A simple parallelization of GEANT4 on a PC cluster with static scheduling for dose calculations

K Sutherland1,2, S Miyajima1,3 and H Date2

1 CREST, Japan Science and Technology Agency, 4-8-1 Honcho Kawaguchi, Saitama, 332-0012 Japan

2 Hokkaido University, Kita-ku N 12 W 5, Sapporo 060-0812 Japan

3 Japan Atomic Energy Agency, Kansai Photon Science Institute, 8-1 Umemidai, Kizu-cho, Soraku-gun, Kyoto, 619-0215 Japan

E-mail: ken@cme.hokudai.ac.jp

Abstract. We discuss our experience improving the speed of Monte Carlo dose distribution calculations with GEANT4. First we customized the core geometry library to navigate voxelized CT volumes more efficiently. This resulted in a speed improvement of more than two orders of magnitude in our case. We then tried to run the simulation in parallel. However, we found that existing parallelizations of GEANT4 such as TOP-C (Task Oriented Parallel C/C++) did not scale well. We determined that the poor performance was due to excessive network traffic. We abandoned the master/slave model employed in TOP-C in favour of “static scheduling” parallelism where the event list is divided evenly among the processes. Our implementation used only a few low-level MPI calls. With this method, we achieved a linear speed improvement on our cluster of 23 machines. The simulation executes efficiently because there is almost no network traffic during the simulation. Using our customized GEANT4 navigation library running on our cluster with our parallelization method, we achieved a combined speed improvement factor of more than 3000, reducing our simulation time from several days to a few minutes. Although we did not consider issues such as fault tolerance or load sharing, our approach is well suited for many kinds of Monte Carlo simulations on a homogenous cluster of local PCs.

1. Introduction

While recent improvements in hardware and software have made Monte Carlo dose simulations feasible on personal computers, it can still take hours, even days to achieve a statistically meaningful result. Although generally less accurate in inhomogeneous media, analytical methods are employed in practice for clinical applications, such as radiation treatment planning. For our research, we wanted to simulate with GEANT4 [1] laser-accelerated proton therapy [2] in patient geometries computed from CT data. Even though our software is intended for research, not clinical use, and thus not constrained by the requirement of near real-time response, execution times were still a factor in our design decisions. Simulating millions of particles through complex patient geometry volumes, which may contain millions of voxels describing the human body matter in 3D, severely tax the capacity of even today’s most powerful PCs.
GEANT4 has been successfully parallelized with “Task Oriented Parallel C/C++” (TOP-C) [3]. TOP-C was developed by Gene Cooperman at Northeastern University originally for symbolic and algebraic computation. It makes it relatively easy to port large serial applications to parallel. It also provides fault tolerance and load sharing, employing a master-slave computational model.

Off-the-shelf components and public domain software make a cluster of PCs on an Ethernet (Beowulf cluster) the least expensive form of “supercomputer”. However, unlike shared memory systems, clusters must work with separate copies of the program and data. Because of the relative low speed of the connection between nodes, overall system performance can suffer greatly if too much data is exchanged between nodes.

In this study, we explore various methods to improve the speed performance of Monte Carlo radiation treatment simulations with GEANT4. We found that in order to achieve good parallel performance, it is necessary to minimize the communication among the nodes during the simulation. We abandoned the master-slave model employed in TOP-C in favor of “static scheduling” where the event list is divided evenly among the nodes. Our implementation used only low-level MPI [4] calls. Although not a general solution, we feel that our method is well suited for many kinds of Monte Carlo simulations on a homogenous cluster of local PCs.

2. Methods

2.1. Hardware and Software.

Our Beowulf cluster consisted of 22 client (slave) machines and a server (master) connected with a Netgear 24 port Gigabit Ethernet switch. We used Intel Pro/1000 network interface cards. Each slave was equipped with a 3 GHz Pentium IV processor and 2 gigabytes of memory. All machines ran Red Hat Linux 9 [5]. For our tests, we used GEANT4 version 8.0p1 and TOP-C version 2.5.0 and MPICH2 (v1.0.3), a popular and free implementation of MPI.

