High Level Languages for GPUs

Mike Houston
Stanford University
High Level Shading Languages

- **Cg, HLSL, & OpenGL Shading Language**
  - **Cg:**
    - http://www.NVIDIA.com/cg
  - **HLSL:**
  - **OpenGL Shading Language:**
Compilers: CGC & FXC

• HLSL and Cg are syntactically almost identical
  - Exception: Cg 1.3 allows shader “interfaces”, unsized arrays

• Command line compilers
  - Microsoft’s FXC.exe
    • Compiles to DirectX vertex and pixel shader assembly only
    • fxc /Tps_3_0 myshader.hlsl
  - NVIDIA’s CGC.exe
    • Compiles to everything
    • cgc -profile ps_3_0 myshader.cg
  - Can generate very different assembly!
    • Driver will recompile code
  - Compliance may vary
Babelshader

http://graphics.stanford.edu/~danielrh/babelshader.html

- Converts between DirectX pixel shaders and OpenGL shaders
- Allows OpenGL programs to use DirectX HLSL compilers to compile programs into ARB or fp30 assembly.
- Enables fair benchmarking competition between the HLSL compiler and the Cg compiler on the same platform with the same demo and driver.

```plaintext
texld r9, t5, s4 ; fetch
dp2add r0.a, r0, r0,-c0.z
rsq r0.a, r0.a
tcp r0.b, r0.a
mad r3, r8, c0.w, -c0.z
mad r8, r3, c4.r, r0
mad r3, r9, c0.w, -c0.z
mad r7, r3, c4.g, r0
mad r1.a, r5.z, c3.x,c3.y
dp3 r4.a, t4, t4
rsq r4.a, r4.a ; 1/view
mul r4, t4, r4.a ;norm ; reflection vector
mul t2.rgb, r0.x, t1
mad t2.rgb, r0.y, t2, t2
mad t2.rgb, r0.z, t3, r2 ;transform bump map normal

TEX r9, t5, texture[41], 2D;
MAD R0.w, R0.x, R0.x, -c0.z;
MAD R0.a, R0.y, R0.y, R0.w;
RSQ r6.a, r0.a;
TCP r0.b, r0.a;
MAD r3, r8, c0.w, -c0.z;
MAD r6, r3, c4.r, r0;
MAD r3, r9, c0.w, -c0.z;
MAD r7, r3, c4.g, r0;
MAD r1.a, r5.z, c3.x,c3.y;
DP3 r4.a, t4, t4;
RSQ r4.a, r4.a;
; 1/view
MUL r4, t4, r4.a;
MUL t2.rgb, r0.x, t1;
MUL t2.rgb, r0.y, t2, t2;
MAD t2.rgb, r0.y, t2, r2;
MAD t2.rgb, r0.z, t3, r2;
```
GPGPU Languages

• Why do you want them?
  - Make programming GPUs easier!
    • Don’t need to know OpenGL, DirectX, or ATI/NV extensions
    • Simplify common operations
    • Focus on the algorithm, not on the implementation

• Accelerator
  Microsoft Research
  http://research.microsoft.com/research/downloads/

• Brook
  Stanford University
  http://brook.sourceforge.net
  http://graphics.stanford.edu/projects/brookgpu

• CTM
  ATI/AMD

• CUDA
  NVIDIA
  http://www.NVIDIA.com/object/cuda.html

• Peakstream
  http://www.peakstreaminc.com

• RapidMind
  Commercial follow-on to Sh
  http://www.rapidmind.net
Microsoft Research Accelerator Project

- GPGPU programming using data parallelism
- Presents a data-parallel library to the programmer.
  - Simple, high-level set of operations
- Library just-in-time compiles to GPU pixel shaders or CPU code.
  - Runs on top of .NET
Data-parallel array library

- Explicit conversions between data-parallel arrays and normal arrays
- Functional: each operation produces a new data-parallel array.
- Eliminate certain operations on arrays to make them data-parallel
  - No aliasing, pointer arithmetic, individual element access
Data-parallel array types

Array\_1[ ... ]

DPArray\_1[ ... ]

library\_calls()

DPArray\_N[ ... ]

Array\_N[ ... ]

CPU

GPU

API/Driver/ Hardware

txtr\_1[ ... ]

pix\_shdrs()

... txtr\_N[ ... ]
Explicit conversion between data-parallel arrays and normal arrays trigger GPU execution.
Functional style: each operation produces a new data-parallel array.
Types of operations

