MEMO: May 14, 1993

TO: Geoffrey

FROM: Gang Cheng

RE: SRC CEM Proposal

CC:

Northeast Parallel Architectures Center

Applying high performance computing technologies to real world problems is the central mission of the Northeast Parallel Architectures Center(NPAC) at Syracuse University. NPAC, in collaboration with national research partners, engages interdisciplinary research, development, education and technology transfer in high performance computing. Core enabling technologies such as parallel algorithms, software systems, compilers, and databases are developed internally and integrated with related national research efforts, then applied in federally and privately funded research application projects by NPAC researchers.

The central mission of NPAC's ACTION program is to accelerate the use of high performance parallel computing and communications in industry. The outreach activities of the program result in close ties with industrial firms such as NYNEX/ New York Telephone, General Electric, IBM, UTC/Carrier, and SIAC. NPAC scientists have expertise and experience in large industrial application projects.

NPAC is a member of a prestigious Center for Research in Parallel Computation(-CRPC), one of the 11 original NSF centers in parallel processing. It is a research consortium of six participating institutions: Argonne National Laboratory, the California Institute of Technology, Los Alamos National Laboratory, Rice University, Syracuse University, and the University of Tennessee. CRPC is committed to a program of research that will make parallel computer systems truly usable - as usable as sequential computer systems are today. CRPC research is both interdisciplinary and interinstitutional. The strength of the research is in the combined efforts in the development of new software tools, new parallel algorithms, prototype implementations of scientific application programs, and shared resources among the six sites. Through our close collaboration with CRPC, NPAC can exploit the newest parallel algorithms for scientific computing and access to the most advanced parallel computing systems today.

The NPAC computational scientists will work together with SRC as a team to evaluate current CEM models, algorithms and numerical methods. The NPAC contribution to this project will be to 1) design and implement state-of-the-science

Gang Cheng
Northeast Parallel Architectures Center at Syracuse University
315-443-4636 gcheng@nova.npac.syr.edu

algorithms and parallel software in the project models, and 2) to provide parallel systems needed to support this project which include both in-house machines for intensive program development and remote-site machines for production run and evaluations.

1. The Application Problem and Computational Requirements

In the moment method which was first studied by R.F. Harlington, the electromagnetic component is modeled by a three-dimensional surface integral equation which formulates the electric current induced on the component's 3D surface. To solve the integral equation, a set of dense matrix equations are computed and its solution yields the radar cross section(RCS).

More specifically for our existing f77 code, the physical problem solved can be expressed as follows:

$$RCS = g(E^s(r), E^{inc}(r)) \quad \text{..........} \; (1)$$

where g is a quadratic function of E^s and E^{inc}, and E^{inc} is the known incident wave. $E^s(r)$ is the scattering electric field at point r, computed by

$$E^s = L(J) \qquad \text{..........} \; (2)$$

where L is a linear operator, and J is the induced electric current to be determined on the surface of the scattering component. The method of moment divides the surface into a collection of small curved triangle patches and uses the following approximation for J:

$$J = \sum_{i=1}^{N} I_i f_i \quad \text{.......} \; (3)$$

where f_i is the *ith* known expansion function and I_i is a unknown coefficient. Based on a boundary condition and a set of chosen testing functions *Ti (i=1,..., N)*, the unknown coefficient vector *I* is computed by solving the matrix equation:

$$[Z][I] = [V] \quad \text{.........} \; (4)$$

where V is an excitation vector and *[Z]* is a *NxN* matrix called impedance matrix. Their elements are determined by:

$$Z_{ij} = \int_s T(i)\, E_t^s(f_j)\, ds \quad \text{.........} \; (4)$$

$$V_i = \int_s T(i)\, E_t^{inc} ds \quad .... (5)$$

A major contribution of moment method is its capability to model the generally unconfined nature (open region) of electromagnetic wave and to give high accurate and stable solution, which has much advantages over other classical analytic methods such as finite element and finite difference. However, because of the nature of dense matrix with its size directly dependent on electrical size of the component, the storage and execution time requirements are proportional to the matrix size ***N***, and usually become the limiting factors in engineering applications. For our discussion of computational requirement and later the proposed approaches, we breakdown the existing code by time and storage complexity into three major steps(or components):

1. Matrix Fill

At this stage, the surface of the scattering component is discretized into small curved triangle patches and elements of the impedance matrix Z and excitation vector V are computed (by the equations (4) and (5) above).

