LA-UR 97-1031

CONF-971005--14

Approved for public release; distribution is unlimited

Title: The DANTE Boltzmann Transport Solve: An Unstructured Mesh, 3-D, Spherical Harmonics Algorithm Compatible with Parallel Computer Architectures

Author(s): John M. McGhee
Randy M. Roberts
Jim E. Morel

Submitted to: Joint International Conference on Mathematical Methods and Supercomputing for Nuclear Applications
Saratoga Springs, New York
October 5–10, 1997

MASTER

Los Alamos
National Laboratory

The DANTE Boltzmann Transport Solver: An Unstructured Mesh, 3-D, Spherical Harmonics Algorithm Compatible with Parallel Computer Architectures

J. M. McGhee, R. M. Roberts, and J.E. Morel

Principal Author: J. M. McGhee
Scientific Computing Group, CIC-19
MS B265, Los Alamos National Laboratory
Los Alamos, New Mexico 87545
Phone: 505-667-9552
Fax: 505-665-5220
E-mail: mcghee@lanl.gov

Submitted to:
Joint International Conference on Mathematical Methods
and Supercomputing for Nuclear Applications
Saratoga Springs, New York, October 5-10, 1997

Session:
Methods for Parallelization of Transport
and Diffusion Calculations
(Invited Paper)

March 14, 1997

The DANTE Boltzmann Transport Solver: An Unstructured Mesh, 3-D, Spherical Harmonics Algorithm Compatible with Parallel Computer Architectures

J. M. McGhee, R. M. Roberts, and J.E. Morel
Radiation Transport Team
Scientific Computing Group
Los Alamos National Laboratory
Los Alamos, New Mexico 87545

Abstract

A spherical harmonics research code (DANTE) has been developed which is compatible with parallel computer architectures. DANTE provides 3-D, multi-material, deterministic, transport capabilities using an arbitrary finite element mesh. The linearized Boltzmann transport equation is solved in a second order self-adjoint form utilizing a Galerkin finite element spatial differencing scheme. The core solver utilizes a preconditioned conjugate gradient algorithm. Other distinguishing features of the code include options for discrete-ordinates and simplified spherical harmonics angular differencing, an exact Marshak boundary treatment for arbitrarily oriented boundary faces, in-line matrix construction techniques to minimize memory consumption, and an effective diffusion based preconditioner for scattering dominated problems. Algorithmic efficiency is demonstrated for a massively parallel SIMD architecture (CM-5), and compatibility with MPP multiprocessor platforms or workstation clusters is anticipated.

I. INTRODUCTION

The Boltzmann Transport Equation describes the transport of particles (neutrons, photons, electrons, ions, etc.) through space and matter. The numerical solution of this important equation is an essential modeling tool in a wide variety of fields. Among these are nuclear reactor design and analysis, inertial confinement fusion, criticality safety, medical therapy and imaging, nuclear weapons safety, and oil well logging. All existing methods for solving the transport equation can be broadly characterized as either stochastic or deterministic. Stochastic or Monte Carlo methods are easily applied in complex 3D geometries, but statistical errors can result in excessive run times for many classes of problems. Deterministic methods do not suffer from statistical error but rather from discretization error. The most popular deterministic method in use today is the discrete ordinates or S_n method. This method has proven to be extremely useful for nuclear reactor analysis, but for a certain class of problems it exhibits a serious lack of rotational invariance commonly known as ray effects. Thus, even though the S_n method is widely used in neutron transport it is unsuitable for many other applications. The only deterministic transport method which rigorously preserves the rotational invariance of the transport operator is the spherical harmonics or P_n method. This is one of the oldest methods for solving the transport equation. However, P_n methods have historically suffered a lack of attention relative to S_n methods, primarily because the S_n equations were much easier to solve on the serial machines of earlier years. However, the advent of efficient Krylov space solvers, ever more powerful microprocessors, and parallel computer architectures promises a vastly increased solution efficiency. With these facts in mind, development of a new P_n algorithm designed specifically for compatibility with parallel computer architectures was undertaken at Los Alamos National Laboratory.

I.A. The DANTE Research and Development Code

DANTE is a deterministic Boltzmann transport solver developed by the Los Alamos National Laboratory (LANL) Scientific Computing Group (CIC-19). Work on the project began in 1994, as part of a continuing research and development effort in 3D parallel particle transport algorithms. The initial versions of the code were written in CM-Fortran for the massively parallel CM-5 computer vended by Thinking Machines Corporation. The code presently consists of approximately 50,000 lines of Fortran 90. Communication is effected through a MPI based communications library provided by a third party vendor. DANTE solves the transport equation in a self-adjoint, second order form. The code utilizes a multi-group energy treatment, and is capable of time-dependent and steady state calculations for both neutrons and photons. The method of solution is standard source iteration. Fully compatible acceleration techniques are provided for the acceleration of scattering and emission/fission sources. Other significant features of the code include:

