Investigation of Portable Event-Based Monte Carlo Transport

COE Phoenix, AZ

Ryan Bleile
Lawrence Livermore National Laboratory
University of Oregon, Eugene

April 20, 2016
Current Landscape of Architectures

- **GPU (NVIDIA)**
  - Sub-architectures:
    - Fermi, Kepler, Maxwell
  - Multiple Memory Types:
    - Global, shared, constant, texture
  - Memory Amount:
    - Up to 12 GB
  - 1000s of threads
    - Grids, blocks, and warps

- **CPU/MIC**
  - Multiple ISAs:
    - Vector Unit Widths:
      - 2, 4, 8 / 16
  - Single Memory Type
    - Shared/private caches
  - Larger Memory Size (CPU)
  - Up to 20/60 threads
    - No explicit organization

*Slide courtesy of Matthew Larsen
University of Oregon, CDUX research group
The Problem

- Forces developers to either:
  - Pick a target architecture
  - Add additional implementations of the same algorithm:

![Diagram showing Algorithm vs Architecture with nodes A, B, C, D, E, F connected to CPU, GPU, MIC]
Data-Parallel Primitives Libraries

- Backend – Implement fast parallel primitive operators for each new architecture
- Frontend – Re-think current algorithms in terms of the primitives

*Slide courtesy of Matthew Larsen
University of Oregon, CDUX research group
Data Parallel Primitives (DPP)

- What are they?
  - Provide a level of abstraction based on Blelloch’s parallel primitive operators
  - Provides node level parallelism

- Big challenge
  - “re-thinking” algorithms to use DPP
  - Not “porting” algorithms to DPP

- Benefits
  - Portable performance
  - Future proof implementations

- What is a DPP
  - If it can be completed in $O(\log N)$ where $N$ is the array size than it can be a DPP
Data Parallel Operations

- **Map**
  - Parallel for each loop

- **Gather / Scatter**
  - Index set array operations

- **Scan**
  - Index creation scheme

- **Reduce**
  - Counting / Narrowing results
Portable Performance – Abstraction Layer

- Previous work done in research group at UO
  - Ray Tracing
    - Promising results
    - Using VTK-m, EAVL, etc...

- Applying this technique to Monte Carlo Transport
  - Many possible avenues to consider
    - Thrust
      - supports data parallel operations
    - RAJA style
      - Supports simplifying key ideas with a template/MACRO definition
Monte Carlo Transport – ALPS_MC

- Models particle transport in a 1D binary stochastic medium
- Particles are created and then tracked through a series of events
- Tallies of multiple types are incremented
  - Single Value: Reflection, Transmission
  - Multi Value (per material): Absorption, Scatter
  - Many Value (per zone): Zonal Flux
- Legacy approach (history-based) did not lend itself to many-core
- Recent work takes a new approach (event-based) that is suitable for many-core systems
  (Investigation of Portable Event-Based Monte Carlo Transport Using the NVIDIA Thrust Library. in press.)
Event based algorithm - overview

- Determine a batch size
  - How many particles fit in GPU memory

- For a given batch
  - Generate all particles in batch
  - While any particles left to compute
    - For each event X
      - Get particles whose next event is X
      - Do event X and compute their next event
    - Delete killed particles

- 3 events tracked
  - Collision
  - Material interface crossing
  - Zonal boundary crossing

- Excluded zonal flux tally as future work to study its effect
AOS and SOA Particle Data Structure

- Particle class contains many variables
  - (3 ints, 1 Long, 6 doubles)
  - Real case scenarios contain even larger classes

- Not all variables used in each kernel
  - Reduce size of memory reads and writes

- Coalesced memory access with SOA

- Reduced memory usage in kernel
New Particle Removal Scheme

- Reorganizing particles is costly
  - More costly than all compute kernels combined

- Only call remove function when it makes an impactful change to array size

- If number to kill >= particles_remaining.size() / 2;
  - Decreases amount of time spent removing particles
  - Increase amount of time needed to establish compute kernels
Details of Implementation

- Explicitly managed GPU memory (cudaMalloc, etc.)
- Modified CUDA version first
  - Made new Thrust, RAJA methods from optimized CUDA method
- Changed particle data structure to allow SOA or AOS
- Kernels read/write strategy changed to ensure - read, compute, write pattern upheld
- New particle removal scheme
# Results – 10 Million Particle Study

