



SIGGRAPH2007

High Level Languages for GPUs Overview

Mike Houston
Stanford University



High Level Shading Languages

- “Old School”
 - Use shading language along with OpenGL/DirectX
- Cg, HLSL, & OpenGL Shading Language
 - Cg:
 - <http://www.nvidia.com/cg>
 - HLSL:
 - http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/directx/graphics/reference/highlevellanguageshaders.asp
 - OpenGL Shading Language:
 - <http://www.3dlabs.com/support/developer/ogl2/whitepapers/index.html>



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



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://graphics.stanford.edu/projects/brookgpu>
- **CTM**
ATI/AMD
<http://ati.amd.com/companyinfo/researcher/documents.html>
- **CUDA**
NVIDIA
<http://www.nvidia.com/object/cuda.html>
- **RapidMind**
Commercial follow-on to Sh
<http://www.rapidmind.net>

Microsoft Research Accelerator Project



SIGGRAPH2007

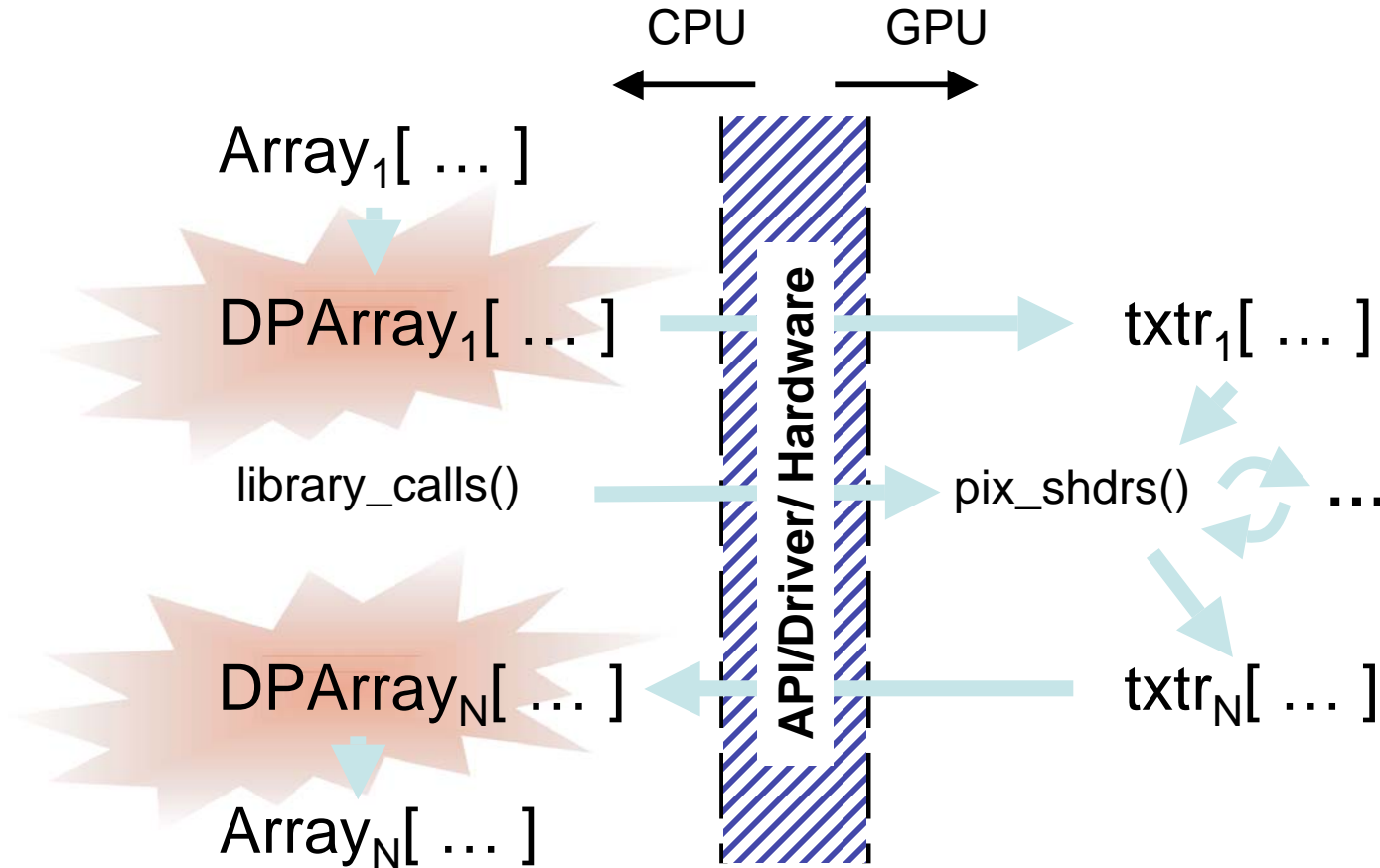
- GPGPU programming using data parallelism
- Presents a data-parallel library to the programmer.
 - Simple, high-level set of operations
- Library just-in-time compiles to GPU pixel shaders or CPU code.
 - Runs on top of .NET



