High Performance Monte Carlo Computing with Tripoli®: Present and Future

F.-X. Hugot, E. Brun, M. Fausto, J.-C. Trama, T. Visonneau, T. Lauffenburger

ANS MC2015 - Joint International Conference on Mathematics and Computation (M&C), Supercomputing in Nuclear Applications (SNA) and the Monte Carlo (MC) Method • Nashville, TN • April 19-23, 2015, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2015)

HIGH PERFORMANCE MONTE CARLO COMPUTING WITH TRIPOLI® : PRESENT AND FUTURE

Francois-Xavier Hugot, Emeric Brun, Fausto Malvagi,
Jean-Christophe Trama, Thierry Visonneau,
CEA Saclay - DANS/DM2S/SERMA,
91191 Gif-sur-Yvette, FRANCE
francois-xavier.hugot@cea.fr
Thomas Lauffenburger, C-S

ABSTRACT

Although the Monte Carlo (MC) codes are natural users of the fast growing capacities in High Performance Computing (HPC), adapting production level codes such as TRIPOLI-4® to the hexascale is very challenging. We present here the dual strategy we follow : new thoughts and developments for the next versions of TRIPOLI-4®, as well as insights on a prototype of next generation Monte Carlo (NMC) designed from the beginning with hexascale in mind.

Keywords: Monte Carlo, High Performance Computing, Parallelism, Random Generator, TRIPOLI, Post-processing

1 INTRODUCTION

1.1 The TRIPOLI® Family

TRIPOLI® is the generic name of a Monte Carlo radiation transport codes family dedicated to shielding, reactor physics with depletion, criticality safety and nuclear instrumentation. This family has been continuously developed at CEA since the mid-60s, at Fontenay-aux-Roses first, then at Saclay. The code TRIPOLI-4® [1-2], the fourth generation of the family, is the cornerstone of the CEA Radiation Transport Software Suite, which also includes APOLLO2 and APOLLO3® [3], a lattice and core family of deterministic codes dedicated to reactor physics analyses, MENDEL/DARWIN, a depletion code, NARMER/MERCURE, a photon point-kernel code with buildup factors, CONRAD [4-5] and GALILEE [6] for nuclear evaluation and data processing. TRIPOLI-4® solves the linear Boltzmann equation for neutrons, photons, electrons and positrons, with the Monte Carlo method, in any 3D geometry. The code uses ENDF format continuous energy cross-sections, from various international evaluations including JEFF-3, ENDF/B-VII, JENDL4 and FENDL3. Its official nuclear data library for applications, named CEAV5.1.1, is mainly based on the European evaluation JEFF-3.1.1. TRIPOLI-4® solves fixed source as well as eigenvalue problems. It has advanced variance reduction methods to address deep penetration issues. Thanks to its robust and efficient parallelism capability, calculations are easily performed on multi-core single units, heterogeneous networks of workstations and massively parallel machines. Additional productivity tools, graphical as well as algorithmic, allow the user to efficiently set its input decks. With its large V&V data base, TRIPOLI-4® is used as a reference code for industrial purposes, as well as a R&D and teaching tool, for radiation protection and shielding, core physics, nuclear criticality-safety and nuclear instrumentation.

TRIPOLI-4® is the reference industrial code for CEA (laboratories and reactors), EDF (operating 58 PWRs), and branches of AREVA. It is also the reference code of the CRISTAL [7] Criticality Safety package developed with IRSN and AREVA.

TRIPOLI-4® is available from the NEA DataBank.

Although the Monte Carlo codes are natural users of the fast growing capacities in High Performance Computing (HPC), adapting production level codes such as TRIPOLI-4® to the hexascale is very challenging. We present here the dual strategy we follow: new thoughts and developments for the next versions of TRIPOLI-4®, as well as insights on a prototype of next generation Monte Carlo (NMC) designed from the beginning with hexascale in mind.

1.2 The Developers Team at CEA Saclay

TRIPOLI-4®, APOLLO3®, MENDEL/DARWIN, NARMER/MERCURE and GALILEE are developed by SERMA (Service d'Etudes des Réacteurs et de Mathématiques Appliquées), a 75 permanent staff R&D Unit of the Nuclear Energy Division (DEN) of CEA, whose focus is fission nuclear energy. More than 10 people contribute to TRIPOLI-4®, their activities covering development, V&V, documentation, user support, distribution and licensing. Although fission is the main focus, fusion is also covered, mainly in the context of radiation shielding, with application to TORE SUPRA and INTOR in the Eighties and now to the ITER magnetic fusion program.