Execution Time: $O(N^2)$ flops. More specifically for our fortran77 code, about $4000N^2$ flops

Storages: $O(N^2)$ bytes.

2. Matrix Equation Solve

A linear solve is used to solve the matrix equation (4) above. Due to LU decomposition used in the linear solver, it can be further breakdown into two stages:

1) *LU factorization*
Instead of solve (4) directly, Z is first factorized into two triangular matrixes $[Z] = [L][U]$.

Execution Time: $O(N^3)$ flops. For the existing fortran77 implementation, it is about $3N^3$ flops.

Storages: $O(N^2)$ bytes.

2) *Matrix Solve*

Forwardward and back-substitution are used respectively to obtain the final solution of the equation.

Execution Time: $O(N^2)$ flops.
Storages: $O(N^2)$ bytes.

3. Field Calculation

Final solution is obtained by finishing the computations of the above equations (3)(2)(1).

Execution Time: $O(N)$ flops.
Storages: $O(N)$ bytes.

Notice that for a filled matrix Z, the step 2 & 3 may be repeated for a number of times, corresponding to different (independent) excitation vector I. This, of course, increases the total execution time of the system.

Total Computational Requirements:

For Phrase 1 benchmark cases with matrix size *N=5,000 - N=6,000:*

Total Execution Time: *200 Gflops - 400 Gflops*, Matrix Fill(50%), LU Factorization(50%).
Total Storages(single precision): *250 Mbytes - 350 Mbytes*

For Phrase 2 benchmark cases with matrix size *N=50,000 - N=60,000:*

Total Execution Time: *385 Teraflops - 663 Teraflops,* Matrix Fill(3%), LU Factorization(97%)
Total Storages(single precision):25 Gbytes -35 Gbytes

2. Discussion of proposed technical approaches

2. 1. Summary of the approaches

Based on evaluation of the nature of this application problem, existing sequential code, its computational requirements and objectives of the proposed study, we plan to develop explicit message passing parallel implementation of the fortran77 model on MIMD (Multiple Instruction and Multiple Data) massively parallel systems. We choose to implement our CEM application on Connection Machine CM5, Intel Machines(Intel iPSC/860, Intel Touchstone Delta and Intel Paragon), and IBM SP-1(optional). Those machines represent the most advanced technologies in massively parallel computing industry. We will evaluate and demonstrate the MPP implementations of the advanced CEM techniques on the newest MPP architectures today.

At the most general level, we will evaluate the issues when implement the parallel model on those MPP systems. More specifically, we must address scalable data decomposition, stable and efficient parallel linear solver for large dense matrix, effective use of memory and optimized computation/communication in the system.

We choose to adopt in our implementation the parallel linear solver from the work of ScaLAPACK[1] developed at Oak Ridge National Laboratory by D. W. Walker's research group. Using this scalable and portable linear algebra package(LAPACK) for general distributed memory parallel systems, we can port our implementations on CM5, Intel Machines and IBM SP-1 with minimum effort. We also plan to use vendor-supplied linear algebra packages, i.e. CMSSL on CM5, ProSolver-DES on Intel MPP systems, to compare and achieve the most efficient dense matrix solver. To solve the storage limitation problem of large matrix size in Phrase 2, we will design an out-of-core scheme for our Phrase 2 algorithm design and implementation, which will make effective use of parallel file system and memory hierarchy of the target MPP system.

In Phrase 1, we will first convert the fortran77 code to CM5 in f77+CMMD(the CM5 message passing library) with matrix equation solver using ScaLAPACK. This then will be used as a working message passing implementation, based on which the next step is, simultaneously, 1) to implement a CMfortran + CMMD version in which the matrix equation solver using the Gaussian elimination routines provided by CMSSL; 2) to port it on Intel iPSC/860 in Fortran77 + NX(the Intel message passing library). Besides the ScaLAPACK version in the Fortran77+NX, we will also implement a Fortran77 + NX version using the ProSolver-DES for the dense matrix solver on Intel machines. As an option, we also plan to port the message passing implementation with ScaLAPACK on a IBM SP-1 system. After the implementation work, we will conduct benchmark experiments on those working machines with different sizes(32-node, 1024-node CM5s, 16-node, 32-node Intel iPSC/860s, 512-node Intel Touchstone Delta, Intel Paragon XP/S, and optional 8-node IBM SP-1), to evaluate scalabilty and performance results.

