M&C 2017 - International Conference on Mathematics & Computational Methods Applied to Nuclear Science & Engineering, Jeju, Korea, April 16-20, 2017, on USB (2017)

PATMOS: A prototype Monte Carlo transport code to test high performance architectures

Emeric Brun*, Stéphane Chauveau†, Fausto Malvagi*

*Den-Service d'Etudes des Réacteurs et de Mathématiques Appliquées (SERMA), CEA, Université Paris-Saclay, 91191 Gif-sur-Yvette, FRANCE
†Nvidia Development France
Corresponding author: emeric.brun@cea.fr

Abstract - *PATMOS is a prototype for Monte Carlo neutron transport under development at CEA. It is dedicated to the testing of algorithms for high performance computations on several modern architectures. The final goal is to demonstrate the feasibility of pin-by-pin full core depletion calculations for large nuclear power reactors with realistic temperature fields. In this paper we show PATMOS hybrid parallelism performances on Intel Xeon and Xeon Phi recent architectures. We also show the vectorisation of the SIGMA1 algorithm for use in our on-the-fly Doppler broadening and its performances. Results with offloading to GPUs in heterogeneous architectures are also presented.*

I. INTRODUCTION

A new prototype (or mini-app) PATMOS for Monte Carlo neutron transport is under development at CEA. PATMOS is complex enough to be representative of a real simulation code and at the same time conceived to be easy to change and adapt. This prototype is dedicated to the testing of algorithms for high performance computations on different modern architectures. The final goal is to perform pin-by-pin full core depletion calculations for large nuclear power reactors with realistic temperature fields.

II. DESCRIPTION OF THE PATMOS PROTOTYPE

PATMOS 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 PATMOS one can mix nuclides with pre-computed Doppler-broadened cross sections and nuclides with on-the-fly Doppler broadening.

Prototyping of PATMOS is first performed in Python for agile programming. So there actually are two versions of PATMOS, one in Python and one in C++. Verification tests assure that the two versions give the same results.

A. Physics

The physics of PATMOS is still simplified. Two types of particles are transported: mono-kinetic pseudo-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. They are mainly used for analytic benchmarks.

For neutron transport, the Python version of PATMOS implements all reactions except the thermal laws scattering data and the unresolved resonance range. The results have been verified by comparisons with TRIPOLI-4® [1] and MCNP5 [2].

In the C++ version, only a subset of physical interactions has been implemented: elastic scattering (MT=2 in ENDF terminology), discrete inelastic scattering (MT=51-90), absorptions (MT=102-117) and fission (MT=18). The cross sections come from ACE files and the scattering anisotropy is fully taken into account: this allows for correct simulation of a number of non-fissile nuclides up to the eventual (n,2n) threshold. All other nuclides can be uploaded, the simulation runs but the results will be unphysical. Exiting neutron distributions from fission are limited to a Maxwellian with a fixed $\bar{\nu}$ value.

A number of methods for cross section access has been tested [3]: binary search, N-ary search, hash tables, fractional cascading and unified grid.

The SIGMA1 method of Doppler broadening [4] has been implemented as an on the fly interrogation at each cross section request, starting from a user-defined temperature (0K and 300K have been tested). It can be assigned on an isotope-by-isotope policy.

Geometry is simplified also. Only a slab geometry with an arbitrary number of heterogeneous regions has been implemented in native mode. However, more complex and realistic geometries can be dealt with by using the ROOT [5] geometry tracking package which has been linked to PATMOS. This option has been used to model the Hoogenboom-Martin benchmark problem [6].

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.

Both fixed-source calculations and criticality calculations can be performed.

B. Hybrid parallelism

PATMOS has been conceived from the start to support two levels of parallelism. The first level, corresponding to the distributed memory parallelism relies on MPI. The second level, corresponding to the shared memory parallelism has been implemented with three different technologies: OpenMP [7], native C++11 threads and Intel TBB [8]. The performances of the three different libraries are equivalent in the cases tested.

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 light objects meant to be duplicated for each thread. This has been done in the geometry, where the non-mutable geometry is separated from the thread-specific navigator which buffers temporary data like current point and current volume, which are needed to optimize tracking and to keep a generic interface. 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 isotope, etc.