We developed a Windows application to investigate the effect of various beam parameters on dose distributions. Our software allows the user to open a series of DICOM CT images, specify the tumor polygon, one or more gantry positions and beam characteristics such as beam diameter, energy spread and angular divergence.

The CT value (in Hounsfield units) of each voxel is used to determine a voxel material. Each material is assigned a density and is composed of chemical elements in ratios following the work of Schneider [6]. The software generates files which specify the voxel materials and an event list containing the initial position (x, y, z), direction (u, v, w) and energy (e) of each particle in the plan. These files are copied to our Linux cluster for simulation. Our GEANT4 program, which runs on the Linux cluster, executes the Monte Carlo simulation then creates a dose distribution file which is read back into our Windows software for visualization and analysis. We based our simulation program on GEANT4 example extended/medical/dicom.

2.2. Test parameters.

We tested with a proton radiation treatment plan for an eye melanoma. The original CT values were sub-sampled by a factor of 4 resulting in a pixel size of 1.25 x 1.25 mm. The slice thickness was 2.5 mm. We used 11 CT slices resulting in a total of 180,224 voxels. There were almost 1 million protons in the plan. The initial proton energies to achieve an SOBP (spread out Bragg peak) within the target volume ranged from 38 to 63 MeV.

In order to achieve maximum processes speed (at the expense of some dose distribution accuracy) we used only the “standard” electromagnetic physics processes (G4MultipleScattering and G4ionIonisation) for protons. We did not simulate secondary electrons. The maximum step length was limited to 1 mm.

2.3. Customizing the navigation library
When GEANT4 needs to know the position or material of a voxel, it calls user-specified call-back functions. This is known as “voxel parameterization”. Although GEANT4 seemed to run fast when targeting non-parameterized water phantoms, we found that when targeting parameterized CT volumes, composed of hundreds of thousands of tiny parameterized voxels, the execution time was much worse.

Following Jiang [7], we modified the geometry navigation library in GEANT4. Due to the way we set up the voxel parameterization, GEANT4 was being forced to perform a linear search through all the voxels in a slice whenever a particle passed from one voxel into an adjacent voxel. For example, a 512 by 512 un-sampled CT slice contains 262,144 voxels resulting in an average of about 130 thousand iterations of the search loop. If a particle travels horizontally to the centre of the volume, it would pass through 256 voxels for a total of about 30 million iterations of the search loop per event. We modified the library to utilize the fact that in our application voxels are numbered systematically: the voxel number of an adjacent voxel can easily be computed, eliminating the need for the search loop.

With the customized geometry navigation library, we achieved a profound speed improvement of more than two orders of magnitude. The effect is more pronounced when the voxel size is small and the initial proton energy is high as these parameters result in more voxel boundary crossings per event.

2.4. Parallelizing GEANT4 with TOP-C

For our first attempt at parallelization, we used TOP-C. Although the parallelized version of our program scaled well when using the original navigation library, we were disappointed with the results when using our customized library. On our cluster of 22 slaves, we observed a speed up of only about a factor of 2. We concluded that the poor performance was due primarily to the master/slave paradigm of TOP-C exceeding the capacity of the network interface.

TOP-C applications must “marshal” (serialize) an event before sending the result of a single event from a slave back to the master. The marshaled event result data is referred to as a “hit list”. A single event consists of a series of steps along a particle path. The maximum step length can be limited in GEANT4. In general, a small step size yields a more accurate simulation, but results in more serialized hit deposit data.

2.5. Parallelizing GEANT4 with static scheduling

In order to reduce network traffic during the simulation, we decided to abandon the master/slave model and allow the slaves to run independently. In this scheme, the “slaves” are referred to as “processes” and the “master” is simply process number 0. Each process is responsible for determining for itself which event to simulate and which to skip from the list of events in the plan. This method is referred to as “static scheduling”.

