Performance Analysis, Modeling and Enhancement of Sandia's Integrated TIGER Series (ITS) Coupled Electron/Photon Monte Carlo Transport Code

Mahesh Rajan, Brian Franke, Robert Benner, Ron Kensek and Thomas Laub
Sandia National Laboratories

Abstract— **ITS is a powerful and user-friendly software package permitting state-of-the-art Monte Carlo solution of linear time-independent coupled electron/photon radiation transport problems, with or without the presence of macroscopic electric and magnetic fields of arbitrary spatial dependence. As one of a few Sandia applications that are targeted for capability class machines like the ASC Red Storm, we have studied extensively the performance of this application using thousands of processors. We have successfully constructed a performance model and verified the model against measurements on a variety of Sandia compute platforms. Use of tools like VAMPIR and PAPI in performance analysis and modeling is discussed. The original algorithm for computing the statistical quantities after each batch of Monte-Carlo computations has been modified to yield improved parallel scaling. Models of alternate message passing algorithms are investigated and validated against measurements on the Red Storm.**

Index Terms—**Performance modeling, performance analysis, Monte Carlo Radiation Transport**

I. Introduction

The Integrated Tiger Series (ITS) code is an evolving Monte Carlo radiation transport code that has been used extensively in weapon-effect simulator design and analysis, radiation dosimetry, radiation effect studies and medical physics research. Many individuals from the DOE labs and NIST have been involved over the years in the development and enhancement of ITS [1]. The different features/sections of the code in ITS: TIGER, MITS, CEPXS, XGEN etc., are applied to an analysis under investigation through the selection of appropriate pre-processor directives when the code is built. Physical rigor for the analysis is provided by employing accurate cross sections, sampling distributions, and physical models for describing the production and transport of the electron/photon cascade from 1.0 GeV down to 1.0 keV. The ITS code is capable of analyzing particle transport through both combinatorial geometry models and CAD models. It also has been significantly enhanced to permit adjoint transport calculations.

For the purposes of this paper we have analyzed the performance using as input, data from a real satellite model. The physical problem solved takes advantage of the MITS multi-group/continuous energy electron-photon Monte Carlo transport code's capability to address realistic three-dimensional adjoint computations. The adjoint transport method is a powerful technique for simulating applications where the knowledge of the particle flux is only required for a restricted region of the phase space, but where this knowledge is required for source parameters spanning a large region of phase space. The run times for simulations for a complex combinatorial geometry model using conventional, or forward, transport are prohibitive and hence the adjoint calculations used in our satellite model [2].

Our performance analysis of the ITS code was initially spurred by the JASONs and NAS review of the ASC programs to assess mapping of a set of DOE applications to architectures. Another reason for this investigation is because of the large percentage of compute resources ITS code users had consumed in the previous years and anticipated similar usage in the future. Recently we investigated the scaling characteristics of ITS to tens of thousands of processors. Execution time measurements have been obtained on various platforms at SNL; ASCI Red, VPLANT (2.4 GHz Xeon cluster with Myrinet), ICC (institutional cluster: 3 GHz Xeon with Myrinet), CPLANT (Alpha cluster with Myrinet) and more recently on the Red Storm. Our performance model attempts to follow a similar approach to that expounded by Kerbyson, et al. [3] and in fact follows closely the model presented by Mathis, Kerbyson and Hoise [4] in their analysis of the MCNP particle transport code. The model develops an analytical expression for the major portions of the execution time, namely, computation, communication and I/O. At the present time our expression for the compute time is obtained by curve-fitting the plot of the measured execution time vs. the number of histories. For the communication time model we focused on the communications at the end of each batch of computations assigned to the processors. This was accomplished with the VAMPIR tracing tool to obtain the message sizes and messaging patterns and later correlated to the MPI calls in the code. For the compute platform

Manuscript received August 26, 2005. This work was supported in part by the U.S. Department of Energy, SNL CSRF project
Authors are with the Sandia National Laboratories, P.O.Box 5800, Albuquerque, NM, 87185 (Contact phone: 505-284-5063; fax: 505-844-2067; e-mail: mrajan@sandia.gov).
Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States National Nuclear Security Administration and the Department of Energy under contract DE-AC04-94AL85000.

communication characteristics such as bandwidth and latency, a set of simple benchmarks were run. The I/O time typically has been a fraction less than 3% and at the present no model has been investigated.