In our approach, which we have carried over from TRIPOLI-4®, all simulations are divided in batches (or cycles in criticality calculation), even if in fixed-source simulations all source particles are independent (for instance, one million histories are typically simulated as 1000 batches of 1000 particles). This allows for robust estimation of confidence intervals, since the batch tally values are averaged values whose distribution converges to the normal distribution. Because of their size, the only non duplicated mutable structures are the scores that are shared objects concurrently modified through OpenMP atomic adds.

In our first multi-threading implementation, the particles of the batch/cycle were dispatched in equal number to the available threads which simulate the particle history and compute its contribution to the score. This parallelization procedure is deterministic, thus assuring reproducibility and facilitating debugging. As an option one can now choose dynamic dispatching of particles to the available threads: this assures load balancing, but in this case the result is non-deterministic. Each thread has its own random number generator, independently initialized at the beginning of the simulation.

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 MPI reduction operations (a call to MPI All_reduce for computing the mean followed by a call to MPI Reduce for the variance calculation).

1. Vectorization

We have chosen the SIGMA1 method for on-the-fly Doppler Broadening (OTF) because it is the reference method used in NJOY and also because it has a good potential for vectorization. Since modern computer architectures heavily rely on vectorization to achieve performances, it is essential for simulation codes to exploit the vector units in order to achieve good performances. Vectorization of the SIGMA1 method has required a few rearrangements of the loops, after which good gains have been observed (see next section).

III. NUMERICAL RESULTS ON DIFFERENT ARCHITECTURES

For the numerical results we use two test cases. The first case, called PointKernel, is a slowing down problem from a 2 MeV source in an infinite medium composed of all the 390 nuclides of the ENDF-BVII.0 library at 900K; the main components of the mixture are H1 and U238 in order to have a classical Pressurized Water Reactor (PWR) spectrum, the other nuclides intervening in small amounts. In this case the computed score is a multigroup flux spectrum with either 10^3 or 10^7 groups. All the times given in the following results are for an average "cycle", which is the simulation of the full history of a number of neutrons (usually 10^4 or 10^5, to keep the simulation time reasonable).

The second test case is the Hoogenboom-Martin benchmark [6], an idealized full Pressurized Water Reactor core with 241 identical 17×17 assemblies composed of 264 identical fuel pins. The fuel composition is representative of a fuel burnup of 24GWd/tHN, but only the 34 most important actinides and fission products have been retained. The objective of the benchmark is to compute the fission power in each of the 100 axial slices of each fuel pin.

PATMOS can run on either Intel Xeon CPUs (we have made performance studies on Sandybridge, Ivybridge, Haswell and Broadwell) and Intel Xeon Phi (MIC) processors (we tested both Knights Corner KNC and Knights Landing KNL). For now, the same source code is used for both Intel lines without platform specific optimizations.

Thanks to a collaboration with Nvidia, a version of PATMOS has been developed where all the neutron simulation is run on CPU and the on-the-fly Doppler broadening is offloaded to one or more GPUs (Nvidia Kepler line K40 and K80 have been tested).

1. Shared memory parallelism

A. Strong scalability

The shared memory parallelism is tested by performing a strong scalability study on the PointKernel test case. This consists in looking at the speedup by keeping constant the amount of work to be done (in our case the number of simulated particles per cycle) and increasing the number of CPU cores utilized. The ideal curve is linear in the number of resources used.

Figure 1 shows the scalability of PATMOS for the PointKernel test case with pre-tabulated cross sections at 900K on three Intel Xeon Broadwell (BDW) machines:

- Blue curve: Intel Xeon E5-2690v4: 2×14 cores at 2.6GHz, Cluster-On-Die not enabled, Turbo mode enabled, Hyperthreading enabled.
- Green curve: Intel Xeon E5-2680v4: 2×14 cores at 2.4GHz, Cluster-On-Die enabled, Turbo mode disabled, Hyperthreading disabled.
- Orange curve: Intel Xeon E5-2697v4: 2×18 cores at 2.3GHz, Cluster-On-Die enabled, Turbo mode disabled, Hyperthreading disabled.

