UCRL-JC-119205
PREPRINT

Conf-9410254--7

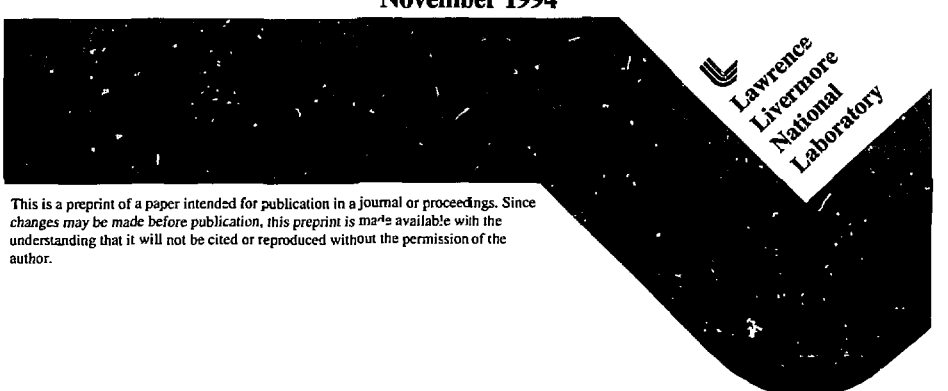
Parallelizing Monte Carlo with PMC

J.A. Rathkopf, T.R. Jones, D.M. Nessett, and L.C. Standberry
University of California
Lawrence Livermore National Laboratory
Livermore, CA 94550

This paper was prepared for submittal to

The Eighth Nuclear Explosives Code Developers' Conference
October 25–28, 1994
Las Vegas, Nevada

November 1994



Parallelizing Monte Carlo with PMC

J.A. Rathkopf, T.R. Jones, D.M. Nessett, and L.C. Stanberry
Lawrence Livermore National Laboratory

PMC (Parallel Monte Carlo) is a system of generic interface routines that allows easy porting of Monte Carlo packages of large-scale physics simulation codes to Massively Parallel Processor (MPP) computers. By loading various versions of PMC, simulation code developers can configure their codes to run in several modes: serial, Monte Carlo runs on the same processor as the rest of the code; parallel, Monte Carlo runs in parallel across many processors of the MPP with the rest of the code running on other MPP processor(s); distributed, Monte Carlo runs in parallel across many processors of the MPP with the rest of the code running on a different machine. This multi-mode approach allows maintenance of a single simulation code source regardless of the target machine. PMC handles passing of messages between nodes on the MPP, passing of messages between a different machine and the MPP, distributing work between nodes, and providing independent, reproducible sequences of random numbers. Several production codes have been parallelized under the PMC system. Excellent parallel efficiency in both the distributed and parallel modes results if sufficient workload is available per processor.

Introduction

With the recent availability of production quality Massively Parallel Processor (MPP) computers, physics simulation codes must make the transition from serial machines in order to exploit effectively the superior performance afforded by MPPs. Many of these codes model a variety of physical processes including particle transport using the Monte Carlo method, perhaps the algorithm that can best take advantage of parallel architectures. PMC (Parallel Monte Carlo) has been developed to simplify the porting of Monte Carlo packages to MPPs. PMC not only distributes the Monte Carlo simulation over many processors of the MPP, but it also permits the distribution of different parts of the simulation over different machines. Computational fluid dynamics (CFD), for example, is difficult to adapt to parallel machines but is very easy to vectorize. With PMC, the CFD portions of the simulation can run on a vector supercomputer while the Monte Carlo simulation can run on a MPP. In this way, algorithms are placed on architectures they can exploit most effectively.

Following a short description of PMC, this paper presents results of the application of PMC to the parallelization of two Monte Carlo codes. In both cases excellent parallel efficiency results with sufficient workload. These performance characteristics can be used as a guide for users of parallel Monte Carlo codes to help them most effectively take advantage of MPPs.

The PMC System

The purpose of PMC is to relieve Monte Carlo code developers of the burden of converting their packages from conventional serial machines to MPPs. With PMC, developers of these "client" codes need not become familiar with the intricacies of various message passing systems or shared memory environments. Rather, instead of calling their Monte Carlo packages directly from a controlling routine, an intermediate PMC routine is called. Because separate processors on a MPP do not necessarily share memory, all of the data required of the Monte Carlo package must be passed to the intermediate PMC routine through subroutine arguments. PMC, in turn, passes control back to the Monte Carlo package on the separate processors by calling a predetermined subroutine supplied by the client code developer. Again, required data are passed through the argument list. This routine calls the rest of the Monte Carlo package which actually performs the transport calculation.