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

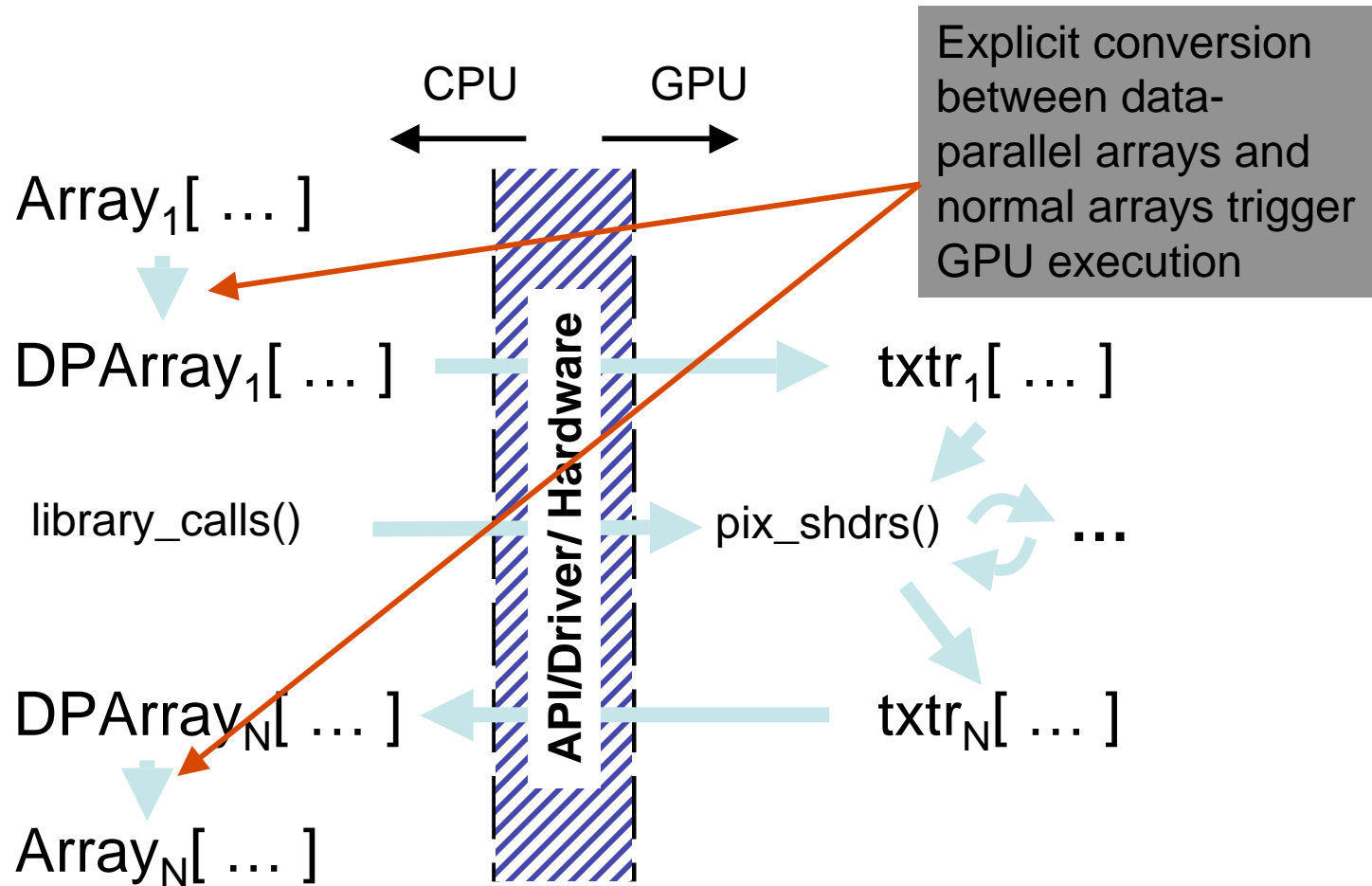
Data-parallel array types



Explicit conversion



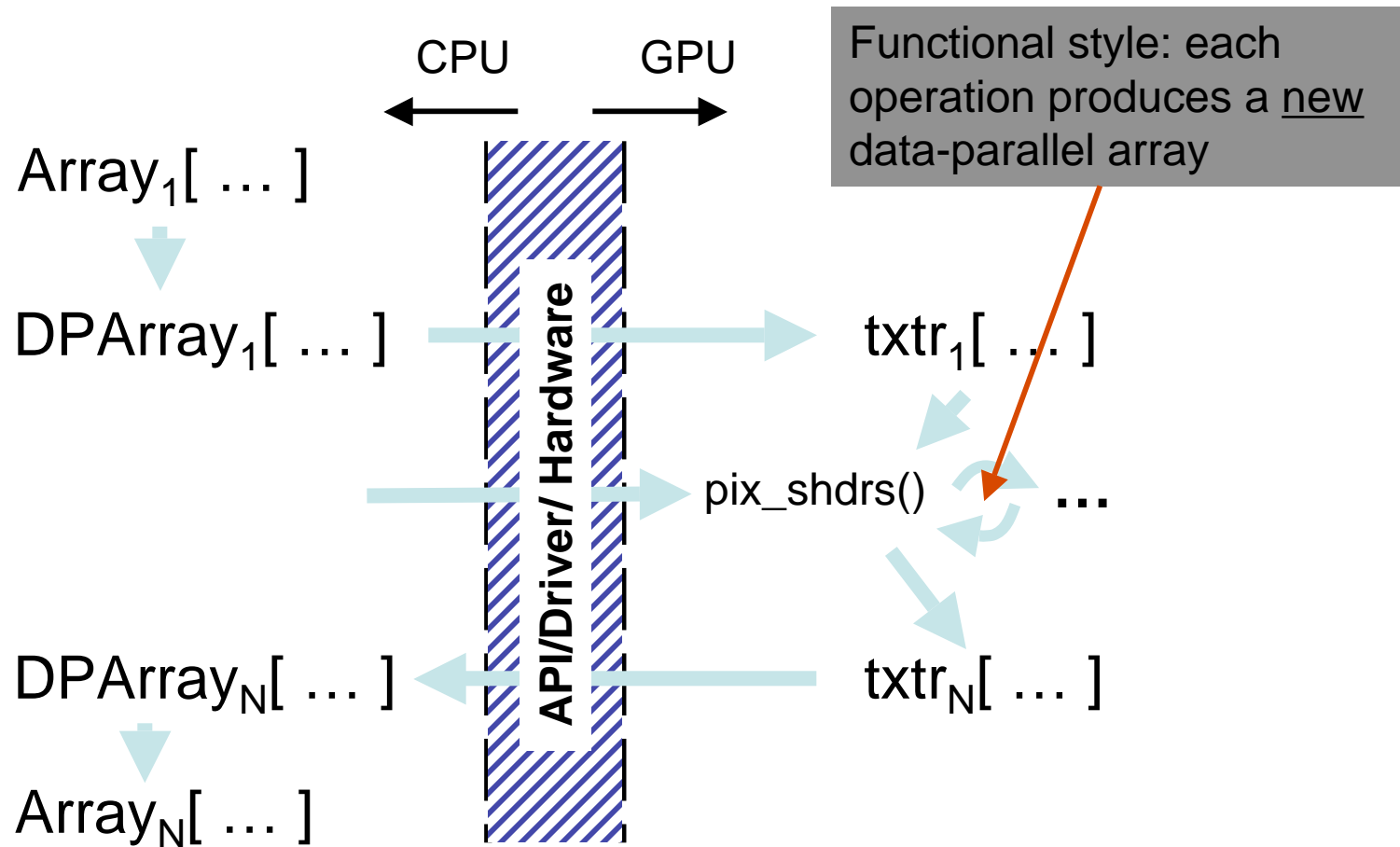
SIGGRAPH2007



Functional style



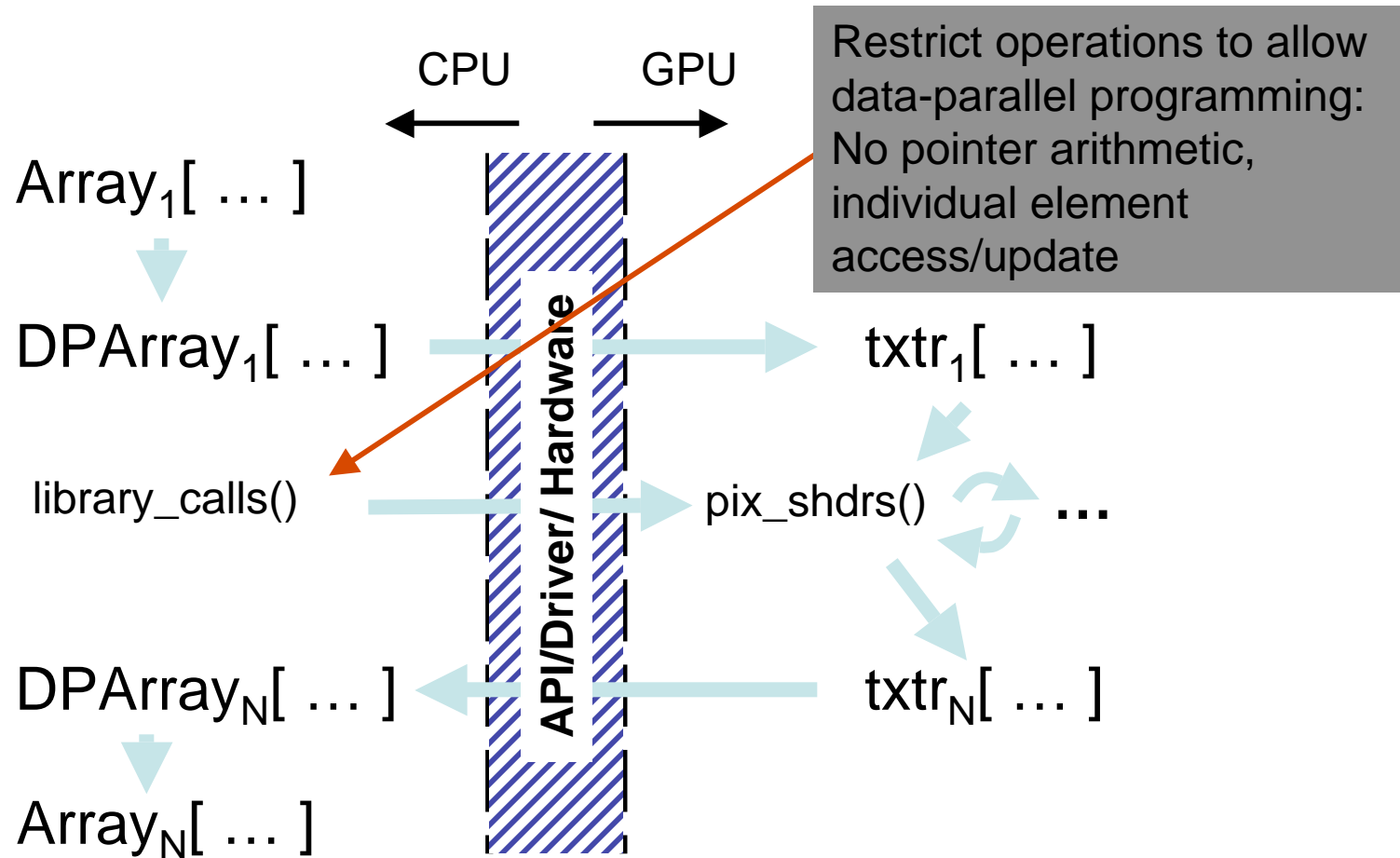