In this paper we present the details of the model and compare measured performance against the model for different architectures. We also present results of tests on the new ASC Red Storm and use the model to predict performance on it to 10,000 processors. The ITS code has been recently enhanced to introduce Fortran 90/95 features and in the process it has also implemented changes in data structures that would improve performance. As will be seen from the scaling plots presented below, ITS can suffer a scaling performance penalty depending on how the history computations are split among the participating processors and the frequency with which the statistical tally of the computations are assembled by the master process. This performance penalty, due to communication cost incurred in the many-to-one communication at the end of each batch of computation, has been remedied by a modified algorithm. We have investigated a simple implementation using MPI collective communication calls and measured its performance on the Red Storm. The MPI collective communication calls (based on MPICH 2.0) are implemented using a binary reduction algorithm and executed using MPI point-to-point operations. We also investigate potential improvements in performance that could result from using Rabenseifner's [5] algorithm that promises better efficiency when message lengths of varying sizes are communicated.

Finally we also present some PAPI hardware performance counter results. We are hoping to use these results in tuning single processor performance and in understanding memory access patterns of the code to evolve a single processor performance model. We believe our analysis and code improvements will enhance the productivity of ITS code users and permit effective usage of new ASC capability class machines like the Red Storm. There is a strong impetus from DOE to push the production use of such systems to utilize upwards of 4000 processors on a regular basis.

II. ITS CODE FLOW AND SATELLITE COMPUTATIONS

Description of the code and details on using ITS can be found in ITS User Guide [6]. In this paper, we present a broad outline of the computation phases and the parallelization strategy used in ITS The interaction between particles and the physical geometry under consideration is analyzed by tracking particle trajectories through the geometry. Typically there is a linear relationship between the number of particle trajectories tracked and the execution time once the number of histories exceeds a few thousand. The ability to conduct independent computation of particle histories on a geometry replicated on every processor, combined with the fact that the statistical uncertainty in any Monte Carlo result decreases as the square root of the number of histories makes it ideally suited for parallel computations. While there exists particle transport codes in which the geometry may also be distributed, like in LLNL's Mercury Code[7], our analysis is simplified because ITS replicates the geometry on all the processors. The basic computation steps are as follows: (1) the master processor reads geometry, problem input and broadcasts to all the worker processes; the number of histories to be computed is divided among the processors and these in turn may be further divided into batches to facilitate break up of computations for convenience in getting intermediate results and keeping the time of computations between batches correlated to the restart dumps, (2) the worker processes perform the Monte Carlo computations, and (3) the master performs Monte Carlo computation and receives and tallies the data after each batch of computations for statistical calculations and outputs the data. Thus based on how the problem is set up we could have many batches of computations assigned to each processor and after completion of each batch there being a communication operation of the tally data from the workers to the master. So although there is no communication at all between the worker processors, there is many-to-one communication which, based on the algorithm used, has significant impact on the parallel scaling characteristics.

As mentioned in the introduction this analysis was undertaken with a satellite combinatorial geometry model. The calculations performed for this work were adjoint point estimation of KERMA (Kinetic Energy Released per unit MAss). This simulation is performed by modeling only photon physics. The adjoint solution allows numerous radiation sources to be assessed for a single detector. In this case, the detector was measuring energy deposition at a point inside of an electronics box. The forward sources assessed were infinite plane-wave sources of photons with a uniform energy distribution between 1 and 50 keV. The average response to these sources was calculated for 472 angular bins of approximately equal solid angle over all 4π possible incident directions. Figure 1 illustrates the dosage computations where the pixels are angular bins of the source directions and the levels are dose values at the same point on the object.