The data in the argument list are specified in terms of messages in several standard formats. Here, messages are collections of blocks of data allocated by the client code. PMC relies heavily on Cray-style FORTRAN pointers (supported by all useful FORTRAN compilers) to minimize the number of redundant memory allocations. Message formats are necessary so that PMC can correctly perform operations such as (in the case of a distributed implementation) translation of data between different internal representations, summing of results for multiple processors, and distributing data among processors.

Following completion of the Monte Carlo transport calculation on each processor, control is returned to PMC as result data are again passed through the argument list. PMC reduces this data as determined by the message type. Reduction is the process where multiple processor data are combined to form a single, meaningful result. Reductions supported by PMC are summation and concatenation. PMC returns to the client controlling routine with the reduced data contained in the subroutine argument list. The client code proceeds with the remainder of the non-Monte Carlo calculation before beginning the next time-step and calling PMC once again.

With PMC, the client code developer achieves parallel/distributed Monte Carlo with minimal code modi-

fications: (1) formulate all the data required of the Monte Carlo package into PMC messages, (2) replace the call to the Monte Carlo package with a call to PMC routine, (3) write a predetermined subroutine which accepts messages from PMC and calls the Monte Carlo package, and (4) formulate all the data resulting from the Monte Carlo package into PMC messages. PMC handles all chores related to control and communication across machines including initialization and startup of MPP processors, data translation between machines, data reduction, providing reproducible, independent random number sequences.

More detailed descriptions of PMC are available in Rathkopf (1992) and Rathkopf et al. (1993).

The current implementations of PMC use the easiest parallelization model imaginable: complete replication. The entire problem to be analyzed, i.e., physical description of the geometry and the physical data (cross sections), is replicated on each MPP processor. Particles, which represent the work of a Monte Carlo calculation, are distributed across processors. Advantages of this approach include its simplicity and the potential for easy use of existing coding without modification. Unfortunately, problem sizes are limited by the memory capacity of a single MPP processor. Future version of PMC will support domain decomposition where the problem geometry is split among several processors. Much larger geometries could be run under this technique, but Monte Carlo tracking routines would require modifications to handle internal boundaries. Particles leaving these boundaries would be reassigned to the appropriate processors by PMC.

Several implementations of PMC exist allowing the use of a variety of computer configurations and message passing systems. Computer configurations supported are serial, Monte Carlo runs on the same processor as the rest of the code; parallel, Monte Carlo runs in parallel across many processors of the MPP with the rest of the code running on other MPP processor(s); and distributed, Monte Carlo runs in parallel across many processors of the MPP with the rest of the code running on a different machine. This approach allows maintenance of a single client code source regardless of the target configuration.

A final advantage of PMC is that it insulates the client code developer from changes in message passing systems and machine architectures as new machines are brought on line. These details are immersed in PMC. Once PMC is converted to a new system or architecture, client codes it supports are converted too.

Implementations of PMC

The various versions of PMC are supported through a single source code. Because PMC's dependencies on message passing systems and machine architectures are isolated to just a handful of routines, the C-preprocessor provides adequate conditional compilation to allow single source maintenance.

Serial. The serial version of PMC is implemented by emulating message passing by copying client code data referenced in PMC messages to memory allocated by PMC. The pointers to the new data are kept in a common block for later reference by PMC. PMC performs this copy (in contrast to simply passing pointers to the original data) so that the client code's message structure is thoroughly debugged in serial mode before proceeding to more difficult, parallel scenarios. This copy appears to add insignificant overhead (Clouse et al., 1994).

Parallel. Two different parallel (i.e., not distributed) versions of PMC have been implemented. The first uses the Livermore Message Passing System (LMPS) (Welcome, 1991). LMPS is available on LLNL's BBN TC2000, where PMC was first developed, as well as LLNL's new Meiko CS-2. A second version uses MPSC, one of the "native" message passing systems provided with the Meiko CS-2. All timing results presented here were found using the LMPS parallel version.

Distributed. In this context, the word *distributed* is slightly misleading. Here it means that part of the calculation runs on a serial computer while the Monte Carlo runs in *parallel* across many processors of a MPP. PMC uses a custom message passing system PMPS (PMC Message Passing System) especially developed for the distributed implementation of PMC. Two versions of PMPS were developed: one uses LLNL's ELROS (Boolootian, 1993) for communication between machines; the other uses the widely available PVM (Begulin, 1991). Both versions use MPSC for communication between processors within the Meiko CS-2. The ELROS version of PMPS has been adopted for permanent use. In addition to allowing calculations to be distributed between distinct machines such as a Cray and the Meiko, PMPS enables the calculation to be distributed between a Meiko processor in the login partition and a number of processors in a parallel partition. No timing results for the distributed version of PMC are presented here.

