ANTENNA ARRAY MODELLING BY PARALLEL PROCESSOR FARMS

Iain Cramb, Daniel H. Schaubert, Richard Beton, James Kingdon, and Colin Upstill*

Roke Manor Research Limited,

Roke Manor, Romsey, Hampshire, SO51 0ZN, UK

ABSTRACT

The fast concurrent implementation of a FORTRAN method of moments analysis of the electromagnetic properties of an array of tapered slot antennae is discussed. Decomposition of an existing FORTRAN algorithm for calculation of the currents induced by an incident radiation field in an infinite array of tapered slot antennae is described. The problem was distributed across an array of INMOS transputers, yielding significant speed-up over a single CPU. This decomposition was relatively simple to implement, can readily be scaled to larger processor arrays virtually indefinitely, and promises linear speed-up with the number of processors in the array.

1. INTRODUCTION

A growing number of electromagnetic analysis problems are being formulated for processing on parallel processing computers, and transputer arrays represent one of the least costly parallel computer systems available to the EM analyst. Recently, tutorial papers have appeared to describe the successes and difficulties one may encounter in solving various types of problems [1, 2]. The objective of this paper is to describe the way in which a FORTRAN moment method analysis was converted from a code for a single CPU to a code that utilises several cpus with *very* high efficiency.

The particular analysis that was converted is for infinite arrays of endfire, tapered slot antennas [3]-[5]. The currents flowing on the metallic fins that comprise an infinite array of tapered slot antennae (figure 1) are determined by using the method of moments to solve the electric field integral equation. Floquet's theorem is used so that only a unit cell of the structure must be considered, and the version of the analysis being performed here uses piecewise sinusoidal *rooftop* basis and testing functions. Most (over 95%) of the time taken to calculate the currents induced in the antennae is spent filling the impedance matrix for the 50 to 200 unknowns that are used to model the current. Since this particular analysis is not the main topic of the paper, only a few sample results are included.

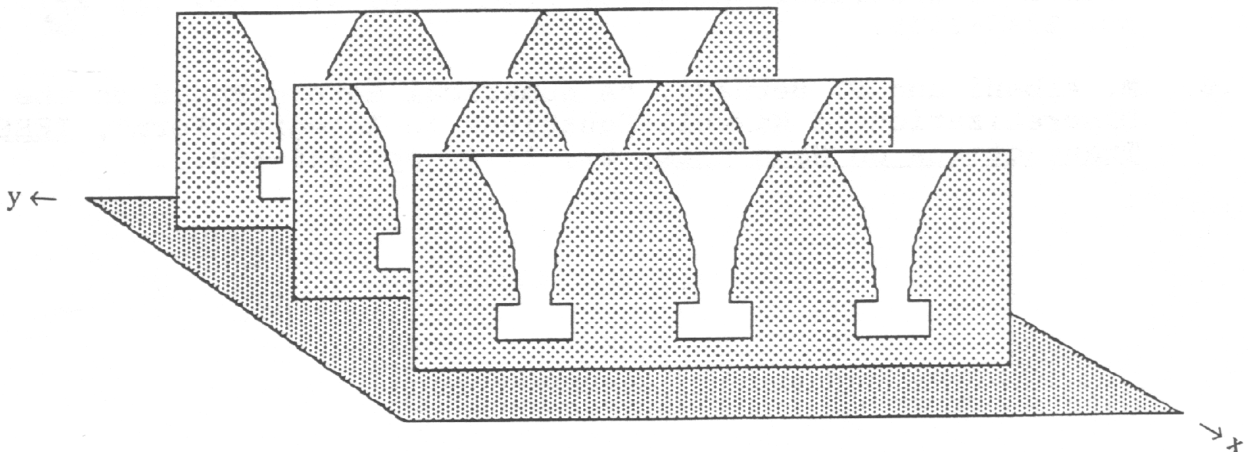

Figure 1 Tapered slot antennae

Use of the numerical model for antenna design and development requires computing the antenna currents for many scan angles and frequencies of operation. Computations for each set of parameters