The combinatorial-geometry (CG) model of the satellite comprised of a total of more than 600 CG bodies. Portions of the satellite are approximated as reduced-density materials distributed over regions that in reality are a combination of void and intricate geometries, but the model is among the most complicated CG models that we have available. Within the satellite, an electronics box is more accurately modeled with more than 200 CG bodies. For purposes of this investigation we focused on a scaled-speedup analysis (weak-scaling) with 3.2 million histories assigned to each processor.

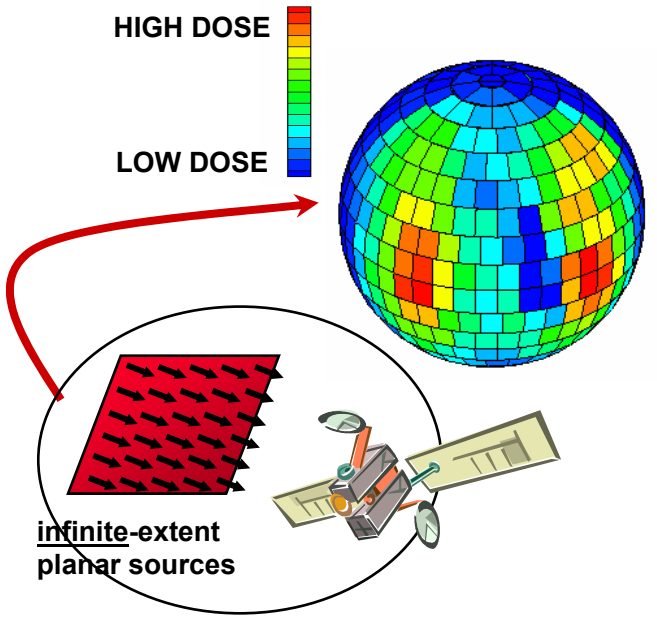

Figure 1. Adjoint calculations for the satellite model assesses the directions of vulnerability

III. Performance Model

The performance model for ITS parallel computation consists of two parts: the computation time model and the communication time model. There is a certain amount of time spent in I/O, but for typical calculations running several hours this turns out to be a negligible fraction of the total run time. Following the approach taken in Ref. [3, 4] the computation model is related to an intrinsic parameter of the computation, which here is simply the number of particle histories, N_{ph} . We measure this quantity using a single processor on each system. The computation model also depends on the geometry and therefore such measurements must be carried out for every geometry. For the satellite geometry the relationship between single processor execution time and number of histories is a linear relationship as shown in Figure 2, for the VPLANT Cluster.

From such measurements the computation time can be calculated as:

$$T_{compute} \equiv N_{ph} * T_{hist} / P$$

where T_{hist}, time for a single particle history, is obtained from the slope of the line in Figure 2. P is the number of MPI processes taken to be equal to the number of processors.

The communication time can be modeled as:

$$T_{communication} = T_{setup} + T_{tally}$$

T_{tally} refers to the worker-master communication time after each batch of Monte Carlo computations to gather the statistics for all the histories computed. This gather operation in ITS was structured as a many-to-one communication resulting in the communication time being directly proportional to the number of processes used in the computation.

