MONTE CARLO NEUTRON TRANSPORT
SIMULATING NUCLEAR REACTIONS ONE NEUTRON AT A TIME

Tony Scudiero
NVIDIA
TAKEAWAYS

- Why Monte Carlo methods are fundamentally different than deterministic methods

- Inherent Parallelism of Monte Carlo Methods is great on GPUs
  - Parallelizing complex monte carlo simulations with highly involved samples on GPUs
  - Approach Monte Carlo Divergence from a Memory Bandwidth Perspective
NEUTRONICS

- Study of the interaction of neutrons with specific materials and geometries.
  - Nuclear Reactor Design
  - Medical imaging
  - Radiation Treatment Design
  - Radiation Shielding Design
COMMON APPLICATION: $K_{\text{EFFECTIVE}}$

- $K_{\text{effective}}$ - Effective Neutron multiplication factor
  - Average number of neutrons resulting from fission which will go on to cause another fission.

$K_{\text{eff}} = 1$  
$K_{\text{eff}} > 1$
NEUTRON TRANSPORT EQUATION

\[
\frac{1}{v(E)} \frac{\partial \psi(r, E, \hat{\Omega}, t)}{\partial t} + \hat{\Omega} \cdot \nabla \psi(r, E, \hat{\Omega}, t) + \sum_t(r, E, t) \psi(r, E, \hat{\Omega}, t) = \]

\[
\chi_p(E) \int_0^\infty dE' \nu_p(E') \sum_f(r, E', t) \phi(r, E', t) + \sum_{i=1}^N \frac{\chi_{fi}(E)}{4\pi} \lambda_i C_i(r, t) +
\]

\[
\int_{4\pi} \int_0^\infty d\Omega' \sum_{s(r, E' \rightarrow E, \hat{\Omega}' \rightarrow \hat{\Omega}, t)} \psi(r, E', \hat{\Omega}', t) + s(r, E, \hat{\Omega}, t)
\]

\[
\frac{\chi_{fo}(E)}{4\pi} \int_0^\infty dE' \nu_o(E') \sum_f(r, E', t) \phi(r, E', t)
\]

\[
\frac{\chi_{s}(E)}{4\pi} \int_{4\pi} \int_0^\infty d\Omega' \sum_{s(r, E' \rightarrow E, \hat{\Omega}' \rightarrow \hat{\Omega}, t)} \psi(r, E', \hat{\Omega}', t) + s(r, E, \hat{\Omega}, t)
\]

NEUTRONICS CODES

- **Determinisitic**
  - NEWT (ORNL)
  - Denovo (ORNL)
  - PARTISN (LANL)
  - DRAGON

- **Monte Carlo**
  - MCNP (Los Alamos)
  - Serpent (VTT Finland)
  - KENO (ORNL)
  - OpenMC (MIT)(ANL)
MONTE CARLO METHODS

- If at first you succeed, try again. And again. And again. And again. And again. And again.

- Useful for approximating solutions to problems with unknown or extraordinarily difficult direct solutions
  - Computational Finance
  - Computational Biology (protein folding, ion transport)
  - Graphics (Ray Tracing, hair/cloth simulation)

- Virtual Lab
  - Dangerous, Expensive, Impossible

https://xkcd.com/242/
SIMPLE MONTE CARLO EXAMPLE

Calculate Pi:

\[
\frac{\text{Area}_{\text{Circle}}}{\text{Area}_{\text{Square}}} = \frac{\pi r^2}{(2r)^2} = \frac{\pi}{4}
\]

Generate \(N\) uniformly distributed random \((x,y)\) points

\[(x, y) \in M : x^2 + y^2 \leq 1\]

3 ops, 1 compare per sample

No other data

K40: 7.9E+10 Samples per second
3.1E+11 Samples per second Quad K40
SHIELDING EXAMPLE

\[ H \left( \frac{XS(Detector)}{2\pi r} \right) \]

Neutron Source

Neutron Detector

Neutron Detector

Neutron Detector

Neutron Detector

Neutron Shield

\[ \frac{XS(Detector)}{2\pi r} \]
SHIELDING EXAMPLE

Not to scale

Tally ➔ Scatter ➔ Absorption ➔ Tally

Neutron Shield Material
CROSS SECTIONS

- Resonance frequency and shape varies by nuclide
- Nuclides have varying degrees of high-frequency resonances in cross section data.
- Aptly named:
  - 1 barn = 1E-28 square meters.
  - 1E-4 square yoctometers