We see that the efficiency when all cores are used can vary from 68% to 85% depending on the machine and its global settings (turbo mode, Cluster-On-Die, ...). Analyzing the effect of each setting will require further investigations. When using pre-tabulated cross sections, the degradation of the efficiency with the increasing number of cores is due to the

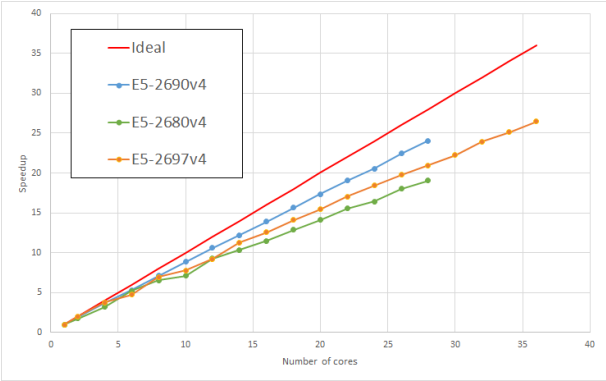

Fig. 1. Strong scalability study on several Intel Xeon Broadwell machines for the PointKernel test case with pre-tabulated cross sections at temperature.

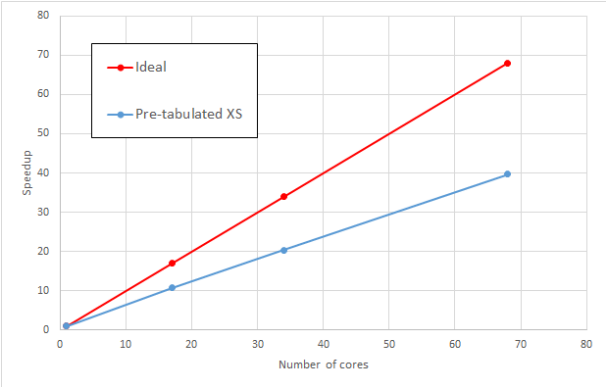

Fig. 2. Strong scalability study on Intel Xeon Phi (KNL) for the PointKernel test case with pre-tabulated cross sections at temperature.

fact the simulation is memory latency bound (caches misses induced by the binary search).

The same study has been performed on an Intel Xeon Phi KNL 7250 with 68 cores at 1.4GHz with up to 4 threads per core (see Fig. 2. The 16GB high bandwidth memory, MCDRAM, is used as a L3 cache and the Quadrant cluster mode is enabled. In this case, we have an efficiency of 53% for the pre-tabulated cross sections simulation.

All the results shown in the remainder of this article concerning the BDW architecture have been obtained on the E5-2690v4 machine.

B. Effect of hyperthreading

The hyperthreading has been enabled on the BDW machine so that each core can run up to 2 threads. Table I shows the influence of hyperthreading on the time necessary to run the same amount of tracked particles. Here, we only show results for the pre-tabulated cross sections simulations. We obtain a gain of 1.7 with 2 threads per core.

Table II shows that the hyperthreading is very effective on the KNL architecture. Indeed, the duration of a cycle is

Nb MPI ranks	threads/core	Cycle time (s)
1	1	14.02
1	2	8.37
2	1	13.93
2	2	8.27

TABLE I. Hyperthreading effects on the Broadwell E5-2690v4 for the PointKernel test case and use of pre-tabulated cross sections

Nb MPI ranks	threads/core	Cycle time (s)
1	1	48.22
1	2	30.01
1	3	23.27
1	4	18.44
4	1	48.77
4	2	29.43
4	3	22.75
4	4	17.88

TABLE II. Hyperthreading effects on KNL for the PointKernel test case and use of pre-tabulated cross sections

divided by a factor 2.7 with 4 threads per core compared to 1 thread per core.

The BDW machine being a bi-socket with 2 NUMA (Non Uniform Memroy Access) domains and the KNL machine being in Quadrant mode, we expected an efficiency gain going from one to two MPI ranks on BDW and a slight gain on KNL going from one to four MPI ranks. Table I and Table II show that this is not the case.

C. Dynamic dispatching of particles

Particles of a cycle are dispatched to all the available threads. This dipatching can be static, all particles are equally distributed to all the available threads at the beginning of the cycle, or dynamic, particles are divided in groups. In our implementation, there are 100 more groups than threads. A dynamically scheduled OpenMP parallel loop takes care of the load balance.

Table III shows no strong effects of dynamic scheduling when using pre-tabulated cross sections.