In Phrase 2, we will choose the best target machine, based on the evaluation of our Phrase 1's work in terms of model accuracy and stability, computational efficiency and projected scalability and performance for large full-scale modeling implementation. We plan to design out-of-core black algorithms using the parallel file system (e.g. DataVault on CM5, CFS on Intel iPSC/860 and Touchstone Delta,or PFS on Intel Paragon), as well as memory hierarchy of the target machine. In our implementation, a vender-supplied out-of-core linear solver (e.g., the out-of-core LU solver to be support in CMSSL 3.1, or the out-of-core ProSolver-DES on Intel MPP systems) will be adopted to achieve high performance.

2.2 Why message passing ?

Message passing allows programmer to have maximum flexibility and control in exploiting domain decomposition and control decomposition of the application problem so that maximum efficiency utilizing the high performance MPP hardware can be achieved by carefully design of explicit message passing parallel algorithms.

One of the most useful features of message passing programs is that they allow the processing nodes to synchronize as frequently or infrequently as required for a given application. A message-passing program is free to arbitrarily distribute program tasks and data among the nodes as necessary, using synchronization and data communication between nodes when it is necessary. This arbitrary load-balancing of tasks and data makes message-passing particularly useful for the CEM applications that require dynamic allocation of tasks and data.

In our CEM application, the matrix fill part happens to be a "loosely synchronous" problem which is spatially irregular but temporally regular. Moreover, driven by the major consideration of scalability and large dense matrix solver in algorithm design, MIMD implementation in explicit message passing paradigm appears to be the natural approach for this application. Further more, we believe MIMD architectures represent the promising future MPP technologies to be adapted in industry applications of high performance computing and communications(HPCC).

2.3 Algorithm Design Issues

In Phrase 1, based on the techniques employed in ScaLAPACK, we will first concentrate on developing a block domain decomposition scheme which should make the message passing algorithms for matrix fill, matrix equation solver and field calculation both scalable(variable problem size and machine size) and efficient.

Block algorithms, which update a block of contiguous vectors instead of one data element at a time, are the major techniques used in ScaLAPACK for LU factorization and matrix-matrix multiplication, and will be adapted in the algorithms for matrix fill and field calculation to have a scalabe and optimized implementation.

Programming in explicit message passing library, we will implement the block algorithms in an pipelined way such that calculations of matrix blocks local to a processing node is pipelined(overlapped) with data communications between nodes, and also pipelined are computations/communications among different stages in the block algorithms.

Through an optimized global algorithm design and implementation, we expect to achieve the most efficient use of working hardware, including the scalable architecture supporting high floating-point vector processing, high-bandwidth internal networks, and later through a out-of-core scheme, balanced parallel I/O, processing and memory bandwidth. We will evaluate the optimized block size of the LU factorization for the overall performance, given different benchmark cases.

We expect that the effort of the migration of Fortran77+CMMD to other MPP systems would be minimum, as the CMMD supports most commonly used message passing paradigms, including MIMD/SPMD distributed memory message passing, global and node-to-node communication, block, nonblock, and active message passing, host-node and hostless, multiple parallel I/O mode, interrupts and polling, etc., and most of the porting work will be confined in syntax while the maximum performance structure in the parallel algorithms can still be maintained.

In Phrase 2, using the required full-scale large model cases as testing demonstrations, those design and implementation issues in Phrase 1 will be revisited and further evaluated. An out-of-core block scheme and a vender supplied out-of-core linear solver will be incorporated into the block algorithm design and implementation. To effectively solve the limiting storage problem of large matrix case, this scheme makes use of target machine’s parallel I/O mass storage system as a 'Virtual Memory' in the memory hierarchy so that the disk-based parallel file system with extremely high storage capacity can be utilized in the algorithm design and implementation, and accessed through direct-connected parallel I/O channels with a sustained high rate. We expect this incorporation would be seamless and efficient, due to the block algorithms and pipelined techniques adopted in our

Phrase 1 approach.

In Phrase 2, we will also investigate more promising analytical techniques and/or software developed under the separate efforts, through our close relationship with ARPAR, NASA Goddard, NSF, CRPC, NASA NRA, and other industry partners.