* This work was performed while D.H. Schaubert was on leave from:
University of Massachussetts at Amherst,
Department of Electrical and Computer Engineering,
Amherst, MA 01003.

are lengthy but tractable with modest CPUs; typical cases require several minutes to a few hours on a 25-MHz 80386/80387 PC. However, typical applications require that dozens, or perhaps hundreds, of parameter sets be analyzed. It is the objective of this paper of demonstrate that EM analysts can readily decompose a FORTRAN algorithm of this type to be distributed to an economical transputer array. The decomposition maintains the input and output features familiar to the user while distributing the computations for each parameter set to a different processor via a "general purpose" communications harness that is easily adapted to different FORTRAN algorithms. A request-driven transputer farming paradigm is used, which provides efficient utilization of the transputer array for this type of computation. Thus, we demonstrate how easily the total processing speed of the transputer array can be used to the benefit of EM analysis codes that were not originally intended for parallel processing as well as to the benefit of new codes that may be written specifically to exploit the parallelism in computations for a single parameter set. The controller/worker model is described in detail in the implementation section below.

2. PARALLEL IMPLEMENTATION

2.1. Paradigms for Parallelism

Parallel systems may be described at various levels of abstraction, and it proves convenient to use the subclasses *conceptual, physical*, and *logical* architectures to describe more accurately what is meant by 'parallel architectures'. This is more than mere taxonomy: understanding parallel processing and effectively managing parallel systems design and implementation requires different views to be taken for different purposes, and thinking in terms of these architectural subclasses has been found very useful in practice.

Conceptual architectures involve the most abstract aspects of parallel processing systems, in particular the type of computation being undertaken. Most conceptual architectures fall into one of four classes: control-driven, data-flow, object-based, and logic-based.

Physical architectures deal with the actual physical implementation of such systems. The categorisation of physical architectures preferred is that introduced by Flynn [6]. It is a classification which distinguishes four basic combinations of instruction streams and data streams (SISD, SIMD, MISD, and MIMD). Flynn's scheme is simpler than most, and is very useful for classifying machines which are clearly different. This work uses a machine in the last of the above four categories - a transputer array.

The *logical architecture* of a system is the architecture seen by applications software designers and programmers. There are three main types of logical architecture appropriate to MIMD machines - *farming, geometric* and *algorithmic* [7,8].

Farming is a controller/worker paradigm in which an overall problem is split into conveniently sized work packets which are distributed over a number of worker processors in such a way as to *dynamically balance* the work load across the processors in the farm. The controller knows about the work that has to be done, and the worker knows how to do it - each worker has an identical copy of the code. The controller sends batches of work to each worker; the latter executes its algorithm on a batch of data then returns the results, starting to work on the next batch pending any message transmission. The primary advantage of the task farm is the fact that it dynamically balances the loading of an overall problem across a network of processors. A potential disadvantage is that each worker has to have a copy of the complete code, which can lead to problems if memory is limited.

An architecture of this type is most appropriate to an application in which firstly, the work packets may require different amounts of processing - that is to say the amount of work for each is packet-dependent. Secondly, and perhaps more obviously, the overall task should naturally lend itself to temporal concurrency. This means that as far as is possible, the processing of any individual packet should not depend upon the processing of any other packet. Complete independence may not actually be possible; packets may have some positional or other relationship to each other. However, such an architecture does demand that the final result does not depend upon the order in which the packets are processed.

In contrast with the automatic run-time load-balancing in a farm, geometric and algorithmic architectures, which exploit data concurrency and algorithm concurrency respectively, have to be load-balanced at design time.

In *geometric parallelism*, the parallelism inherent in the data is exploited. The problem will generally have an underlying geometrical structure (this is often very similar to SIMD array processing), and when this is the case, the array topology reflects the data topology. In geometric parallelism, as in farming, each processor has an identical copy of the complete code for the whole program, but each works on separate pre-defined portions of the data space, i.e., each processor is responsible for a specific area or volume of the data set.