2 HPC WITH TRIPOLI-4®

TRIPOLI-4® is mainly written in C++. It contains some parts in C (geometry) and Fortran (evaluation reading). It is written in a portable way and has always been ported without difficulties on new OS and machines (in single or parallel mode). Along the years, it has been ported on various parallel computers (CRAY, DEC/COMPAQ, Blue Gene). It is still being ported on Solaris-Sparc, OSF1, Linux/x64, MacOS. We are running night/week checks with the following compilers: Clang (LLVM), Intel suite, GCC…

2.1 Parallelism Model

The parallelism scheme has been laid in the early stage of TRIPOLI-4 development [8-9]. It uses three kinds of processes:

- a monitor which only does process management
- a scorer which collects batch results
- any number of simulators which track particles grouped in batches and send batch results at the end of the batches.

This scheme is described in Fig.1. Communication is done at each batch and is the bottleneck for score (tally) intensive computation.

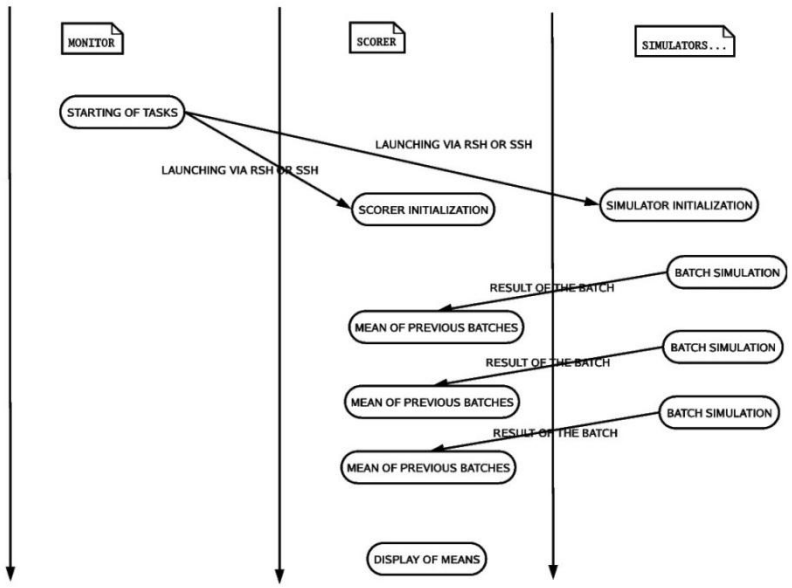

Figure 1: scheme of the parallelism mode of TRIPOLI-4. The time arrow goes downward. Each column describes actions on one type of process (monitor, scorer, simulator). Arrows between processors indicate communication.

The current development version of TRIPOLI-4 has three parallelism modes:

- BSD, the oldest, based on BSD sockets, implements a mailbox scheme for asynchronous communications [8]. It is still widely used on large computers, but is limited to a maximum of about 1000 processors due to a limitation on the select system call. The asynchronous property is especially useful in shielding calculations with weight reduction to provide the load balancing necessary with this kind of simulations.
- MPI has been recently added to TRIPOLI-4®. This port is still in development but has shown good properties in tests up to 65000 processors on the CCRT (CEA DIF).
- The NOCOM mode can use any number of processors without any communications. The results are usually collected through the standard XML output of the code and processed with the R&D tool T4ListXml which will be described later in this paper. This mode can play any parallelism/sequential scheme for testing or debugging. It allows for example to launch the n^{th} task of a simulation without launching the whole calculation and it permits the use of standard debuggers.

An important option for parallelism is the PACKET_LENGTH capability. Although it has been released in TRIPOLI-4 version 7 as one of the most practical way to correct the effect of the intercycle correlations on the variance [14,15], it also has the huge effect of dividing the amount of data sent by the provided length factor. It simply condensates each batch result in groups of length <n>, and sends the grouped result when completed. It allows the user to increase the apparent throughput of his tally collection by one or two orders of magnitude. It can also be seen as a cheap but effective way of doing a two-level collection scheme.

