Distributed Computational Electromagnetics Systems

Gang Cheng
Geoffrey Fox
Kenneth Hawick
Gerald Mortensen

CRPC-TR94628
August 1994

Center for Research on Parallel Computation
Rice University
6100 South Main Street
CRPC - MS 41
Houston, TX 77005

Also available as *SCCS-635* from the Northeast Parallel Architectures Center at Syracuse University.

Distributed Computational Electromagnetics Systems*

Gang Cheng[†] Kenneth A. Hawick[†] Gerald Mortensen[‡] Geoffrey C. Fox[†]

Abstract

We describe our development of a "real world" electromagnetic application on distributed computing systems. A computational electromagnetics (CEM) simulation for radar cross-section(RCS) modeling of full scale airborne systems has been ported to three networked workstation cluster systems: an IBM RS/6000 cluster with Ethernet connection; a DEC Alpha farm connected by a FDDI-based Gigaswitch; and an ATM-connected SUN IPXs testbed. We used the ScaLAPACK LU solver from Oak Ridge National Laboratory/University of Tennessee in our parallel implementation for solving the dense matrix which forms the computationally intensive kernel of this application, and we have adopted BLACS as the message passing interface in all of our code development to achieve high portability across the three configurations. The performance data from this work is reported, together with timing data from other MPP systems on which we have implemented this application including an Intel iPSC/860 and a CM-5, and which we include for comparison.

1 Introduction

Traditional electromagnetic engineering simulations are largely limited by memory requirements, as well as by sequential processing time. Most electromagnetic applications on high performance computing systems so far implemented have been on either massively parallel processors or traditional vector-based supercomputers [5, 6]. The advances in workstation cluster technology, represented by clusterable IBM RS/6000 systems and Digital's Alpha farm with both high performance node processors and large memory capacity, provides new opportunities for computational electromagnetic applications which are both CPU time **and** memory intensive. Together with the emergence of portable distributed linear algebra packages such as BLACS/ScaLAPACK and portable message passing interfaces such as PVM and MPI, cluster-based computational electromagnetics systems provide an attractive solution to numerical simulations in engineering electromagnetic analysis and design with modest problem sizes. They achieve better cost/performance than the Massively Parallel Processors (MPP) solutions and this is especially important to real world applications where workstation clusters are more readily accessible to engineers.

One of the major motivations for our work was to port the CEM application to **multiple** MPP and distributed computing systems and to evaluate and demonstrate their capabilities. Portability, as well as performance scalability, was among our top objectives [7]. Our use of the distributed linear algebra packages ScaLAPACK[3, 8] and BLACS[1] in our parallel implementation allowed us to achieve true code portability across: Intel machines (the Intel Delta, an Intel Paragon and a Intel iPSC/860); an IBM SP-1 with either Ethernet or switch connection, **and** the three distributed configurations described above. We focus our discussion in this paper on portability issues and benchmarking aspects of this CEM application on cluster systems. Results of the related MPP efforts will be reported separately.

---

[*] This work is sponsored by Syracuse Research Corporation and the agency set forth: Plans and Programs Directorate (Signature Technology) Wright Laboratory Air Force Material Commend (ASC) US Air Force, Wright-Patterson AFB OH 45433-5000.

[†] Northeast Parallel Architectures Center, Syracuse University, Syracuse, NY 13244

[‡] Syracuse Research Corporation, Syracuse, NY 13244

2 The RCS Application Problem

The CEM application used in this work is a well-established CEM package from Syracuse Research Corporation (SRC), named ParaMoM which stands for Parametric patch Method of Moments (MoM). The most distinctive feature of the ParaMoM is its use of basis functions that conform to parametric surfaces with curvature and thus generate accurate and stable simulation results.

ParaMoM utilizes a general parametric surface formulation. All surface integrals are performed on the curved surface rather than on a flat facet approximation as has been traditionally done. The basis (expansion) functions are a parametric surface generalization of the well-known Rao-Witon-Glisson functions[12]. The domains of these functions are curved three-sided surface patches which conform exactly to the underlying global surface representation. The parametric surface patch approach is a foundation to which a comprehensive set of capabilities have been developed and included in ParaMoM 1.0[11]. These include: acceptance of IGES 114 and IGES 128 parametric surfaces, flat facets and IGES 110 wires; wire antenna modeling for user-defined loads, feed points, and connections to surfaces; wires may radiate or scatter; E, H, and combined-field formulations; up to three symmetry planes may be used to reduce memory and computation requirements for problems with geometric symmetry; impedance-sheet formulation for treatment of resistive cards (R-cards); graphics-based preprocessor to perform gridding and other model preparation functions not included in a general-purpose CAD package.