Algorithmic parallelism is the case in which the parallelism inherent in the algorithm is exploited, e.g. with a pipeline. In a distributed memory machine, each processor has its own segment of the code, and this is different from processor to processor. The topology of the machine typically reflects the topology of the data flow graph on which the algorithm is based.

Many architectures are, in fact, hybrids of two or all three types, but our decomposition is purely a farm. The logical architecture is described in more detail a little later. Before giving the description we present a brief summary of the various techniques used in implementation of parallel software.

2.2. Implementing Parallel Software

There are several distinct approaches to implementing parallel software. The quickest, and the one used in this case, is to use existing codes within a specially created communication harness. Possible parallelism is identified at the procedure level, separate code segments are isolated, and then a communication harness is constructed. Slight modifications may be needed to the existing code - the extent of this depends on the manner in which the original code was written. According to how well the existing code is written, this approach usually requires less effort than either implementing the same algorithm in a more appropriate language (e.g., Occam) or than making a completely fresh start with respect to algorithm and language, and provides a significant return.

2.3. Transputers and Parallelism

Confusion is often caused in discussions of transputer-based software by the ambiguous use of the term parallelism. Three distinct forms of parallelism may be observed in a multi-transputer implementation.

Firstly there is the case of separate transputers running separate processes. This is genuine simultaneous operation of concurrent or parallel processes, similar to having two or more independent computers running side by side.

Secondly, a single transputer may be running several processes in a time-sliced mode. This is analogous to the forms of pseudo-parallelism performed on many conventional machines, e.g., those running processes under an operating system such as Unix or VAX VMS. Time slicing is performed on a time scale which is short compared to the process lifetimes. It should be remembered that each of the parallel processes on a single transputer is competing for cpu time, and in the same way that another user on a VAX makes operation seem slower, every extra process will slow down the operation of those already there.

The time-slice mechanism of the transputer is implemented in hardware, and is particularly efficient compared to many similar systems. In particular, processes waiting for communication are removed from the active process list (descheduled) and incur no cpu overheads.

The third form of parallelism concerns the hardware of the transputer. The T800 contains a fixed-point cpu, a floating-point processor, and four bidirectional serial communications link engines; all of these elements can function simultaneously and independently. This is particularly important in relation to communications. If the time required to pass a message is less than the processing time required to do the work generated by the previous message, then the communications can be done in the background with little impact on the time required to generate the desired results.

The link engines operate by DMA. For a message to pass between two processors, the cpu must provide the link engine with a start address and a byte count. This cpu overhead for communications means that it is more efficient to pass long messages (e.g., a few kbytes) than short ones (e.g., single bytes).

In this paper we are primarily concerned with the exploitation of the genuine parallelism of running separate processes on separate processors.

3. ALGORITHM FOR THE CPU

The analysis was originally coded in FORTRAN for implementation on a single CPU. The code is comprised of a main program which performs data input and output, and controls the computations by calling subroutines that fill the impedance matrix and solve the system of equations. A typical matrix element is obtained by evaluating an expression of the form [9]:

$$Z_{pq} = \sum_{n=1}^{N} \sum_{m=-M}^{M} K_{mn} B^{q}_{mn} T^{p}_{mn}$$

$$K_{mn} = \frac{(n\Delta V_m)\sin(\beta_{mn}a)}{(\beta_{mn}a)\,[\cos(U_o a) - \cos\beta_{mn}a]}$$

$$B^{q}_{mn} = \frac{\sin(V_m W_q)\cos(n\Delta Z_q)[\cos(n\Delta h_q) - \cos(kh_q)]}{V_m(k^2-n^2\Delta^2)\sin(k\,h_q)}\, e^{jV_m y_q}$$

