2×3×1	101.7	5.7	94%
8	50	12:48	2×2×2	102.4	7.5	94%
8	50	12:40	2×4×1	101.3	7.6	95%
8	50	12:41	1×8×1	101.5	7.6	95%
16	40	8:12	2×4×2	105.0*	14.6*	91%

†Assumes ideal scaleability for ≤4 processors. *Clock frequency (f_c) normalised to 50 MHz.

The head-telephone interaction model was ultimately run using the parallel FDTD code on a 128-processor IBM SP-2 parallel computer to assess the effectiveness of the parallel port. Figure 4 shows three sets of results for speed-up of the code. Test 1 was for a 2.5 mm resolution model of the head, hand and mobile telephone, Tests 2 and 3 being at resolutions of 2 mm and 1 mm respectively. Table 4 shows the FDTD problem size for each of these test cases and Fig. 5 shows a typical slice of the computed output.

It should be noted that it was not possible to run the 1 mm problem on fewer than eight processors on this computer since the data quantity then became too large for the memory of the processors in use. Therefore the results for one, two and four processors were extrapolated from the result at eight processors using the efficiencies found in

the 2 mm test case. The curve labelled 'ideal' in Fig. 4 illustrates the speed-up that would be achieved with a 100% efficient system.

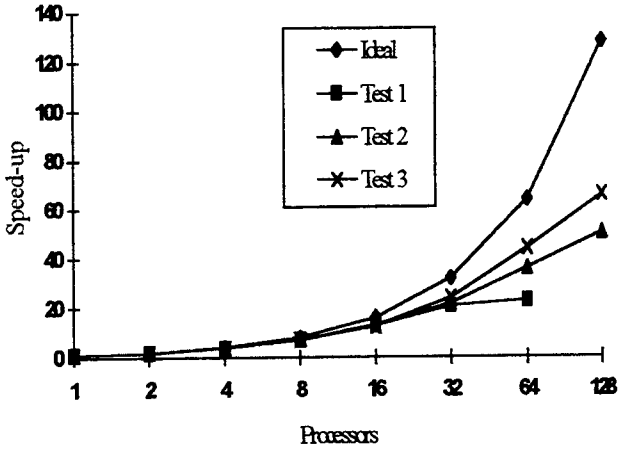

Figure 4: Speed-up results from the head-telephone test cases, running on a 128-processor IBM SP-2.

Table 4. Memory Requirements for Test Problems

Test No.	Resolution	Problem Size (cells)	Memory Required
1	2.5 mm	1.3×10^6	45 MBytes
2	2 mm	2.5×10^6	90 MBytes
3	1 mm	20×10^6	720 MBytes

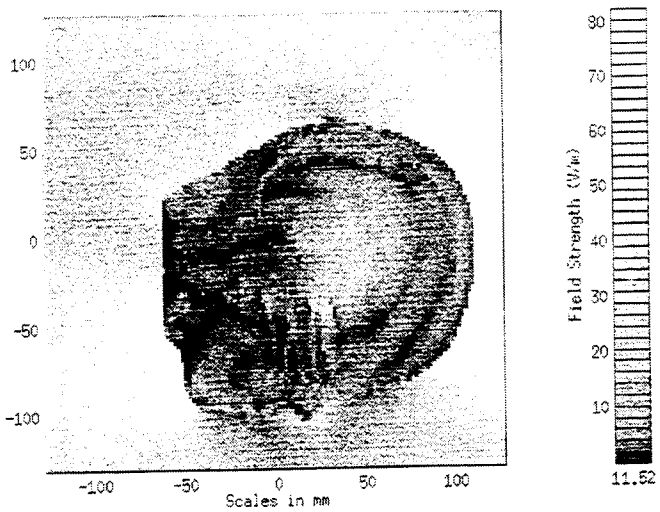

Figure 5 A typical slice of the computed output: electric field magnitude through the centre of a head adjacent to a 1.8GHz mobile telephone handset.

Figure 4 demonstrates that efficient massively-parallel processing of electromagnetic problems is now a mature and viable technique, whereas early efforts frequently showed little gain after a very small number of processors was exceeded. It also shows two very important effects: firstly, the larger the computational task, the greater is the efficiency of the parallelisation, because the ratio of computational to communications tasks is increased. Secondly, when a task is subdivided over too many processors the efficiency of the parallelisation reaches a limit because the computational task is attenuated to a point where it becomes of the same order as the communications task: this is clearly seen in the results for the smallest test case (Test 1) with 64 processors.

The objective of portability has been demonstrated by use of this software on Meiko and IBM computers, but it was also successfully tested on one by Parsytec, and on networks of UNIX workstations.

5 CONCLUSIONS

Electromagnetic field computation is inherently a SPMD (single instruction, multiple data) process which is very appropriate for execution on vector and parallel supercomputers. Automatic algorithms for efficient exploitation of these are not yet fully developed and some care is thus necessary in the parallelisation of traditional serial software, taking account of the structure of the software and the way in which it interacts with the detailed architecture of the computer. Ways of doing this have been reviewed: for differential-equation based methods the PVM or MPI algorithms appear to offer a useful and portable approach which is accepted over a wide range of platforms.