- Spatial differencing is accomplished on an arbitrarily connected mesh using a Galerkin finite element technique[5]. Options are provided for lumped (i.e. single point) source and removal terms for stability in thick problems. Options are available for a variety of finite elements in one, two and three dimensions. Additional elements can be added with a minimum of effort, if desired.
- Options exist for discrete-ordinates (S_n, simplified spherical harmonics (SP_n), and spherical harmonics (P_n) angular discretization. Options are also available for even-parity, odd-parity, or simultaneous even-parity and odd-parity solution modes. The P_n and SP_n angular differencing methods provide rotationally invariant solutions free of the "ray effects" sometimes associated with discrete ordinates codes.
- For the S_n and SP_n modes, a standard within group source iteration technique is used to implement arbitrary order anisotropic scatter. A fully compatible diffusion synthetic accelerator (DSA) is included. No within-group scattering source iteration is required in the P_n angular differencing mode, as arbitrary order anisotropic scatter is provided through direct inclusion of the scattering terms in the transport operator.
- For the P_n method, the constants appearing in the spatially-continuous spherical harmonics equations are efficiently generated in-line via an exact quadrature integration of products of the spherical harmonics functions.
- For the P_n method, an efficient Marshak boundary condition treatment is provided through an exact numerical quadrature integration procedure. This treatment can be applied on arbitrarily oriented boundary faces.
- Solution of the transport operator is effected by a preconditioned conjugate gradient algorithm[3]. Options are provided for both a simple point Jacobi preconditioner, and a more sophisticated, diffusion based, block diagonal preconditioner that is particularly effective for optically thick, scattering dominated problems.
- An in-line matrix construction option is provided that greatly reduces the memory required to store the P_n transport operator, at the cost of a moderate (50%) increase in run-time.
- Fission, emission, and down-scatter sources are provided. Fission and emission sources are included using a standard source iteration procedure. A fully compatible, diffusion based accelerator is provided for both fission and emission sources.
- Operator construction and solution techniques are tailored for parallel machine architectures. Data layout is designed to take advantage of mesh partitioning algorithms, which can greatly reduce inter-processor communication costs, thus improving run time performance.

I.B. Paper Organization

Although DANTE has options for S_n and and SP_n angular differencing, it was designed primarily as a P_n code. This article will focus on the parallel implementation of the 3D, even-parity P_n angular differencing technique in DANTE. The remainder of this article is organized as follows. A Parallelization Principals section discusses the general philosophies of parallelization behind the DANTE algorithm. A Numerical Implementation section discusses the application of these principals to the DANTE code and develops a computational model. Finally, a Computational Results section presents results of the implementation on the CM-5 parallel computer.

II. PARALLELIZATION PRINCIPALS

Today's computing environment is vastly different from the environment of even 10 years ago. Rapid improvements in workstation and networking capabilities have produced desktop machines that rival the capabilities of yesterday's mainframe supercomputers. Massive parallelization via a networked collection of "off the shelf" desktop computers is now widely believed to be a pathway to ultra high performance. This philosophy is reflected in recent computer purchases at both Lawrence Livermore National Laboratory (LLNL) and LANL for the US Department of Energy Advanced Super-computing Initiative program (ASCI). These architectures (and their anticipated follow ons) can be characterized as distributed memory massively parallel processors (MPP's) with peak network bandwidths on the order of 80 MBytes/s-node or more and powerful processing nodes capable of 250 Mflops/node. The LANL Scientific Computing Group is tasked with performing transport calculations on these machines. A typical problem might consist of order one hundred thousand spatial cells, expected to run efficiently on a configuration with hundreds of processing nodes. An efficient parallel algorithm design is expected to distribute arithmetic work evenly among all available processors, and also to minimize interprocessor communication. In general terms then, our objective is to develop efficient parallel algorithms that are also portable, easily maintained and debugged, and easily modified. In support of this objective we have adopted several general principals which are discussed below.

Parallelize primarily over the spatial axis. Typical transport problems of interest at LANL represent a phase space characterized by order 10^6 spatial cells, 10^2 angles, and 10^1 energy groups. For these problems, we find that dividing the number of spatial cells evenly among all the processing nodes usually provides more than enough computational work to keep all nodes busy, even for modest problems sizes. Moreover, we anticipate future growth in both the number of processors available, and the number of spatial cells in problems of interest. We therefore elect to use a fine grained parallelization over the spatial axis as our primary means of parallelization. This strategy is compatible with current machine/problem configurations, and maintains parallel scalability with anticipated future machine configurations and problem sizes. Other coarse grained parallelizations, i.e. over angle and/or energy can provide valuable parallelization opportunities, and their importance should not be discounted; however, for our purposes we feel that they should remain secondary to spatial parallelization. Moreover, if the processors are already saturated, additional parallelization over that provided in the spatial axis is of no benefit and may complicate the code unnecessarily.

Minimize inter-processor communication via an efficient distribution of data among processors. Interprocessor communication represents an additional work load for parallel machines that is not present on a serial machine. In addition, the cost of communication is, in general, relatively expensive time-wise compared to an on-chip arithmetic computation. As a result, parallel efficiency demands that interprocessor communication be kept to a minimum.

Identify repeated communication patterns, perform a setup for these patterns once, and amortize the cost of this setup over many communication calls. The cost of a communication operation can be divided into two parts or phases. One, the creation of a communication "pattern" that is independent of the data being transmitted, and two, the actual transmission of the data. In the case of an algorithm that repeatedly reuses the same "pattern", the setup phase can be performed once, the results stored, and the setup cost amortized over the subsequent data transmissions. This can result in significant overall savings in communication time.