$$T^{p}_{mn} = \frac{\sin(n\Delta z_p)\sin(n\Delta W_p)[\cos(V_m h_p) - \cos(k\,h_p)]}{(n\Delta)(k^2-V_m^2)\sin(k\,h_p)}\, e^{jV_m y_p}$$

$$\beta_{mn} = \sqrt{k^2 - V_m^2 - n^2\Delta^2}, \quad \mathrm{Im}\{\beta_{mn}\} \le 0$$

$$V_m = k\sin\theta_o\sin\phi_o + m\frac{2\pi}{b}$$

$$U_o = k\sin\theta_o\cos\phi_o$$

$$k = \omega\sqrt{\mu_o\varepsilon_o}$$

θ_o, ϕ_o = main beam direction

a,b = array grid spacings in x and y directions

Δ = increment of the spectral variable

y_q, z_q = coordinates of center of basis function q

y_p, z_p = coordinates of the center of testing function p

h_q, W_q = half-length and half-width of basis function q

h_p, W_p = half-length and half-width of testing function p

The upper limits of the summations required for convergence are typically +/- 15 for m and 500-1500 for n. The large range of the upper limit for n is due to this parameter being sensitive to the antenna geometry. The FORTRAN code uses the common practice of storing various terms that comprise K_{mn}, B^{q}_{mn} and T^{p}_{mn} so that they are not recomputed when needed for successive sets of the indices.

Total storage requirements are kept to a reasonable level (1-3 MBytes) by proper nesting of the loops on m, n, p and q. The problem possesses an inherent parallelism because the computations are usually performed for several scan angles, which provides a simple way to divide it up into independent sub-problems. Except for a few initialising calculations, the various scan angles do not share any common data, but the code that is executed for each angle is identical.

4. THE FORTRAN FARM

4.1. Toplevel architecture

The architecture used in this case is shown in figure 2 - a ring. Data and work packets pass round the farm in the same direction. All file access and control is provided by the driver and the calculations are performed by the worker processors. The communications harness was written in the parallel processing language Occam, and all the original file access, control and calculation code was retained. The result is about 600-700 lines of Occam and about 2000 lines of FORTRAN. Occam was chosen for the communications harness because it is a natural language for describing parallel communications within a computer program and it was the quickest means available at the time of creating an efficient communications harness. Further data on the transputer system are contained in table 1.

Table 1 System characteristics

Hardware	Compliers
Meiko M10 computing surface	Inmos Occam compiler for the communications harness
T800 transputers (20 MHz, 4 cycle RAM) T800 maximum computing power = 1Mflop VAX3800 maximum computing power ≈ 3-4 Mflops	Meiko Fortran 77 compiler for the main program

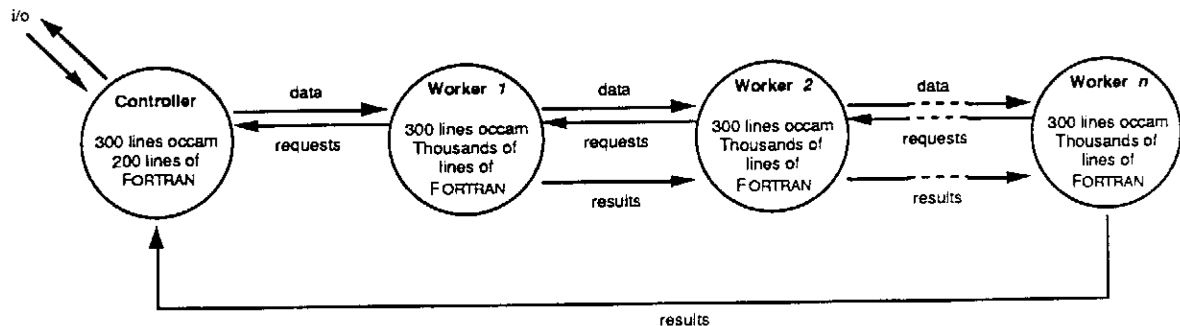

figure 2 Toplevel architecture

In this case the most convenient decomposition was to perform the calculations for different scan angles on different processors. This is the simplest approach possible, and is also the one which yields the highest ratio of computation load to communications load. The work packets passed out to the farm consist of little more than the scan angle at which the calculations are to be performed - a few bytes of data at most, whereas the calculations for each angle require between one and two hours of T800 cpu time.