Machine	Static dispatch.	Dynamic dispatch.	Gain
BDW	8.37	7.99	1.05
KNL	18.41	17.88	1.03

TABLE III. Effect of dynamic dispatching of particles on KNL and BDW for the PointKernel test case and use of the pre-tabulated cross sections at temperature. Configuration with 1 MPI rank and 48 threads for BDW and 272 threads for KNL.

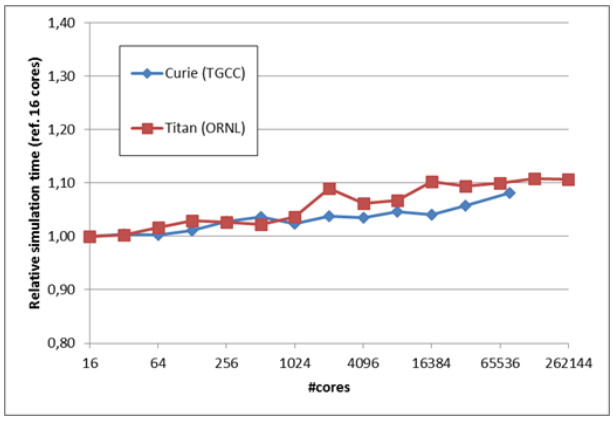

Fig. 3. Weak scalability study for the PointKernel test case on the CURIE machine and the TITAN machine.

2. Distributed memory parallelism

The distributed memory parallelism is tested by looking at the weak scalability. This is measured by doubling the amount of work each time one doubles the number of cores used. The ideal curve is flat versus the number of resources used.

Figure 3 shows the weak scalability for the PointKernel test case with 10 million scores (representative of a pin-by-pin power map of PWR core with 100 axial zones). The simulations have been performed on the CURIE machine (Très Grand Centre de Calcul, CEA) and on the Oak Ridge TITAN machine. The penalty is only 10% when running at 79,200 cores on CURIE and 262,144 cores on TITAN.

We have recently run the Hoogenboom-Martin benchmark on CURIE up to 65,000 cores with again a 10% penalty (see Fig. 4).

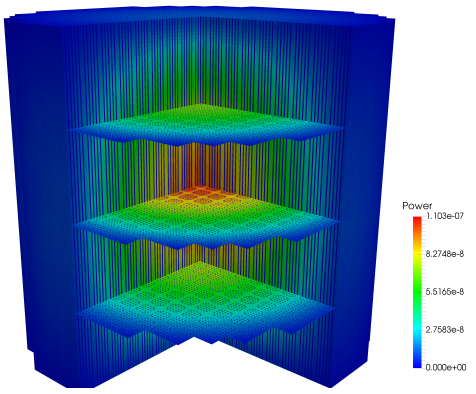

Fig. 4. Pin-by-pin fission power map for the Hoogenboom-Martin benchmark.

3. Acceleration of the SIGMA1 method

A. Vectorization

The SIGMA1 method used for on-the-fly Doppler broadening slows down the computing time by a factor of 10 when compared to simple table look-up on pre-broadened cross sections. This figure seems high, but we have been able to count the floating point operations (by analyzing the instruction mix given by Intel Software Development Emulator) of two simulations with pre-tabulated cross sections and on-the-fly Doppler broadening with the SIGMA1 algorithm. The number of floating point operations is 150 times higher for on-the-fly Doppler broadening compared to pretabulated cross sections simulations. This highlights the fact that in the standard tabulated search, most of the time of the floating point units is spent waiting for data to come.

In order to vectorize the main loop of the SIGMA1 algorithm, we resorted to manual software pipelining. The original algorithm looks, in a simplified way, like this:

```
for (i=0; i<n; i++) {
  Fa=F(i)
  Fb=F(i+1)
  H = Fb - Fa
  SIGMA += G(i,H)
  }
```

where the function $G(i, H)$ has the interesting property of being linear in the second variable, that is $G(i, X + Y) = G(i, X) + G(i, Y)$. The fact that both indexes i and $i + 1$ are present in the loop, prevents the compiler from vectorizing it. A standard technique is to reshuffle the computations by moving the $i + 1$ part to the next iteration:

```
Fa=F(0)
SIGMA -= G(0, Fa)
for (i=0; i<n-1; i++) {
  Fb=F(i+1)
  SIGMA += G(i,Fb)
  Fa=F(i+1)
  SIGMA -= G(i+1,Fa)
  }
Fb=F(n)
SIGMA += G(n-1, Fb)
```