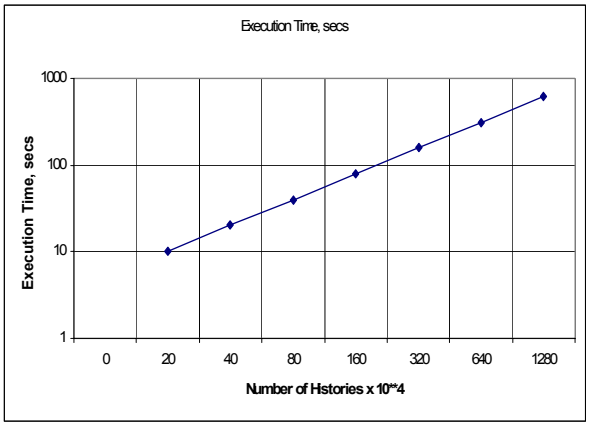

Figure 2. Measurement of single processor execution time on SNL's VPLANT Cluster

ITS has the option of assigning all the history computations to the worker processes leaving the master to coordinate the computations, this option was not used in our scaling studies. Therefore p-1 processes serially send the tally data to the master. The amount of data that is gathered by the master processor is related to the output of physical quantities requested by the analyst via the input deck and the size of these data elements depend again on the problem under investigation as they are related to the size of the physical arrays. In ITS the size of most of the important physical arrays are controlled through an include file in static fashion. Initially in building the model, we used the VAMPIR performance analysis tool to obtain the details of the communication and later correlated it with MPI calls in the code.

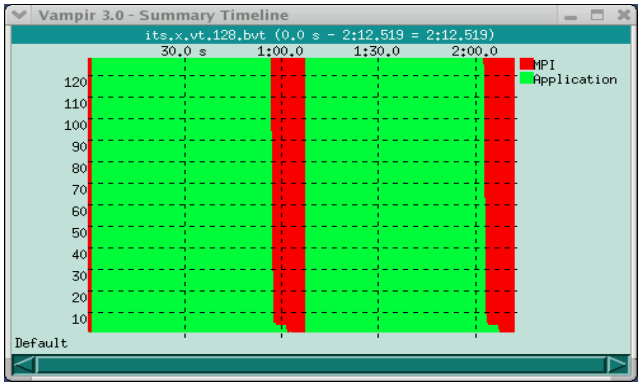

Figure 3. A VAMPIR summary timeline on SNL's ICC cluster

One could zoom into the red region in Figure 3 to gather the communication details. For the satellite model under consideration the details of the messages between a typical

worker and the master is captured in Figure 4.

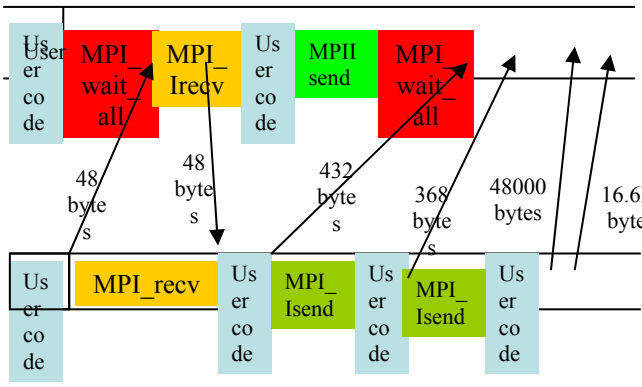

Figure 4. Worker-Master tally data communication calls

As seen from Figure 3, the communication time for setup is a small fraction. Therefore in our model we have chosen to ignore it although one could build a model for it following a similar procedure. So the analytical expression for the communication time is:

Tcomm.= {2 * T48 + T48000 + T432 + T16M + T368 } * num_batches * (p-1)

Each processor at the end of a batch of computation sends to the master a data structure coded in an 8 byte integer array called batch_info(6) that signals the master on completion of its task and some book keeping information such as the execution time for that batch. The master receives the data and using the same data structure instructs the worker process to transmit the tally data for the physical quantities (such as TAU, ALINE, etc. in the code). The 432, 368, 48000, and 16M byte messages corresponding to the various physical quantities are then received by the master from each worker processor one after another in a serial fashion, from the p-1 processes.

To compute the different terms in the communication time expression above, we measure the point-to-point communication time as a function of the message size using simple MPI benchmarks. This measured information, shown in Table 1, on bandwidth and latency are used in the calculation of the communication time as a function of the number of processors and batches. The total execution time is readily tabulated against the number of processors, from which parallel efficiency is calculated. Figure 5 compares the parallel efficiency obtained with our model against performance measurements on ASCI Red (janus), CPLANT, VPLANT and SNL's institutional cluster – ICC-LIBERTY.