4.2. Communications paradigm

The simplest type of communications harness for a farming decomposition is one in which the driver processor passes work packets out to the farm as quickly as possible, and each worker processor accepts data if it is free to do some work, or passes it on if not. This flood-fill approach to supplying the workers with data for processing is not appropriate in this case, because it is necessary for each worker to have buffer processes in which work packets must be stored temporarily whilst the processor decides whether to pass on the data or accept it for processing - the minimum possible

number of buffers is one. This is not usually a problem for decompositions in which the number of processors in the farm is one or two orders of magnitude *less* than the total number of work packets. However, in a case where the total number of work packets to be processed is only twice or three times the number of processors in the farm, the flood-fill approach can have very serious disadvantages. Work packets are often left stranded in buffers in such a way that at the end of a processing run the last two or three packets are processed one after the other by the same processor. This is not noticeable in cases where there are hundreds, or even thousands, of individual work packets, each of which takes a few milliseconds to process. If there are only a few tens of packets and a similar number of processors, as there are in this particular decomposition, then the total compute time can be significantly greater than if the farm were used with the maximum possible efficiency.

We have adopted an approach which removes the necessity for any buffering on the worker processors - the farm is request driven. The driver only passes work packets out to the workers when they specifically request it. Each packet is tagged with the address of the requesting worker and no other processor may accept that packet. This approach can reduce farm efficiency if the work packets are large because there is then a significant delay between a worker sending out a request and receiving its work package, during which time the processor stands idle; but for this problem the work packets are at most ten bytes, and for a transmission time around ten microseconds per processor the communications delay is insignificant.

4.3. Performance

Typical processing runs were over 20 to 55 scan angles with 10 to 20 worker processors available. The results below are for a particular example of 52 scan angles ($\theta = 0^o$ to 85° in 5° steps and $\phi = 0^o$ to 90° in 45° steps) distributed over 17 processors: the time taken for the processing run was 7.61 hours. The same processing run on a VAX 3800 with a floating point accelerator took 37.32 hours: even without the benefit of the extremely efficient FORTRAN compiler available on the VAX, the transputer system is four times as fast. This increase in compute power allowed overnight production of results which would otherwise have taken two days of dedicated VAX 3800 cpu time.

The VAX 3800 is chosen for comparison because its characteristics are widely publicized and these machines, or similar ones, are often available to EM analysts. Obviously, newer workstations could outperform the VAX 3800, but newer transputers also could outperform the T800.

4.4. Sample Computed Results

Typical results for the input impedance of the antenna depicted in figure 3 are shown in figures 4 and 5. These curves are similar to those one might obtain for any well-behaved phased array antenna. The resistance versus scan is reasonably constant near broadside and then drops to zero at endfire. In the H-plane ($\phi = 0$) and the E-plane ($\phi = 90$), the reactance changes are opposite. The same array, operating at 3.725 GHz, exhibits at grating lobe at 54.9^o in the principal planes. The reflection coefficient for this case is shown in Figure 6. The data in Figures 4, 5 and 6 were computed by using 77 rooftop-like current modes on the metal fin. These modes and a closely related formulation of the electromagnetic problem are described in [9], as are issues related to convergence and validation of the analysis. To summarize those findings, the summation limits mentioned above are adequate for computing the impedance to an accuracy that agrees within 1 or 2 ohms with published data for test cases, such as dipoles and monopoles, and also agrees well with waveguide simulators for other cases.