A further simplification can be achieved by noting that now $Fa = Fb$ inside the loop, and so the final version looks like:

```
Fa=F(0)
SIGMA -= G(0, Fa)
for (i=0; i<n-1; i++) {
  Fa=F(i+1)
  SIGMA += G(i,Fa) - G(i,Fb)
  }
Fb=F(n)
SIGMA += G(n-1, Fb)
```

One of the F functions mentioned above calls the erfc (complementary error) function. In order to achieve vectorization, the compiler needs a version of erfc callable in a vectorized loop. This is the case for Intel compiler via the Short Vector Math Library (SVML). The loop can then be vectorized either by resorting to an OpenMP SIMD directive or even auto-vectorization since Intel 2016.

In order to analyze the performance of the vectorization on BDW and KNL, three binaries of PATMOS have been produced with different compiler switches:

	BDW (s/cycle)	KNL (s/cycle)
Binary search	8.37	18.41
OTF	213.76	295.01
OTF -lm	106.85	197.82
OTF vectorized	36.39	44.56
Vectorization gain	2.94	4.44

TABLE IV. PointKernel test case on Xeon (Broadwell) and Xeon Phi (KNL). Results with pre-tabulated cross sections, non-vectorized SIGMA1 and vectorized SIGMA1. The maximal theoretical vectorization gain is 4 for Broadwell and 8 for KNL.

- OTF vectorized : "-O2 -xMIC-AVX512" on KNL and "-O2 -xCORE-AVX2" on BDW
- OTF: vectorization disabled by "-no-vec -no-simd -qno-openmp-simd"
- OTF -lm : vectorization disabled and "-lm" compiler switch to use the GNU Math library instead of the Intel Math library for the scalar version on the erfc function. This was done because the Intel erfc function appears to be four times slower than the GNU version.

Cycle times obtained with these three different version of the binary are reported in Table IV. The vectorization gain is the ratio between the vectorized SIGMA1 cycle duration (OTF vectorized) and the non-vectorized version of SIGMA1 resorting to GNU Math library (OTF -lm). The vectorization of SIGMA1 main loop has permitted a speed-up of a factor of 2.9 on the Broadwell machine (out of a maximum of 4) and of almost 4.5 on the KNL (out of a maximum of 8) (see Tab. IV). The vectorization gain is limited, especially for the KNL, by the instructions of the SIGMA1 algorithm which are not inside the (vectorized) loop.

The penalty for the on-the-fly Doppler broadening is now a factor of 4 on CPU and a factor of 2 on KNL compared to the use of pre-tabulated cross sections.

B. Strong scalability

Figures 5 and 6 show the strong scalability results for the simulation with on-the-fly Doppler broadening on BDW and KNL, respectively. The efficiency is much higher than for the pre-tabulated cross sections: we get close to 90% for the BWD and 97% for the KNL using all the cores (but no hyperthreading). This is due to the fact that now the simulation is much more computing intensive, and the cores can perform more floating points operations with the data in cache.

C. Effect of hyperthreading

Hyperthreading is less effective for simulations with on-the-fly Doppler broadening. The speedup is now of only 1.47 on BWD with two threads per core, and of 2.2 on KNL with four threads per core. Those results are consistent with the fact that the simulation is more computing-intensive, and there is less to gain by switching thread while waiting for data.

Again, from table V, we do not see NUMA effects on BDW. On the other hand, table VI shows a 15% improvement

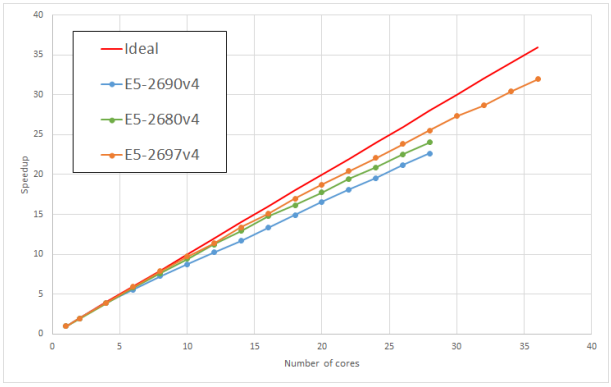

Fig. 5. Strong scalability study on several Intel Xeon Broadwell machines for the PointKernel test case with on-the-fly Doppler broadening from 0K.

