Examples of GPU optimisation for wave propagation problems

Dr. Sofya Titarenko, PDRA in LIDA
Summary

• This talk will focus on NVDIA GPUs;
• Not every problem can be solved using GPU;
• Different problems gain different speedup when moving from CPU onto GPUs;
• It may take effort to find the best method of optimisation;
• For many problems it is probably easiest to start with NVIDIA libraries.
Q: When do we choose to use GPU?
Answer: The problem has to include a routine calculation applied to a big set of data.

Single Instruction Multiple Data (SIMD) principle!
Examples: FFT, Large Matrix-Vector operations, Finite Difference calculations...

Solving wave equation, heat equation etc...
Wave propagation through elastic media with a fracture
Wave propagation through elastic media

\[ \begin{align*}
\rho \frac{\partial \dot{u}_1}{\partial t} &= \frac{\partial \sigma_{11}}{\partial x_1} + \frac{\partial \sigma_{12}}{\partial x_2}, \\
\rho \frac{\partial \dot{u}_2}{\partial t} &= \frac{\partial \sigma_{22}}{\partial x_2} + \frac{\partial \sigma_{12}}{\partial x_1}, \\
\frac{\partial \sigma_{11}}{\partial t} &= \left( \lambda + 2\mu \right) \frac{\partial \dot{u}_1}{\partial x_1} + \lambda \frac{\partial \dot{u}_2}{\partial x_2}, \\
\frac{\partial \sigma_{12}}{\partial t} &= \mu \frac{\partial \dot{u}_1}{\partial x_2} + \mu \frac{\partial \dot{u}_2}{\partial x_1}, \\
\frac{\partial \sigma_{22}}{\partial t} &= \lambda \frac{\partial \dot{u}_1}{\partial x_1} + \left( \lambda + 2\mu \right) \frac{\partial \dot{u}_2}{\partial x_2},
\end{align*} \]

**Finite Difference** approach, leap-frog approach

Applying FD formula for the whole array of \( \sigma_{11}, \sigma_{12}, \sigma_{22}, \dot{u}_1, \dot{u}_2 \)

**SIMD principle! Good for GPU**

How we solve it on GPU?

1. Move data from **host (CPU)** to device (GPU)

   *It is relatively slow, should be minimized. Use asynchronous transaction?*

   **Global memory** is large (ex. Tesla K80, 12GB), but **slowest**
   **Shared memory** is small (ex. Tesla K80, 64KB), but **fast**

   It is more efficient to divide data into blocks so the data you work with can fit in shared memory.
Running on GPU

- All threads on GPU can be represented as grid of blocks;

- Problem is run on GPU threads; parallelism
Wave propagation problem on GPU

Divide dataset on blocks
Wave propagation problem on GPU

Divide dataset on blocks

One block
Wave propagation problem on GPU

Divide dataset on blocks

One block

Grid points
Wave propagation problem on GPU

Divide dataset on blocks

One block

Threads

Grid points
Wave propagation problem on GPU

Divide dataset on blocks

One block

Threads

Grid points
Tips for acceleration

- Use shared memory;
- Use block sizes so they could fit in shared memory; In my example it is [32X32]
- Avoid bank conflicts;

Use NVIDIA profiler to get the idea where bottlenecks are

http://cuda-programming.blogspot.co.uk/2013/02/bank-conflicts-in-shared-memory-in-cuda.html
### Speedup on single GPU (GeForce GTX 760, CPU Intel core i7, 6 cores, Sandy Bridge)

**Problem:** Wave propagation through elastic medium *with cracks*, 1000 steps

<table>
<thead>
<tr>
<th>N, size=NXN</th>
<th>-O3, no OMP, sec</th>
<th>OMP, -O3, sec</th>
<th>OMP, -O3, improved, sec</th>
<th>GPU (using CUDA), sec</th>
<th>Speed up vs the most optimised</th>
</tr>
</thead>
<tbody>
<tr>
<td>32X32X2</td>
<td>1164.440 (19.41 min)</td>
<td>227.556 (3.79 min)</td>
<td>125.662 (2.09 min)</td>
<td>10.571</td>
<td>11.89</td>
</tr>
<tr>
<td>32X64X2</td>
<td>4426.665</td>
<td>882.383</td>
<td>460.963</td>
<td>41.640</td>
<td>11.07</td>
</tr>
</tbody>
</table>

### Conclusion:

Speedup depends on the problem: it could be higher for the more complicated problems!
Acceleration on multiple GPUs

**Problem:** Wave propagation through elastic medium *with cracks*, 1000 steps

<table>
<thead>
<tr>
<th>size</th>
<th>/OMP best, sec</th>
<th>GeForce GTX 760, sec</th>
<th>GeForce GTX 770, sec</th>
<th>2 GeForce cards, sec</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>32X32X2</td>
<td>125.662</td>
<td>10.571</td>
<td>7.435</td>
<td>5.925</td>
<td>21.209</td>
</tr>
<tr>
<td>32X64X2</td>
<td>460.963</td>
<td>41.640</td>
<td>31.338</td>
<td>23.951</td>
<td>19.24</td>
</tr>
</tbody>
</table>

*Estimation:* for two GeForce GTX 770 cards I would expect speedup ~ 30 times!

