Monte Carlo Simulation With The GATE Software Using Grid Computing
Romain Reuillon, David Hill, Christophe Gouinaud, Z. El Bitar, Vincent Breton, I. Buvat

To cite this version:

HAL Id: in2p3-00367374
http://hal.in2p3.fr/in2p3-00367374
Submitted on 12 Mar 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
ABSTRACT
Monte Carlo simulations needing many replicates to obtain good statistical results can be easily executed in parallel using the “Multiple Replications In Parallel” approach. However, several precautions have to be taken in the generation of the parallel streams of pseudo-random numbers. In this paper, we present the distribution of Monte Carlo simulations performed with the GATE software using local clusters and grid computing. We obtained very convincing results with this large medical application, thanks to the EGEE Grid (Enabling Grid for E-science), achieving in one week computations that could have taken more than 3 years of processing on a single computer. This work has been achieved thanks to a generic object-oriented toolbox called DistMe which we designed to automate this kind of parallelization for Monte Carlo simulations. This toolbox, written in Java is freely available on SourceForge and helped to ensure a rigorous distribution of pseudo-random number streams. It is based on the use of a documented XML format for random numbers generators statuses.

Categories and Subject Descriptors

General Terms

Keywords
Monte Carlo, Grid computing, GATE simulation.

1. INTRODUCTION
Monte Carlo simulations (MCS) are widely used in emission tomography; for protocol optimization, design of processing or data analysis methods, tomographic reconstruction, or tomograph design optimization. GATE [1] is a Monte Carlo simulation tool based on the Geant4 package and dedicated to Single Photon Emission Computed Tomography and Positron Emission Tomography simulations. It was designed to be flexible and precise, thus GATE simulations are computer intensive and cannot be used in a clinical context. This work presents a distributing method and a tool for the parallelization of MCS. This method is then applied to a practical application in image reconstruction using GATE and execution times are given for clusters and the EGEE European grid environment.

2. MATERIAL AND METHODS
MCS are commonly considered to be naturally parallel [2]. It is widely assumed that with N processors executing N replicates of a Monte Carlo calculation, the pooled result will achieve a variance N times smaller than a single instance of calculation in the same time [3]. In the next sections we discuss why we changed the default Pseudo-Random Number Generator (PRNG) of the GATE software and will also present how we separate experiments to avoid correlations that could slow down the convergence.

2.1 A. Using GATE with a better Pseudo-Random Number Generator (PRNG)
GATE simulations were initially based on the “James Random” algorithm [4], [5] as implemented in the “Class Library for High Energy Physics” (CLHEP) [6]. This generator is 21 years old and has been shown to have poor statistical properties. We checked that it succeeded in only 36 tests out of 122 using the recent and already well-known statistical test battery “TestU01” of L’Ecuyer [7]. We therefore modified GATE to use the Mersenne Twister 19937 [8] as implemented in CLHEP. This generator is recent, has a huge period of $2^{19937}$ and is equidistributed in 623 dimensions. It passes almost all the tests of the test battery TestU01 and it is fast.