Experiences with PMC

PMC has achieved excellent parallel performance for three Monte Carlo applications. Experiences with a Monte Carlo photonics demonstration code and a Monte Carlo neutronics package are described below. Information regarding the third application can be found in Clouse et al. (1994).

Monte Carlo photonics demonstration code

The xPHOT (Martin et al., 1986) series of codes has been used for a number of years to examine alternative algorithms for Monte Carlo particle transport on a variety of computer architectures, including vector supercomputers and parallel processors. These codes simulate the transport of photons in a high temperature plasma on an axially-symmetric, two-dimensional, logically rectangular (KL) mesh. They include realistic opacity data. The version TPHOT allows investigation of methods for parallelizing time-dependent Monte Carlo codes. Initially TPHOT called the message passing library LMPS directly. Now TPHOT uses PMC. Essentially no difference in parallel efficiency has been seen between the two implementations.

The results presented here are for a single time step of a two-region test problem divided into 1960 zones (49x40) with a specified source. A number of calculations were performed varying the workload, W, which is specified by the number of particles, over an increas-

ing number of processors (1, 4, 8, 16, 32, 64, 115). The number of particles for each value of W is shown in Table 1. Each curve in the figures represents a constant number of particles spread over processors.

Table 1. Particles in photonics workload.

W	num. particles
4	24,000
5	240,000
6	2,400,000
7	2.4E+7
8	2.4E+8
9	2.4E+9

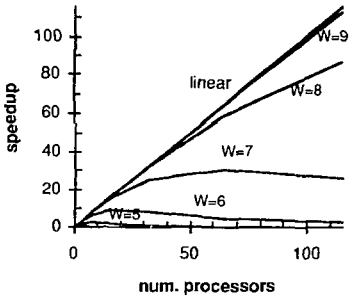


Figure 1. Parallel speedup for photonics demonstration code for various workloads.

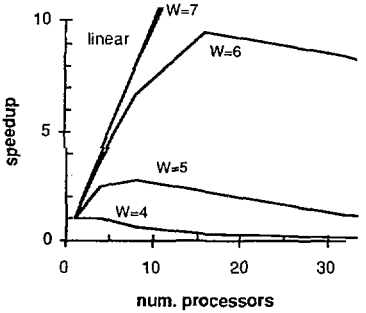


Figure 2. Parallel speedup for photonics demonstration code for various workloads (enlarged).

PMC achieves nearly perfect linear speedup with even the maximum number of processors of 115 for the largest workload tried as seen in Figure 1. For smaller workloads, linear speedup is seen as long as the number of processors is not too large. For very small workloads (see Figure 2, an enlargement of Figure 1), using more than just a few processors actually causes the problem to run slower than it does on a single processor, truly a case of misuse of resources.

Efficiency, the fraction of linear speedup obtained, is plotted in Figure 3.

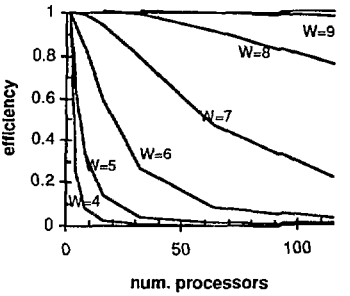


Figure 3. Parallel efficiency for photonics demonstration code for various workloads.

Users of future parallel production codes will have to carefully balance the need for quick turn-around, the desire for accurate results, and the availability of many processors in order to effectively utilize resources. By drawing a horizontal line across a graph such as Figure 4, users can predict how many particles can be simulated (which is directly related to accuracy) in a given elapsed time as a function of number of processors used. By finding the minimum in a particular curve, users can deduce the optimal number of processors for a calculation of specific accuracy.

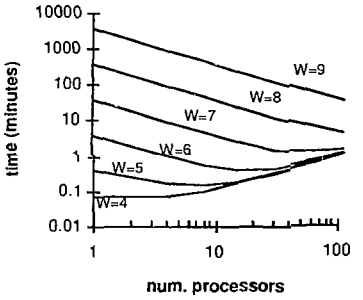


Figure 4. Total elapsed run times for photonics demonstration code for various workloads.

Monte Carlo neutronics package

A new Monte Carlo neutronics package also achieved good parallel performance under PMC. This package performs Monte Carlo neutron transport on axially-symmetric, two-dimensional, arbitrarily connected mesh. Production quality neutron kinematics,

sources, and tallies are part of the package. The increased sophistication of the neutron collision physics together with other complexities result in a 100-fold increase in CPU time per particle history. As will be seen, this increase favorably impacts parallel performance.