From: http://wwwndc.jaea.go.jp/j33fig/j33fig10.html
KORD SMITH CHALLENGE

- Delivered at Mathematics & Computation 2003
- Monte Carlo simulation of power for fuel burnup calculations
  - 40-60 million tallies with standard deviation of <1%
  - Predicts 2030 a single processor can do this in 1 hr.
- Mathematics & Computation 2007: Bill Martin changes update to 2019
HOOGENBOOM AND MARTIN BENCHMARK

- Very specific: Geometry, Materials, concentrations, designs.
- Implement in any MC Neutronics app. (MCNP, OpenMC)

Figures from
OPENMC

- Recently Developed MC Neutronics code from MIT
- Active Research at Center for Exascale Simulation of Advanced Reactors (CESAR) ANL
- ~30k lines of F90
- Can compute the H-M benchmark
- In use as a neutronics virtual laboratory
  - Primarily computational experiments thus far
MAJOR ALGORITHM SUBSECTIONS

- **Geometry**
  - Identify the material in the neutron’s location
  - Constructive Solid Geometry

- **Physics**
  - Probabilistic Collisions
  - Collision Modeling:
    - Inelastic Scattering
    - Elastic Scattering
    - Fission
      - Delayed Neutron
      - Prompt Neutron

- **Cross Section Calculation**
MONTE CARLO NEUTRONICS ON GPUs: DIVERGENCE

While (neutron.alive)
  Sample Next Event
    Collision
      Scatter
        Inelastic
        Elastic
      Fission
        Delayed
        Prompt
      Absorption

Outer Loop Divergence

Inner Loop Divergence
OPENMC H-M SMALL ANALYSIS

OpenMC: H-M Small Benchmark Profile
Percent of wall time

Percentage of walltime spent in specific functions.
XSBENCH

- DOE Benchmark for Material Cross Section calculation
- Derived from OpenMC
- ~1k lines of C99
- Developed by John Tramm at ANL
- Used in NVIDIA’s research on MC for future architectures
MATHEMATICAL FORMULATION

\[ \Sigma_t(E, \text{mat}) = \sum_{i=0}^{M} \rho_i \ast \sigma_{t,i}(E) \]

- \( \rho_i \) - Concentration of nuclide \( i \) in material
- \( M \) - Number of different nuclides in material
- \( \sigma_{t,i}(E) \) - Cross section of Nuclide \( i \) at energy \( E \)
  - Approximated by linear interpolation of proximal samples.
- \( \Sigma_t \) - Material cross section of type \( t \).
  - Total, Fission, Elastic, Inelastic, Capture
- \( E \) - \textit{random} energy
- \( \text{mat} \) - \textit{random} material
COMPUTATIONAL ANALYSIS

\[ \Sigma_t(E, mat) = \sum_{i=0}^{M} \rho_i \cdot \sigma_{t,i}(E) \]

- 5 different t’s. M ranges from 4 to 321
- Using linear interpolation for \( \sigma_{t,i}(E) \) involves loading 2 values, 3 flops
  - 10 loads and 15 flops per t
  - Additionally: 1 load for \( \rho_i \), 2 loads for E, 3 flops for interp. coefficient
- 104 bytes, 23 flops for each i =1:M
DIVERGENCE WITHIN XSBENCH

- Threads drop out as nuclide (M) loop completes.
**XSBENCH PERFORMANCE COMPARISON**

### Processor Lookups / Sec Speedup vs IVB10

<table>
<thead>
<tr>
<th>Processor</th>
<th>Lookups / Sec</th>
<th>Speedup vs IVB10</th>
</tr>
</thead>
<tbody>
<tr>
<td>SB6(HT)</td>
<td>4,452,404</td>
<td>0.62397</td>
</tr>
<tr>
<td>SB8</td>
<td>4,684,446</td>
<td>0.656488</td>
</tr>
<tr>
<td>SB8x2</td>
<td>6,253,792</td>
<td>0.87642</td>
</tr>
<tr>
<td>IVB 10</td>
<td>7,135,611</td>
<td>1</td>
</tr>
<tr>
<td>IVB 10 x2</td>
<td>11,939,573</td>
<td>1.673238</td>
</tr>
<tr>
<td>K40 Original</td>
<td>17,941,073</td>
<td>2.514301</td>
</tr>
<tr>
<td>K40 Tuned</td>
<td>39,761,679</td>
<td>5.572288</td>
</tr>
</tbody>
</table>