2.2 Random Generators

TRIPOLI-4® provides three different random generators

- In single processor mode the drand48 (standard Unix random generator) provides a simple but full-fledge and well parameterized congruential generator with full length period (due to its additive constant and the nature of the multiplicator) of 2^48. Its 48 bits length words are almost the right amount of bits required to initialize the double precision numbers used throughout the simulation. With TRIPOLI-4®, it provides seeding and checkpoint restarts.
- In parallel mode the GFSR [10-11] (Generalized Feedback Shift Register) is a generator which provides a scheme for initializing full 32 bits words (using delayed replicas) and the leap-frog technique for leaping multiple terms of the random stream very efficiently for use in parallel. It is implemented with a checkpoint restart feature as long as the number of processors is not dramatically changed between the two runs. The current parameterization of this generator has been limited to 1023.
- In parallel mode starting with the next version (10), the Mersenne-Twister (MT19937) [12] will be available in TRIPOLI-4®. Its 32 bits implementation and seeding strategy are directly inspired from the author's implementation. The seeding strategy involves a hash function to initialize the multiple streams of the parallel mode. It provides a full-fledge generator with both single processor and parallel capabilities, checkpoint restart and seeding. Its current implementation is limited to about 1000000 streams by the seeding algorithm.

2.3 Verification&Validation

TRIPOLI-4® has different parallelism V&V suites here ordered by increasing CPU requirement:

- Elementary cases are run in parallel on nightly or weekly basis to check the internal development version.
- All major V&V suites (criticality, shielding, reactor physics, photon transport, geometry etc.) are run on 8 processors at least at each release. As the same calculations are also done in single processor, it yields statistically independent runs which can be checked against one another.
- The Extended Parallelism suite is run at each release from 8 to 1000 processors. This special suite has been set to test a new parameterization of the GFSR generator and to develop the Mersenne-Twister. The results of all these tests are kept and can be reused as a nearly automatized non-regression basis of high precision for new tests.This suite was designed to spot very small discrepancies between versions of code or results with different random seeds, therefore the runs are long enough to reach very low standard deviations (for example down to 1 pcm for criticality safety). A careful selection of 18 cases belonging to the other V&V suites was made to cover the major calculations domains of TRIPOLI-4®. For each case, three kinds of simulations are used:

  - One simulation in single processor, run once for several months in order to obtain very converged results. The checkpoint restart feature was used to avoid losing all results in the case of a network or power failure.

- Simulations in parallel with GFSR at various scales between 8 and 1000.
- Simulations in parallel with the Mersenne-Twister at 64 processors

Then all possible cross comparisons are made to check if all simulations are statistically coherent.

One of the configurations tested in this suite is the GODIVA benchmark (ICSBEP benchmark). The number of histories per batch (cycle) is 50000. The K-effectives (means and relative standard deviations) are presented in table 1 as an illustration

Table 1 : GODIVA Keff in various parallelism configurations

	GFSR / 512 processors	GFSR / 1000 processors	MT19937 / 64 processors
# of batches	72900	77600	53000
K-collision	0.996783 (1.6 pcm)	0.996768 (1.6 pcm)	0.996794 (1.9 pcm)
K-track	0.996786 (1.4 pcm)	0.996778 (1.4 pcm)	0.996788 (1.7 pcm)
K-combined	0.996776 (1.4 pcm)	0.996786 (1.4 pcm)	0.996789 (1.6 pcm)

It is worth noting that the precision of the results here are much more converged than experimental errors and all kind of biases on entry data (cross-sections, compositions …). These precisions are only useful to detect very small biases in code which could be amplified in other configurations. Due to the slow convergence of the Monte-Carlo method, these tests are the only one sufficient to reveal problems with the random generators directly in the code. The K-combined is the best statistical mix of the Track, Collision and Step (generation) K-effectives.

2.4 Parallel Outputs Post-Processing

2.4.1 T4ListXml

TRIPOLI-4® has an XML output of all batch results. In order to process these results, some R&D tools have been developed. One of these tools, named T4ListXml and currently used as an external tool, provides various statistical estimators useful to assess the quality of a TRIPOLI-4® simulation. Among these estimators we can mention: mean, standard deviations, median, sextiles, variance of variance, normality estimators (Kolmogorov-Smirnov, von Mises-Cramer-Smirnov), intercycle correlation estimators, sensitivity estimator.