Isolate communication operations from computation operations for maximum efficiency of both. Optimizing compliers generally require large code blocks for maximum efficiency. The presence of calls to communication primitives in a code block can severely inhibit automatic compiler optimizations. In addition, there is usually a latency or minimum cost per call associated with a communication operation. In order to minimize these effects, we attempt to isolate communication operations and arithmetic operations into large independent blocks. This strategy facilitates the construction of large messages, which minimize network latency effects, and presents the compiler with relatively large code blocks that are more easily optimized for arithmetic efficiency.

Minimize computational workload by utilizing an arbitrary mesh capable of accurately modeling complex geometries with a minimum number of spatial cells. Real world problems often have complicated geometries that can be very expensive to accurately represent with structured grids. Since the required computational work is proportional to the number of computational cells in a problem, there is a strong incentive to minimize the number of cells required for an accurate solution. We elect to use an unstructured computational mesh in order to achieve this goal. Using an unstructured approach, we can add resolution as required in any area of the problem without affecting the mesh elsewhere. This results in considerable savings in total cells required verses a structured mesh approach.

Enhance portability by using a standard programming language and by isolating communication functions in a separate library which can be replaced or upgraded with a minimum of impact on the rest of the code. We attempt to maximize portability by using standard language features wherever possible. However, interprocessor communication routines require functionality which are not provided by modern programming languages. As a result, interprocessor communication often seems to be the stumbling block in terms of portability, and often has a strong impact on performance as well. By isolating all communication primitives in their own subroutines/library, it is possible to port a code by simply porting the communication library. If more than one code is utilizing the same library, the cost of porting the library can be amortized over all the users. Likewise, if a substitution of communication subroutines appropriate to the machine at hand is required, then this is most easily accomplished if the communication routines are isolated in their own library. This approach also facilitates the rapid evaluation of multiple communication libraries.

We find these principles to be useful in a general sense in that they can be utilized in the design and optimization of any parallel algorithm. We now describe their application to the development of the DANTE transport code.

III. NUMERICAL IMPLEMENTATION

This section provides a brief discussion of some of the more significant details of the actual numerical implementation of the DANTE algorithm.

III.A. Fundamental Operations

We begin by identifying the fundamental operations contained in the DANTE algorithm. As previously mentioned, DANTE uses a conjugate gradient algorithm to effect a solution of the transport equation. The majority of the computational work in a conjugate gradient algorithm consists of a matrix-vector multiplication, and therefore we focus on this operation for purposes of solver optimization. An additional consideration arises due to the large size of the P_n matrix. In order to conserve memory we elect to generate this matrix piecewise during the course of each matrix-vector multiply. For large problems, we find that these two operations account for over 90% of the total solution time. Optimization of the DANTE solution algorithm can therefore be considered to be the optimization of two process: in-line matrix generation, and a matrix vector multiply. Both of these processes exhibit a great deal of data parallelism, and we utilize this fact in the design of our algorithm.

III.B. Data Layout and Mesh Partition

For a typical data array in DANTE we distribute information across the available processors in one of two ways: cell-wise or node-wise. The majority of communication that occurs in DANTE is thus between cells and nodes. By placing cells and nodes that are closely positioned in space on the same processor, the amount of data that must be moved between processors is minimized. There are many readily available mesh partitioners that can efficiently accomplish this ordering. On the CM-5 we utilize the spectral-bisection partitioner available in the CMSSL[2]. On other machines we have found the METIS partitioner[4], available from University of Minnesota, to be extremely efficient. Since the majority of communication operations are based on a cell to node, or node to cell pattern, a single reordering optimizes communication for all important operations in the generation and solution of the transport matrix.

III.C. Matrix Generation

DANTE uses a standard finite element Galerkin procedure for spatial discretization. This procedure places the primary unknowns (the moments of the angular flux) on the mesh nodes, and produces a large, sparse, SPD matrix. If the unknowns are arranged into a vector which varies most rapidly in the spatial index, then the complete transport matrix can be viewed as a (n x n) block matrix where n is the number of moments in the P_n expansion. Each block has an identical non-zero pattern. For example, the even parity, 3D, P_3 matrix has 6 even moments on each node, and the P_3 matrix can be viewed as a 6x6 block matrix where each block has the same non-zero pattern. In order to conserve memory, we construct this matrix one block at a time.

The data required to construct a block of the transport matrix consist of cell geometric characteristics, cell material properties, and moment-to-moment coupling constants. By storing all the data associated with a particular

cell as a unit, and then distributing these units over the processor space, the cell by cell contributions to the transport matrix can be generated without communication, and the arithmetic workload of matrix generation is distributed evenly among the processors. Complete in-line matrix construction from scratch using standard finite element integration rules was found to be too expensive for our purposes. As a compromise, the construction of the Hessian matrix required by the finite element treatment is carried out once at setup time, and stored for future use. The Hessian matrix is dependent only on the cell geometry and is the same for each block in the system. The ready availability of the Hessian matrix reduces the amount of work required to generate the matrix elements to, what we consider to be, a more reasonable level. Using this approach, matrix generation on large problems typically accounts for about 50% of the total solution time.