SIGGRAPH2007



Types of operations



SIGGRAPH2007





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.



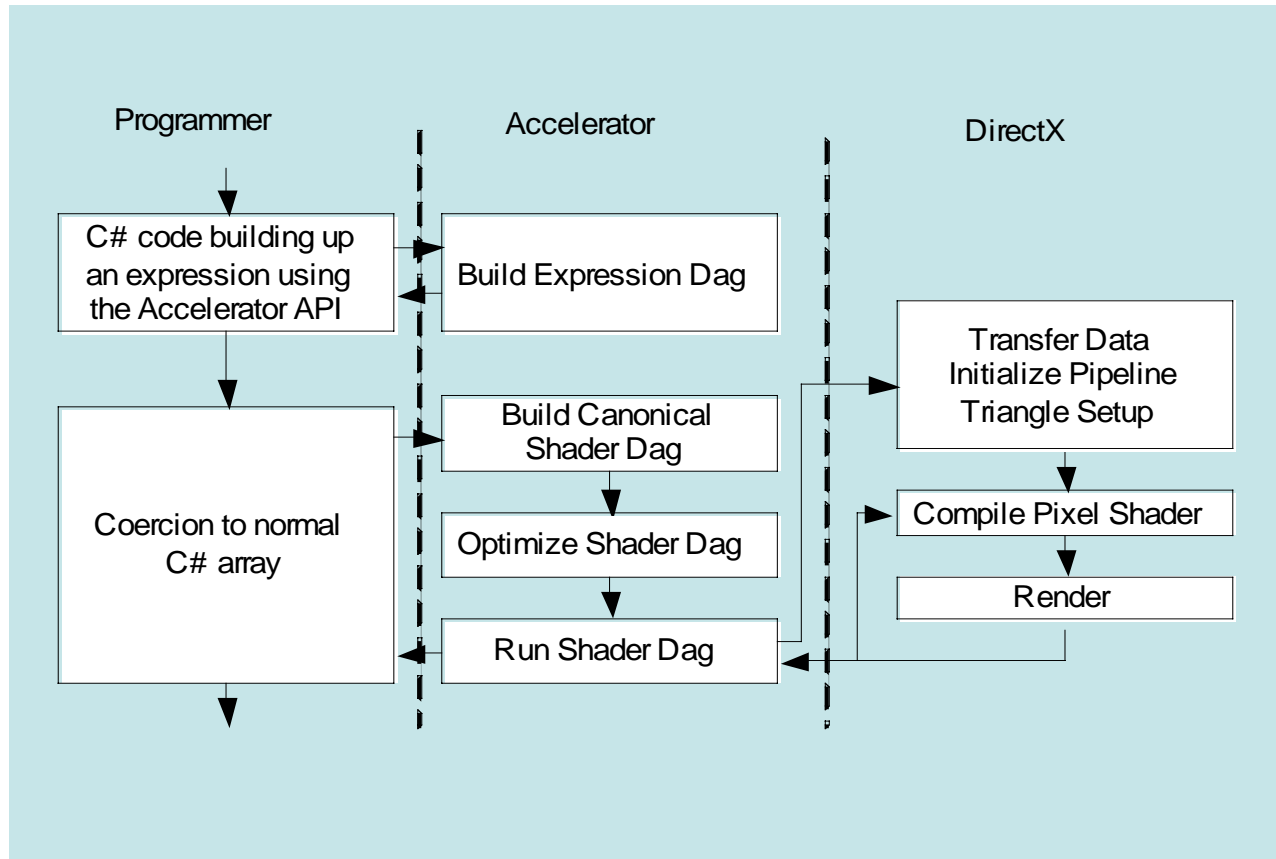
Example: 2-D convolution

```
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



SIGGRAPH2007



Availability and more information



SIGGRAPH2007

- Binary version of Accelerator available for download
 - <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>

Brook: General Purpose Streaming Language



SIGGRAPH2007

- Stream programming model
 - GPU = streaming coprocessor
- C with stream extensions
- Cross platform
 - ATI & NVIDIA
 - OpenGL, DirectX, CTM
 - Windows & Linux

GP GPU





Streams

- Collection of records requiring similar computation

- particle positions, voxels, FEM cell, ...