It also provides various methods to join series coming from different simulations into one series. It can sum up parts of these series in packets to mitigate the effect of intercycle dependencies on variance ; correct these dependencies estimating the correlation with correlation matrices ; estimate the correlation queue with a one-dimension Markov Chain model. It has been extensively used to investigate the correlation on fission sources with TRIPOLI-4® in the study [15]. It can also easily be used to observe the convergence toward the normal law predicted by the Central Limit theorem. It thus gives information about the correspondence between the standard deviation and the confidence interval.

It uses a SAX parser to process the XML format. XML is a well formatted ASCII format and, as such, is well suited to preservation (persistence) of data. However, being ASCII, it can yield voluminous files and the tool can perform “on the fly” decompression using the external tool gunzip. As compression is very CPU intensive, the “on the shelf” tool named pigz can also be used to perform parallel compression.

The main language of the tool is Ocaml (INRIA), a high level French functional language which is fast and well suited for rapid prototyping [22].

The first task of the tool when reading XML files is to serialize the results with the standard (binary) serialization solution of the language to speedup further access to the data. When the processing of the data has been done, the marshalled (serialized) data may be discarded leaving only compressed XML files for long term preservation.

TRIPOLI-4® producing exactly one XML file per process, the tool provides a way to process all the files of a parallel computation in one call. However, the tool can also be used with only one file at each call but using mpirun to process a whole bunch of XML with independent parallelism.

2.4.2 Parallelization/Vectorization of T4ListXml

The CPU intensive core of this tool has recently been redeveloped in C++. We are using the Thrust (NVIDIA) template “library” to parallelize/vectorize the code in a portable and elegant way on multiple “backends”. The currently tested backends are

- GPU (CUDA version 6)
- OpenMP
- TBB (Intel).

The first GPU backend has been tested with success on the Fermi Nvidia GPU card of a simple desktop machine (GF108) and with the Kepler Nvidia GPU cards from the Airain machine (K20) at the CCRT (CEA DIF). For CPU backends, only the TBB backend has shown interesting results (correct scaling). The Mic (Intel) backend, has not been tested with this code, however first tests with other codes using Thrust show good scaling and the backend looks promising.

The Thrust template library uses methods very similar to those in the STL (for example “algorithm”, or “numeric”). All the operations are done on containers like vectors. The algorithm used by the developer indicates to the compiler the dependency on the data. The compiler then decides what parallelization/vectorization scheme should be used. Only minimum hardware calls (and no pragma) are needed into source code. The code is just as clear as in single processor mode and can run in single processor with only cosmetic changes. For backends like TBB or Mic, the Nvidia compiler is not even required (standard Gcc or Mic Intel Compiler can be used for example).

The algorithms that were vectorized in the tool are the following:

- calculation of first moments (means and standard-deviation)
- sorting the series to estimate the median, the sextiles and the empirical repartition function which is very useful for some estimators (normality estimators cited above).

For the GPU backend, it was necessary to begin with a copy of the data on the GPU.

This copy is always a penalty when using GPU and without the need of the sorting task, the GPU would not be competitive.

Another potential issue when dealing with GPU is the size of the memory. On the two tested cards (Fermi and Kepler microarchitectures), the size of the memory is about a few GigaBytes. As the series are sent one by one, this is not considered an issue in this case. A second point to be considered with memory seems to be the allocation time (at least of the very first allocation). In our implementation, some special "design patterns" were created to avoid unnecessary allocations but keeping our code clear, simple and portable.

The time of processing in function of the quantity of data is shown on Fig. 2 The different curves show how this time evolves if the same process is done with more than one serie (which is always the case with standard TRIPOLI-4® calculations). This clearly shows the initialization penalty of the GPU device. Under about 50 series to process a very low number of terms per series, the GPU is not competitive. With several hundred series, the penalty becomes negligible and the GPU is considerably faster than CPU.