3 Porting the Application

The ParaMoM code consists of five major consequent processing phases which have different computational requirements in terms of CPU time and memory consumption, shown in Table 1 in terms of the number of unknowns N which is the number of basic functions and is proportional to the target surface area.

Table 1

Processing Phases of Sequential ParaMoM and Their Computational Requirements

Phase	Component	CPU time	Memory
1	Setup	$O(N)$	$O(N)$
2	Matrix fill	$\boldsymbol{O(N^2)}$	$\boldsymbol{O(N^2)}$
3	RHS vector fill	$O(N)$	$O(N)$
4	Matrix factor/solve	$\boldsymbol{O(N^3)}$	$\boldsymbol{O(N^2)}$
5	Far-field computation	$O(N)$	$O(N)$

The role of the setup phase is to read in the geometric target description (i.e. the parametric surface representation along with the triangulation, material, and wire connection information) and to precompute specific geometry information required by the fill algorithm which include positions and tangent vectors for the surface points used in matrix fill expansion and testing integrals. We used a block cyclic data partition so that each node precomputes evenly its piece of the geometry arrays and broadcasts its results to the other nodes. Each node then maintains a private copy of these arrays of the manageable size ($O(N)$) in its **local** memory.

Our algorithm design is focused on Phase 2 of filling the impedance matrix (Z-matrix), while ScaLAPACK LU routines are used to factor and solve the single complex dense Z-matrix in Phase 3. The major task in our implementation is to partition the computation of Z-matrix elements amongst processors by distributing its elements amongst processors. Since the ScaLAPACK LU solver uses a scattered square block (SSB) partition scheme, in order to avoid the communication overhead of redistributing the Z-matrix between Phase 2 and Phase 3, we choose to use the **same partition** scheme for the Z-matrix in the two phases.

The Basic Linear Algebra Communication Subprograms (BLACS) [9] is a library providing simple, portable message-passing for matrix-based operations in parallel linear algebra programs on distributed memory machines. It supports both point-to-point operations between processors and collective communications on a virtual two-dimensional processor grid. ScaLAPACK is a two-

dimensional distributed memory version of the LAPACK[10] package and relies on calls to the Basic Linear Algebra Subprograms (BLAS) for local computation, and calls to the PB-BLAS[4] for global computation. For portability reasons, communications takes place inside the PB-BLAS through calls to the BLACS.

We were fortunate in gaining early access to the ScaLAPACK library and adopted an early release of the REAL variable arithmetic version in writing our COMPLEX routines. It is important to parallelize the *full* applications code so as to optimize and balance the overall performance of this application[2]. The BLACS interface also provided a suitable level of abstraction for us to implement the "Setup" and "Far-field" components. The full schematic of our parallelized ParaMoM code is shown in Figure 2.

The internal storage mechanisms of ScaLAPACK have been well described elsewhere [3], however it is necessary to explain how the matrix and vectors involved are blocked and scattered so as minimize the inter-node communication requirements. There are three major user tunable parameters in BLACS/ScaLAPACK: P, Q and N_b. They represent how a two-dimentional matrix with a block size N_b is partitioned on a virtual processor grid of $P \times Q$. Consider a simplified example shown in Figure 1 where a vector of length 15 (can be either a single vector or all columns in each row of a matrix) is distributed across the memory belonging to four processor nodes which are conceptually arranged in a row ($P = 1$, $Q = 4$). If a block size N_b of the full vector length of 15 is used, all vector elements are in fact stored in the first processor's memory only. The opposite extreme is if an N_b of 1 is used, then alternate elements are stored in all four processors' memories as shown. Different blocked partitions are shown for N_b of 2 and 3.

The MoM matrix filling is well suited to distributed memory systems. Figure 3(a) gives the sequential filling algorithm in pseudo-Fortran code. The sequential algorithm has the following features which are well utilized and highlighted in our Single Program Multiple Data (SPMD) parallel algorithm.