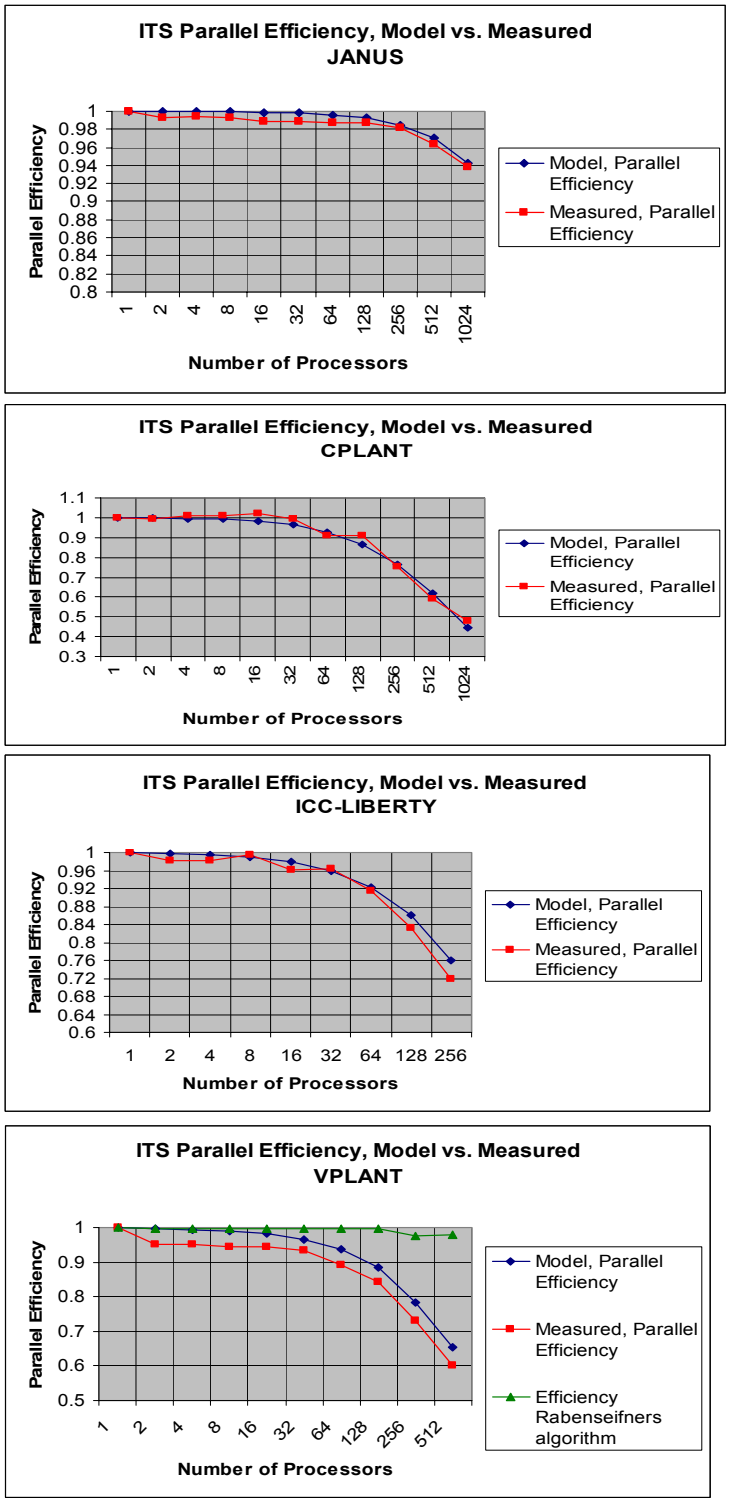

Figure 5. Comparison of parallel efficiency; Model vs. Measurements on ASCI Red, CPLANT, ICC, and VPLANT.