2.4 Description of tools to be used in this project

1). ScaLAPACK

ScaLAPACK is a distributed memory version of the LAPACK[5][6] software package for dense and banded matrix computations[1][2][3]It is designed to be scalable, portable, flexible and ease of use on MIMD parallel computers[1][2][3][4], is developed at Oak Ridge National Laboratory / University of Tennessee by a research group headed by Jack Dongarra and David W. Walker. Using distributed versions of the Level 3 BLAS[4] as building blocks, a square block scattered decomposition scheme and an object-based interface to the ScaLAPACK routines, this package takes advantage of surface-to-volume effects and allow effective reuse of data in local memory. With other techniques such as a two-dimensional data mapping, increasing the granularity of the algorithms, overlapping communication and calculation, and maximizing the size of submatrices multiplied in each processor, this approach can lead to optimal algorithms of linear solvers on distributed-memory machines. Portability is achieved through a collection of routines that are portable across a wide range of architectures without sacrificing performance.

2). CMMD 3.0 and NX

CMMD is a library of message-passing routines for the CM5. Programs that use CMMD typically operate in a message-passing style such that each of the processing nodes of the CM5 runs an independent copy of a single program and manages its own computations and data layout, and communication between nodes is handled by calls to CMMD message passing functions. CMMD supports a variety of message passing styles, including point-point message, cooperative message passing, asynchronous message passing, virtual communication channels, global operations, interrupts and polling, and active message(CMAML).

In addition, CMMD programs written in data parallel languages such as CMFortran use both the microprocessor and the vector units. The parallel language provides the concept of a “node-level parallel” array, in which a parallel array is stored on a single node, with its values spread across the memory of the node’s four vector units, thus can take advantages of the VUs for high-speed float-point computations.

NX is the message passing library used for communication among processes in an Intel MPP system. As independent processor/memory pairs, the nodes do not share physical memory. Even node processes running on the same node maintain distinct memory spaces. If the node processes need to communicate, they do so via NX message passing calls. NX supports common message passing paradigms, including synchronous/asynchronous message passing, interruption message passing and global operations.

3). CMSSL 3.1 and ProSolver/DES

The Connection Machine Scientific Software Library(CMSSL) is a set of numerical routines that support computational applications while exploiting the massive parallelism of the CM system. CMSSL provides data parallel implementations of familiar numerical routines such as linear algebra, providing new solutions to problems of both performance and algorithm choice and design. Many of the routines have been implemented to allow parallel computation on either multiple independent objects or a single large object. For dense matrices, CMSSL offers two methods of solving a linear system represented as a real or complex general matrix. The first method uses the Gauss-Jordan linear system solver, while the second method uses the QR factorization operation in combination with the QR solver. A out-of-core implementation of LU and QR solvers will be released in CMSSL 3.1 in June, 1993 by Thinking Machines Co..

ProSolver-DES is a dense solver for the Intel iPSC/860 and Paragon-XP/S, with both in-core and out-of-core implementations. It is a library for solving large, dense and/or disk resident, systems of linear equations, AX=B. ProSolver-DES implements a variant of block Gaussian elimination to compute the block LU factorization of A and then uses block forward and back substitution to solve for the unknown matrix X. ProSolver-DES was carefully optimized for high performance on iPSC/860. In the out-of-core implementation, asynchronous I/O is used to overlap most of the I/O time with computation and the maximum problem size which can be solved by ProSolver-DES depends on the size of the file system.

3. NPAC's Capabilities and Relevant Experience

3.1 MPP machines

. NPAC's 32-node CM5, 16-node Intel iPSC/860, 8-node IBM SP-1 for development in Phrase 1 & 2

. CRPC facilities:

. Rice Univ. 32-node Intel iPSC/860

. CalTech 512-node Intel Touchstone Delta

. Los Alamos National Laboratory or Pittsburgh Supercomputing Center 1024-node CM5

. Oak Ridge National Laboratory Intel Paragon XP/S (through CRPC's participation in Concurrent Supercomputing Consortium)

3.2 NPAC's relevant experience (to be elaborated ...)

. NPAC's Action Program, industry and application projects
. in-house 32-node CM5 for phrase 1, 16-node Intel iPSC/860 for Phrase 2.
. Block LU project, EMS projects funded by DARPA, AVS+EMS work(a heterogeneous approach)
. NPAC's relationship with CRPC (1024-node CM5, relationship with Jack Dongarra's group at Argonne National Lab., and JPL)
. NPAC's collaboration with Dr. Harlington's group

4. References (to be added ...)

[1] J. Choi, J.J. Dongarra, R. Pozo and D. W. Walker, "*ScaLAPACK: A Scalable Linear Algebra Library for Distributed Memory Concurrent Computers,*" Proceedings of 4th Symposium on the Frontiers of Massively Parallel Computation (McLean, Virginia, October 19-21, 1992), IEEE Computer Society Press, Los Alamitos, California, 1992, pp. 120-127.

