PROGRAMMABILITY: DESIGN COSTS AND PAYOFFS USING AMD GPU STREAMING LANGUAGES AND TRADITIONAL MULTI-CORE LIBRARIES

7/30/09
Rick Weber, Robert Harrison, Greg Peterson
Multi-core Processors

- Put multiple CPUs on a single chip
- Multi-core becoming the norm in computing
  - Helps overcome power wall
  - Sometimes helps overcome memory wall
  - More efficient use of transistors than complex branch prediction, scoreboarding, etc.

- Usually requires explicit software support to reap benefits (in a single application)
  - Unlike many CPU performance features, speedups don’t come for “free”
  - Requires developers to exploit thread level parallelism (usually explicitly)
  - OpenMP, pthreads, Intel Threading Blocks
General Purpose Graphics Processing Unit (GPGPU)

- Traditionally used for gaming and CAD
  - Fixed pipeline replaced with programmable shader units
  - Lots of simple processors for computing pixels in parallel
  - Large memory bandwidth for sampling textures

- Now being used in general purpose computing
  - CUDA, Brook+, CAL, OpenCL all target GPUs
  - Can achieve exploit massive parallelism
  - Double precision now supported on some cards
    - Firestream series
    - Tesla 1060
ATI GPU architecture
ATI GPU Architecture

- Stream processor bundles SIMD Engines
- SIMD Engine
  - Controls thread processor execution
  - All thread processors within SIMD engine execute same instruction
  - Analogous to Nvidia’s SIMT Multiprocessors
  - Very long instruction word (VLIW)
- Thread processors
  - Analogous to Nvidia’s thread processors
  - 5 floating point units (stream cores) per processor
Firestream 9170

- 320 Stream cores
- \( \sim 500 \text{Gflops/s} \) peak (single precision)
- \( \sim 100 \text{Gflops/s} \) peak (double precision)
  - 4 SP units are fused to perform one double operation
  - T-unit is not used in double
- 51.2GB/s memory bandwidth
Programming AMD GPUs

**Computation Oriented APIs**
- Brook+
- Intermediate Language (IL)
- Compute Abstraction Layer (CAL)
- Brook+ runtime
- Assembly Language

**Graphics Oriented APIs**
- OpenCL
- DirectX
- OpenGL
- HLSL
- GLSL
Streaming Model

Set of elements in output stream is called **domain**

Code snippet:

```c
kernel void myKernel(float a<>,
float b<> , out float c<>) {
float tmp;
tmp = a + b;
c = tmp;
}
```
Mapping Streaming Model onto AMD GPUs
APIs

**CAL**

- Low level API for stream programming
- C frontend
  - Very verbose
- Intermediate Language (IL) kernels
  - Pseudo-assembly language
- SIMD data types

**Brook+**

- High level C-like language based on Brook project
- Streaming model
- Very abstract!
  - Read and write to streams
Compute Gaussian basis functions to approximate Slater functions

Used in Hartree-Fock method to approximate ground state wave function of n-body quantum mechanical system

exp(-a r^2), where a is spread and r^2 is distance between two orbitals
Grid Potential

n Basis functions (a)
m coordinates (x, y, z)

\[ r^2 = x^2 + y^2 + z^2; \exp(a \times r^2) \]

mxn grid potentials
Naïve C implementation

```c
for(i = 0; i < npt; i++)
{
    float r = x[i] * x[i] + y[i] * y[i] + z[i] * z[i];

    for(j = 0; j < nbas; j++)
    {
        basis2[j*npt+i] = exp(alpha[j] * r);
    }
}
```
Naïve GPU implementations

Brook+

```c
1 kernel void computeBasisFunction(float alpha[],
2 float xCoord[],
3 float yCoord[],
4 float zCoord[],
5 out float basis->
6 {
7     float2 index = indexof(basis).xy;
8     float x2;
9     float y2;
10    float z2;
11    float r2;
12
13    x2 = xCoord[index.y] * xCoord[index.y];
14    y2 = yCoord[index.y] * yCoord[index.y];
15    z2 = zCoord[index.y] * zCoord[index.y];
16
17    r2 = x2 + y2 + z2;
18
19    basis = exp(alpha[index.x] * r2);
20 }
```

CAL

```c
kernel char* basisFunc =
"ll_ps_2_B
" "dl_output_generic.c.in"
"dl_literall.0, 2.7102815264590423536, 2.7102815264590423536, 2.7102815264590423536, 2.7102815264590423536, 2.7102815264590423536, 2.7102815264590423536, 2.7102815264590423536, 2.7102815264590423536"
"dl_input_position_intels(linear_interpolate) yWinCoord0.xy
." """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " """ " ""`
Optimizations

<table>
<thead>
<tr>
<th>Optimization</th>
<th>x86</th>
<th>Brook+</th>
<th>CAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel unrolling</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SIMD</td>
<td>Yes (Intel MKL VML)</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Precompute radii</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Cache alphas</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Cache block coords</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
Performance in grid potentials / sec

Platform Performance

Naive implementation
Optimized implementation

0.02
0.86

Brook+   CAL   8xXeon5355@2.66Ghz

Gex/s
Final Speedup

**Total Performance Increase over Optimized x86**

- X86: 1.00
- Brook+: 3.08
- CAL: 11.83

**Naive Performance Speedup**

- X86: 1.00
- Brook+: 107.96
- CAL: 413.96

Tennessee Advanced Computing Laboratory 7/30/09
Conclusions

- Streaming is an elegant solution to SOME applications
  - Data must be completely independent
  - Data must be input or output (no inout)
- Brook+ allows for quick and dirty implementation and can give modest speedups over CPU
  - But compiler-introduced overhead hinders performance in short kernels
- CAL is difficult to program but allows more control (and hence more performance)