At this point a conventional approach would require the assembly of the cell-by-cell contributions to the transport matrix into a final compact matrix form. However, this step entails interprocessor communication, and we elect to avoid this communication by leaving the matrix in its unassembled form. The trade off is, that as a result, there is more communication and math to perform during the matrix vector multiply. Whether or not this is a winning strategy depends on the relative cost of computation verses communication, which is platform dependent. We plan future experiments to investigate this issue on a variety of platforms. While there is considerable room for "fine-tuning" in this area, we do not expect the resolution of this question to seriously impact the overall utility of the method.

III.D. Matrix Vector Multiply

Having completed our discussion of matrix construction, we turn now to the actual matrix-vector multiply. This is accomplished as a five stage procedure. First, a vector gather is performed that copies the complete moment vector from each mesh node to its corresponding cell vertex. This node-to-cell gather is followed by the construction of a single block of the transport matrix, as described in the previous section. Third, the matrix construction is followed by a (nvrtx x nvrtx) matrix vector multiply on each cell, where nvrtx is the number of vertexes on a cell. The construction and multiplication steps are then repeated for each block in the transport matrix. (36 blocks for P_3). Fourth, the results of the construction-multiplication step are scattered-with-add from the cell vertexes to the mesh nodes. Finally, to complete the matrix vector multiply, boundary conditions accommodated in a follow on procedure similar to the previous four steps. Memory usage is minimized, as storage is required for only one block of the transport matrix at any given time.

III.E. Computational Model

We have constructed a computational model to predict the performance of the DANTE solver. Analysis based on this model is useful in predicting, understanding, and optimizing the solution algorithm. The model is somewhat crude, but can still provide useful information. We begin the description of our computational model with the following definitions:

- ncells = number of cells in the problem
- p = number of processors
- nvrtx = number of vertexes on a cell
- nvrpf = number of vertexes on a cell face
- tmath = time to perform a floating point operation (s/flop)
- tcomm = time to perform an interprocessor tcomm = (tdata nwords) + tlat (s)
- nwords = number of words in a interprocessor message
- tdata = interprocessor data transfer coefficient (s/word)
- tlat = interprocessor communication latency (s)
- nblocks = number of blocks in the transport matrix

n	1	3	5	7	9	11	13	15
nblocks	1	36	175	434	813	1312	1931	2570

Table I: Number of moment-to-moment coupling blocks in a 3D even-parity P_n matrix

- n = P_n order
- nevmo = number of even moments
- ncgiter = number of conjugate gradient iterations

The number of even moments for a P_n calculation is given by the following formula:

$$\text{nevmo} = n(n+1)/2 \tag{1}$$

The total number of blocks in the P_n matrix increases rapidly with increasing P_n order, approaching (30 nevmo) for large n. The actual number of blocks for odd values of n less than 15 is presented in Table I. The time required to perform the math in a single matrix construction varies somewhat from one P_n order to another, but can be approximated as:

$$\text{tmatrix} = \text{nblocks tmath ncells 5 } (\text{nvrtx}^2)/p \tag{2}$$

The time required for a single interprocessor gather and scatter operations can be estimated as:

$$\text{tgather} = \text{tscatter} = \text{nmom (tdata 3 nvrpf } (\text{ncells/p})^{2/3} + \text{tlat}) \tag{3}$$

The time required for the math in a single matrix-vector multiply is:

$$\text{tmvm\_math} = \text{nblocks tmath 2 ncells } (\text{nvrtx}^2)/\text{p} \tag{4}$$

and the total time for solution is estimated as:

$$\text{t} = \text{ncgiter (tmatrix} + \text{tgather} + \text{tscatter} + \text{tmvm\_math}) \tag{5}$$

IV. COMPUTATIONAL RESULTS

In order to investigate the parallel performance of the DANTE transport algorithm, we constructed a benchmark test suite consisting of a series of computational timing studies. Timing data was obtained for a simple one-group, steady state problem while varying both the number of spatial cells and the number of processing nodes.

In this section we describe results from this series of test problems. We begin with a brief description of the platforms used in the benchmark.

IV.A. CM-5 Description

As a representative distributed memory parallel platform, we selected the CM-5, vended by Thinking Machines Corporation (TMC). Our team has ready access to such a machine at Los Alamos National Laboratory. The Los Alamos National Laboratory CM-5 consists of 1024 processing nodes, each having a maximum memory of 32 Mbytes. Each processing node is comprised of a SPARC CPU and four vector processors. The nodes are connected by a "fat tree" data network with a maximum bandwidth of 5 MBytes/s for general communication. Users access the machine via one of several "partition managers" which provide access to 32, 64, 128, 256, or 512 node partitions. A more complete description of the CM-5 is available in the CM-5 Technical Summary[1]. For this benchmark, we operate the CM-5 in the data parallel SIMD mode, and utilize the "*partition_vector_gather/scatter*" utilities from the TMC supplied CMSSL[2] for interprocessor communication.