5. CONCLUSIONS

We calculate that the ratio of communications to compute load for this particular application is so small (a few milliseconds versus half an hour) that the farm size could easily be increased to several hundred processors without significant (or even visible) degradation of efficiency. This implies linear speed-up. This assumes either that a more finely grained decomposition for this algorithm could be found without excessively increasing the communications load on the farm, or that one would desire to compute the antenna response either for many thousands of scan angles, or for several tens of scan

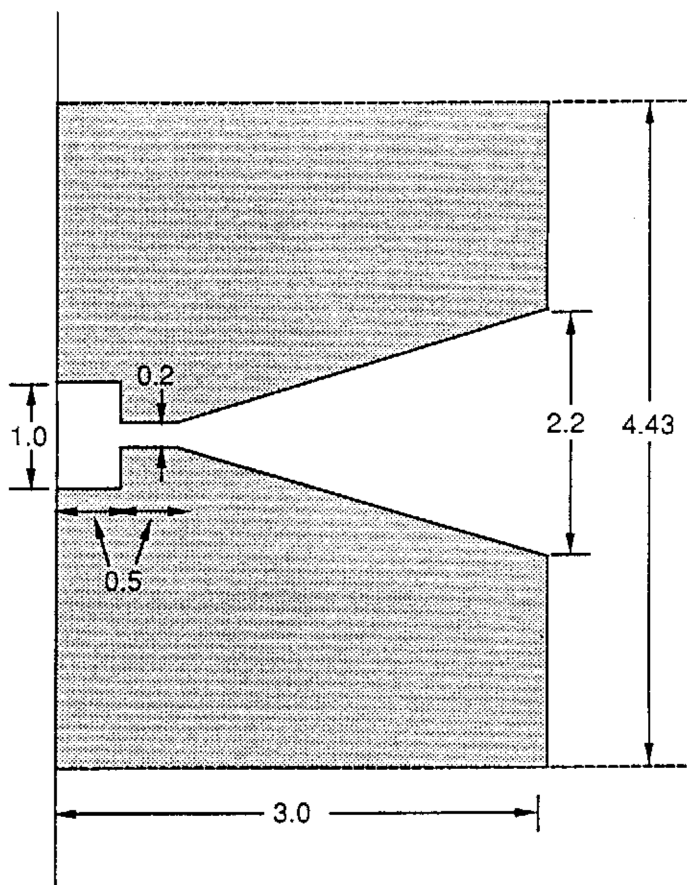

figure 3 Unit cell of linearly tapered slot antenna. Dimensions in cm and H-plane spacing = 4.43 cm.