Restrict operations to allow data-parallel programming:
No pointer arithmetic, individual element access/update
Operations

- Array creation
- Element-wise arithmetic operations: +, *, -, etc.
- Element-wise boolean operations: and, or, >, < etc.
- Type conversions: integer to float, etc.
- Reductions/scans: sum, product, max, etc.
- Transformations: expand, pad, shift, gather, scatter, etc.
- Basic linear algebra: inner product, outer product.
Example: 2-D convolution

```c
float[,] Blur(float[,] array, float[] kernel) {
    using (DFPA parallelArray = new DFPA(array)) {
        FPA resultX = new FPA(0.0f, parallelArray.Shape);
        for (int i = 0; i < kernel.Length; i++) { // Convolve in X direction.
            resultX += parallelArray.Shift(0,i) * kernel[i];
        }
        FPA resultY = new FPA(0.0f, parallelArray.Shape);
        for (int i = 0; i < kernel.Length; i++) { // Convolve in Y direction.
            resultY += resultX.Shift(i,0) * kernel[i];
        }
        using (DFPA result = resultY.Eval()) {
            float[,] resultArray;
            result.ToArray(out resultArray);
            return resultArray;
        }
    }
}
```

Just-in-time compiler

Programmer

C# code building up an expression using the Accelerator API

Coercion to normal C# array

Accelerator

Build Expression Dag

Build Canonical Shader Dag

Optimize Shader Dag

Run Shader Dag

DirectX

Transfer Data
Initialize Pipeline
Triangle Setup

Compile Pixel Shader

Render
Availability and more information

- Binary version of Accelerator available for download
  - [http://research.microsoft.com/downloads](http://research.microsoft.com/downloads)
- Available for non-commercial use
  - Meant to support research community use.
  - Licensing for commercial use possible.
- Includes documentation and a few samples
- Runs on Microsoft.NET, most GPUs shipping since 2002.
- More information:
  - ASPLOS 2006 “Accelerator: using data-parallelism to program GPUs for general-purpose uses”, David Tarditi, Sidd Puri, Jose Oglesby
  - [http://research.microsoft.com/act](http://research.microsoft.com/act)
Brook: General Purpose Streaming Language

• Stream programming model
  - GPU = streaming coprocessor

• C with stream extensions

• Cross platform
  - ATI & NVIDIA
  - OpenGL, DirectX, CTM
  - Windows & Linux
Streams

- Collection of records requiring similar computation
  - particle positions, voxels, FEM cell, ...

  \[ \text{Ray } r<200>; \]
  \[ \text{float3 } \text{velocityfield}\langle100,100,100\rangle; \]

- Similar to arrays, but...
  - index operations disallowed: \[ \text{position}[i] \]
  - read/write stream operators
    \[ \text{streamRead} (r, \ r\_ptr); \]
    \[ \text{streamWrite} (\text{velocityfield}, \ v\_ptr); \]
Kernels

• Functions applied to streams
  - similar to for_all construct
  - no dependencies between stream elements

kernel void foo (float a<>, float b<>,
    out float result<>) {
    result = a + b;
}

float a<100>;
float b<100>;
float c<100>;

foo(a,b,c);

for (i=0; i<100; i++)
    c[i] = a[i]+b[i];
Kernels

• Kernel arguments
  - input/output streams

```c
kernel void foo (float a<>,
                float b<>,
                out float result<>)
{
    result = a + b;
}
```
Kernels

- Kernel arguments
  - input/output streams
  - gather streams

```c
kernel void foo (... , float array[] ) {
    a = array[i];
}
```
Kernels

- Kernel arguments
  - input/output streams
  - gather streams
  - iterator streams

```c
kernel void foo (... , iter float n<> ) {
    a = n + b;
}
```
Kernels

• Kernel arguments
  - input/output streams
  - gather streams
  - iterator streams
  - constant parameters

```c
kernel void foo (... , float c ) {
    a = c + b;
}
```
Reductions

- Compute single value from a stream
  - associative operations only

```c
reduce void sum (float a<>,
    reduce float r<>)
    r += a;
}

float a<100>;
float r;
sum(a,r);
```

```c
r = a[0];
for (int i=1; i<100; i++)
    r += a[i];
```
Reductions

- Multi-dimension reductions
  - stream “shape” differences resolved by reduce function

```c
reduce void sum (float a<>,
    reduce float r<>)
    r += a;
}
```

```c
float a<20>;
float r<5>;
sum(a,r);
```

```c
for (int i=0; i<5; i++)
    r[i] = a[i*4];
for (int j=1; j<4; j++)
    r[i] += a[i*4 + j];
```
Stream Repeat & Stride