Calculation of any Z-matrix element is completely independent of other elements in the matrix and is only dependent on its memory allocation. This property makes it possible to perform the Z-matrix calculations locally "in-place" on the processors on which the Z-matrix elements are stored and the performance of the parallel algorithm solely depends on the locality of the Z-matrix partition and not upon the networking topology of physical processors.

```
c loop over source patch
      Do isource = 1, nfaces
c loop over field patch
          Do ifield = 1, nfaces
c calculate integration points for patch pair (ifield,isource)
              call patch-interaction(ifield,isource)
c loop over basis functions of the source patch
              Do isrcnode = 1, 3
c look up the index in pre-defined table
                  j = lookup(isource,isrcnode)
c loop over integration points of the field patch
                  Do ipnt = 1, 4
c calculate potential integrals for the patch pair - main routine
                      call pots(isource,ifield,isrcnode)
c loop over basis functions of the field patch
                      Do ifldnode = 1, 3
c look up the index in pre-defined table
                          i = lookup(ifield,ifldnode)
c calculate intermediate result
                          tmp = exp(ifield,ifldnode,ipnt)
c accumulate the Z-matrix element where cz is the N by N Z-matrix
                          cz(i,j) = cz(i,j) + tmp
                      Enddo ifldnode
                  Enddo ipnt
              Enddo isrcnode
          Enddo ifield
      Enddo isource
```

Fig. 3(a): The Sequential Matrix Filling Algorithm

Our algorithm is strongly influenced by the characteristics of the numbering of basis functions among patches since the calculation for a patch pair $(isource, ifield)$ is based on the 9 associated basis functions $(isrcnode = 1 : 3, ifldnode = 1 : 3)$ and is eventually filled into the Z-matrix as

shown in Figure 3(b), where j, l are column-index related to the source path (*isource*) and i, k row-index related to the field patch (*ifield*). From Figure 3(b), it is clear that a column-major blocked partition of the Z-matrix achieves the best locality for a patch pair. It guarantees at least 6 Z-matrix elements mapped to a processor and the maximum number of processors used to calculate the total 9 elements is 2. In an average case when a general SSB partition is used, the maximum number of processors for a patch pair is 4. Compared to the ideal partition in which all the 9 elements are mapped to a single processor, the only overhead for the column-major block partition is one redundant computation of the routine "patch-interaction" which takes less than 20% of the total computation for a patch pair.

The locality issue examined above is for a single patch pair. We must also investigate the load-balance issue for all patch pairs to see if the overall Z-matrix computation is evenly distributed to all processors. Fortunately, the numerical model ensures that statistically the Z-matrix computation of all patch pairs is evenly distributed. Actually, for all Z-matrix elements except those related to boundary patches, an element's final value is accumulated from four patch pairs and we know a patch pair contributes to nine different Z-matrix elements in most cases. Our measurements in testing cases also proved this is true.

It becomes clear that a larger Z-matrix size gives better data locality, and a larger machine size makes a relatively better balanced load of the Z-matrix amongst processors. Therefore, the parallel filling algorithm will guarantee scalable performance in terms of machine size and problem size.

Our complete parallel implementation includes: (1) In the setup phase, parallel file input for target geometry data and a parallel algorithm for precomputation in which only global broadcast operations are used; (2) An embarrassingly parallel algorithm for Z-matrix filling which requires some redundant calculation of patch-patch interactions but no inter-node communication; (3) A similar embarrassingly parallel algorithm for RHS excitation vectors filling which has no inter-node communication; (4) A COMPLEX parallel ScaLAPACK LU solver was implemented and adopted; and (5) A parallel algorithm for the far-field calculation which has only global summations.

4 Performance Comparison and Discussion

We report in this section timings for the Z-matrix filling and LU factorization parts which take more than 95% of the total execution time. The filling part is almost 50% dominated by MFLOPS and 50% by memory access, while the LU part is more than 90% dominated by MFLOPS. Elapsed time for filling and both elapse time and MFLOPS for LU are listed.

The DEC Alpha farm consists of 8 Alpha model 4000 workstations running at 133 MHz and with 64 MB memory per processor, interconnected by a dedicated Gigaswitch which provides full FDDI bandwidth and low latency switching to every processor in the farm. The Ethernet-based IBM cluster has 8 IBM RS/6000 model 370 processors running at 62.5 MHz each, and with 64 MB memory per processor. The ATM link between two SUN IPXs operates at a peak bandwidth 155 Mbps(OC3c). Each SUN IPX has 32 MB memory and runs at 40 MHz.