The key to achieving static scheduling parallelism is in our implementation of `G4VUserPrimaryGeneratorAction::GeneratePrimaries` (figure 1). This function is called by GEANT4 at the beginning of each event. From this function we normally call `G4ParticleGun::GeneratePrimaryVertex` with the initial position, direction and energy to “shoot” the particle from the gun and start it simulating. However, if the program is running in parallel, we first check the event ID with the current process number and total number of processes. If the event is not assigned to the current process, we simply return from `GeneratePrimaries` without calling `GeneratePrimaryVertex`.

```c++
void MyPrimaryGeneratorAction::GeneratePrimaries(G4Event* pEvent)
{
    if (((pEvent->GetEventID() % NumProcess)) != ProcessNum)
        return; // Skip this event on this node!
    // Otherwise, process this event normally.
```
Figure 1. Code to skip events not assigned to the current process. This code is run on all processes in the simulation. Each process only needs to know its own process number (ProcessNum) and the total number of processes running (NumProcess).

After each process is finished simulating its share of the events, the dose distribution is left spread across the nodes of the cluster. The partial dose distributions must be collected and summed up to achieve the total dose distribution. We used MPI functions MPI_Send and MPI_Recv to send the partial dose buffers from the sub-processes to process 0. Process 0 then writes the completed dose distribution to an output file. Thus instead of many small marshaled single event data transfers during the simulation, we perform one large dose distribution data transfer after the simulation has completed. If the number of nodes is large, the buffers can be efficiently collected and summed up with a single MPI function call: MPI_Reduce.

3. Results
We first measured the amount of time necessary to simulate the plan on a single CPU with the original GEANT4 geometry navigation library and our customized geometry navigation library. Results are listed in table 1.

|                      | Total Time (h:mm:ss) | Events per Second | Speedup Factor |
|----------------------|----------------------|-------------------|----------------|
| Original library     | 110:00:00            | 2.2               | 1.0            |
| Custom library       | 0:48:00              | 320               | 142            |

Our customization of the voxel navigation library resulted in a speed improvement of more than two orders of magnitude reducing the execution time from four days to less than an hour.

Next we compared the performance of TOP-C and our static scheduling parallelization on our cluster. Figure 2 shows the execution speed (in events per second) of the TOP-C and static scheduling versions of the simulation. The TOP-C version peaked at about 12 slaves. The static scheduling version scaled linearly up to 23 nodes. Although we did not have access to a cluster with more than 23 nodes to test, the static scheduling version should be infinitely scalable because there is very little communication among the nodes during the computation.
Figure 2. Execution speed of TOP-C and static scheduling versions of the simulation as a function of the number of slaves. The execution speed of TOP-C peaks at about 12 slaves. With our static scheduling method, we observe a linear increase in execution speed as we add slaves. Because there is almost no network traffic during the simulation, our method should be infinitely scalable.

Figure 3 shows the CPU usage for the TOP-C version of the simulation. The CPU usage on the master reaches 100% at about 11 slaves due to excessive serialized hit data. This result explains the saturation of the TOP-C curve in figure 2. The average CPU usage on the slaves gradually declines as we add more slaves because they are forced to wait before sending their hit data, resulting in low efficiency and poor performance.
Figure 3. CPU usage for the TOP-C version of the simulation as a function of a number of slaves. As we increase the number of slaves with TOP-C, we observe that the CPU usage on the master increases linearly until about 11 slaves. Above 11 slaves, the CPU usage on the slaves gradually decreases as they are forced to wait before sending their serialized hit data.

We confirmed that the dose distribution computed with the original GEANT4 navigation library and the distribution computed with our customized library were exactly the same. As long as the voxel search routine in our customized version of the navigation library returns the same voxel number, the results will be identical. The customized version only requires less iteration. We also checked the dose distribution computed in parallel. By setting random number generator seeds at the beginning of each event, it is possible to recompute the simulation exactly regardless of the number of nodes.

4. Discussion
We have made no attempt to make our system robust. If one of the nodes crashes or loses network connection during the simulation, the program will hang in the sum-up loop and must be restarted from the beginning. TOP-C can handle this situation nicely by reassigning events to other slaves on-the-fly. Our method could be improved to handle a single node failure by reallocating the failed node’s events to the remaining nodes to complete the simulation.