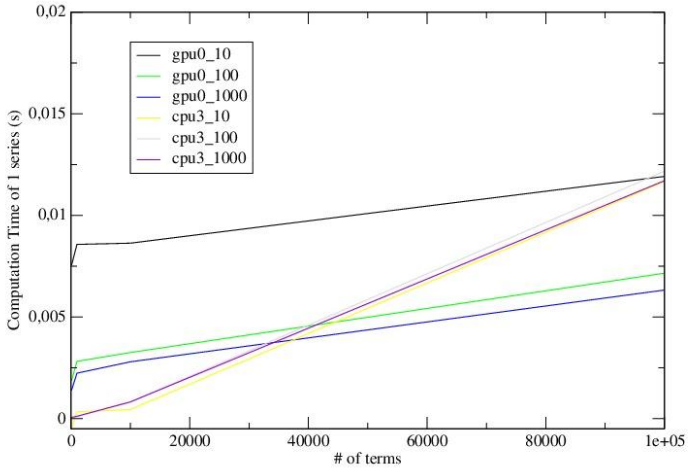

Figure 2: Computational time on GPU and CPU by series depending on the number of terms. Two sets of curves are drawn corresponding to the time of computations vectorized on GPU and to the time of computation in single processor (CPU) mode. In each set, the number attached to the curve is the number of series processed in one calculation. The time plotted is then the total time divided by the number of series of the calculation. The plot shows that the initialization penalty is clear with GPU computations with low number of series (short calculations). With more than 100 series this penalty becomes negligible. For more than 100 series and above fourty thousand terms, the interest of the GPU is clear and increasing.

2.4.3 Map/Reduce Parallel Scheme

Some adaptations of the tool were developed to help processing large data. For testing purpose, a sample of a previous simulation with 2000 processors in NOCOM mode with XML output was used as input of the post-processing tool. The very simple Godiva configuration was used with a

mesh result with sufficiently fine grid. Only the first few hundred batches were simulated to get a precise view of the source convergence phase.

The amount of data produced was 1 TeraByte. This corresponded to 200 MegaBytes of doubles in memory (8 Bytes floating point results) which could not be fitted completely in memory on the test machine.

The options of the tool were enhanced to condensate all the results in a range of batches into one batch and serialized. Using mpirun on 2000 processors took less than 5 minutes to process the TeraByte of data. This phase is often called the "map" phase. The "reduce" phase gathers all the results of the previous phase and publishes the user results (mean, standard deviation, …)

If we compare with the speed of a standard quality disk (take 200 MegaByte/second if data is contiguous on disk), the whole processing would take more than one hour just to read the data. A coefficient of around 10 can be counted to take into account the effect of reading compressed ASCII data (even XML) instead of just loading data into memory.

The efficiency of the tool has not been thoroughly investigated, but it has already been shown that its modularity allows fast post-processing of large quantity of XML which would be painful to process in single processor mode, or would not even fit into memory. It is thus useful to users as a module for the post-processing of a simulation. We also want to emphasize the fact that splitting calculations schemes into two phases (one map, one reduce) with a minimum possible reduce phase, enables modular schemes in which the map phase can be easily parallelized. The reduce phase, introducing concurrency, is often more difficult to parallelize and should just be kept minimal. However, some reducing algorithms are also known to be enhanced by adding more processing to break dependencies between data (eg sorting, see also reduce_by_key).

3 HPC WITH NMC

3.1 The New Monte Carlo (NMC) Prototype

TRIPOLI-4® being a complete and complex code of about $5x10^5$ lines of instructions, its structure is rather unyielding to serve as a basis for experimenting with different logical architectures in order to optimize the execution on hexascale computers. The decision has thus been taken to develop a prototype (or mini-application) complex enough to be representative of a real simulation tool and at the same time conceived to be easy to adapt.

The NMC prototype is entirely written in C++11 but meant to be used in Python via a SWIG-generated interface for all user–accessible objects. It is object oriented and makes heavy use of polymorphism in order to always allow the choice between competing algorithms: as an example, in NMC one can mix isotopes with pre-computed Doppler-broadened cross sections and isotopes with on-the-fly Doppler broadening. Prototyping of NMC is first performed in Python for agile programming.

The physics of NMC is simplified. Two types of particles are transported: mono-kinetic particles (MKparticles) and neutrons. MKparticles travel at constant speed and they suffer only three types of collisions: absorption, scattering and branching. The angular distribution of the outgoing particles, whether coming from diffusion or branching is always isotropic.

For the neutron transport only two interactions have been implemented: radiative absorption (MT=102 in ENDF-speak) and elastic scattering (MT=2). On the other hand, the cross sections

come from ACE files and scattering anisotropy is taken fully into account: this allows for correct simulation of the two isotopes H1 and He4 (and this has been verified with respect to TRIPOLI-4®). All other isotopes can be uploaded, but the simulation will be unphysical.

The sigma-1 [16] method of Doppler broadening has been implemented for an on the fly interrogation at each cross section request, starting from a user-defined temperature (usually 0°K or 300°K, assuming those temperatures are available). It can be assigned on an isotope-by-isotope policy. It has been verified by comparisons with NJOY.