Mesh ID	Points	Tets	a (cm)	a (mfp)
con_test0	71	192	35.35	2.000
con_test1	429	1536	17.68	1.000
con_test2	2969	12228	8.85	0.500
con_test3	22065	98304	4.42	0.250

Table II: Test Mesh Descriptions. A single material cube 100 cm on edge. "a" is defined as the mesh average tet edge length.

Mesh ID	IBM 590, 1 node	CM-5, 32 nodes	CM-5, 64 nodes	CM-5, 128 nodes	CM-5, 256 nodes
con_test0	0.08	0.75	1.03	1.16	–
con_test1	1.08	1.11	1.24	1.39	–
con_test2	17.31	2.70	2.55	2.58	2.72
con_test3	271.62	20.06	12.47	8.33	6.73

Table III: Benchmark Problem Timing Results (s)

IV.B. IBM Description

For purposes of comparison with the CM-5, we selected a IBM 590/RS6000 workstation as a representative serial machine. The IBM 590 is a modern, cache based super-scalar workstation with a SPECfp92 flop rate of approximately 250 Mflops. Additional information on the IBM 590 system architecture can be found in the IBM 590 user documentation[6]. The machine used in our benchmark was equipped with 512 Mbytes of memory, which was sufficient to prevent any significant swapping for the benchmark problem.

IV.C. Test Problem Description

The benchmark problem consisted of a single material cube 100 cm on edge with a uniform isotropic distributed source and vacuum boundary conditions. The source strength was set to 1.0 ($p/cm^3 - s$). The total cross section was set at $5.657 \times 10^{-2}\ cm^{-1}$. Scattering was isotropic, with a value of $2.8285 \times 10^{-2}\ cm^{-1}$. These opacities resulted in a problem mean free path of approximately 17.7 cm making the problem approximately 5.66 mfp on edge.The problem was discretized using four meshes, as shown in Table II. The meshes were essentially identical in all respects other than tetrahedra dimensions. All tetrahedra were uniform and well shaped. Testing was performed using a early, one-group version of DANTE limited to linear tetrahedral finite element meshes. The DANTE code was compiled on both the IBM 590 and the CM-5 with compiler optimization switches activated. Run times were measured on the four meshes for four different machine sizes: 32, 64, 128, and 256 processing nodes. All runs were made in the even-parity mode with P_n order set to three (P_3). Timing results were obtained using the "*cmf_timer*" utility on the CM-5 and the "*etime*" utility on the IBM. Wall clock times were also recorded as an additional reference. Although neither machine was "dry" (free from other users) during the testing, this was not believed to significantly affect the results, as wall clock and CPU times were observed to be nearly identical.

IV.D. Results

Results from our timing runs are presented in Table III. A plot of the relative performance of the IBM 590 verses the 32 node CM-5 is presented in Figure 1. This figure dramatically demonstrates the startling reduction in run time that can be achieved through parallel algorithms.

Based on the timing data, we estimate the following effective parameters for the CM-5, per processing node:

- tlat = 5000 μs
- tdat = 1.8 μs/8-byte-word (approximately 4.6 Mbytes/s, 90% peak)
- tmath = 0.021 μs/flop (approximately 48 Mflops, 38% peak)

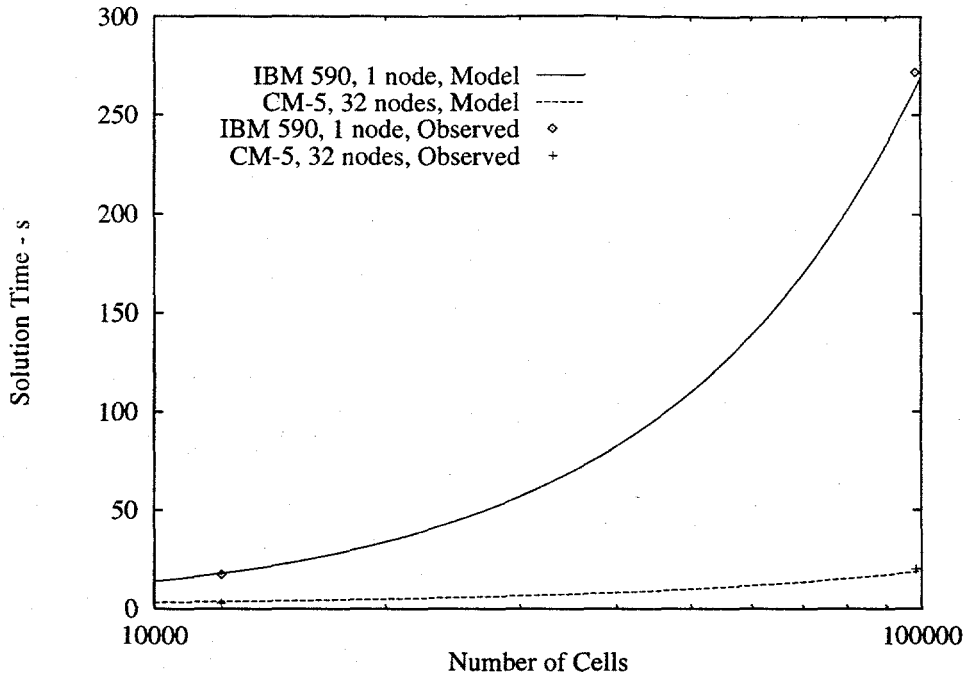

Figure 1: CM-5 Solution Time verses IBM 590 Solution Time.

