Experiences with CUDA Streaming in Teton’s Linear Sweep

Steven Rennich*, Teresa Bailey, Aaron Black, Peter Brown, Robert Chen, Terry Haut, Adam Kunen, John Loffeld, Paul Nowak, Bujar Tagani, Ben Yee

Lawrence Livermore National Laboratory, *NVIDIA

Introduction

While GPUs are enabling the scaling of new performance peaks, their limited memory sizes are still an obstacle to codes using large quantities of data. Teton, a thermal radiative transfer code, reads and writes data many times the size of GPU memory, which results in more memory transfer overhead. To alleviate this overhead, the Teton team prototyped a CUDA-C streaming version of its linear sweep, that enables more data to be processed than the GPU can usually hold. This capability allows Teton to run larger problem sizes, and increases performance by overlapping memory transfers with computation.

Background

Linear sweep characteristics:
- Large number of unknowns.
- Dependencies between variables.
- Large % of runtime.

Fortran Implementation:
- Uses OpenMP 4.5 target offload to run sweep on GPU.
- “Mapped” host-to-device and device-to-host memory transfers.
- Memory transfers not overlapped with computation.

CUDA C Implementation:
- 3D space grid
- 2D angular grid
- 1D energy grid

Temperature Iteration Loop

Check Convergence

- Term inology:
  - 10% = time during computation when not all kernels are active
  - Ramp Up = time during computation when not all kernels are active
  - Ramp Down = time during computation when not all kernels are active

Implementation details:
- Each sweep kernel computes 1 angle, and each angle is associated with a specified number-of-groups (groups per block).
- The number of groups per block needs to be scaled by the user such that GPU memory usage is not exceeded.
- Each hyperplane contains a zone loop.
- Zones are batched to scale with the number of groups per block, into chunks. The sweep kernel logic iterates over these chunks.
- A small amount of data is directly copied to the GPU per kernel launch. Although this is not optimal, the copies are small enough to be overlapped with computation. Further optimization or caching of these data structures would have a modest impact on performance.
- Converted initialization, update, and 2 help functions into separate streaming kernels.

Problem Configuration

- 146k zones, 56 groups, 80 angles
- Mapping to GPUs:
  - 4 energy groups/SM
  - 20 simultaneous angles (kernels) per loaded zone
  - 2 IBM POWER9 CPUs, 256GB memory
  - 4 NVIDIA Volta GPUs, 16GB memory

Notes:
- Specifying shared memory variables in OpenMP 4.5 is difficult. Can be specified in 5.1.
- Streaming register count could be reduced further with more optimization of temporary variables.
- Streaming threads per block was capped at 128, but could be increased to ~800.
- Streaming sweep algorithm based on original non-parallelized version of sweep.

Introduction

Streaming code design:
- CUDA-C conversion from Fortran interface.
- Sweep variable sets within an SM to minimize synchronization overhead.
- Asynchronously stream data through GPU to minimize transfer overhead.

Overlapped Memory Transfers

• Linear sweep characteristics:
  - problem sizes, and increases performance by overlapping memory transfers with computation.

• As much as possible, keep sweep logic intact.

• Each sweep kernel computes 1 angle, and each angle is associated with a specified number-of-groups (groups per block).

• The number of groups per block needs to be scaled by the user such that GPU memory usage is not exceeded.

• Each hyperplane contains a zone loop.

• Zones are batched to scale with the number of groups per block, into chunks. The sweep kernel logic iterates over these chunks.

• A small amount of data is directly copied to the GPU per kernel launch. Although this is not optimal, the copies are small enough to be overlapped with computation. Further optimization or caching of these data structures would have a modest impact on performance.

• Converted initialization, update, and 2 help functions into separate streaming kernels.

Results

Table 1: Fortran vs. Streaming CUDA-C

<table>
<thead>
<tr>
<th></th>
<th>Fortran OpenMP 4.5</th>
<th>Streaming CUDA-C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8 GB (mem)</td>
<td>24 GB (mem)</td>
</tr>
<tr>
<td>Register count</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>Threads per block</td>
<td>512</td>
<td>128</td>
</tr>
</tbody>
</table>

Figure 1: Nvidia Performance Profile of one cycle (with 4 iterations) of streaming linear sweep

Figure 2: Nvidia Performance Profile of last iteration of linear sweep

Figure 3: Nvidia Performance Profile of last iteration of streaming linear sweep

Figure 4: Nvidia Performance Profile of one cycle (with 4 iterations) of streaming linear sweep

Figure 5: Scaling study of streaming linear sweep, maintaining constant GPU memory usage, while increasing zone count

Future Work

- Find limit of maximum number of zones which can be handled by streaming.
- Integrate streaming sweep as option into main codebase.
- Implement Fortran OpenMP 5.1 sweep with streaming and shared memory.