Geometry is simplified too. Only two specialized geometries have been implemented: an infinite homogeneous medium and a slab with an arbitrary number of heterogeneous regions. However, more complex and realistic geometries can be dealt with using the ROOT [17] geometric tracking package which has been linked to NMC.

Scoring has been separated from the simulation: a particle is tracked from birth to death and its history recorded in a dedicated object. This history is then passed to the scorer for actual tally computation. For the moment the only available score is the volumetric flux spectrum, with the two estimators track and collision.

Only fixed-source calculations are performed for now.

3.2 Two Level Parallelism

NMC has been conceived from the start to support two levels of parallelism. The first level with distributed memory has been implemented in MPI. The second level with shared memory has been implemented with OpenMP and native C++ threads. The performances of the two different libraries are equivalent in the cases tested.

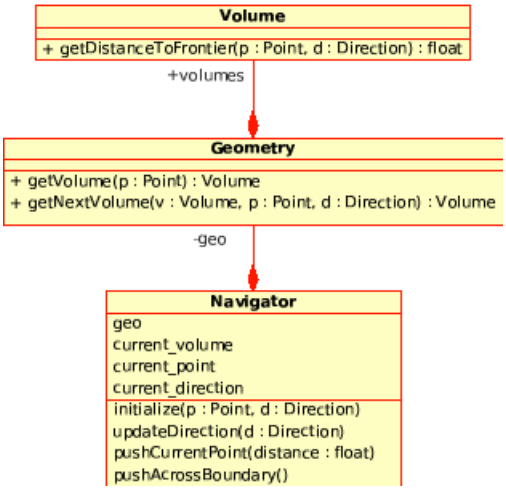

Figure 3: the geometry navigation class hierarchy

In order to achieve good performances and avoid race conditions in shared memory, care must be taken in separating non-mutable data in objects that will be shared from mutable data which is encapsulated in duplicated objects. This has been done in the geometry, were we use a non-mutable geometry which contains most of the data and which answers the usual tracking questions of a) in which volume is a point given its coordinates; b) given a volume, a point inside

it, and a direction of flight at what distance is the next bounding surface; c) given that a boundary is crossed, which volume is on the other side. This shared object is pointed to by thread-specific navigators which bufferize temporary data like current point and current volume, which are needed to optimize tracking and to keep a generic interface. This is illustrated by Fig.3. The same logic has been applied to the cross sections, where shared objects contain the data read from the ACE files and duplicated objects contain temporary data like the current neutron energy, current shocked isotope, etc.

In our approach, which we carried over from TRIPOLI-4®, all simulations are divided in batches, even if in fixed-source simulation all particles are independent (one million histories are typically simulated as 1000 cycles of 1000 particles). This allows for robust estimation of confidence intervals, since the cycle tally values are averages which are supposed to be normally distributed. In our multi-threading implementation, the particles of the cycle are dispatched in equal number to the available threads and the scores are computed sequentially at the end of each cycle (this will change shortly for distributed tally calculations). Each thread has its own random number generator, independently initialized at the beginning of the simulation. This parallelization procedure is deterministic, thus assuring reproducibility and facilitating debugging. Load balance has not proven a problem so far.

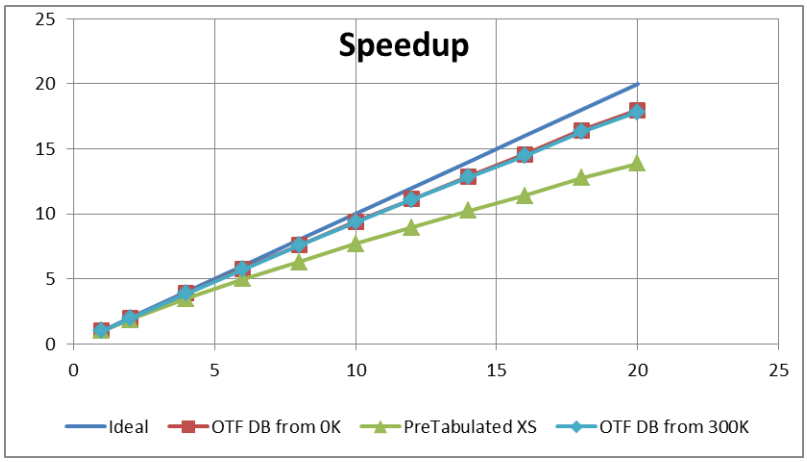

Figure 4 : NMC speedup for a slowing down problem in U238 as a function of the number of threads

