Portable Monte Carlo Transport Performance Evaluation in the PATMOS Prototype

Tao CHANG

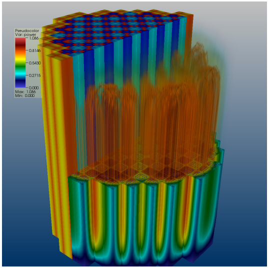
[1] DEN-Service d'Etudes des Réacteurs et de Mathématiques Appliquées (SERMA)

November 27, 2019

Outline

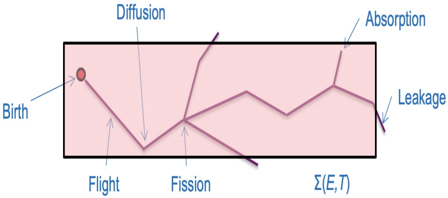
Monte Carlo Neutron Transport

- In the nuclear field, Monte Carlo (MC) simulation is widely used to compute physical quantities such as:
  - density of particles
  - reaction rates
  - fission power
  - ...
- List of MC codes:
  - **TRIPOLI-4**® (CEA, France)
  - MCNP-5 (LANL, USA)
  - OpenMC (MIT, USA)
  - SERPENT (VTT, Finland)
  - RMC (Tsinghua, China)
  - ...



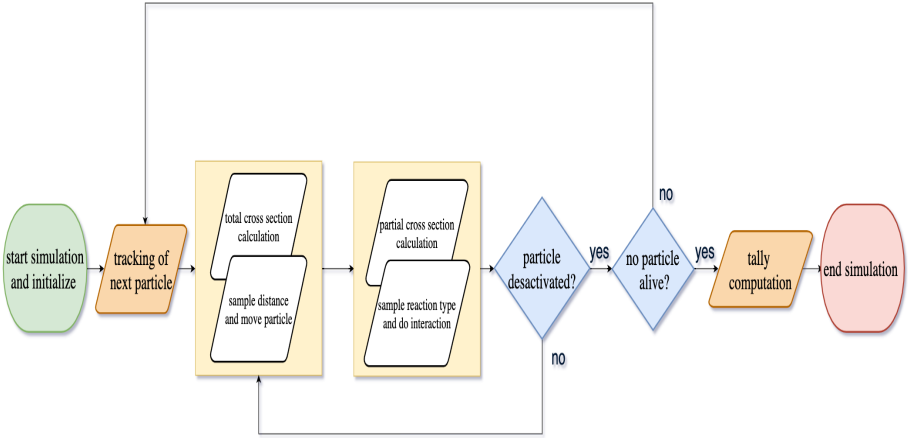
Credit: ANS Nuclear Cafe

Monte Carlo Neutron Transport



- The Monte Carlo transport codes simulate the life of a particle from birth to death
- A succession of transports and collisions
- Advantages:
  - precision, few approximations
  - complex geometries
- Drawbacks:
  - high computational cost

Monte Carlo Neutron Transport



Monte Carlo Neutron Transport

Cross section

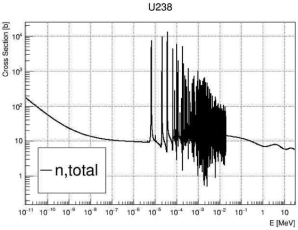
- Address the interaction probability of the particle with the different nuclides composing the material

$$\Sigma(E,T) = \sum_i N_i * \sigma_{t,i}\,(E,T)$$

Macroscopic cross section (cm^{-1})

Atomic density of nuclide i (at/cm^3)

Total microscopic cross section of nuclide i (cm^2)



- Pre-tabulated method (load precalculated total cross sections at (E, T))
- On-the-fly Doppler Broadening method (calculate cross sections at **(E, T)** before each random flight)

Monte Carlo Neutron Transport

Run time percentage

- Total macroscopic cross section is the most consuming part

Processing Step	**Run Time Percentage** (%)
Total Cross Section	95.4
exp	17.6
erfc	49.4
`binary_search`	2.4
`compute_integral`	79.2
Partial Cross Section	1.7
exp	0.2
erfc	0.6
`binary_search`	0.1
`compute_integral`	1.4
Initialization	1.8
`buildMedium`	1.5
Others	1.1

Outline

PATMOS

- A prototype dedicated to the testing of algorithms for high performance computations on modern architectures
- Prepare next generation of TRIPOLI
- Written in C++
- A subset of neutron physics is implemented but representative for performance analysis

PATMOS

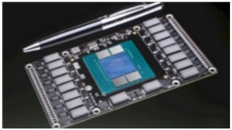
- A prototype dedicated to the testing of algorithms for high performance computations on modern architectures
- Prepare next generation of TRIPOLI
- Written in C++
- A subset of neutron physics is implemented but representative for performance analysis
- Hybrid parallelism: MPI + OpenMP + GPU offload
- GPU version written in CUDA
- Only the microscopic cross section calculation is offloaded

Outline

1. Introduction
   - Monte Carlo Neutron Transport
   - PATMOS
   - Objective
2. Implementations
3. Tests
4. Conclusions

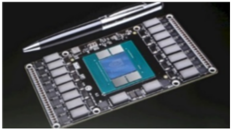
Objective

- The implemented CUDA version in PATMOS is not "portable" as it is only for Nvidia GPU
- A variety of architectures to address:
  - Many-core:
    - Intel Xeon Phi
    - Arm
  - Heterogeneous architecture
    - Intel + Nvidia GPU
    - OpenPower + Nvidia GPU
    - AMD + GPU
    - ...