2.2 Parallelization of PRNG
For quantitative Monte Carlo simulations the MRIP or “Multiple Replication In Parallel” parallelization approach ([13], [14]), allows a maximum speed up if many replications of the same experiment have to be made in order to obtain a good approximation of the result. However, when parallelizing the underlying pseudo-random number generator (RNG), correlations within and between the random numbers streams generated in each processor have to be avoided [2]. Different parallel generation techniques of pseudo random numbers can be found in the following documentation: [17]. In the “central server approach”, a central RNG generator provides numbers for all simulation jobs. This approach is the natural one but doesn’t fulfill the requirements for a good parallel RNG [18] and creates a bottleneck that slows down the distributed simulation. The “sequence splitting” or “blocking” consists in splitting the RNG cycle into non-overlapping contiguous sections [19]. This
technique must be used with caution because long range
correlation in the parallelized generator might become short range
inter-sequences correlations. Instead of unrolling the generator,
one might consider to randomly generate states of the pseudo-
random number generator (a status is archiving a precise state).
The average minimal distance between n statuses should be in this
case 1/n times the distributed generator period [20]. This is
possible using a cryptographic generator or a hash function to
generate the statuses. The distribution of the Mersenne Twister
19937 algorithm is achieved that way in the library SPRNG [3].
The “leap frog” technique distributes the sequences to the
processor like a deck of cards to card players. Each process of the
distributed simulation uses 1 number out of n in the original
sequence. This last technique requires a generator that allows
cycle division [1]. With this technique long range correlations in
the original sequence might also become short range inter-
sequences correlations if the interval of sampling in the original
sequence is not chosen carefully [20]. The “independent
sequences” technique produces different cycles of numbers
depending on the initial seed. This technique is available for a few
generators like some lagged Fibonacci pseudo-random number
generators. [19]. This last technique is close to the
“parameterization” technique that might be used with some RNG
like the Mersenne Twister [16] or linear congruential generator
with Mersenne or Sophie-Germain prime moduli [21]. It generates
algorithm parameters leading to the generation of highly
independent random number streams. Within the current state of
the art, we are not able to provide a theoretical proof of
independence between pseudo-random number streams. However,
various approaches can be tested empirically, implying heavy
computation that can be achieved once for many applications
under a precise experimental framework. The Mersenne Twister
19937 has a very long period of $2^{19937}$ drawings. It is an already
parameterized version of the generic Mersenne Twister algorithm
and it has no efficient cycle division technique available. The
Mersenne Twister 19937 generator is well suited to a
parallelization using the “sequence splitting” technique. To
achieve the non-overlapping condition of the “sequence splitting”
parallelization technique, we first estimated the number of random
draws drawn using a simulation job designed to have an
average execution time of 12 hours (on an average working node
of the European grid). This estimation led to 12 billion drawings
per job. Then we generated over 6000 statuses for the Mersenne
Twister spaced by 15 billion numbers each. We used a similar
approach in [15]. The generated status were archived and
converted into a documented XML format, in order to be reused
with different implementations of the Mersenne Twister 19937
algorithm.

2.3 Creation of a generic parallelizing tool for
MCS

We designed and implemented an open source software tool in
Java called “DistMMe”, which is dedicated to parallelize stochastic
simulations. This tool contains a status database and is able to
create jobs for various distributed environments independently
from the random number generation library. It is based on the
intensive use of a documented XML generic format for the
pseudo-random number generator statuses [22]. DistMMe is fully
usable and its sources can be found on the Internet
(http://sourceforge.net/projects/distmne). With this tool, we could
generate GATE jobs for any distributed execution system: basic
scripts (using ssh for instance), bags of work for the European
Grid using JDL descriptors (the European grid Job Description
Language) and more specially “OpenPBS” (using Portable Batch
System scripts). A tutorial is available (www.isima.fr/~reuillon)
as well as 9000 statuses for the Mersenne Twister 19937
algorithm, spaced of 15 billion drawings each.

2.4 Hardware

We could access 650 worker nodes of the EGEE European
computing grid (Enabling Grid for E-sciencE, www.eu-egee.org)
mainly in France, United Kingdom, Netherlands and Poland. We
also had at our disposal two clusters hosted by local research
laboratories (the LIMOS/ISIMA cluster composed of 14 bi-
processors and the LAMI/IFMA cluster composed of 28 bi-
processors managed by an “OpenPBS” system. Each processor of
the cluster is an Intel Xeon 3 GHz with hyper-threading.

2.5 Merging of simulation results

Each simulation generates two binary output files requiring about
10 megabytes of storage space. Simulation output files produced
on the grid were automatically registered and copied on a Storage
Element (SE). When all the simulations were completed, a script
using grid commands retrieved these files from the SE into a local
machine. When the computing was performed on a local cluster,
the retrieving of the simulation output files was achieved using a
regular and local FTP commands (File Transfer Protocol). The
merging of all files was performed using a simple C code and
required less than 5 minutes for less than 30 Gigabytes (on a local
desktop computer – Xeon 3 GHz with simple SATA disk).

3. RESULTS

3.1 PRNG Parallelization

The computation of the 6000 PRNG status is not possible in
parallel and took around 80 days on a single node of the ISIMA
cluster running at 2.4 GHz. Once the list of status is generated, it
is important to test the resulting random number series. Indeed, a
good parallel PRNG must behave like several good sequential
PRNG. Each sequence was then tested using the statistical tests
battery for sequential PRNG TestU01. As shown in Table I, only
2% of sequences failed in more than 5 tests and no sequence
failed in more than 10 tests out of the 122 tests of the battery. This
calculation has been made on the ISIMA cluster and took 35 days
at full cluster load. If the work has been made on a single machine
it would have taken around 3 years.