Figure 4 shows the speedup of NMC as a function of the number of threads used on a cluster where each node is made of two Intel Xeon E5-2680 CPUs clocked at 2,8 Ghz with 10 cores each. The problem is a slowing down from a 2 MeV source in an infinite medium composed of all the 390 isotopes of the ENDF-BVII.0 library at 900°K; the main components of the mixture are H1 and U238 in order to have a classical PWR spectrum, the other isotopes intervening as trace elements. The three curves with markers refer to a calculation using cross sections pre-tabulated at 900°K (green line), on-the-fly Doppler broadening from 0°K cross sections (red line), and on-the-fly Doppler broadening from 300°K cross sections (blue line). The blue line represents the ideal perfect scaling. We see that the on-the-fly broadening scales very well as a function of the number of threads, while the pre-tabulated cross sections scales somewhat worse with a speedup of 14 for 20 threads. This is consistent with the results reported by other teams. [18].

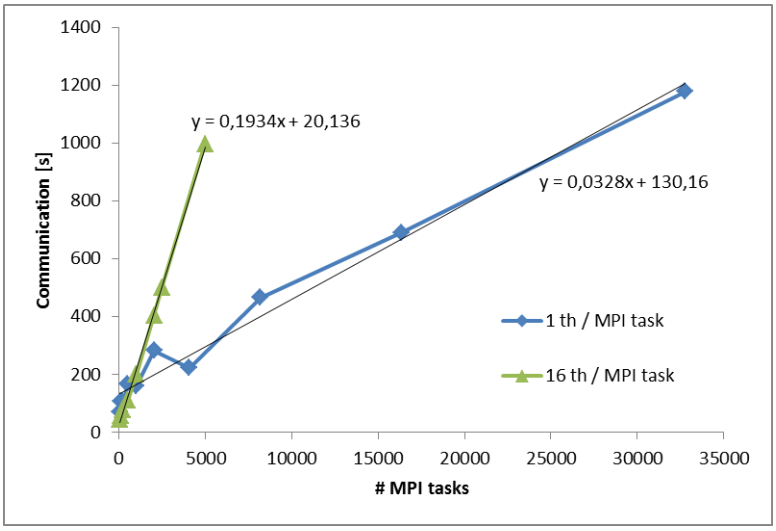

Figure 5: scaling of communication time as a function of the number of MPI tasks

When distributing the simulation with MPI, each MPI process executes an independent simulation; at the end of the run the results of all simulations are aggregated via two calls to MPI reduce (once for the mean and once for the variance calculation). This simulation has successfully run over the 5000 nodes of the TGCC, each equipped with two 8-cores Intel Xeon E5-2680 for a total of 80000 processors. Figure 5 shows the communication time as a function of MPI processes; the green line shows the results for 16 threads per MPI task (one task per node and one thread per core) while the blue line shows the results for one thread per MPI task (and consequently 16 MPI tasks per node). About one thousand scores, relating to the flux spectrum, are computed. We see that the communication time scales linearly with the number of processes. The scaling with one thread per MPI process (blue line) is more effective (less steep) since the total communication is partially intra-node for the 16 processes running on the same node. Of course a linear scaling is not acceptable for a large number of nodes and a better scaling will be sought, ideally logarithmic. In any event, when the number of scores increases, like in full core calculations, the tallying approach must be profoundly revisited by the use of tally servers.

3.3 Future Work

Now that the parallelization of the simulation appears to give good results, the next step is the treatment of criticality calculations, where the fission bank needs to be efficiently distributed over several nodes. The work done in OpenMC [19] is in our opinion one of the best approaches to the issue.

On-the fly Doppler broadening other than the sigma-1 methods have been suggested and it is our intention to test them in NMC.

The big issue of course is the treatment of the terabytes of tallies needed in full-core depletion calculations. The use of the concepts of tally servers and domain decomposition is certainly a must, but the specific implementations are still much under research [20-21]. We hope to communicate about NMC performances on the issue soon.

For the moment we have concentrated on building the prototype NMC in CPU architectures. It is our intention to extend our investigations also to recent accelerator architectures such as GPUs and Intel MICs.