An objective of our performance modeling effort is not only to understand performance characteristics of applications, but

to implement changes in algorithm that would improve performance. Towards this end alternate implementations of the gather phase of the computation using some binary reduction algorithms were investigated. Thakur and Gropp [8] discuss efficiency of various algorithms for MPI collective operations which are now part of the MPICH release (version 1.2.6 and higher). Before we actually implemented a change we analyzed the potential improvement in parallel efficiency using Rabenseifner's [5] algorithm. The expression for the reduction operation for this algorithm is given by:

$$T_{communications} = 2\ln(p)\alpha + 2(p-1)\beta n/p + (p-1)n\gamma/p$$

Here p is the number of processors, α is the latency, β is the message transfer time per byte and γ is the local reduction time per byte and n is the message length. The improved scaling that would result from implementing this algorithm is seen in Figure 5 for the data corresponding to the VPLANT cluster.

Table 1. System parameters used in the computation model and comparison of parallel overhead at 512 processors

System	Pt-to-pt BW MB/s	Pt-to-pt Latency, usec	Comp time, secs	Comm time, secs	Overhead, Parallel Efficiency, f & (1/1+f)
Red Storm	1156	7	246.92	19.44	0.078, 0.927
Janus	330	18	1673	53.20	0.03, 0.97
ICC	245	6.8	108	69	0.63, 0.61
VPLANT	209	7.9	156	83	0.53, 0.65
CPLANT	76	40	334	237	0.70, 0.58

Recently we have applied the performance model to the new ASC Red Storm[9] and used the model to predict parallel efficiciency to 10000 processors. ITS code has also recently undergone major rewrite to take advantage of the new FORTRAN 90/95 features. This rewrite has created new data structures that has facilitated investigation of the alternate message passing scheme to replace the *O(p)* dependent communication scheme to *O(ln(p))* communication algorithm. In Red Storm the currently installed MPI software based on MPICH takes advantage of the improved collective communication calls. So the code was modified to simply use MPI collective operations. The results of this improvement can be seen with measured parallel efficiency curve with the 'new code' in Figure 6. If the calls to MPICH collective operations are replaced with Rabenseifner's MPI collective algorithm we should see further improvement in performance as it promises to be more efficient with both long and short messages. Measurements and models with alternate algorithms will be included in the final presentation.

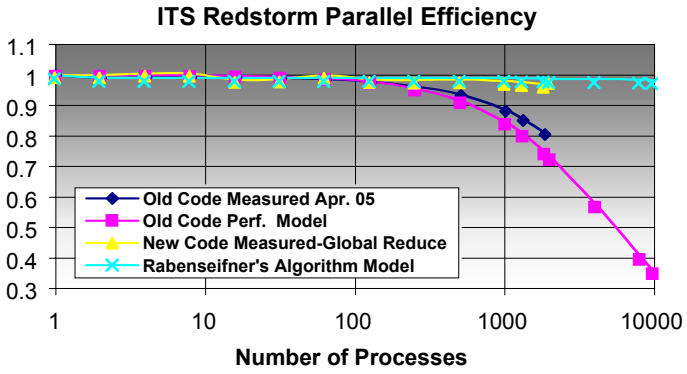

Figure 6. Red Storm; model and measured data showing improved performance with the 'new code'

IV. SINGLE NODE PERFORMANCE ANALYSIS

The single node performance of ITS was investigated on the Red Storm node: AMD Opteron 2GHz processor, ASC White node: IBM Power3 375 MHz processor and SGI Altix node: Intel Itanium-2 1.4GHz processor. On the Opteron the PGI compiler was used, on the Power3 the IBM compiler and on the Itanium the Intel compiler. Table 1 summarizes the performance of a representative computation on each processor with identical input data consisting of the satellite model with 32000 history computations.