Different modest problem sizes are used to show the performance comparison and scalability. For comparison purpose, timings of this application on an Intel iPSC/860 (40 MHz, 16 MB/node) and a CM5 (25 MHz, 16 MB/node) are also listed. The Intel implementation, with BLACS linked to its NX version, has exactly the same source code as that on the clustered systems, while the CM5 implementation used a completely different approach in which the fill part was written in a message-passing CMMD program and the LU part was a data parallel CMFortran program using a vendor-supplied CMSSL LU solver to utilize CM5's high-performance vector units. The CM-5 implementation used out-of-core DataVault (or SDA) disk memory to hold and transfer the Z-matrix between the two different components. For some problem sizes, more processors have to be used on the iPSC/860 and CM-5 due to the memory requirement. Notice that a block size $N_b = N/N_p$, $P = 1$ and $Q = N_p$ are used for all the tests reported here, where N_p is the total number of processors used, N the matrix size and $P \times Q$ the virtual processor grid. As discussed in Section 3, this partition is optimized for the matrix fill algorithm on all the platform and good for the LU part on the three distributed systems, but it is not optimized for the LU solver on the iPSC/860.

Table 2 gives an overall breakup of timings which indicates the two dominated components

and relatively balanced results. In Table 3-4, the clustered systems show compatible or even better performance than the MPP platforms with comparable number of processors, although the LU part is poorer, limited by the networking bandwidth and high latency of message-passing protocols on the cluster systems. The LU performance on the ATM-based configuration (Table 4-5) is extremely poor and non-scalable with the problem size. There may be two reasons for this: (1) TCP/IP is used instead of ATM's own protocol, so that high latency compounded by that of PVM dominate the overall communications performance; (2) no optimized (CPU chip-specific) BLAS library is utilized. The scalability of fill and LU parts are shown in Figure 4-5.

ScaLAPACK can make use of any system supplied BLAS routines that will help performance optimization at the node level. This is important since many of the recent parallel architectures employ vector operations at the node level, and processors such as the i860 employed by the Intel iPSC/860 is well endowed with vector registers and can carry out BLAS operations very efficiently. Furthermore, even on the super-scalar nodes used by machines like the IBM RS/6000 cluster, the BLAS operations provide a valuable level of abstraction for node level optimization.

5 Conclusion

Our work shows that both high portability and performance can be achieved in electromagnetics applications on distributed system by adopting a portable parallel linear algebra package such as BLACS/ScaLAPACK and minimizing the use of communication primitives. The BLACS in particular we found to be invaluable since it allowed us to design a degree of software portability into the MoM application code that would otherwise have been impossible.

Finally, we note the suitability of our parallel application for benchmarking purposes based upon its distinct but well characterised computational and memory access proprties.

Acknowledgements It is a pleasure to thank: J. Choi, J. J. Dongarra, A.Petitet and R.C.Whaley at UTK/ORNL for early access to ScaLAPACK/BLACS/PB-BLAS and for help and advice in employing it; X. Shen at NPAC for the CM5 implementation; and C. Cha and J. Lauer at SRC for useful discussions.

References

[1] E. Anderson, A. Benzoni, J. J. Dongarra, et al, *Basic Linear Algebra Communication Subprograms,* Proc. of the 6th Distributed Memory Computing Conference, 1991.

[2] G. Cheng, G.C. Fox, and K. A. Hawick, *A Scaleable Paradigm for Effectively-Dense Matrix Formulated Applications*, Proc. of the European Conference on High-Performance Computing and Networking, Munich, Germany, 1994.

[3] J. Choi, J. J. Dongarra, R. Pozo, and D. W. Walker, *Scalapack: A scalable linear algebra library for distributed memory concurrent computers*, Proc. of the 4th Symposium on the Frontiers of Massively Parallel Computation, pp. 120-127, 1992.

[4] J. Choi, J. J. Dongarra, and D. W. Walker, *PB-BLAS: A Set of Parallel Block Basic Linear Algebra Subprograms,* Proc. of the 1994 Scalable High Performance Computing Conference, May, 1994.

[5] T. Cwik, J. Patterson, D. Scott, "Electromagnetic Scattering Calculations on the Intel Touchstone Delta," Proc. of Supercomputing '92, IEEE Press, 1992, pp. 538-542.