[2] J. Choi, J.J. Dongarra, and D. W. Walker,"*The Design of Scalable Software Libraries for Distributed Memory Concurrent Computers,*" CNRS-NSF Workshop on Environments and Tools for Parallel Scientific Computing (Saint Hilaire du Touvet, France, Sept. 7-8, 1992), Elsevier Science Publishers, 1992.

[3] J. Demmel, J.J. Dongarra, R. van de Geijn, and D. W. Walker, "*LAPACK for Distributed Memory Architectures: The Next Generation,*" Proceeding of 6th SIAM Conference on Parallel Processing for Scientific Computing(Norfolk, Virginia, March 22-24, 1993) (R. F. Sincovec, eds.), SIAM, Philadephia, 1993.

[4] J. Choi, J.J. Dongarra, and D. W. Walker, "*Level 3 BLAS for Distributed Memory Concurrent Computers,*" CNRS-NSF Workshop on Environments and Tools for Parallel Scientific Computing (Saint Hilaire du Touvet, France, Sept. 7-8, 1992), Elsevier Science Publishers, 1992.

[5] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. J. Dongarra, J. DuCroz, A.

Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *"Lapack: A Portable Linear Algebra Library for High Performance Computers*," Proceeding of Supercomputing '90, pp. 1-10, IEEE Press, 1990.

[6] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK Users' Guide*, SIAM Press, Philadephia, 1991.

[7] G. C. Fox, S. W. Otto, *Matrix Algorithms on a Hypercube I: Matrix Multiplication*, Parallel Computing, Vol. 4, pp. 17-31, 1987.

[8] G. C. Fox, M. A. Johnson, G. A. Lyzenga, S. W. Otto, J. K. Salmon, and D. W. Walker, *Solving Problems on Concurrent Processors*, Volume 1, Prentice Hall, Englewood Cliffs, N.J., 1988.

[9] G. Laszewski, M. Parashar, A. G. Mohamed, G. Fox, "*On the Parallelization of Blocked LU Factorization Algorithms on Distributed Memory Architectures,*" Proceedings of Supercomputing '92, IEEE Press, 1992.

[10] G. C. Fox, "*The Architecture of Problems and Portable Parallel Software Systems*," SCCS technical Report 134, also CRPC - TR91172, 1991.

[11] G. Fox, "*Parallel Computing in Industry: An Initial Survey*", in Proc. of Fifth Australian Supercomputing Conference, World Congress Centre, Melbourne, Australia, December, 1992.

[12] R. F. Harrington, *Field Computation by Moment Methods*, the Macmillan Co., New York (1968). Reprinted by Krieger Publishing Co., Malabar, FL (1982).

[13] G. Cheng, K. Mills and G. Fox, "*An Interactive Visualization Environment for Financial Modeling on Heterogeneous Computing Systems*," Proceeding of 6th SIAM Conference on Parallel Processing for Scientific Computing(Norfolk, Virginia, March 22-24, 1993) (R. F. Sincovec, eds.), SIAM, Philadephia, 1993.

[14] Y. Lu, A. G. Mohamed, G. Fox and R. F. Harrington, "*Implementation of Electromagnetic Scattering from Conductors Containing Loaded Slots on the Connection Machine CM-2*," Proceeding of 6th SIAM Conference on Parallel Processing for Scientific Computing(Norfolk, Virginia, March 22-24, 1993) (R. F. Sincovec, eds.), SIAM, Philadephia, 1993.

[15] G. A. Mohmad, G. Fox, G. Laszewski, M. Parashar, T. Haupt, K. Mills, Y. Lu, N. Lin, and N. Yeh, "*Applications Benchmarking Set for Fortran-D and*

High Performance Fortran," Technical Report, Syracuse Center for Computational Science 327, June, 1992, also CRPC-TR92260.

[16] G. Cheng, Y. Lu, G.C. Fox, K. Mills and T. Haupt, "*An Interactive Remote Visualization Environment for an Electromagnetic Scattering Simulation on a High Performance Computing System*," submitted to Supercomputing '93, Portland, OR, Nov. 14-17, 1993.(in review).