Provided a sufficiently large task is addressed, it has been shown that efficient massively-parallel processing of electromagnetic problems is now a viable and mature technique, at least for differential-equation based formulations, whereas early efforts frequently showed little gain after a very small number of processors was exceeded.

It was found that as the problem size increased the efficiency of the code increased, the results tending towards the ideal. It is a paradox that it is difficult to accurately assess the efficiency of very large test cases, as they are too large to be run on a small number of processors. It was observed that when a task is subdivided over too many processors the efficiency of the parallelisation reaches a limit because the computational task is reduced to a point where it becomes of the same order as the communications task.

the 2 mm test case. The curve labelled 'ideal' in Fig. 4 illustrates the speed-up that would be achieved with a 100% efficient system.

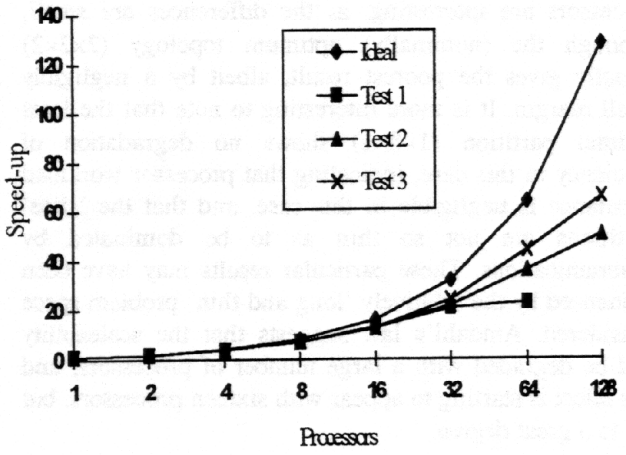

Figure 4: Speed-up results from the head-telephone test cases, running on a 128-processor IBM SP-2.

Table 4. Memory Requirements for Test Problems

Test No.	Resolution	Problem Size (cells)	Memory Required
1	2.5 mm	1.3×10^6	45 MBytes
2	2 mm	2.5×10^6	90 MBytes
3	1 mm	20×10^6	720 MBytes

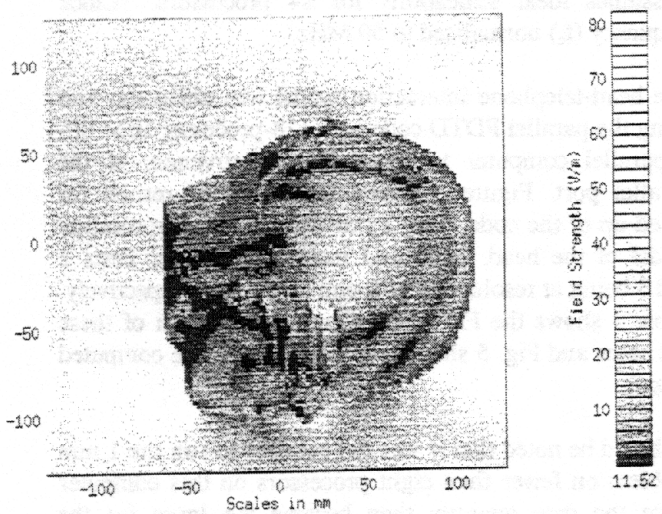

Figure 5 A typical slice of the computed output: electric field magnitude through the centre of a head adjacent to a 1.8GHz mobile telephone handset.

Figure 4 demonstrates that efficient massively-parallel processing of electromagnetic problems is now a mature and viable technique, whereas early efforts frequently showed little gain after a very small number of processors was exceeded. It also shows two very important effects: firstly, the larger the computational task, the greater is the efficiency of the parallelisation, because the ratio of computational to communications tasks is increased. Secondly, when a task is subdivided over too many processors the efficiency of the parallelisation reaches a limit because the computational task is attenuated to a point where it becomes of the same order as the communications task: this is clearly seen in the results for the smallest test case (Test 1) with 64 processors.

The objective of portability has been demonstrated by use of this software on Meiko and IBM computers, but it was also successfully tested on one by Parsytec, and on networks of UNIX workstations.

5 CONCLUSIONS

Electromagnetic field computation is inherently a SPMD (single instruction, multiple data) process which is very appropriate for execution on vector and parallel supercomputers. Automatic algorithms for efficient exploitation of these are not yet fully developed and some care is thus necessary in the parallelisation of traditional serial software, taking account of the structure of the software and the way in which it interacts with the detailed architecture of the computer. Ways of doing this have been reviewed: for differential-equation based methods the PVM or MPI algorithms appear to offer a useful and portable approach which is accepted over a wide range of platforms.

Provided a sufficiently large task is addressed, it has been shown that efficient massively-parallel processing of electromagnetic problems is now a viable and mature technique, at least for differential-equation based formulations, whereas early efforts frequently showed little gain after a very small number of processors was exceeded.