We also observe an additional "unidentified serial time" of 0.08 seconds per conjugate gradient call. This time does not seem to be a strong function of either the problem size or the number of processors, but depends only on the number of conjugate gradient iterations. We account for this time in our model by adding a factor of (+0.08 ncgiters) to the total time predicted for a CM-5 calculation. This time is insignificant for large problems.

Using the above parameters, we plot, in Figure 2, the performance predicted by our model against the observed timings. Considering the crudeness of the computational model, we find the agreement between model and experiment to be very reasonable.

Figure 3 displays the relative importance of the different components of our computational model to the total solution time. For large problems the computational time is dominated by the generation of the transport matrix.

A plot of the relative parallel speedup predicted by our model is presented in Figure 4. The serial reference time used was the predicted DANTE solution time for a single CM-5 processing node, without communication. A speed up of 1.0 would represent perfect parallel efficiency, a speed up equal to the number of processors. The efficiency of the CM-5 is reduced for small problems sizes, primarily as a result of the previously mentioned additional "unidentified serial time". For comparison, we also present the predicted performance of our algorithm on a IBM 590/SP2 cluster. Parallel efficiency on this network is predicted to be much higher. For the IBM 590/SP2 combination, the following values are considered to be representative and were used in the model:

- tlat = 75. μs
- tdat = 0.33 μs/8-byte-word (approximately 24 Mbytes/s, 30% peak)
- tmath = 0.013 μs/flop (approximately 80 Mflops, 30% peak)

As a point of comparison with a conventional first order code, we compare DANTE on the IBM 590 with our unstructured tet mesh code ATTILA[7] in Figure 5. This comparison indicates that, for the benchmark problems, P_n solutions can be obtained for costs similar to that of standard S_n solutions . We note however that the solution time for the P_n algorithm is a function of the cell optical thickness.

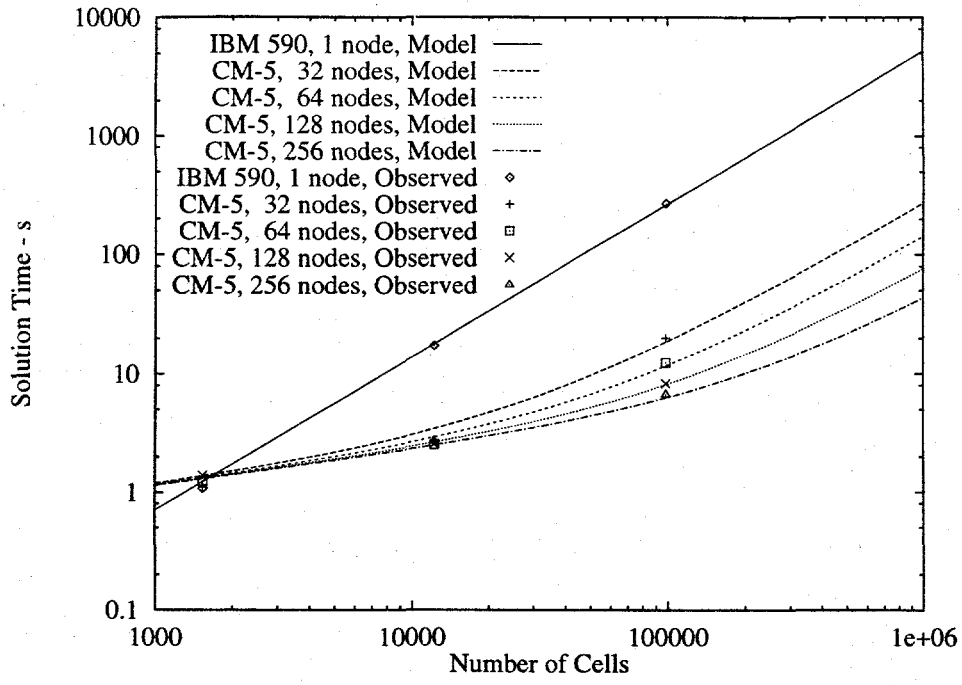

Figure 2: Comparison of CM-5 computational model predictions with numerical results.