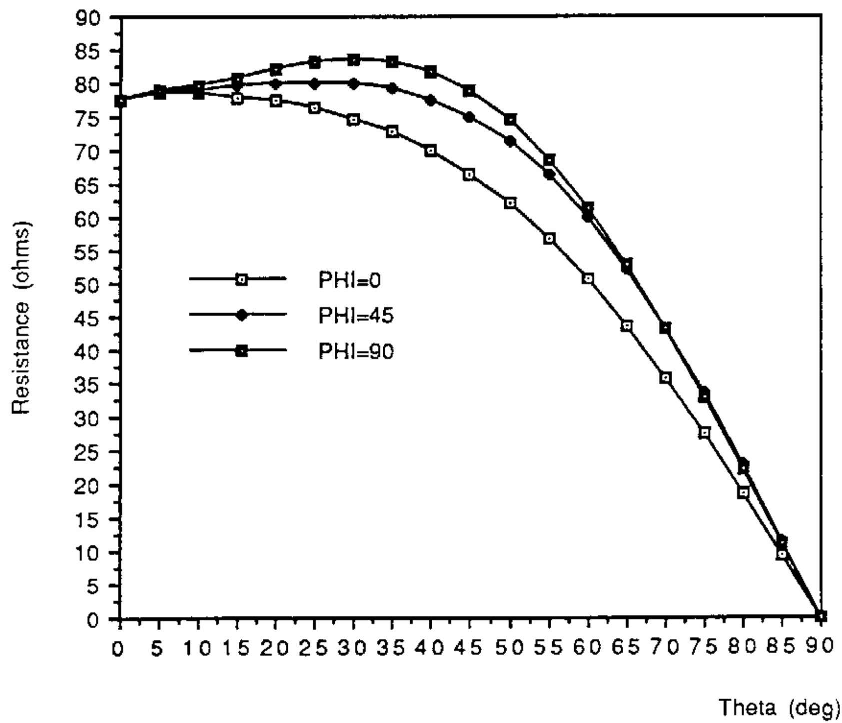

figure 4 Resistance against scan angle for three values of phi. Phi = 0 is H-plane and frequency = 2.54GHz.

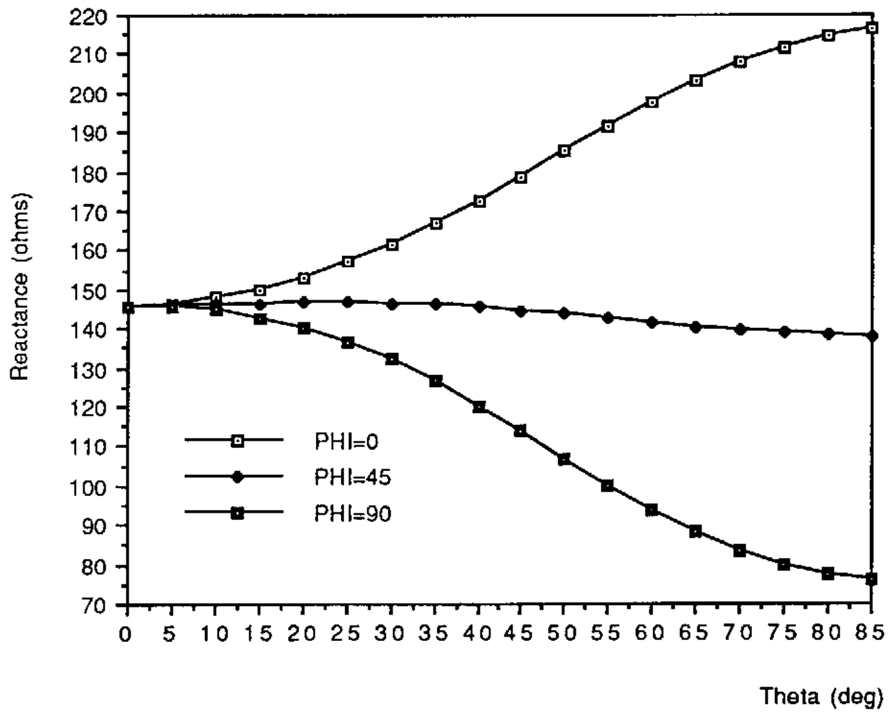

figure 5 Reactance against scan angle for three values of phi.

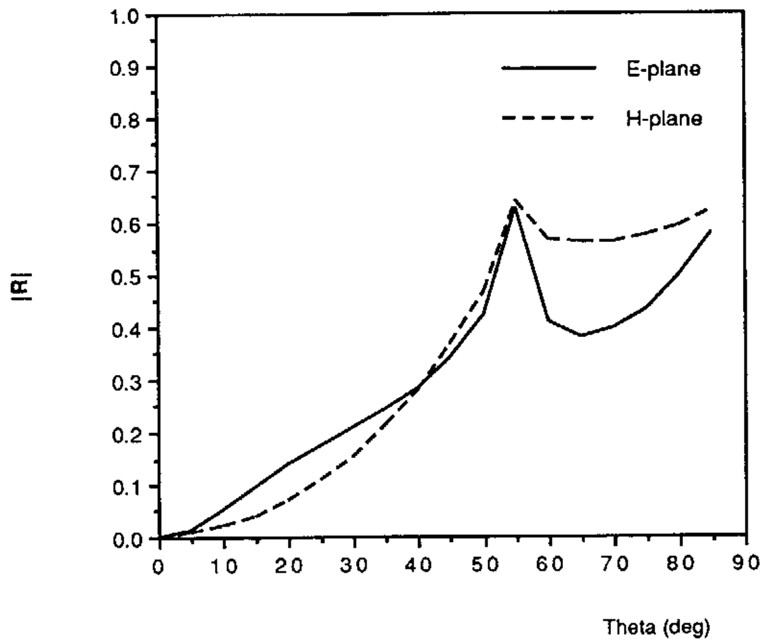

figure 6 Magnitude of reflection coefficient for array of figure 5 operated at 3.725 GHz.