### Processor Lookups / Sec Speedup vs IVB10

<table>
<thead>
<tr>
<th>Processor</th>
<th>Lookups / Sec</th>
<th>Speedup vs IVB10</th>
</tr>
</thead>
<tbody>
<tr>
<td>SB6(HT)</td>
<td>1,321,943</td>
<td>0.619345</td>
</tr>
<tr>
<td>SB8</td>
<td>1,453,827</td>
<td>0.681134</td>
</tr>
<tr>
<td>SB8x2</td>
<td>1,646,447</td>
<td>0.771379</td>
</tr>
<tr>
<td>IVB 10</td>
<td>2,134,420</td>
<td>1</td>
</tr>
<tr>
<td>IVB 10 x2</td>
<td>3,214,495</td>
<td>1.506027</td>
</tr>
<tr>
<td>K40 Original</td>
<td>5,059,439</td>
<td>2.370405</td>
</tr>
<tr>
<td>K40 Tuned</td>
<td>11,955,176</td>
<td>5.601136</td>
</tr>
</tbody>
</table>
GPU OPTIMIZATION TECHNIQUES USED

- **Load outside Inner Loop**
  - Use large datatypes (double2)

- **Use LDG intrinsic**
  - L1 Caching helps out AOS loads
  - Lower load granularity

- **Unroll outer loop**
  - Mitigate outer loop divergence
  - Maximize loads in flight

- **Pseudosort**
  - “Warp Specialization” for fuel

~60% of performance difference
~30% of performance difference
<10% of performance difference
for(int i=0; i<numNuclides[material]; i++) {
    double r[5]; // results
    f = (E - ND[0])/ (ND[6] - ND[0]);
}
double r[5]; // results double2 d[6]; // register storage
for(int t=0; t<6; t++) {
    d[t] = __ldg(&N0[t]);
}
f = (E - d[0].x) / (d[3].x - d[0].x);
r[0] = d[0].y + f*(d[3].y - d[0].y);
for(int i=0; i<numNuclides[material]; i++) {
    ...
}
UNROLL OUTER LOOP

```c
double r[5]; // results
double2 d[12]; // register storage
for(int t=0; t<6; t++) {
    d[t] = __ldg(&ND0[t]);
    d[t+6] __ldg(&ND1[t]);
}
```

f0 = (E - d[0].x) / (d[3].x - d[0].x);
... // same compute as before
f1 = (E - d[6].x) / (d[9].x - d[6].x);
... // Same compute with indexes+=6;

- Increases loads in flight before stall
- Mitigates divergence effect on BW
- Minute effect decreasing loop overhead
- Extra conditional code (not shown)
- Not just a #pragma unroll
PSEUDOSORT
**FUEL-SORT SPECIALIZATION**

- $O(tb)$ scan moves all fuel lookups to first threads
  - Expect < 32 fuel lookups => all fuel in warp0
  - “Warp Specialization” approach to divergence mitigation
- In-core(SM) sort, $tb \leq 256$ => low sorting cost
- Warp0 runs significantly longer than other warps
  - Other warps do not clear SM, but are not scheduled
- ~10% improvement in memory bandwidth utilization
  - ~2-5% performance improvement
  - cost of scan reduces overall gains
XSBENCH PERFORMANCE COMPARISON