```
Ray r<200>;
```

```
float3 velocityfield<100,100,100>;
```

- Similar to arrays, but...

- index operations disallowed: `position[i]`

- read/write stream operators

```
streamRead (r, r_ptr);
```

```
streamWrite (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

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



Kernels

- Kernel arguments
 - input/output streams
 - gather streams

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



Kernels

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

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



Kernels

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

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



Reductions

- Compute single value from a stream
 - associative operations only

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

```
float a<100>;
float r;
```

```
sum(a,r);
```

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



Reductions

- Multi-dimension reductions
 - stream "shape" differences resolved by reduce function

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

```
float a<20>;
float r<5>;
```



```
sum(a,r);
```

```
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])
```



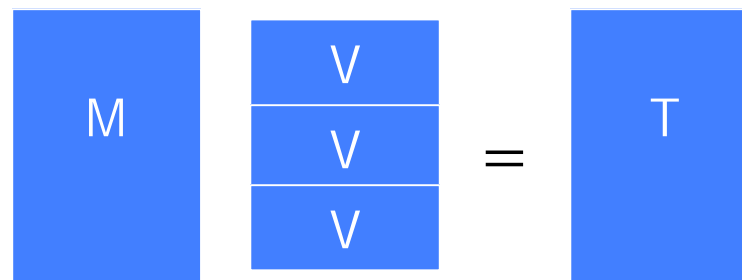

Matrix Vector Multiply

```
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);
```





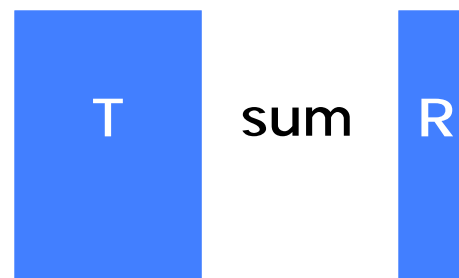
Matrix Vector Multiply

```
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);
```



- **Accessing stream data for graphics apps**
 - Brook runtime api available in C++ code
 - autogenerated .hpp files for brook code