**Conclusion:** speedup depends on the graphics card is used. But it make sense to upgrade a system with extra GPU card, even if it is less powerful
Wave propagation through layered media


Grid-characteristic monotonic difference method

TVD (Total Variation Diminishing) Finite Difference Scheme

Applying TVD finite difference formula to all 9 arrays of variables \( \{ \dot{u}_1, \dot{u}_1, \dot{u}_1, \sigma_{11}, \sigma_{12}, \sigma_{13}, \sigma_{22}, \sigma_{23}, \sigma_{33} \}^T \)

**SIMD principle! Good for GPU**

<table>
<thead>
<tr>
<th>GPU</th>
<th>Cores</th>
<th>Clock rate, MHz</th>
<th>GFLOPS - single precision</th>
<th>SP:DP</th>
<th>GFLOPS - double precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>GeForce GT 640</td>
<td>384</td>
<td>900</td>
<td>691</td>
<td>24</td>
<td>29</td>
</tr>
<tr>
<td>GeForce GT 480</td>
<td>480</td>
<td>1401</td>
<td>1345</td>
<td>8</td>
<td>168</td>
</tr>
<tr>
<td>GeForce GTX 680</td>
<td>1536</td>
<td>1006</td>
<td>3090</td>
<td>24</td>
<td>129</td>
</tr>
<tr>
<td>GeForce GTX 760</td>
<td>1152</td>
<td>980</td>
<td>2258</td>
<td>24</td>
<td>94</td>
</tr>
<tr>
<td>GeForce GTX 780</td>
<td>2304</td>
<td>863</td>
<td>3977</td>
<td>24</td>
<td>166</td>
</tr>
<tr>
<td>GeForce GTX 780 Ti</td>
<td>2880</td>
<td>876</td>
<td>5046</td>
<td>24</td>
<td>210</td>
</tr>
<tr>
<td>Tesla M2070</td>
<td>448</td>
<td>1150</td>
<td>1030</td>
<td>2</td>
<td>515</td>
</tr>
<tr>
<td>Tesla K40m</td>
<td>2880</td>
<td>745</td>
<td>4291</td>
<td>3</td>
<td>1430</td>
</tr>
<tr>
<td>Tesla K80</td>
<td>2496</td>
<td>562</td>
<td>2806</td>
<td>1.5</td>
<td>1870</td>
</tr>
<tr>
<td>Radeon HD 7950</td>
<td>1792</td>
<td>800</td>
<td>2867</td>
<td>4</td>
<td>717</td>
</tr>
<tr>
<td>Radeon R9 290</td>
<td>2560</td>
<td>947</td>
<td>4849</td>
<td>8</td>
<td>606</td>
</tr>
</tbody>
</table>

- **CPU**
  - Intel Xeon E5-2697 2.7 GHz
  - Compilers: icc
  - Compiler Options:
    - -mavx
    - -fopenmp (auto vectorization)
    - -O2

- **GPU**
  - Compilers: nvcc, gcc
  - Compiler Options:
    - -O2
    - -use_fast_math
“Applying CUDA and OpenCL technology for modelling seismic processes using grid-characteristics methods”, Andrey Ivanov, Nikolay Khokhlov, Moscow Institute of Physics and Technology, Russian Supercomputing Days 2016.

<table>
<thead>
<tr>
<th>GPU</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radeon R9 290 (AMD)</td>
<td>£314</td>
</tr>
<tr>
<td>Tesla K80 (NVIDIA)</td>
<td>£4,962</td>
</tr>
<tr>
<td>GeForce GTX 780</td>
<td></td>
</tr>
<tr>
<td>GeForce GTX 770</td>
<td>£429.98</td>
</tr>
<tr>
<td>GeForce GTX 760</td>
<td>£149.99</td>
</tr>
</tbody>
</table>
Conclusions: Radeon AMD cards are actually work **better** both for single precision and double precision while cost much less than NVIDIA Tesla! GeForce can work better than Tesla for single precision, Tesla works better than GeForce for double precision.
Speedup on multiple GPUs, CUDA (for multiple GPU processor setup)

"Applying CUDA and OpenCL technology for modelling seismic processes using grid-characteristic methods",

Andrey Ivanov, Nikolay Khokhlov, Moscow Institute of Physics and Technology, Russian Supercomputing Days 2016.

**Problem:** wave propagation through irregularly layered medium

**Conclusions:** multiple Teslas work a little bit better than multiple GeForces.
Conclusions

• GPU parallelisation can help sufficiently speed up calculations;
• GPU parallelisation can help sufficiently speed up calculations while keeping hardware and software expenses low;
• Using multiple GPUs where it is possible can help further speed up of calculations.

Some tips

• The formulation of problem we want to calculate does matter;
• Try to formulate problem using SIMD principle;
• Try to minimise exchange between CPU host and GPU;
• Try to use fast shared memory where it is possible; Block size fit in shared memory;
• Try to avoid “if” construction in your loop and use threads efficiently;
Thank you for your attention!

Any questions?