Table 2: Comparison of single processor execution on Opteron, Power3, and Itanium

Processor	Opteron, 2Ghz	Power3 375MHz	Itanium 1.4GHz
Cache	L1=64KB, L2=1MB	L1=64KB (Data); 32KB (Ins), L2=8MB	L1=32KB, L2=256KB, L3=3MB
Exec. time, secs	1.69	6.51	3.91
Gprof : *gg, loczon, locbod*	70%,9%,6%	44%,6%,3%	56%,10%,7%

As seen in Table 2, the cache sizes of the three processors are different. These comparisons of execution time with just one geometry model may not be conclusive and must be extended in the future by considering different model geometry data. Larger L3 cache on and L2 cache on Power3 do not result in a significant reduction in the execution time. This suggests that cache has a lesser impact and computations may require frequent memory access. A linear relationship between the execution time and the number of history computations suggests that this other important input parameter does not influence the cache behavior.

A code profile using *Gprof* shows that the three top routines, *gg, loczon*,*locbod,* as indicated in Table 2, account

for majority of the compute time: 85% on the Opteron and 53% on the Power3 and 73% on the Itanium. Most of the time in ITS is spent in routine *gg* which computes the distance a particle travels within a combinatorial geometry body before it escapes that body. The functions *loczon* and *locbod* determine if the initial position plus the displacement at a step, puts a particle inside or outside a given body and for this computation they call the routine *gg*. Subroutine *gg* mainly consists of branches for different geometries such a polyhedron, sphere, cone, cylinder, etc. Further within the computations for each geometrical body there are branches to compute intersection of particle trajectory lines with geometry component surfaces and for different directions of travel. Therefore the nature of the code suggests that this application would stress the load/store units as the different geometry data is accessed as particle history is tracked. As the size of the geometry data exceeds the cache size, all the geometry information is not held in cache during the entire history computations. Because the main loop of computation is over the particles, the cache temporal locality is not very high as different particles move through different geometries, while the cache spatial locality is dictated by the size of the data describing a given body's geometry. For the reasons discussed above, and further analysis of the dominant routines leads one to the conclusion that ITS has a lot of load/store and branching instructions. While computation of distance involves the square root operation and a small amount of associated floating point computations, ITS is not dominated by floating point operations unlike many scientific codes.

For further investigations the code was instrumented with the PAPI tool [10] to obtain hardware performance counts. Several runs of the program were required because various processor events could not be measured simultaneously. Although the differences in the processor architecture and instruction sets would impact the measurements, it is instructive to compare the percentage of each instruction type between these different processors. However, this goal is limited by the lack of certain hardware counts, such as integer operations, across all these processors. Despite this limitation, we attempted to understand the nature of this application by collecting and comparing whatever information the PAPI tool provides. This result is summarized in Table 3, showing the fractional percentage of load, store, branch, floating point, and integer instructions to the total instructions. For the Opteron and the Itanium as integer operation count is not available under PAPI, and also the load/store count not available for the Opteron. We postulate that the majority of instructions listed under 'unaccounted' for the Opteron is integer and load/store operations. For the IA-64 it is reasonable to assume that the unaccounted operations might be Integer operations.

It is interesting to note that for this application that the cycles-per-instruction for both the Power 3 and Opteron is close to 0.83, while it is 0.65 for the Itanium. The fact that the clock speed of the Opteron is almost five times that of the Power3 and yet the execution time ratio is 1/3.8 suggests higher efficiency of the operations for Power3.

Table 3. Hardware Performance counter-PAPI data with *–O3* optimization

PAPI DATA	IA-64, 1.4GHz	Power3, 375MHz	Opteron, 2.0GHz
TOTAL CYCLES	5,471,391,792	2,524,426,100	3,841,925,011
TOTAL INSTRUCTIONS	8,348,552,835	3,022,782,250	4,627,544,804
% Floating point ins or ops	0.026	0.052	0.040
% Load instructions	0.305	0.312	N/A
% Store Instructions	0.251	0.235	N/A
% Branch Instructions	0.084	0.137	0.199
% Integer Instructions	N/A	0.376	N/A
% Unaccounted ins	0.334	-0.112	0.761

We also conclude from the data in Table 3, that this application is not floating point computation intensive and therefore MFLOPS measures of a processor are poor indicators of how this code would perform. For the Power3, which has the best selection of hardware counters under PAPI, a negative value for the unaccounted instruction merely points to the redundant counts of some operations amidst the categories listed.