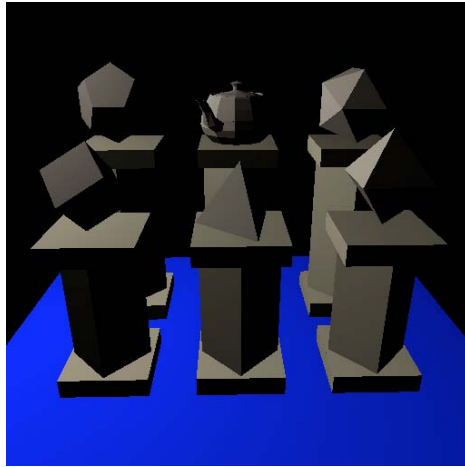
```
brook::initialize( "dx9", (void*)device );

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

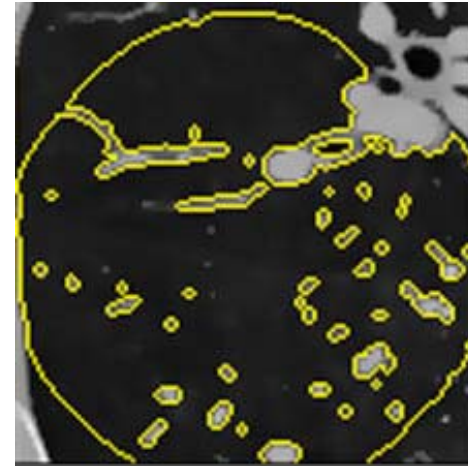
// Get a handle to the texture being used by
// the normal stream as a backing store
normalTexture = (IDirect3DTexture9*)
    normalStream->getIndexedFieldRenderData(0);

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

Applications



ray-tracer

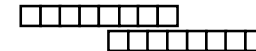


segmentation

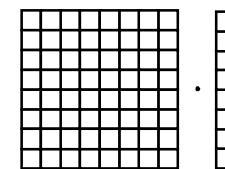


fft edge detect

SAXPY



SGEMV



linear algebra



Brook for GPUs

- Release v0.3 available on Sourceforge
 - CVS tree *much* more up to date
- Project Page
 - <http://graphics.stanford.edu/projects/brook>
- Source
 - <http://www.sourceforge.net/projects/brook>
- Paper:
 - Brook for GPUs: Stream Computing on Graphics Hardware
 - Ian Buck, Tim Foley, Daniel Horn, Jeremy Sugerman, Kayvon Fatahalian, Mike Houston, Pat Hanrahan



CTM - AMD



SIGGRAPH2007

- See Justin Hensley's talk to follow
- Web information
 - <http://ati.amd.com/companyinfo/researcher/documents.html>
 - http://ati.amd.com/companyinfo/researcher/documents/ATI_CTM_Guide.pdf



CUDA - NVIDIA

- See Mark Harris's talk to follow
- Web information
 - <http://www.nvidia.com/object/cuda.html>

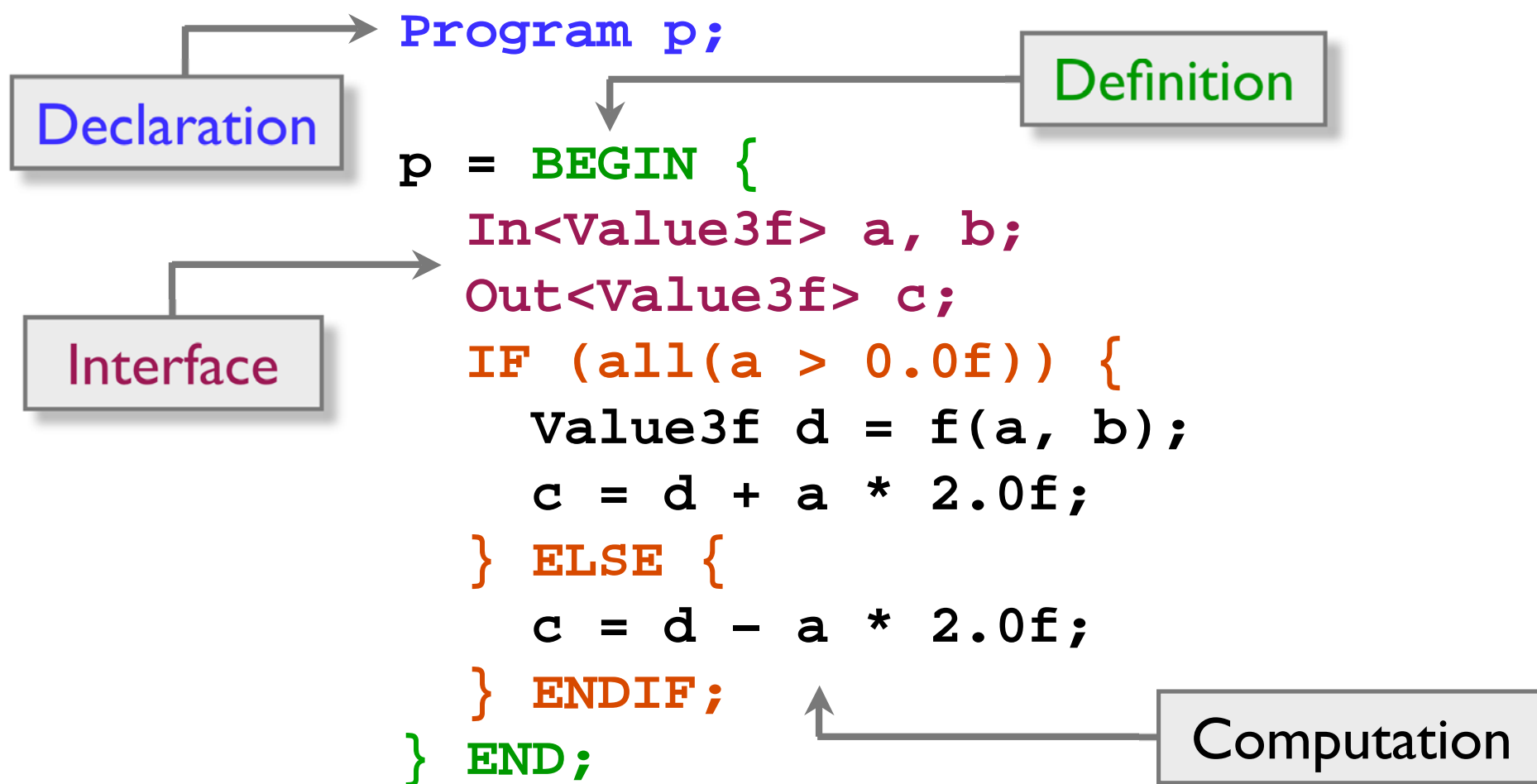


Introduction to RapidMind

- <http://www.rapidmind.net>
- 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



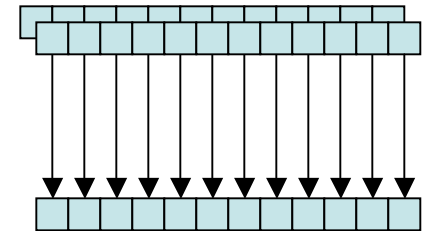
SPMD Data Parallel Programming Model



SIGGRAPH2007