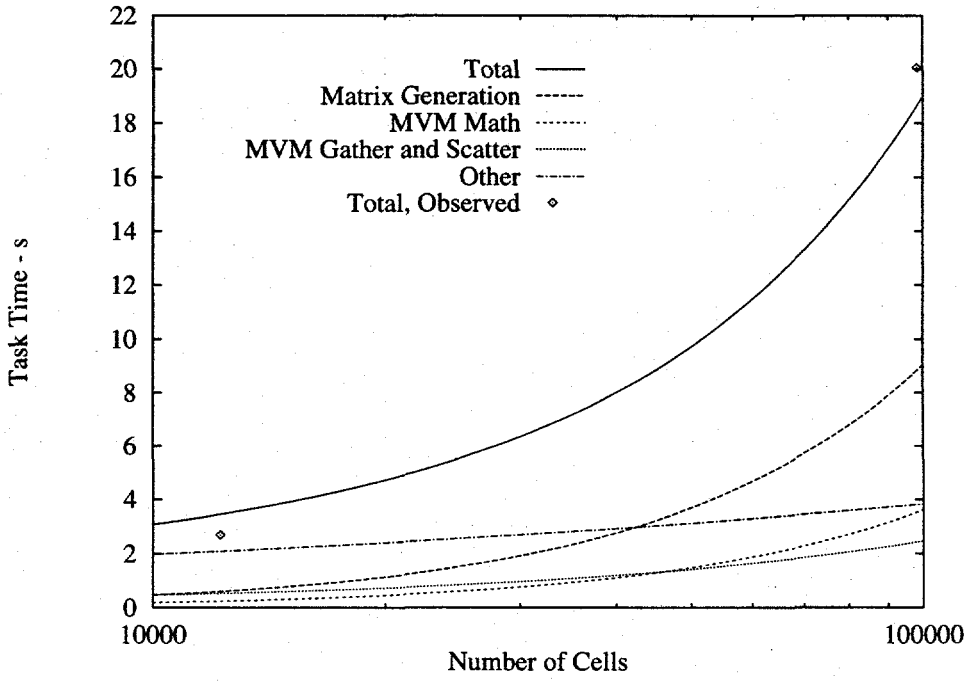

Figure 3: CM-5 computational model predictions of fraction of total solution time spent in each computational task. 32 nodes.

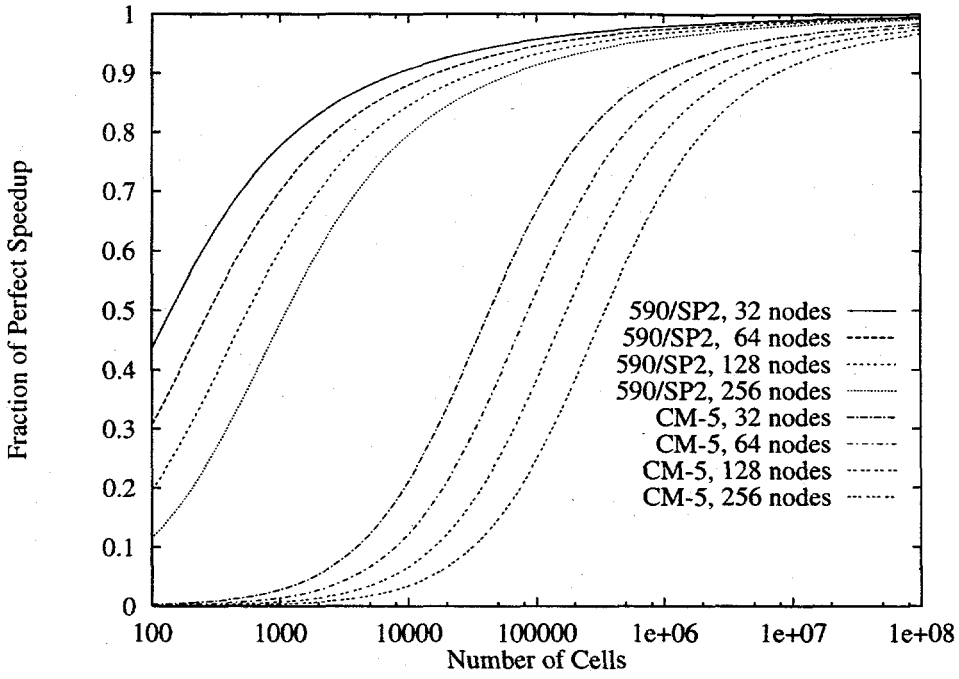

Figure 4: Speedups predicted by CM-5 and IBM 590/SP2 computational models. A "perfect speedup of 1.0 means that a parallel calculation with n processors would require time t/n where t is the time to perform the calculation on one processor.