Figure 5: Scaling study of streaming linear sweep, maintaining constant GPU memory usage, while increasing zone count

Using Streaming Linear Sweep (CUDA-C) Last Cycle Timing (seconds)

<table>
<thead>
<tr>
<th></th>
<th>Iter 1</th>
<th>Iter 2</th>
<th>Iter 3</th>
<th>Iter 4</th>
<th>Iter 5</th>
<th>Iter 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run Time</td>
<td>0.00312492</td>
<td>0.00253443</td>
<td>0.00337925</td>
<td>0.0040936</td>
<td>0.00179817</td>
<td>0.00179817</td>
</tr>
<tr>
<td>Ramp Up</td>
<td>0.04371169</td>
<td>0.04371169</td>
<td>0.04371169</td>
<td>0.04371169</td>
<td>0.04371169</td>
<td>0.04371169</td>
</tr>
<tr>
<td>Ramp Down</td>
<td>0.04371169</td>
<td>0.04371169</td>
<td>0.04371169</td>
<td>0.04371169</td>
<td>0.04371169</td>
<td>0.04371169</td>
</tr>
<tr>
<td>Total Compute</td>
<td>0.13014451</td>
<td>0.13014451</td>
<td>0.13014451</td>
<td>0.13014451</td>
<td>0.13014451</td>
<td>0.13014451</td>
</tr>
<tr>
<td>Overhead of Total Compute = 27.97%</td>
<td>Overhead of Total Compute = 27.97%</td>
<td>Overhead of Total Compute = 27.97%</td>
<td>Overhead of Total Compute = 27.97%</td>
<td>Overhead of Total Compute = 27.97%</td>
<td>Overhead of Total Compute = 27.97%</td>
<td></td>
</tr>
</tbody>
</table>

Overlap = Average time of computing kernel and moving data to or from the GPU.

- Ramp time scales to the amount of GPU memory used (time required to fill GPU memory), and is the same regardless of problem size.
- Ramp down can be considered the streaming overhead (14.25% per cycle) for this problem configuration, which is similar to the OpenMP overhead (16.21%).

Table 2: Cycle Statistics

<table>
<thead>
<tr>
<th></th>
<th>Iter 1</th>
<th>Iter 2</th>
<th>Iter 3</th>
<th>Iter 4</th>
<th>Iter 5</th>
<th>Iter 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grind Time Per Configuration (unknowns/second)</td>
<td>25125240</td>
<td>34912985</td>
<td>41103230</td>
<td>20125240</td>
<td>34912985</td>
<td>41103230</td>
</tr>
<tr>
<td></td>
<td>Iter 1</td>
<td>Iter 2</td>
<td>Iter 3</td>
<td>Iter 4</td>
<td>Iter 5</td>
<td>Iter 6</td>
</tr>
<tr>
<td>GPU Memory Usage Per Configuration</td>
<td>56655737</td>
<td>60160000</td>
<td>56655737</td>
<td>60160000</td>
<td>56655737</td>
<td>60160000</td>
</tr>
<tr>
<td></td>
<td>Iter 1</td>
<td>Iter 2</td>
<td>Iter 3</td>
<td>Iter 4</td>
<td>Iter 5</td>
<td>Iter 6</td>
</tr>
<tr>
<td></td>
<td>80 SMs used</td>
<td>80 SMs used</td>
<td>80 SMs used</td>
<td>80 SMs used</td>
<td>80 SMs used</td>
<td>80 SMs used</td>
</tr>
<tr>
<td></td>
<td>Iter 1</td>
<td>Iter 2</td>
<td>Iter 3</td>
<td>Iter 4</td>
<td>Iter 5</td>
<td>Iter 6</td>
</tr>
<tr>
<td></td>
<td>1.2M zones</td>
<td>1.2M zones</td>
<td>1.2M zones</td>
<td>1.2M zones</td>
<td>1.2M zones</td>
<td>1.2M zones</td>
</tr>
<tr>
<td></td>
<td>Iter 1</td>
<td>Iter 2</td>
<td>Iter 3</td>
<td>Iter 4</td>
<td>Iter 5</td>
<td>Iter 6</td>
</tr>
<tr>
<td></td>
<td>600k zones</td>
<td>600k zones</td>
<td>600k zones</td>
<td>600k zones</td>
<td>600k zones</td>
<td>600k zones</td>
</tr>
<tr>
<td></td>
<td>Iter 1</td>
<td>Iter 2</td>
<td>Iter 3</td>
<td>Iter 4</td>
<td>Iter 5</td>
<td>Iter 6</td>
</tr>
<tr>
<td></td>
<td>150k zones</td>
<td>150k zones</td>
<td>150k zones</td>
<td>150k zones</td>
<td>150k zones</td>
<td>150k zones</td>
</tr>
</tbody>
</table>

- Streaming implementation is bounded by system memory, rather than GPU memory.
- 4+ groups/SM enables coalesced memory accesses for better performance.
- Spreading problem over more SMs begets better throughput.
- Can fit problems onto GPU by scaling SM use (# angle kernels), and/or groups per SM.