It was found that as the problem size increased the efficiency of the code increased, the results tending towards the ideal. It is a paradox that it is difficult to accurately assess the efficiency of very large test cases, as they are too large to be run on a small number of processors. It was observed that when a task is subdivided over too many processors the efficiency of the parallelisation reaches a limit because the computational task is reduced to a point where it becomes of the same order as the communications task.

Acknowledgements

Primary financial support was provided by the UK Engineering and Physical Sciences Research Council (EPSRC). The human head dataset was created by Peter Olley, of Bradford University, using MRI data from the University of North Carolina. Access to the KSR-1 computer was provided by the Manchester Computer Centre. The parallelisation of EMA3D was funded by the ESPRIT III EUROPORT2 project and undertaken jointly with British Aerospace (Military Aircraft) Ltd and EMA Inc. Access to IBM and Meiko parallel computers was arranged by Smith System Engineering Ltd and provided by CERFACS (Toulouse), CERN (Geneva), Cornell Theory Center (Ithaca) and KTH (Stockholm).

References

[1] Olley, P., and Excell, P. S., 1995: 'Classification of a High-Resolution Voxel Image of a Human Head', ***International Workshop on Voxel Phantom Development***, National Radiological Protection Board, Chilton UK, 16-23.

[2] Davidson, D.B. and Ziolkowski, R.W., 1995: 'A Connection Machine (CM-2) Implementation of a Three-Dimensional Parallel Finite Difference Time-Domain Code for Electromagnetic Field Simulation', ***Int. J. Num. Modelling: Electronic Networks, Devices and Fields***, **8**, 221-232.

[3] Rodohan, D.P., Saunders, S.R., and Glover, R.J., 1995: 'A Distributed Implementation of the Finite Difference Time-Domain (FDTD) Method', ***Int. J. Num. Modelling: Electronic Networks, Devices and Fields***, **8**, 283-291.

[4] Buchanan, W.J., Gupta, N.K., and Arnold, J.M., 1993: 'Simulation of Radiation from a Microstrip Antenna using Three-Dimensional Finite-Difference Time-Domain (FDTD) Method', ***Proc. IEE Int. Conf. on Antennas and Propagation, Edinburgh,*** **2,** 639-642.

[5] Amdahl, G.M., 1967: 'Validity of Single-Processor Approach to Achieving Large-Scale Computing Capability', ***Proc. AFIPS Conf., Reston, VA, USA***.

[6] Tatalias, K.D., and Bornholdt, J.M., 1989: 'Mapping Electromagnetic Field Computations to Parallel Processors', ***IEEE Trans. Magnetics,*** **25,** 2901-2906.

[7] Beguelin, A., et al, 1991: 'A Users' Guide to PVM - Parallel Virtual Machine', ***Oak Ridge National Laboratory***, USA, Report No. ORNL/TM-11826.

[8] Heath, M.T., and Etheridge, J.A., 1991: 'Visualizing the Performance of Parallel Programs', ***IEEE Software***, **8**, 29-39.

[9] Excell, P.S., Porter, G.J., Tang, Y.K., and Yip, K.W., 1995: 'Re-working of Two Standard Moment-Method Codes for Execution on Parallel Processors', ***Int. J. Numerical Modelling: Electronic Networks, Devices and Fields***, **8**, 243-248.

[10] Gedney, S.D., and Barnard, S., 1995: 'Efficient FD-TD Algorithms for Vector and Multiprocessor Computers', *in* ***Taflove, A., 'Computational Electrodynamics: The Finite-Difference Time-Domain Method'***, Boston: Artech House.

[11] Holland, R., 1977: 'THREDE: A Free-Field EMP Coupling and Scattering Code', ***IEEE Trans. Nuclear Science***, **NS-24**, 2416-2421.

[12] Gedney, S.D., 1995: 'Finite-Difference Time-Domain Analysis of Microwave Circuit Devices on High Performance Vector/Parallel Computers', ***IEEE Trans. Microwave Theory & Tech.,*** **43,** 2510-2514.

[13] Varadarajan, V., and Mittra, R., 1994: 'Finite-Difference Time-Domain (FDTD) Analysis using Distributed Computing', ***IEEE Microwave and Guided Wave Lett.,*** **4,** 144-145.

[14] Perala, R., Whittle, S., Hargreaves, M., and Kerton, P., 1995: 'An Introduction to PEPSE ('Parallel Electromagnetic Problem Solving Environment') and Considerations for Paralellization of a Finite Difference Time-Domain Solver', ***Int. J. Num. Modelling: Electronic Networks, Devices and Fields***, **8**, 187-204.

[15] Excell, P. S., Olley, P., and Jackson, N. N., 1996: 'Modelling of an Arbitrarily-Oriented Mobile Telephone Handset in the Finite-Difference Time-Domain Field Computation Method', ***Applied Computational Electromagnetics Society Journal***, **11**, 55-65.