Application Brief
Intel Scientific Computers

Porting an Electromagnetic Scattering Code to the Intel iPSC™

Cleve Moler and David Scott
Intel Scientific Computers

PROBLEM DESCRIPTION

Professor Don Wilton, of the University of Houston, together with several colleagues and students, has developed a FORTRAN program for calculating the electromagnetic scattering and radar cross section of objects of arbitrary shape in three dimensions [1]. The program finds approximate solutions to Maxwell's equations using a finite element method applied to the electric field integral equation.

The surface of an object is modeled with planar triangular patches. A complex-valued electric current is associated with each edge in the collection of triangles. These currents, i, are related to each other by a system of simultaneous linear equations generalizing Ohm's Law.

$$Z\,i = e$$

Here e is a vector computed from an externally imposed electric field and Z is a large, dense, complex matrix whose elements are obtained by numerical quadrature of functions involving the distances between all possible pairs of surface triangles. After this equation has been solved for i, far-field quantities - such as radiation patterns and radar cross sections - can be computed by taking the inner products of i with a number of independent vectors, q_k. This is essentially a large matrix-vector product,

$$s = Q\ i$$

The number of unknowns, n, is equal to the number of distinct interior triangular edges, which is approximately 3/2 the number of triangles. Values of n in the range of a few thousand are needed to accurately model complicated objects. Solution of a non-sparse, non-symmetric, complex linear system of order 2000 requires over 32 megabytes of storage and over 20 trillion floating point operations.

PARALLEL PROGRAMS

The original program, intended for a conventional sequential computer, uses LINPACK subroutines CGEFA and CGESL to solve the complex linear system. Parallel generalizations of these subroutines are now available for the Intel iPSC™, so the question becomes:

How much effort is required to parallelize the rest of this program?

The program has four main sections. The time spent in each section increases at different rates with increasing n.

Step		Work	Parallelization
1.	Geometry initialization	$O(n)$	Don't bother
2.	Assemble Z and e	$O(n^2)$	How?
3.	Solve $Zi = e$	$O(n^3)$	Already done -- LINCUBE
4.	Compute Qi	$O(qn)$	Perfectly parallel

Step	Description
1. Geometry Initialization	This step takes so little time that the parallel program simply duplicates the calculation on each processor.
2. Matrix Assembly	Performing the second step, the matrix assembly, in parallel, requires some thought and analysis, but very little actual code change. The computation involves a double loop over the surface, examining all possible pairs of triangles. For each pair, several integrals are evaluated by numerical quadrature. Each of these integrals affects the 9 elements of Z that describe the interactions of the 3 edges on one triangle with the 3 edges of the other. LINCUBE expects Z to be distributed by columns, with column j stored on the processor (j-1) $mod\ p$. Consequently, a test is inserted near the beginning of the body of the double loop. Each processor computes only those integrals which affect its matrix elements.
3. Solve $Zi = e$	This step dominates the execution time as n increases. The call to the LINPACK equation solver in the original program is changed to a call to a LINCUBE version of the same subroutine. These subroutines are described in another Intel Scientific Computers Application Brief.
4. Far-Field Calculations	This step is almost trivial to convert to a parallel form, but the approach involved applies to many other algorithms. The main loop is:

```
      DO 499 I = 1, NFACES
      ...
499 CONTINUE
```

This is changed to

```
      ID = MYNODE()
      P = 2**CUBEDIM()
      DO 499 I = ID+1, NFACES, P
        ...
499 CONTINUE
      CALL GSUM ( ... )
```

The system functions **MYNODE** and **CUBEDIM** return the processor identification number and the hypercube dimension, so **P** is the number of processors. The loop body is executed for successive values of **I** on different processors and all processors execute the loop body the same number of times, plus or minus one. Finally, the library subroutine **GSUM** combines the quantities generated on the individual processors.

It took a few hours of discussion involving the various authors of different portions of the program to understand the modifications necessary to make it parallel, but the actual textual changes made involve only about a dozen lines out of roughly 1400 lines of code.

The performance of the program that resulted could still be improved. In the matrix assembly step, most of the integrals are computed by three different processors. This redundancy could be nearly eliminated by using a different edge numbering scheme. In addition, a rearrangement of some of the loops in the matrix assembly step would improve the vectorization within an individual processor.

---

[1] S. M. Rao, D. R. Wilton and A. W. Glisson, "Electromagnetic Scattering by Surfaces of Arbitrary Shape," *IEEE Trans. Antennas Propagation*, vol. AP-30, No. 3, 1982, pp. 409-418.