3.2 Simulation execution

The two local clusters achieved 600 jobs, which all succeeded and
2300 jobs were executed on the EGEE European grid, (with 1811
usable results). Fig. 1 shows how the jobs were executed on the
different calculation units. The IFMA cluster hosted 400 jobs, 200
jobs ran on the ISIMA cluster, 200 on the Polish worker nodes,
499 on the Dutch ones, 922 in England and 190 in France. A
variance study on the final results showed that after the execution of
2000 jobs a convergence was reached. The curve on Fig. 2
shows an asymptotic behavior around 2000 jobs. Executing more
jobs was then un-necessary.

3.3 Distribution of the computing time

The time required for the total execution of the simulation on a
single sequential computation unit (Intel Xeon 3 GHz) is 906
days/CPU (Central Processing Unit).
The execution time of the distributed simulation on clusters is inversely proportional to the number of processors from which they are composed since the migration time of the jobs is neglected compared to the total execution time. Hence, the gain factor was 84 since the number of local bi-processors is 42 (14 bi-processors on ISIMA cluster; 28 bi-processors on IFMA cluster) resulting in 84 execution units running in parallel.

Table 1. Failed tests of the battery testu01 for the 6000 random numbers sequences

<table>
<thead>
<tr>
<th>Number of Failed Tests</th>
<th>Number of Sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>632</td>
</tr>
<tr>
<td>1</td>
<td>1415</td>
</tr>
<tr>
<td>2</td>
<td>1676</td>
</tr>
<tr>
<td>3</td>
<td>1255</td>
</tr>
<tr>
<td>4</td>
<td>289</td>
</tr>
<tr>
<td>5</td>
<td>107</td>
</tr>
<tr>
<td>6</td>
<td>28</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
</tbody>
</table>

![Figure 1. Repartition of the jobs](image)

For the grid the problem is a bit trickier. The execution power of the grid is virtually unlimited, supposing that the number of worker nodes available in our real execution environment was 650. It is greatly inferior to the number of jobs we had to execute. This means, that we could only execute 650 jobs concurrently. This impact negatively on the virtual execution time of our global simulation jobs, increasing it to 70,722 hours by job in average, thus the average practical gain factor is 316.

The end-user might be interested in the global execution time of the simulation corresponding to the time between the submission of the jobs and the return of the results from the last job. Supposing a concurrent submission the simulation ends when the last part of the results is returned. From the log files, the longest job with the longest execution time over all jobs is 36 hours. This leads to a minimal theoretical gain of 621. Taking into account the fact we are limited by the number of the worker nodes the virtual length of a jobs increase to 103,2 hours and the minimal practical gain is about 217. The different gain values are summarized hereafter:

- ideal proportional gain: 1813
- average theoretical gain: 906
- average practical gain: 316
- minimal theoretical gain: 621
- minimal practical gain: 217

4. CONCLUSION AND DISCUSSION

By distributing the calculation on many execution units our nuclear medicine simulation was achieved in a few days. It would have taken more than three years on a single powerful computer without distributing the simulation using the MRIP approach. We have not repeated this simulation to study the grid and cluster overhead, since we may obtain different execution times with different grid/cluster loads. The simulation results were directly used by scientists working in nuclear medicine [10], [11].

Improvements can be made in the following directions: the different gain factors might be improved using more worker nodes...
of the grid and optimization techniques for stochastic simulations distribution like the “N out of M” strategy presented in [23]. Furthermore, each random number sequence has been tested individually with the best test battery presently available, but tests have to be done to check that the correlation between the sequences is acceptable using the parallel PRNG tests described in [1] and implemented in SPRNG. It represents a huge amount of calculation and it will be achieved using the internet computing platform BOINC [12]. The use of the DistMe toolbox requires the downloading of the statuses from the internet and a manual operation to insert them in a database on the local computer running DistMe. To simplify this task for the end user, the statuses and the results will be published via a central web service and DistMe will gain a transparent access to this web service. Last but not least, it might be interesting to optimize the status generation phase, by combining the sequence splitting technique with highly independent random numbers sequences obtained using a “parameterization” technique and then generating the statuses for each sequence in parallel. A pseudo-random number generation library, “DistRNG”, is being implemented and already allow the use of cutting edge parallelization techniques.

5. ACKNOWLEDGMENTS
The authors would like to thank the Auvergne Regional council (France) for its research sponsoring, the EGEV consortium and also ISIMA/LIMOS & IFMA/LAMI laboratories which provided an easy access to their local clusters.

6. REFERENCES