The static scheduling parallel method assumes that the nodes in the cluster are homogenous. If one node is significantly slower, the total simulation speed will be limited by the one slow machine. Static scheduling also assumes that the time to simulate each event is the same on average. If one machine is assigned many events that take more time (e.g. particles of high initial energy), the overall performance will likewise suffer. The particle list could be shuffled to distribute high and low energy particles among the nodes. Although this does not guarantee even load balancing, we have found that in practice each node completes its allocation of events almost simultaneously. Because the initial particle energy spread is fairly small in the kind of dose distribution simulations we are performing, the time to simulate each event is approximately equal.
Task-parallelism employed in TOP-C is based on the assumption that the time to simulate an event (task) is much larger than the time necessary to transmit the result (Cooperman 1996). By modifying the voxel navigation library, this assumption is no longer valid. A single event simulates on the order of 1/100 of a second. The slaves generate so much marshaled data so fast that either the network connection or the capacity of the CPU on the master is exceeded. We therefore propose that static scheduling is better suited for dose simulations.

Message Passing Interface (MPI) is the de facto standard for developing parallel applications. Although it is sometimes referred to as the “assembly language” of parallel programming, it is surprisingly easy to use. Though we were at first daunted with the prospect of abandoning TOP-C and parallelizing GEANT4 ourselves, we found it straight-forward with MPI. We only had to add a few lines of code in our main function to initialize and shut down MPI, a couple of lines in GeneratePrimaries to skip events not assigned to the current process and wrote a short routine to collect and sum up the partial dose distributions. Our parallelization of GEANT4 used just six MPI functions.

Our approach demonstrates the advantage of static scheduling for particle-in-matter Monte Carlo simulations. This is possible because load balancing and fault tolerance are considered low priority. Because each event requires approximately the same amount of time to simulate, load balancing is only expensive overhead for our application. As we are running on a dedicated local cluster, it is rare to see a node crash. Fault tolerance is more important in large scientific simulations involving thousands of nodes spread across continents. Fault tolerance may also be considered more important in a clinical rather than in our research environment.

5. Conclusions
By running the simulation in parallel with our custom geometry navigation library we observed a combined speed improvement of more than three orders of magnitude (about 3,000 times faster) for our melanoma plan. The time necessary to simulate the plan was reduced from several days to a few minutes. Furthermore, our approach should be infinitely scalable by simply adding more nodes. Although there is some overhead at the end of the calculation to sum up the dose distribution buffers, we feel that our method is well suited for many kinds of simulations on a local cluster of PCs.

Acknowledgements
The authors would like to thank Gene Cooperman for his help and guidance. This work was supported by the Japan Science and Technology Agency.

References
[1] Agostinelli S et al 2003 GEANT4—a simulation toolkit Nucl. Instrum. Methods Phys. Res. A 506 250–303
[2] Bulanov S and Khoroshkov V 2002 Feasibility of Using Laser Ion Accelerators in Proton Therapy Plasma Physics Reports Vol. 28, No 5 (2002) 453-456
Fourkal F, Li J S, Xiong W, Nahum A and MA C-M 2003 Intensity modulated radiation therapy using laser-accelerated protons: a Monte Carlo dosimetric study Phys. Med. Biol. 48 (2003) 3977-4000
Malka V et al 2004 Practicability of protonotherapy using compact laser systems Med. Phys. 31 (6) June 2004 1587-1592
[3] Cooperman G 1996 5th. International Symposium on High Performance Distributed Computing (HPDC-5). IEEE Press 141-150
[4] Pacheco P 1997 Parallel Programming with MPI Morgan Kaufman, (San Francisco) ISBN 1-55860-339-5
[5] Negus C 2003 Red Hat Linux 9 Bible Wiley Publishing, Inc. (Indianapolis) ISBN 0-7645-3938-8
[6] Schneider W, Bortfeld T and Schlegel W 2000 Correlation between CT numbers and tissue
parameters needed for Monte Carlo simulations of clinical dose distributions *Phys. Med. Biol.* **45** (2000) 459–478.

[7] Jiang H and Paganetti H 2004 Adaptation of GEANT4 to Monte Carlo dose calculations based on CT data *Med. Phys.* **31** (10), October 2811-2818