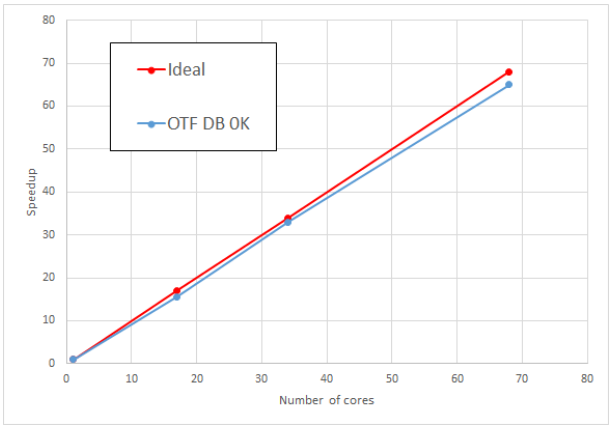

Fig. 6. Strong scalability study on Intel Xeon Phi (KNL) for the PointKernel test case with on-the-fly Doppler broadening from 0K.

Nb MPI ranks	threads/core	Cycle time (s)
1	1	53.70
1	2	36.39
2	1	53.29
2	2	36.11

TABLE V. Hyperthreading effects on Broadwell for the PointKernel test case and on-the-fly Doppler broadening.

Nb MPI ranks	threads/core	Cycle time (s)
1	1	98.37
1	2	60.26
1	3	51.43
1	4	44.56
4	1	103.23
4	2	59.75
4	3	45.74
4	4	38.70

TABLE VI. Hyperthreading effects on KNL for the PointKernel test case and use of on-the-fly Doppler broadening.

Machine	No prefecth.	Prefetch.	Gain
BDW	36.39	36.12	1.00
KNL	44.56	41.30	1.08

TABLE VII. Effect of memory prefetching on KNL and BDW for the PointKernel test case and use of on-the-fly Doppler broadening. Configuration with 1 MPI rank and 48 threads for BDW and 272 threads for KNL.

when going from one to four MPI rankon the KNL.

D. Memory prefetching

Once the energy grid indexes corresponding to the upper and lower bounds of the Doppler integral have been computed, it is possible to prefetch these memory segments in the last level cache. This can be done while the half fist iteration outside of the vectorized loop is computed. results with memory prefetching are reported in table VII. One can see that there is no gain on the BDW machine but a 8% on the KNL.

E. Dynamic dispatching of particles

Dynamic dispatching of particles has a bigger effect (10% on KNL) with on-the fly Doppler broadening, as shown in table VIII. This is due to the fact that a neutron simulation time now has a big dispersion, according to how many times the SIGMA1 routine is called. The source being at 2MeV, but the Doppler threshold at 20keV, load unbalancing can be large between neutrons that slow down to low energies and neutrons that are absorbed at higher energies.

Machine	Static dispatch.	Dynamic dispatch.	Gain
BDW	36.12	35.07	1.03
KNL	41.30	37.44	1.10

TABLE VIII. Effect of dynamic dispatching of particles on KNL and BDW for the PointKernel test case and use of on-the-fly Doppler broadening. Configuration with 1 MPI rank and 48 threads for BDW and 272 threads for KNL. The memory prefetching is enabled.

Configuration	Time (s/cycle)
CPU ; binary search	18.1
CPU + 1 K80 OTF	32.4
CPU + 2 K80 OTF	19.8
CPU + 3 K80 OTF	17.9

TABLE IX. PointKernel test case with on-the-fly Doppler broadening using the GPU as a cross-section server. The simulation part is run on the CPU with 2x16 threads.

F. GPU version

In this version, realized in the framework of a collaboration with Nvidia, the on-the-fly Doppler broadening has been ported to GPU by "flattening" the cross sections objects into `C` arrays and rewriting the SIGMA1 kernel in CUDA. Thus all the particle geometric tracking, collisions, and scoring are still performed on the CPU, and the GPU essentially works as a microscopic total cross sections server providing on-the-fly Doppler broadened cross sections every time the macroscopic total cross section of the material is computed. The total cross sections of all the nuclides of the current material are called at once, to maximize the GPU occupancy.