- Studies in CUDA to understand performance

<table>
<thead>
<tr>
<th>(runtime in seconds)</th>
<th>SOA</th>
<th>AOS</th>
<th>SOA (kill/2)</th>
<th>AOS (kill/2)</th>
<th>SOA (sort)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collision</td>
<td>0.77</td>
<td>0.89</td>
<td>0.93</td>
<td>1.03</td>
<td>0.92</td>
</tr>
<tr>
<td>Zone Boundary</td>
<td>0.62</td>
<td>0.79</td>
<td>0.75</td>
<td>0.93</td>
<td>0.74</td>
</tr>
<tr>
<td>Material Interface</td>
<td>0.70</td>
<td>1.11</td>
<td>0.92</td>
<td>1.33</td>
<td>0.91</td>
</tr>
<tr>
<td>Compute Total</td>
<td>2.09</td>
<td>2.80</td>
<td>2.59</td>
<td>3.28</td>
<td>2.57</td>
</tr>
<tr>
<td>Remove / Sort</td>
<td>3.95</td>
<td>2.31</td>
<td>0.36</td>
<td>0.42</td>
<td>1.08</td>
</tr>
<tr>
<td>Total Time</td>
<td>6.04</td>
<td>5.11</td>
<td>2.95</td>
<td>3.70</td>
<td>3.65</td>
</tr>
</tbody>
</table>
Results – 100 Million Particle Study

- Using one GPU device (½ K80)
  - Results From Paper:

<table>
<thead>
<tr>
<th>(runtime in seconds)</th>
<th>AOS</th>
<th>% slowdown</th>
<th>SOA</th>
<th>% slowdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial</td>
<td>508.74</td>
<td>-</td>
<td>31.43</td>
<td>-</td>
</tr>
<tr>
<td>Thrust</td>
<td>234.30</td>
<td>33%</td>
<td>33.84</td>
<td>8%</td>
</tr>
<tr>
<td>CUDA</td>
<td>48.39</td>
<td>8%</td>
<td>31.92</td>
<td>2%</td>
</tr>
</tbody>
</table>
Conclusion

- Spending time to make the SOA changes and directly managing CUDA memory paid off in performance for all versions.

- Starting with CUDA, backing out an abstraction layer was simple.

- Initial pass abstraction layer attempt suffered significant performance degradation.
  - Lessons learned now can pay off down the line in future attempts at starting with an abstraction layer.

- DPP portable performance approach promising for event based Monte Carlo transport.
Acknowledgements

- This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

- Funding for this work was provided by the LLNL Livermore Graduate Scholar Program.

- Research advisors Patrick Brantley (LLNL), Hank Childs (UO), Matt O’Brien (LLNL).
### Results – 80 Million Particles Study

- **Using 1 GPU device (½ K80)**

<table>
<thead>
<tr>
<th>(runtime in seconds)</th>
<th>AOS</th>
<th>% slowdown</th>
<th>SOA</th>
<th>% slowdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA</td>
<td>1.00</td>
<td>-</td>
<td>0.79</td>
<td>-</td>
</tr>
<tr>
<td>Thrust</td>
<td>1.99</td>
<td>49%</td>
<td>1.38</td>
<td>43%</td>
</tr>
<tr>
<td>RAJA Like</td>
<td>1.17</td>
<td>15%</td>
<td>0.79</td>
<td>0%</td>
</tr>
</tbody>
</table>

- **Using 4 GPU devices (2 K80s)**

<table>
<thead>
<tr>
<th>(runtime in seconds)</th>
<th>AOS</th>
<th>% slowdown</th>
<th>SOA</th>
<th>% slowdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA</td>
<td>0.27</td>
<td>-</td>
<td>0.23</td>
<td>-</td>
</tr>
<tr>
<td>Thrust</td>
<td>0.91</td>
<td>70%</td>
<td>0.78</td>
<td>71%</td>
</tr>
<tr>
<td>RAJA Like</td>
<td>0.32</td>
<td>16%</td>
<td>0.23</td>
<td>0%</td>
</tr>
</tbody>
</table>
Using 4 GPU devices (2 full K80s)

<table>
<thead>
<tr>
<th></th>
<th>AOS</th>
<th>SOA</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA</td>
<td>17.74 [s]</td>
<td>15.84 [s]</td>
</tr>
<tr>
<td>Thrust</td>
<td>18.34 [s]</td>
<td>11.37 [s]</td>
</tr>
<tr>
<td>RAJA Like</td>
<td>18.64 [s]</td>
<td>15.92 [s]</td>
</tr>
</tbody>
</table>

Thrust SOA method scaling on multiple devices more effectively

Only minor performance losses using RAJA over direct CUDA
Results – CPU Portability

- 100 Million Particle Study – Done on CPU
  - [ SOA AOS ]
  - Thrust: XXX.XX XXX.XX
  - RAJA like: XXX.XX XXX.XX
  - Thrust History: XXX.XX
  - OMP History: XXX.XX

- Comment on OMP results
- Comment on Portability of event versus history

[results not yet determined]