[6] T. Cwik and J. Patterson (Eds.) *Progress in Electromagnetic Research: Computational Electromagnetics and Supercomputer Architecture*, EMW Publishing, Cambridge, MA, 1993.

[7] *Development and Implementation of Computational Electromagnetics Techniques on Massively Parallel Architectures*, Vol. 1-5, final project reports, SRC/NPAC, Syracuse, NY, 1994.

[8] J. J. Dongarra, R. A. van de Geijn and D. W. Walker, *A look at scalable dense linear algebra libraries*, Proc. of the Scalable High-Performance Computing Conference, pp. 372-379, 1992.

[9] J. J. Dongarra, R. A. van de Geijn and R. Clint Whaley, *A Users Guide to the BLACS*, Technical Report, ORNL/UTK, November 1993.

[10] *LAPACK Users' Guide*, E.Anderson, Z.Bai, C.Bischof, J.Demmel, J.Dongarra, J. Du Croz, A.Greenbaum, S.Hammarling, A.McKenney, S.Ostrouchov and D.Sorensen, SIAM, 1992.

[11] *Parametric Method of Moments (ParaMoM) RCS Prediction Packages, Version 1.0, User's Manual*, Syracuse Research Corporation, TD92-1321, October, 1992.

[12] S.M. Rao, *Electromagnetic Scattering and Radiation of Arbitrarily-Shaped Surfaces by Triangular Patch Modeling*, PhD Dissertation, University of Mississippi, August 1980.

TABLE 2

Timing Comparisons for $N = 4889$ *(in second)*

Platform	N_p	Fill	LU	LU(mflop)	Setup	RHS+Field	Total
Alpha(FDDI)	8	1420.0	1119.9	295.5	12.3	18.0	2570.2
IBM RS(Ethernet)	8	1500.7	1804.7	183.4	51.4	28.2	3385.0
Intel iPSC/860	64	525.6	280.9	1178.8	45.4	53.0	904.9
CM-5	32	3295.2	170.7	1938.8	21.1	4.3	3491.3

TABLE 3

Timing Comparisons for $N = 3060$ *(in second)*

Platform	N_p	Fill	LU	LU (mflop)
Alpha(FDDI)	8	450.4	713.5	91.6
IBM RS(Ethernet)	8	495.1	1210.3	54.0
Intel iPSC/860	16	830.9	175.6	372.1
CM-5	32	1207.0	79.5	821.9

TABLE 4

Timing Comparisons for $N = 1504$ *(in second)*

Platform	N_p	Fill	LU	LU (mflop)
Alpha(FDDI)	8	114.9	76.2	110.5
IBM RS(Ethernet)	8	127.1	131.8	63.9
Intel iPSC/860	8	478.6	61.5	136.9
CM-5	32	358.9	11.0	824.3
SUN(ATM)	2	2087.4	1941.4	4.3
Alpha(FDDI)	2	411.0	278.6	30.2
IBM RS(Ethernet)	2	475.6	262.0	32.1
Intel iPSC/860	2	1950.7	93.9	89.7

TABLE 5

Timings of Different N *on 2 SUN IPXs Interconnected by an ATM Link*

N	Fill	LU	LU (mflop)
1504	2087.4	1941.4	4.3
988	981.1	560.6	4.3
672	449.3	176.4	4.5
468	229.2	60.7	4.3
368	147.8	29.9	4.1
187	45.4	4.0	4.1

Fig. 4 Fill Time vs. Processors(Matrix Size=988)

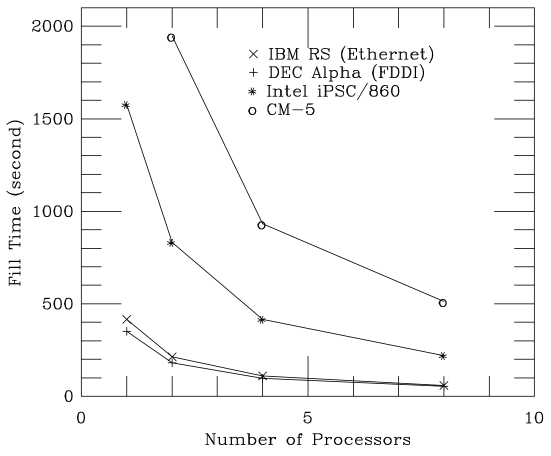

Fig. 5 LU Time vs. Processors(Matrix Size=988)