The GPU version of PATMOS has been tested on a Intel Xeon Haswell server E5-2698v3 (2×16 cores at 2.3GHz) equipped with 4 Nvidia K80 GPUs. All the cases were run with 32 threads. We see in Table IX that the optimal performance is achieved with 2 GPUs fed by 32 threads. After that the number of threads per GPU is not enought to have a good GPU occupancy and the part of work not offloaded becomes preponderant. The results show that the slow down due to on-the-fly Doppler broadening is a of 1.7 with one K80 and roughly equal time is achieved with two K80.

CONCLUSIONS AND PERSPECTIVES

A new prototype for Monte Carlo neutron transport is under development at CEA mainly devoted to the issues of High Performance Computing. Its object-oriented architecture allows easy implementation and testing of different algorithmic options and parallel libraries. Simulation results show that the hybrid parallelism in PATMOS scales well on the latest Intel architectures of both the Xeon and the Xeon Phi lines. Further work is planned to improve KNL performances which is 2 times slower than the Broadwell for pre-tabulated cross section simulations and 22% slower for on-the-fly Doppler broadening cross sections simulations. We showed results with 10% penalty for distributed memory simulations up to 262,144 cores.

The SIGMA1 method for on-the-fly Doppler broadening has been vectorized on Intel Xeon and Xeon Phi architectures and now the slow-down is limited to a factor between 2 and 3.

An "heterogeneous" version of Patmos has been developed which can offload the on-the-fly Doppler broadening to one or several GPUs. Our results show the SIGMA1 kernel can be run at no extra cost with respect to standard pre-tabulated cross sections with the use of two Nvidia K80 GPUs.

All the performances reported in this work need to be reviewed as new hardware is made available by constructors.

We are currently testing PATMOS on OpenPower architecture made of two Power8 processors each one attached to two Nvidia Pascal P100 GPU.

From an algorithmic point of view, our next goal concerns the distribution of the ten billion (10^{10}) scores, needed for full core pin-by-pin PWR burn-up computations, across several nodes of a HPC machine.

ACKNOWLEDGMENTS

Part of this work was performed in the framework of "Cellule de Veille Technologique" of GENCI (Grand Equipenment National de Calcul Intensif) with the participation of Intel, Atos/Bull, IBM and Nvidia experts.

REFERENCES

1. E. BRUN, F. DAMIAN, C. DIOP, E. DUMONTEIL, F.-X. HUGOT, C. JOUANNE, Y.-K. LEE, F. MALVAGI, A. MAZZOLO, O. PETIT, J.-C. TRAMA, T. VISONNEAU, and A. ZOIA, "TRIPOLI-4, CEA, EDF and AREVA reference Monte Carlo code," *Annals of Nuclear Energy*, **82**, 151–160 (2015).
2. X-5 MONTE CARLO TEAM, "MCNP - A General N-Particle Transport Code, Version 5," Tech. rep., LA-UR-03-1987, Los Alamos National Laboratory (Updated 2005).
3. Y. WANG, E. BRUN, F. MALVAGI, and C. CALVIN, "Competing Energy Lookup Algorithms in Monte Carlo Neutron Transport Calculations and Their Optimization on CPU and Intel MIC Architectures," *Procedia Computer Science*, **80**, 484 – 495 (2016), international Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA.
4. D. E. CULLEN and C. R. WEISBIN, "Exact Doppler Broadening of Tabulated Cross Sections," *Nuclear Science and Engineering*, **60**, *3*, 199 – 229 (1976).
5. R. BRUN and F. RADEMAKERS, "ROOT - An object oriented data analysis framework," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, **389**, 81 – 86 (1997).
6. J. E. HOOGENBOOM, W. R. MARTIN, and B. PETROVIC, "A Proposal for a Benchmark to Monitor the Performance of Detailed Monte Carlo Calculation of Power Densities in a Full Size Reactor," in "Core. Proc. Int. Conf. Mathematics, Computational Methods, and Reactor Physics, Saratoga Springs, NY," (2009).
7. L. DAGUM and R. MENON, "OpenMP: an industry standard API for shared-memory programming," *IEEE computational science and engineering*, **5**, *1*, 46–55 (1998).
8. A. KUKANOV and M. J. VOSS, "The Foundations for Scalable Multi-core Software in Intel Threading Building Blocks." *Intel Technology Journal*, **11**, *4* (2007).