angles at many different frequencies. Increasing the number of processors is easy, finances permitting - in the software, a single constant defining the number of workers is changed and the communications harness is recompiled - the FORTRAN is completely unaffected. We estimate that a 32 processor system, offering an order of magnitude speed-up over a VAX 3800, complete with all necessary software and additional hardware could be purchased for less than half the price of the VAX.

An alternative to increasing the number of processors, or redesigning and recoding the algorithm to suit a finer grained decomposition, is to use different hardware. Examples of high performance cpus available, or soon to be available, are the INTEL i860 and the INMOS T9000 transputer. These would both offer performance improvements over the T800 of at least a factor of ten - the i860 is already available (with communications handled by transputers), and the T9000 is expected to be available late in 1991. The i860-transputer hardware can be obtained for around £10,000 per i860; as yet the truly exceptional performance of this chip (upto 60Mflops - 60 x T800 transputer power) is only available to those who are willing to hand craft their algorithms in the appropriate assembler. Currently available compilers achieve between 5 and 10 Mflops. Between the times of writing the original manuscript and the revisions, the cost of i860 boards has been reduced more than 50 percent. However, at present, it appears that the 30 Mflops/£20,000 cost effectiveness of the transputer farm employing T800's is superior to that of multiple i860's that would be required to achieve 30 Mflops with currently available compliers. This means that the i860 is, as yet, no more cost effective than the transputer - we are not yet able to comment on the T9000.

6. REFERENCES

[1] Hafner, C., 1989, "Parallel Computation of Electromagnetic Fields on Transputers", *IEEE Ant. Prop. Soc. Newsletter*, **31**, no 5, 6-12.

[2] Davidson, D.B., 1990, "A Parallel Processing Tutorial", *IEEE Ant. Prop. Soc. Magazine*, **32**, no 2, 6-19.

[3] Lewis, L.R., Fasset, M., and Hunt, J., 1974, "A Broadband Stripline Array Element", *Digest of 1974 IEEE Symp. Ant. Prop.*, 335-337, Atlanta, GA.

[4] Gibson, P.J., 1979, "The Vivaldi Aerial", *Digest of 9th Eur. Microw. Conf.*, 120-124, Brighton, UK.

[5] Yngvesson, K.S., Korzeniowski, T.L., Kim, Y.S., Kollbert, E.L., and Johansson, J., 1989, "The Tapered Slot Antenna – A New Integrated Element for Millimeter Wave Applications", *IEEE Trans. Microw. Th. Tech.*, **MTT-37**, 365-374.

[6] Flynn, M.J., 1966, "Very high speed computing systems", *Proc IEEE*, **54**, No 12, 1901-1909.

[7] Pritchard, D.J., 1987,"PARLE Vol 1", *Lecture Notes in Comp. Sci.* No. 258, Springer-Verlag.

[8] Beton, R.D., Turner, S.P., Upstill, C., 1989, "Hybrid Architecture Paradigms in a Radar ESM Data Processing Application", *Microprocessors and Microsystems*, **13**, No. 3, 160-164.

[9] Cooley, M.E., Schaubert, D.H., Buris, N.E. and Urbanik, E.A., 1991, "Radiation and Scattering Analysis of Infinite Arrays of Endfire Slot Antennas with a Ground Plane", to appear in *IEEE Trans. Ant. Prop.*, **AP-39**, No. 11.