Objective

- The implemented CUDA version in PATMOS is not “portable” as it is only for Nvidia GPU
- A variety of architectures to address:
  - Many-core:
    - Intel Xeon Phi
    - Arm
  - Heterogeneous architecture
    - Intel + Nvidia GPU
    - OpenPower + Nvidia GPU
    - AMD + GPU
    - ...
- Develop portable codes on a large variety of architectures
- Evaluate the different programming models in terms of performance of implemented benchmark



Outline

Programming Model

- Only consider intra-node parallelism
- OpenMP thread + {**X**}
- {**X**} can be any languages or libraries which are capable of parallel programming on modern architectures, such as:
  - Low-level:
    - CUDA
  - High-level:
    - OpenACC
    - OpenMP
    - Kokkos
    - SYCL

Outline

Algorithms

Algorithm 1: History-based algorithm

```
Each MPI Rank
foreach batch or generation do
    initialize particle state from source;
    OpenMP Thread Level
    foreach particle in batch do
        while particle is alive do
            calculation of macroscopic cross section:
              • do microscopic cross section lookups ⇒ offloaded;
              • sum up total cross section;
            sample distance, move particle, do interaction;
        end
    end
end
```

Algorithms

Algorithm 2: Microscopic cross section lookup

Input: randomly sampled a group of N tuples of materials, energies and temperatures, $\{(m_i, E_i, T_i)\}_{i \in N}$

Result: caculated microscopic cross sections for N materials, $\{\sigma_{ik}\}_{i \in N, k \in |m_i|}$

CUDA Threadblock Level

`#pragma acc parallel loop gang` or

`#pragma omp target teams distribute`

for (n_{ik}, E_i, T_i) *where* $n_{ik} \in m_i$ **do**

  $\sigma_{ik} = pre\_calcul();$

  CUDA Thread Level

  `#pragma acc loop vector` or

  `#pragma omp parallel for`

  foreach *thread in warp* **do**

    $\sigma_{ik} \mathrel{+}= compute\_integral();$

  end

end

Algorithms

- **History-based (HB)** algorithm on GPU:
  - Too many small data transfers
  - Many memcpy calls
  - Small kernel
- Tuning solutions:
  - Reduce memcpy calls, enlarge kernel size
  - A new method called **pseudo event-based (PEB)** algorithm

Algorithms

Algorithm 3: Pseudo event-based algorithm

```
Each MPI Rank
foreach batch or generation do
    initialize particle state from source;
    OpenMP Thread Level
    foreach bank of N particles in batch do
        while particles remain in bank do
            foreach remaining particle in bank do
                bank required data;
            end
            • do microscopic cross section lookups ⇒ offloaded;
            foreach remaining particle in bank do
                • sum up total cross section;
                sample distance, move particle, do interaction;
            end
        end
    end
end
```

Outline

Benchmark

slabAllNulides

- Fixed source MC simulation
- Slab geometry
  - 10,000 volumes, 900K
  - each material ⇒ 355 nuclides
  - main components: H1 and U238
  - Pressurized Water Reactor (PWR) spectrum
- On-the-fly Doppler broadening method

Outline

Parameters

Machine

- **Ouessant:** 2× 10-core IBM Power8, SMT8 + 4× Nvidia P100 (GENCI IDRIS)
- **Cobalt-hybrid:** 2× 14-core Intel Xeon E5-2680 v4, HT2 + 2× Nvidia P100 (CEA-CCRT)
- **Cobalt-V100:** 2× 20-core Intel Skylake + 4× Nvidia V100 (CEA-CCRT)

slabAllNuclides

- **Inputs**: 20,000 particles, 10 cycles, 100 as bank size
- **Outputs**: particles/sec (higher is better)

Environment

	GCC	Intel Compiler	PGI	XLC	CUDA
Ouessant	7.3.0		18.10	16.1.0	9.2
Cobalt-hybrid	7.1.0	17.0.6	18.7		9.0
Cobalt-V100	7.1.0	17.0.6	18.7		9.2

Outline

Results

OMPth + $\{X\}$

Machine		Programming Model	slabAllNuclides ($\times 10^2$ particles/s)	
			HB	PEB
Ouessant	CPU (20 cores, SMT8)	OMPth	4.7	4.7
		OMPth+ACC	4.6	4.5
		OMPth+offload	3.7	3.7
	1P100	OMPth+CUDA	6.7	27.2
		OMPth+ACC	2.7	22.2
		OMPth+offload	2.5	3.6
	2P100	OMPth+CUDA	13.0	47.5
		OMPth+ACC	4.5	40.2
		OMPth+offload	4.3	6.7
	4P100	OMPth+CUDA	23.7	65.8
		OMPth+ACC	9.4	52.4
		OMPth+offload	5.0	12.2
Cobalt-hybrid	CPU (28 cores, HT2)	OMPth	10.1	8.7
		OMPth+ACC	5.6	5.0
	1P100	OMPth+CUDA	6.8	25.4
		OMPth+ACC	2.7	18.5
	2P100	OMPth+CUDA	16.4	48.5
		OMPth+ACC	5.6	34.5