• Kernel arguments of different shape
  - resolved by repeat and stride

```
kernel void foo (float a<>, float b<>,
                 out float result<>);

float a<20>;
float b<5>;
float c<10>;

foo(a,b,c);
```

foo(a[0], b[0], c[0])
foo(a[2], b[0], c[1])
foo(a[4], b[1], c[2])
foo(a[6], b[1], c[3])
foo(a[8], b[2], c[4])
foo(a[10], b[2], c[5])
foo(a[12], b[3], c[6])
foo(a[14], b[3], c[7])
foo(a[16], b[4], c[8])
foo(a[18], b[4], c[9])
Matrix Vector Multiply

```
kernel void mul (float a<>, float b<>,
    out float result<>) {
    result = a*b;
}

reduce void sum (float a<>,
    reduce float result<>) {
    result += a;
}

float matrix<20,10>;
float vector<1, 10>;
float tempmv<20,10>;
float result<20, 1>;

mul(matrix,vector,tempmv);
sum(tempmv,result);
```
Matrix Vector Multiply

kernel void mul (float a<>, float b<>,
    out float result<>) {
    result = a*b;
}

reduce void sum (float a<>,
    reduce float result<>) {
    result += a;
}

float matrix<20,10>;
float vector<1, 10>;
float tempmv<20,10>;
float result<20, 1>;

mul(matrix,vector,tempmv);
sum(tempmv,result);

GPGPU
• Accessing stream data for graphics aps
  - Brook runtime api available in C++ code
  - autogenerated .hpp files for brook code

brook::initialize( "dx9", (void*)device );

// Create streams
fluidStream0 = stream::create<float4>( kFluidSize, kFluidSize );
normalStream = stream::create<float3>( kFluidSize, kFluidSize );

// Get a handle to the texture being used by
// the normal stream as a backing store
normalTexture = (IDirect3DTexture9*)
    normalStream->getIndexedFieldRenderData(0);

// Call the simulation kernel
simulationKernel( fluidStream0, fluidStream0, controlConstant,
    fluidStream1 );
Applications

- ray-tracer
- segmentation
- fft edge detect
- linear algebra

SAXPY

SGEMV
Brook for GPUs

- **Release v0.3 available on Sourceforge**
  - CVS tree *much* more up to date

- **Project Page**

- **Source**

- **Paper:**
  - Brook for GPUs: Stream Computing on Graphics Hardware
  - Ian Buck, Tim Foley, Daniel Horn, Jeremy Sugerman, Kayvon Fatahalian, Mike Houston, Pat Hanrahan
CTM - ATI

- See Mark Segal’s talk to follow
- Web information
CUDA - NVIDIA

- See Ian Buck’s talk to follow
- Web information
Peakstream

- C/C++ library based
- Extended operators
- See Peakstream’s talk later this afternoon
- http://www.peakstreaminc.com
Introduction to RapidMind

• A software development platform for multi-core and stream processors, such as GPUs and the Cell Broadband Engine

• Embedded within ISO Standard C++
  - No new tools, compilers, preprocessors, etc.

• Portable core
  - Exposes platform specific functionality to also allow tuning for specific platforms

• Integrates with existing programming models
Program Definition

Program p;

p = BEGIN {
    In<Value3f> a, b;
    Out<Value3f> c;
    IF (all(a > 0.0f)) {
        Value3f d = f(a, b);
        c = d + a * 2.0f;
    } ELSE {
        c = d - a * 2.0f;
    } ENDIF;
} END;
SPMD Data Parallel Programming Model

- Parallel application:
  - Returns a new array: \( C = p(A, B) \)
  - Programs may have control flow
  - Programs may perform random reads from other arrays

- May operate on subarrays

- Collective operations:
  - Reduce: \( a = \text{reduce}(p, A) \)
  - Gather: \( A = B[U] \)
  - Scatter: \( A[U] = B \)
  - others...
Step 1: Replace Types

```cpp
#include <cmath>

float f;
float a[512][512][3];
float b[512][512][3];

float func(
    float r, float s
) {
    return (r + s) * f;
}

void func_arrays() {
    for (int x = 0; x<512; x++) {
        for (int y = 0; y<512; y++) {
            for (int k = 0; k<3; k++) {
                a[y][x][k] = func(a[y][x][k], b[y][x][k]);
            }
        }
    }
}
```

```cpp
#include <rapidmind/platform.hpp>

Value1f f;
Array<2,Value3f> a(512,512);
Array<2,Value3f> b(512,512);

Value3f func(
    Value3f r, Value3f s
) {
    return (r + s) * f;
}
```