- **Parallel application:**

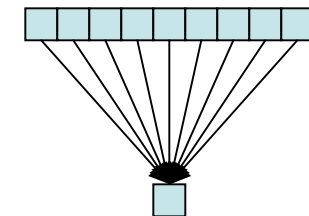
- Returns a new array: $C = p(A, B)$
- *Programs may have control flow*
- *Programs may perform random reads from other arrays*



- **May operate on subarrays**

- **Collective operations:**

- Reduce: $a = \text{reduce}(p, A)$
- Gather: $A = B[U]$
- Scatter: $A[U] = B;$
- *others...*



Reduce

Step 1: Replace Types



SIGGRAPH2007

```
#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]);
            }
        }
    }
}
```

```
#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;
}
```

Step 2: Capture Computation



SIGGRAPH2007

```
#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]);
            }
        }
    }
}
```

```
#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;
    . . .
}
```



Step 3: Parallel Execution



SIGGRAPH2007

```
#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]);
            }
        }
}
```

```
#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



SIGGRAPH2007

- Usage:
 - Include platform header
 - Link to runtime library
- Data:
 - Value tuples
 - Arrays
 - *Remote data abstraction*
- Programs:
 - Defined dynamically
 - Execute on coprocessors
 - *Remote procedure abstraction*

```
#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



SIGGRAPH2007

- Complete standard library
- Full C++ integration
- Expresses general purpose computations
- Multiple platforms
 - Multi-core
 - Cell
 - GPUs
- 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



Acknowledgements

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