The objective of the single processor analysis is to find opportunities for performance improvement. From the large number of call counts to subroutine *gg*, in-lining this routine (*-Q+gg* flag) was first attempted on the ASC White. Unfortunately on the ASC White this version of the optimized code failed to complete execution, giving a core dump. However using the maximum optimization *–O5* which also invokes inter-procedural analysis (*-ipa=lvl2*), the execution time reduced to 3.46 seconds, a performance improvement of 47%. PAPI data with this higher optimization showed that this improvement is a consequence of 41% reduction in the total cycles and a 27% reduction in the total instructions. The instruction mix with this higher optimization did not show large changes in the percentage of instructions listed in Table 3. Similar attempts at compiler optimization for the Opteron and the Itanium with in-lining and inter-procedural optimization did not result in a significant improvement in performance. Even attempts at profile-based-optimization (*-pfo*) on the Opteron yielded little improvement in performance.

This study suggests that greatest improvement in performance would come from improving the cache temporal locality to reduce load/store instructions. We are hoping to drill down into some of the compute intensive routines like *gg* to identify places where the cache locality can be improved. We will continue investigating opportunities for this without a major restructuring of the code.

V. Conclusion

We have successfully built and used an analytical performance model for ITS to understand it's performance on all of Sandia's major computational resources. Our approach follows closely the approach expounded by LANL's PAL team. The performance model and analysis has helped us to identify bottlenecks in communication which were limiting the scalability of this application. Along with other code improvements we have modified the communication algorithm permitting good scaling of this application to *O(10K)* processors.

References

[1] J.A. Halbleib, et al., "ITS Version 3.0: The integrated TIGER Series of Coupled Electron/Photon Monte Carlo Transport Codes," Technical Report SAND91-1634, Sandia National Laboratories, 1992.

[2] R.P. Kensek, et al., "DTRA High Performance Computing for Testable Hardware Initiative Final Report," Sandia National Laboratories, SAND2001-2900 (2001).

[3] Kerbyson, D.J. et al., " Predictive Performance and Scalability of Modeling of a Large-Scale Application", In the Proceeding of the IEEE/ACM conference on Supercomputing Sc '01, Denver, CO, October 2001.

[4] M.M. Mathis, D.J. Kerbyson, A. Hoise, "A performance Model of nondeterministic Particle Transport on Large-Scale Systems" In Future Generation Computer Systems, To Appear 2005, LA-UR 02-7313.

[5] R. Rabenseifner, "Optimization of Collective Reduction Operations", International Conference on Computational Science, June 7-9, Krakow, Poland, LCNS, Springer-Verlag, 2004.

[6] B.C. Franke, R.P. Kensek, T.W. Laub, "ITS Version 5.0: The Integrated TIGER Series of Coupled Electron/Photon Monte Carlo Transport Codes," Technical Report SAND2003-4173, Sandia National Laboratories, 2004.

[7] M. O'Brien, J. Taylor, R. Procassini, "Dynamic Load Balancing of Parallel Monte Carlo Transport Calculations," The Monte Carlo Method: Versatility Unbounded In A Dynamic Computing World, Chattanooga, Tennessee, April 17–21, 2005, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2005).

[8] Thakur, R., Gropp, W, Improving the Performance of Collective Operations in MPICH. In:Euro PVM/MPI 2003, Springer-Verlag, LNCS Vol. 2840 (2003) 257267

[9] Hoise, et al., "An initial Performance Analysis of the Red Storm Architecture" LANL PAL on-line distribution March 14, 2005.

[10] P.J. Mucci, S. Browne, C. Deane, and G. Ho. ***PAPI**: A Portable Interface to Hardware Performance Counter**s**.* In Proceedings of the Department of Defense HPCMP Users Group Conference, Monterey, CA, June 7-10, 1999.