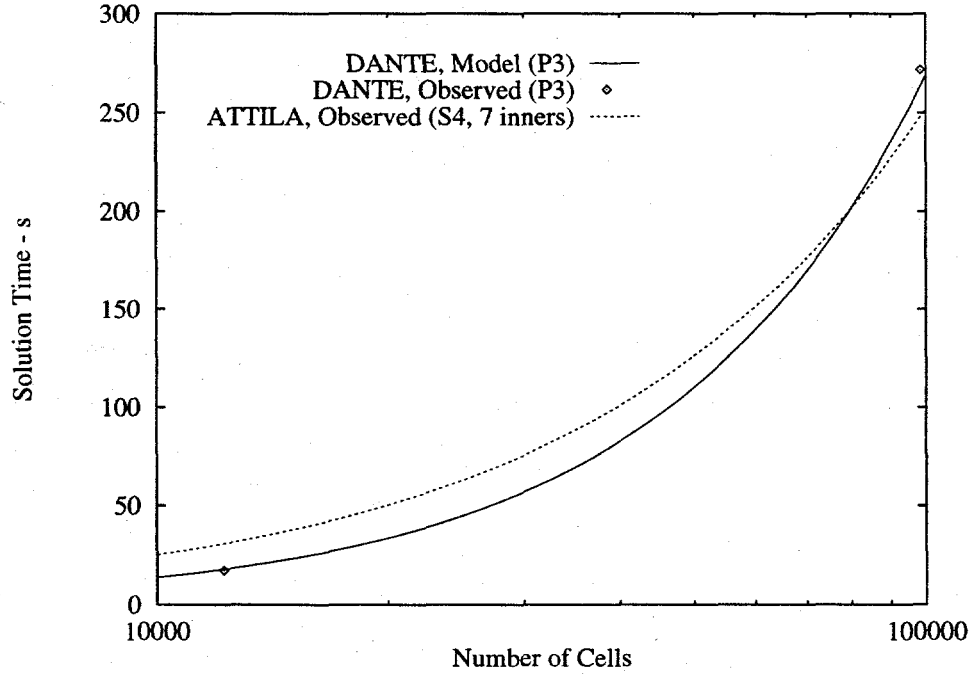

Figure 5: Comparison of serial solution times for a first order S_n code (ATTILA) and a second order self-adjoint P_n code (DANTE). Benchmark problem of section IV.C.

P_n Order	nblocks	Memory (MBytes)	
		Compute In-Line	Store Full Matrix
1	1	7.196	9.30
3	36	11.98	38.10
5	175	21.30	155.90
7	434	36.60	360[1]
9	813	59.56	670[1]

Table IV: Memory usage of in-line and full matrix storage schemes, con_test2 mesh, (Mbytes) IBM 590, Note 1. Estimated, insufficient memory to store full matrix.

The observed memory usage on the IBM 590 is presented in Table IV for a variety of P_n orders. Memory usage was determined using the AIX "*ps*" utility during the actual operation of the DANTE program. Memory requirments are observed to be reduced by as much as a factor of 10 for the higher P_n orders. The price of this reduction is the time spent in matrix generation, which was between 50 and 60 % of the total solution time for all P_n orders.

IV.E. Discussion

Our experince with the DANTE algorithm is limited to date, but we are able to make some general observations. Some of the advantages of our approach include:

- The second order self-adjoint formulation is easy to accelerate.
- Good parallel efficiency is theoretically possible.
- The efficiency of the conjugate gradient solver is excellent for time dependent problems, as these problems include a large "time-absorption" term on the matrix diagonal, and have an excellent initial guess for the solution from the previous time cycle.
- There is a large community of researchers working on SPD solvers. Significant developments in this field should apply directly to DANTE.
- The P_n or SP_n solution modes are rotationally invariant.
- We find it relatively easy to try new communication libraries, and port to new platforms.

Some of the disadvantages that we have noted are:

- The conjugate gradient solver iteration count is problem dependent. Problems with a large number of cells, or optically thin cells can require many CG iterations. Problem scattering ratio can also affect iteration count.
- No sophisticated parallel preconditioners for the conjugate gradient solver are presently in hand.
- Storage of the Hessian matrix and one block of the transport matrix, even though greatly reduced over the full P_n matrix, still represents a significant memory requirement for large problems.
- Vacuum cells are problematic, as the second order self-adjoint form of the transport equation includes $1/\sigma$ terms.

V. FUTURE WORK

Our research and development efforts continue. Investigations are currently underway or planned for a variety of improvements, modifications, and extensions of the current algorithm. Among these are:

- Investigate/optimize performance on the LLNL and LANL ASCI MPP clusters.

- Investigation of performance of alternate communication libraries.
- Investigation of alternative parallel solvers, i.e. algebraic multi-grid solver, et. al.

VI. CONCLUSION

A spherical harmonics based transport algorithm has been developed for parallel computer architectures. Good parallel performance has been demonstrated on the CM-5 massively parallel computer operating in a SIMD mode. As good or better performance is expected on MPP clusters.

ACKNOWLEDGEMENTS

This work was performed with funding provided by the LANL LDRD and ASCI programs, and was performed in part using resources located at the Advanced Computing Laboratory of Los Alamos National Laboratory. Harold Trease (LANL) provided the tetrahedral mesh generation software used to create the computational meshes for all benchmark problems.

REFERENCES

1. Thinking Machines Corporation, CM-5 Technical Summary, Nov 1993.
2. Thinking Machines Corporation, Connection Machine Scientific Subroutine Library, 1994.
3. Golub and Van Loan, "Matrix Computations", John Hopkins University Press, Baltimore, 1989.
4. Karypis and Kumar," Parallel Multilevel k-way Partitioning Scheme for Irregular Graphs", Technical Report, Department of Computer Science, University of Minnesota, 1996.
5. Zienkiewicz and Taylor, "The Finite Element Method", McGraw-Hill, London, 1994.
6. IBM Corporation, "AIX Version 3.2 for RISC System/6000, Optimization and Tuning Guide for Fortran, C, and C++, Chap 4. RISC System/6000 Architecture", Armonk NY, 1993
7. Wareing, T. A., J. M. McGhee, and J. E. Morel, " ATTILA: A Three-Dimensional Unstructured Tetrahedral Mesh Discrete-Ordinates Transport Code", Vol. 75, p.146, Trans. Amer. Nucl. Soc., American Nuclear Society Annual Winter Meeting, Washington, D.C., 10-14 Nov 1996.