Introduction to GPU programming

for

PDC summer school 2008

by

Tomas Oppelstrup
Outline

- An overview of the CPU
- What is a graphics card?
- Why do we want to use it?
- How can it be used in scientific computing?
- How hard is it to use?
- Example of algorithm changes necessary
- Examples of successful applications
The mess of a modern CPU

- Out-of-order execution (72 in-flight instructions)
- Exceptions (address, floating point, ...)
- Address translation (TLB, protection)
- Branch prediction
- Register renaming
- Cache management
- ...all this requires a lot of silicon
Closeup of an AMD Opteron

Only a fraction of the chip does computations
What is a graphics card?

- A circuit board that draws pretty pictures
- A specialized piece of hardware
- Designed for maximum performance in image drawing
- Modern games require enormous performance
- Basis for GPU computing is that GPU's are becoming more versatile
If we go back to the early 90's...

- Fastest machines were CRAY's...
  - Vector machine
  - Fast memory streams vectors through a very fast processor
- ...and Connection Machines
  - Massively parallel architecture
  - Very many slower processors each compute on one element of the result vector.
Specialized machines

• Specialized machines like the CRAY's and CM's devote hardware to do certain computations extremely well (e.g. linear algebra).

• Today's CPU's are versatile and general prupose. They do a lot of things with rather well.
  – They are fabulous because they are so versatile
The return of the vector machine

A modern GPU is both a vector machine and massively parallel!
Why graphics cards?

**Pros**
- Fast
- Cheap
- Low-power
- Future is streaming anyway?

**Cons**
- Specialized
- Hard to program
- Bandwidth problems
- Rapidly changing
GPU's are much faster than CPU's

A 200 Gflops card is $150

Performance history:

AMD FireStream 9250: 1Tflops, 8 Gflops/Watt
Fastest machine in the world is a PlayStation

1993

Roadrunner at Los Alamos

2008
GPU's are fast in reality too

Folding @ Home statistics:

<table>
<thead>
<tr>
<th>OS Type</th>
<th>Current TFLOPS*</th>
<th>Active CPUs</th>
<th>Total CPUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>199</td>
<td>209394</td>
<td>2137161</td>
</tr>
<tr>
<td>Mac OS X/PowerPC</td>
<td>7</td>
<td>8246</td>
<td>118873</td>
</tr>
<tr>
<td>Mac OS X/Intel</td>
<td>20</td>
<td>6384</td>
<td>59056</td>
</tr>
<tr>
<td>Linux</td>
<td>62</td>
<td>36336</td>
<td>326100</td>
</tr>
<tr>
<td>ATI GPU</td>
<td>354</td>
<td>3215</td>
<td>6331</td>
</tr>
<tr>
<td>NVIDIA GPU</td>
<td>1350</td>
<td>12273</td>
<td>18055</td>
</tr>
<tr>
<td>PLAYSTATION®3</td>
<td>1109</td>
<td>39332</td>
<td>583874</td>
</tr>
<tr>
<td>Total</td>
<td>3101</td>
<td>315180</td>
<td>3249450</td>
</tr>
</tbody>
</table>

- 1-3 Gflop/s average from PC's
- 100 Gflop/s average from GPU's
The GPU Devotes More Transistors to Data Processing
nVidia GPU architecture

- Many processors
  - Striped together
- No cache coherency
- 256kb cache used by 128 processors
- 300 flop/s
- 80 Gb/s
How hard is it?

- High level languages
  - Brook+, CUDA (C)
- Available libraries:
  - BLAS, FFT, sorting, sparse multiply
- Add-ons for MATLAB and Python
- Rapidly growing community
- ...but still hard for complicated algorithms
Algorithm example 1: Summation

- Traditionally
  ```
  sum = 0.0;
  for(i = 1; i<N; i++) sum = sum + x[i];
  ```

- Have to exploit parallelism on GPU
  
  - Use tree reduction:
    ```
    /* p is thread id */
    b = N / 2;
    while (b > 0) {
      if(p < b)
        x[p] = x[p] + x[p+b];
      b = b / 2;
      __syncthreads();
    } /* x[0] contains sum of x[i] */
    ```
Example 2: Heat equation

• Want to solve heat equation in time:
  \[ u(i,j) := u(i,j) + D \left( u(i,j-1) + u(i,j+1) + u(i-1,j) + u(i+1,j) - 4u(i,j) \right) \]

• Naïve algorithm is bandwidth limited
  – 7 flops and 6 memory accesses.

• GPU's can do 20 flops per memory access

• Must make use of fast memory (cache)
Heat equation continued

• Loading a tile into cache enables reuse of data
• Happens automatically on the CPU
• Has to be programmed on the GPU
• Strategy:
  – Load MxM tile of u into cache.
  – Compute result of (M-2)x(M-2) internal points
  – Store (M-2)x(M-2) result tile back
• 7 flops per 2 memory accesses for large tile
Heat equation continued

- 3.5 flops / memory access not good enough
  - Have large overlap between tiles
  - Can run several timesteps before writing result back to memory

- Reuse of data in cache reduces memory bandwidth at the cost of extra flops
Multi timestep analysis

- 2D Heat equation
- 32x32 grid point tiles
- Can get 12 useful flops per memory access
- Wasted flops are free; waiting for memory anyway
- Beneficial on CPU's too
In summary: People are doing it...

PS3GRID / BOINC

Examples from CUDA Zone

Accelerating Density Functional Calculations with Graphics Processing Unit

Koji Yasuda

Graduate School of Information Science, Nagoya University, Chikusa-ku, Nagoya 464-8601, Japan

Received March 27, 2008

Abstract: An algorithm is presented for graphics processing units (GPUs), which execute single-precision arithmetic much faster than commodity microprocessors (CPUs), to calculate the exchange-correlation term in ab initio density functional calculations. The algorithm was implemented and applied to two molecules, taxol and valinomycin. The errors in the total energies correlation latter is resulting a factor functional.