GPGPU

• Apply consumer parallel graphics hardware for general purpose (GP) computing
• GPU almost comes with every PC
• Let’s focus on two approaches:
  – Shader programming
  – CUDA
Shader Programming

- GPU is not originally designed for GPGPU, but for graphics
- Shader (program)
- Shading language (specialized language, C-like)
- A graphics “shell” is needed to perform your GP program
Programming as “Drawing”

• Every program must be a “drawing” even you draw nothing
• Two dummy triangles to cover the screen
Programming as “Drawing” (2)

- Then, rasterization (discretization to pixels)
- Each pixel triggers a shader
Pixel as Chromosome

- For EC, it is natural to have each pixel being a chromosome
- Each shader evaluates the objective function
CUDA

- A tailormade platform for GPGPU on GPU
- No dummy graphics “shell”
CUDA Architecture

- shader => kernel
- Shared memory
- Thread synchronization
- Communication!
Shader vs CUDA

- **Learning curve:**
  - **Shader:** Dummy graphics “shell” needed, and specialized shading language
    => Longer learning curve for non-graphics people
  - **CUDA:** Just like multi-thread programming, basically C language
    => easier to catch up for most people
Shader vs CUDA

• Communication among processes:
  - **Shader**: No communication
    => multiple passes, read & write textures for data sharing
  - **CUDA**: Yes, via shared memory & synchronization
    => less passes, more efficient and flexible
Shader vs CUDA (2)

- **Logical number of instances**
  - **Shader**: Strongly coupled with screen resolution
    No. of pixels = No. of shader instances
    = No. of chromosomes
    => Straightforward problem formulation
  - **CUDA**: Depends on hardware limit
    No. of threads < No. of chromosomes
    => Each thread handles multiple chromosomes
Shader vs CUDA (3)

- **Efficiency**
- In theory, CUDA should be as efficient as shader programming
Shader vs CUDA (4)

• Standardization
  – **Shader**: There are standards
    GLSL (OpenGL shading language)
    HLSL (MS DirectX high level shading language)
    => cross-platform (can be ATI or nVidia)
  – **CUDA**: Standard is still forming
    CUDA is basically supported by vendor nVidia,
    not sure whether it will be supported by ATI
Shader vs CUDA (5)

- Access to graphics specific functionalities
- Mipmapping, Cubemap look-up
  - **Shader**: Accessible
    => fast evaluation (lookup) of spherical functions
    => fast downsampling and upsampling
  - **CUDA**: No access
Debugging Shader

• So far, quite limited
• printf-style visual debugging (graphics)
• Microsoft Shader Debugger – MS DirectX shaders can be debugged
  – Shader emulation on CPU, not debugging on actual GPU
  – seldom use as we stick to OpenGL for backward compatibility
Debugging Shader (2)

- NVIDIA Shader Debugger for FX Composer
  - recently released in April 2008, as a plugin for FX composer!?

- glsldevil, OpenGL GLSL Debugger
  [http://www.vis.uni-stuttgart.de/glsdevil/](http://www.vis.uni-stuttgart.de/glsdevil/)
Debugging Shader (3)

- Execution cycle needed for a shader can be determined offline

```
nvshaderperf -a G70 -f main shader.cg
```

![nvshaderperf output](image)


T. T. Wong

5 June 2008, CIGPU, WCCI 2008
Debugging CUDA

- CUDA can be executed in device emulation mode => threads are executed sequentially
- Set break point is feasible
- Currently, debugging tools are still quite scarce
Debugging CUDA (2)

- VC++ debug modes
  - EmuDebug, Debug
- Kernel codes are traceable in EmuDebug (emulation) mode, not on actual hardware
- gdb debugger (not yet released)
Debugging CUDA (3)

• Profiling in CUDA

By enabling CUDA_PROFILE: to enable (1) or disable (0)

./shaderprogram -N1024
method=[ memcopy ] gputime=[ 1427.200 ]
method=[ memcopy ] gputime=[ 10.112 ]
method=[ memcopy ] gputime=[ 9.632 ]
method=[ real2complex ] gputime=[ 1654.080 ] cputime=[ 1702.000 ] occupancy=[ 0.667 ]
method=[ c2c_radix4 ] gputime=[ 8651.936 ] cputime=[ 8683.000 ] occupancy=[ 0.333 ]
method=[ transpose ] gputime=[ 2728.640 ] cputime=[ 2773.000 ] occupancy=[ 0.333 ]
method=[ c2c_radix4 ] gputime=[ 8619.968 ] cputime=[ 8651.000 ] occupancy=[ 0.333 ]
method=[ c2c_transpose ] gputime=[ 2731.456 ] cputime=[ 2762.000 ] occupancy=[ 0.333 ]
method=[ solve_poisson ] gputime=[ 6389.984 ] cputime=[ 6422.000 ] occupancy=[ 0.667 ]
method=[ c2c_radix4 ] gputime=[ 8518.208 ] cputime=[ 8556.000 ] occupancy=[ 0.333 ]
method=[ c2c_transpose ] gputime=[ 2724.000 ] cputime=[ 2757.000 ] occupancy=[ 0.333 ]
method=[ c2c_radix4 ] gputime=[ 8618.752 ] cputime=[ 8652.000 ] occupancy=[ 0.333 ]
method=[ c2c_transpose ] gputime=[ 2767.840 ] cputime=[ 5248.000 ] occupancy=[ 0.333 ]
method=[ complex2real_scaled ] gputime=[ 2844.096 ] cputime=[ 3613.000 ] occupancy=[ 0.667 ]
method=[ memcopy ] gputime=[ 2461.312 ]
Debugging CUDA (4)

• Occupancy -- amount of shared memory and registers used by each thread block
• CUDA occupancy calculator computes the multiprocessor occupancy of the GPU by a given CUDA kernel

Panel Discussions

- Components needed for GPGPU from the perspective of EC community
- Debugging experience
- Standardization of GPGPU platforms and languages
- Any other topics