The results presented here are for a single time step of a two-region test problem divided into 280 zones (21x15) with a specified source. A number of calculations were performed varying the workload, W, which is specified by the number of particles, over an increasing number of processors (1, 2, 5, 10). The number of particles for each value of W is shown in Table 2. Each curve in the figures represents a constant number of particles spread over processors. The results presented here are *very* preliminary. So preliminary that not all combinations of workload and number of processors complete without crashing.

Table 2. Particles in neutronics workload.

W	num. particles
a	1000
b	10,000
c	50,000
d	100,000

Even for as few as 10,000 particles (W=b), a speedup of 4.7 is seen in Figure 5 for the neutronics calculation run on five processors. The photonics calculation required 2,400,000 particles for a similar speedup for similar number of processors. It is the total computational work done by each processor that overcomes parallelization and serial overhead to achieve good parallel efficiency.

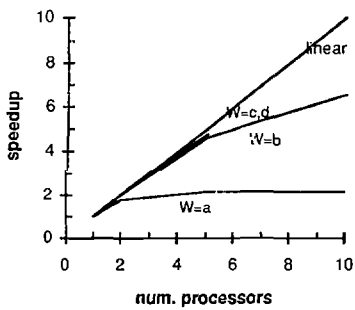


Figure 5. Parallel speedup for neutronics package for various workloads.

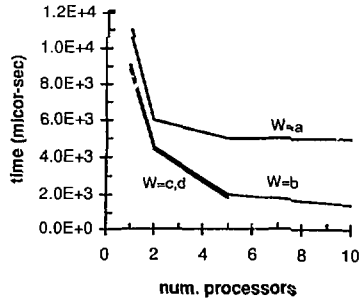


Figure 6. CPU time per particle for neutronics package for various workloads.

Comparing the time spent transporting each neutronic particle (Figure 6) with that required of photonic particles (Figure 7) vividly illustrates the increased complexity of the neutronic calculation and the corresponding advantage to parallel efficiency.

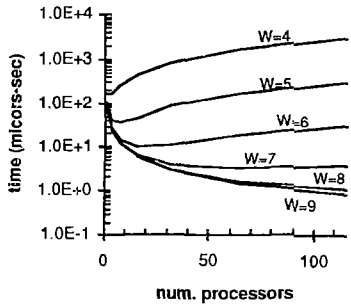


Figure 7. CPU time per particle for photonics demonstration code for various workloads.

Conclusions

The PMC system provides Monte Carlo code developers a simple way of exploiting parallel processing either solely on a MPP or distributed between another computer and a MPP. As seen in the timing results for the two applications analyzed above, an interesting dynamic exists between workload, number of processors, and parallel efficiency. In the future, when users are running parallel Monte Carlo applications for production calculations, graphs such as those presented here should be available to help guide them in their choices of workload and machine configurations.

Acknowledgments

The authors gratefully acknowledge the assistance of Dale Slone, Bill Biester, Chris Clouse, George Zimmerman, Doug Dickson, and Rex Evans. This work was

performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract #W-7405-Eng-48.

References

Begulin, A., Dongerra, J., Geist, G.A., Manchek, R., and Sunderam, V., *A User's Guide to PVM: Parallel Virtual Machine*, Oak Ridge National Laboratory, Oak Ridge, TN, ORNL/TM-11826 (1991).

Boolootian, M.R., Branstetter, M.L., Guse, J.A., Nessett, D.M., and Stanberry, L.C., *An ELROS Primer*, Lawrence Livermore National Laboratory, Livermore, CA, UCRL-MA-108543 (Rev. 2) (1993).

Clouse, C.J., Biester, W.C., and Rathkopf, J.A., "Parallel Monte Carlo neutronics on the Meiko CS-2," *Proceedings of NECDC-94*, Los Alamos National Laboratory, Los Alamos, NM (1994).

Martin, W.R., Nowak, P.F., and Rathkopf, J.A., "Monte Carlo Photon Transport on a Vector Supercomputer," *IBM J. Res. and Dev.*, 30, 193 (1986).

Rathkopf, J.A., "PMC: A shared short-cut to portable parallel power," *Proceedings of NECDC-92*, (also in UCRL-112311) Lawrence Livermore National Laboratory, Livermore, CA (1992).

Rathkopf, J.A., Martin, W.R., Majumdar, A., and Litvin, M., "Experiences with different parallel programming paradigms for Monte Carlo particle transport leads to a portable toolkit for parallel Monte Carlo," *Proceedings of the Joint International Conference on Mathematical Methods and Supercomputing in Nuclear Applications*, (also in UCRL-JC-111846) Karlsruhe, Germany (1993).

Welcome, T.S., *Programming in LMPS*, Lawrence Livermore National Laboratory, Livermore, CA, UCRL-MA-107031 (1991).