GPGPU
Step 2: Capture Computation

```cpp
#include <cmath>

float f;
float a[512][512][3];
float b[512][512][3];

float func(
    float r, float s
) {
    return (r + s) * f;
}

void func_arrays() {
    for (int x = 0; x < 512; x++) {
        for (int y = 0; y < 512; y++) {
            for (int k = 0; k < 3; k++) {
                a[y][x][k] = func(a[y][x][k], b[y][x][k]);
            }
        }
    }
}
```

```cpp
#include <rapidmind/platform.hpp>

Value1f f;
Array<2,Value3f> a(512,512);
Array<2,Value3f> b(512,512);

Value3f func(
    Value3f r, Value3f s
) {
    return (r + s) * f;
}

void func_arrays() {
    Program func_prog = BEGIN {
        In<Value3f> r, s;
        Out<Value3f> q;
        q = func(r,s);
    } END;
    . . .
}
```
```cpp
#include <rapidmind/platform.hpp>

Value1f f;
Array<2,Value3f> a(512,512);
Array<2,Value3f> b(512,512);

Value3f func(
    Value3f r, Value3f s
) {
    return (r + s) * f;
}

void func_arrays() {
    Program func_prog = BEGIN {
        In<Value3f> r, s;
        Out<Value3f> q;
        q = func(r,s);
    } END;
    a = func_prog(a,b);
}
```

```cpp
#include <cmath>

float f;
float a[512][512][3];
float b[512][512][3];

float func(
    float r, float s
) {
    return (r + s) * f;
}

void func_arrays() {
    for (int x = 0; x<512; x++)
        for (int y = 0; y<512; y++) {
            for (int k = 0; k<3; k++) {
                a[y][x][k] = func(a[y][x][k],b[y][x][k]);
            }
        }
}
```

---

Step 3: Parallel Execution

```cpp
#include <rapidmind/platform.hpp>

Value1f f;
Array<2,Value3f> a(512,512);
Array<2,Value3f> b(512,512);

Value3f func(
    Value3f r, Value3f s
) {
    return (r + s) * f;
}

void func_arrays() {
    Program func_prog = BEGIN {
        In<Value3f> r, s;
        Out<Value3f> q;
        q = func(r,s);
    } END;
    a = func_prog(a,b);
}
```
# Usage Summary

- **Usage:**
  - Include platform header
  - Link to runtime library

- **Data:**
  - Value tuples
  - Arrays
  - *Remote data abstraction*

- **Programs:**
  - Defined dynamically
  - Execute on coprocessors
  - *Remote procedure abstraction*

```cpp
#include <rapidmind/platform.hpp>

Value1f f;
Array<2,Value3f> a(512,512);
Array<2,Value3f> b(512,512);

Value3f func(
    Value3f r, Value3f s
) {
    return (r + s) * f;
}

void func_arrays() {
    Program func_prog = BEGIN {
        In<Value3f> r, s;
        Out<Value3f> q;
        q = func(r,s);
    } END;
    a = func_prog(a,b);
}
```
Summary

- Complete standard library
- Full C++ integration
- Expresses general purpose computations

Application spaces:
- Financial modeling
- Image processing
- Oil and Gas
- Scientific Computation
- Content Creation

Example applications:
- FFT
- BLAS
- Black-Scholes
- Raytracing
- Crowd simulation
- Shape detection
- Sorting
- Coupled Map Lattice Simulation
RapidMind@SC06

• **Tutorial with IBM**
  - Programming Using RapidMind on the Cell BE
  - Monday, 8:30 AM - 12:00 PM, Room 17

• **Poster with HP**
  - Performance Evaluation of GPUs Using the RapidMind Development Platform
  - Poster Reception, Tuesday, 5:15 PM - 7:15 PM

• **Demo at the IBM booth ( #1205 )**

• [http://www.rapidmind.net](http://www.rapidmind.net)
Acknowledgements

- Ian Buck - based off of his previous talks
- RapidMind
  - Michael McCool
  - Stefanus Du Toit
- Stanford
  - The entire BrookGPU team
- Microsoft
  - David Tarditi