4 CONCLUSION

In this article, we have exposed our strategy to adapt the TRIPOLI code to hexascale computing. This strategy is twofold: we investigate new methods in the new Monte-Carlo code NMC designed for rapid prototyping, and we work on TRIPOLI-4 directly to adapt its structure to achieve hexascale computations. Eventually, new methods developed in NMC which prove successful will be integrated in TRIPOLI-4.

We also have exposed the CPU intensive part of the development process of TRIPOLI which is the parallelism V&V suite. This suite was used to check the fundamental part of a Monte-Carlo computation which is the new random number generator of the code (Mersenne-Twister 19937). TRIPOLI's random number generators were also presented in this article.

5 ACKNOWLEDGMENTS

We gratefully acknowledge EDF longtime partnership with TRIPOLI® as well as support from AREVA.

6 REFERENCES

1. TRIPOLI-4® Project Team, "*TRIPOLI-4® User Guide*", CEA R 6316, Saclay, 2012
2. TRIPOLI-4® Project Team, "TRIPOLI-4® CEA, EDF and AREVA Reference Monte Carlo Code", *SNA+MC 2013*, Paris (2013)
3. R. Sanchez et al., "APOLLO2 Year 2010", *Nuclear Engineering and Technology*, **V42**, No 5 (2010)
4. C. De Saint Jean et al., *International Conference on Nuclear Data for Science and Technology*, Nice (2007)
5. P. Archier, C. De Saint Jean et al., *International Conference on Nuclear Data for Science and Technology*, New-York City (2013)
6. M. Coste Delclaux, "GALILEE: A Nuclear Data Processing System for Transport, Depletion and Shielding Codes", *PHYSOR*, Interlaken (2008)
7. J.M Gomit et al., "CRISTAL Criticality Package 12 Years Later and New Features", *ICNC*, Edinburgh (2011)
8. Y. Peneliau, J.P. Both, "Parallelization of the Monte Carlo Code TRIPOLI-4®", *Mathematical and Computation, Reactor Physics and Environmental Analysis in Nuclear Applications*, Madrid (1999)

9. J.C. Trama, F.X. Hugot, "TRIPOLI-4®: Parallelism Capability", *Transactions of the American Nuclear Society*, **V97**, p. 522 (2007)
10. S. Aluru, G.M. Prabhu, and J. Gustafson, "A Random Number Generator for Parallel Computers", *Parallel Computing 18* (1992)
11. T.G. Lewis, and W.H. Payne, "Generalized Feedback Shift Register Pseudorandom Number Algorithm", *Journal of the Association for Computing Machinery*, **V20**, No. 3 (1973)
12. M. Matsumoto, T. Nishimura, "Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator", *ACM: Transaction on Modeling and Computer Simulations: Special Issue on Uniform Random Number Generation* (1998)
13. Lee Y.K., Hugot F.X., "Verification Tool for TRIPOLI-4® Criticality Code Validation", *ICNC*, Saint-Petersbourg (2007)
14. F.X. Hugot, Y.K. Lee, F. Malvagi, "Recent R&D around the Monte-Carlo code TRIPOLI-4® for Criticality Calculation", *PHYSOR*, Interlaken (2008)
15. N. Petrov, N.P. Kolev , G. Todorova, F.X. Hugot, T. Visonneau, "TRIPOLI-4® Solutions of VVER-1000 Assembly and Core Benchmarks", *M&C*, Saratoga Springs (2009)
16. D.E. Cullen, C.R. Weisbin, "Exact Doppler broadening of tabulated cross sections," *Nuclear Science and Engineering*, **V60**, pp.199-229 (1976).
17. R. Brun, F. Rademakers, "ROOT- an object oriented data analysis framework," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, **V389**, pp.81-86 (1997).
18. J.R. Tramm, *et al.*, "XSBench, "The development and verification of a performance abstraction for Monte Carlo reactor analysis," *PHYSOR*, Kyoto (2014).
19. P.K. Romano, B. Forget, "Parallel Fision Bank Algorithms In Monte Carlo Criticality Calculations," *Nuclear Science and Engineering*, **V170**, pp.125-135 (2012).
20. N. Horelik, *et al.*, "Monte Carlo Domain Decomposition for Robust Nuclear Reactor Analysis," *Parallel Computing*, **V40**, pp.646-660 (2014).
21. P.K. Romano *et al.*, "On the Use of Tally Servers in Monte Carlo Simulations of Light-Water Reactors," *SNA + MC*, Paris, (2013).
22. http://caml.inria.fr (Ocaml web site)