Table: Particle traking rate via different programming models on Ouessant and Cobalt-hybrid

Results

OMPth + $\{X\}$

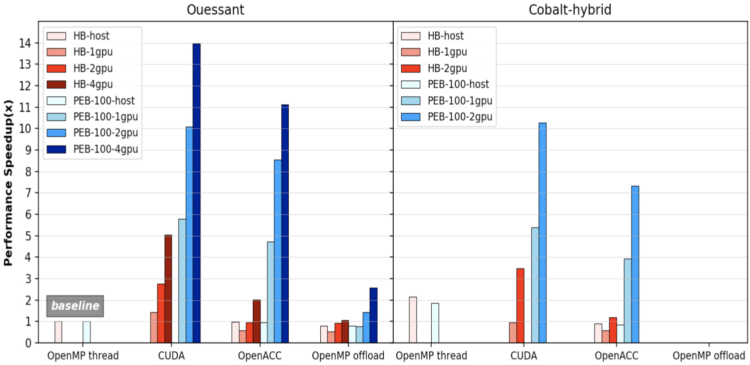


Figure: Comparision of performance speedup for slabAllNuclides on Ouessant and Cobalt-hybrid

Results

OMPth + {X}

Machine		Programming Model	slabAllNuclides ($\times 10^2$ particles/s) HB	PEB
Ouessant	CPU (20 cores, SMT8)	OMPth	4.7	4.7
		OMPth+ACC	4.6	4.5
		OMPth+offload	3.7	3.7
	1P100	OMPth+CUDA	6.7	27.2
		OMPth+ACC	2.7	22.2
		OMPth+offload	2.5	3.6
	2P100	OMPth+CUDA	13.0	47.5
		OMPth+ACC	4.5	40.2
		OMPth+offload	4.3	6.7
	4P100	OMPth+CUDA	23.7	65.8
		OMPth+ACC	9.4	52.4
		OMPth+offload	5.0	12.2
Cobalt-V100	CPU (40 cores)	OMPth	14.7	13.3
		OMPth+ACC	6.9	6.1
	1V100	OMPth+CUDA	7.8	56.0
		OMPth+ACC	3.1	27.7
	2V100	OMPth+CUDA	16.8	89.7
		OMPth+ACC	6.7	42.5
	4V100	OMPth+CUDA	32.5	134.2
		OMPth+ACC	11.8	54.8

Table: Particle traking rate via different programming models on Ouessant and Cobalt-V100

Results

OMPth + {X}

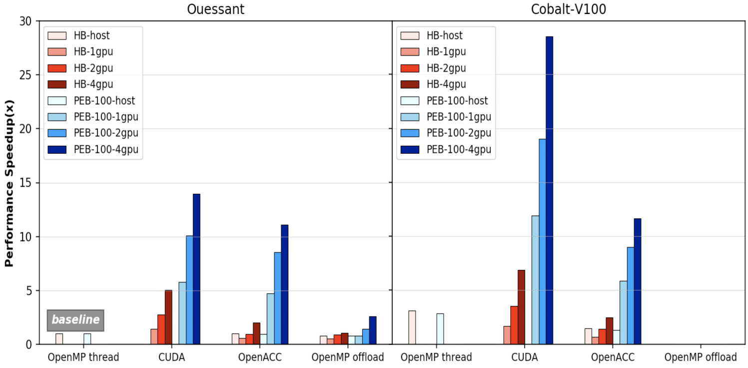


Figure: Comparision of performance speedup for slabAllNuclides on Ouessant and Cobalt-V100

Sum-up

- PEB is more suitable than HB, for PEB:
  - The CUDA version can reach up to **28.5x**.
  - The OpenACC version can attain a factor of **11.6x**, there is no large difference between performances on Ouessant and Cobalt-V100.
  - The OpenMP offload version is limited to **2.5x** performance speedup due to the underdeveloped implementation of OpenMP offload functionalities of XLC 16.1.

Outline

CUDA Profiling

HB vs PEB

History-based Method	
Block size	(32, 2, 1)
Registers/Thread	68
Theoretical Warps/SM	28
Occupancy	8.8%
FLOP Efficiency	4.4%

Pseudo event-based Method	
Block size	(32, 2, 1)
Registers/Thread	68
Theoretical Warps/SM	28
Occupancy	31.6%
FLOP Efficiency	21.9%

Outline

Conclusions

- The GPU performance via PEB surpasses significantly HB
- The OpenACC version can be competitive to the CUDA version with PEB
- The performance of OpenMP offload version is limited due to the underdeveloped support of CUDA asynchronous streams

Conclusions

Future work

- Implement other high-level programming languages such as SYCL
- Perform more tests to cover a wider range of architectures
- Adopt several metrics for the evaluation of portability and performance portability

Thank you