### HM Small

<table>
<thead>
<tr>
<th>Processor</th>
<th>Lookups / Sec</th>
<th>Speedup vs IVB10</th>
</tr>
</thead>
<tbody>
<tr>
<td>SB6(HT)</td>
<td>4,452,404</td>
<td>0.62397</td>
</tr>
<tr>
<td>SB8</td>
<td>4,684,446</td>
<td>0.656488</td>
</tr>
<tr>
<td>SB8x2</td>
<td>6,253,792</td>
<td>0.87642</td>
</tr>
<tr>
<td>IVB 10</td>
<td>7,135,611</td>
<td>1</td>
</tr>
<tr>
<td>IVB 10 x2</td>
<td>11,939,573</td>
<td>1.673238</td>
</tr>
<tr>
<td>IVB 10 x2</td>
<td>11,939,573</td>
<td>1.673238</td>
</tr>
<tr>
<td>K40 Original</td>
<td>17,941,073</td>
<td>2.514301</td>
</tr>
<tr>
<td>K40 Tuned</td>
<td>39,761,679</td>
<td>5.572288</td>
</tr>
</tbody>
</table>

### HM Large

<table>
<thead>
<tr>
<th>Processor</th>
<th>Lookups / Sec</th>
<th>Speedup vs IVB10</th>
</tr>
</thead>
<tbody>
<tr>
<td>SB6(HT)</td>
<td>1,321,943</td>
<td>0.619345</td>
</tr>
<tr>
<td>SB8</td>
<td>1,453,827</td>
<td>0.681134</td>
</tr>
<tr>
<td>SB8x2</td>
<td>1,646,447</td>
<td>0.771379</td>
</tr>
<tr>
<td>IVB 10</td>
<td>2,134,420</td>
<td>1</td>
</tr>
<tr>
<td>IVB 10 x2</td>
<td>3,214,495</td>
<td>1.506027</td>
</tr>
<tr>
<td>K40 Original</td>
<td>5,059,439</td>
<td>2.370405</td>
</tr>
<tr>
<td>K40 Tuned</td>
<td>11,955,176</td>
<td>5.601136</td>
</tr>
</tbody>
</table>
FROM XSBENCH BACK TO OPENMC

- OpenMC on I7-3930k CPU (SB6 + HT)
  - Small: 14.9k neutrons/second (OMP_NUM_THREADS=12)
  - Large: 4.1k neutrons/second (OMP_NUM_THREADS=12)
  - Git Cloned on Feb 21st, 2014

- K40 (full Boost) Projection
  - Small: [ 39.4k - 133k ] neutrons / second
  - Large: [ 9.0k - 37.07] neutrons / second
    - Lower Bound: No acceleration of other code areas
    - Upper Bound: Equal acceleration of other code areas

- Grain of Salt:
  - Parallelizing the rest of the code generates additional memory traffic.
A LIVING CODE

- CUDA Conversion started in late 2012
- OpenMC data structures have changed since original snapshot
- OpenMC CPU performance has changed as well.
- OpenMC Capabilities have expanding
WHAT’S NEXT

- Integrate into New Versions of OpenMC
  - New version more efficient for XS calculations
- Explore new Cross Section storage and calculations
- Explore other MC Neutronics
- Explore other MC transport
  - (Photonics)
SUMMARY

- GPUs are very good at Monte Carlo Processes
  - “Embarrassingly Parallel” on the surface
  - Complex samples get... complicated.

- Monte Carlo Neutronics
  - Require Out of Core operation, but do well on GPUs
  - Complex simulation geometry is GPU’s bread and butter

- OpenMC: Still in progress. Initial results are promising

- XSBench: Really really good on GPUs.
MORE MONTE CARLO ON GPU

- S4554 - GPU Acceleration of a Variational Monte Carlo Method - Wednesday 16:30 212A
- S4156 - Mining Hidden Coherence for Parallel Path Tracing on GPUs - Thursday 15:00 LL21C
- S4154 - Pricing American Options with Least Square Monte Carlo simulations on GPUs - Thursday 15:00 210C

- S4259 - Optimization of a CUDA-based Monte Carlo Code for Radiation Therapy - Wednesday 9:00 212A
- S4784 - Monte-Carlo Simulation of American Options with GPUs - Wednesday 10:30 210C
- S4261 - Massively-Parallel Stochastic Control and Automation: A New Paradigm in Robotics Wednesday 15:30
TAKEAWAYS

- Why Monte Carlo methods are fundamentally different than deterministic methods

- Inherent Parallelism of Monte Carlo Methods is great on GPUs
  - Parallelizing complex monte carlo simulations with highly involved samples on GPUs
  - Approach Monte Carlo Divergence from a Memory Bandwidth Perspective