High level languages

Ian Buck

Graphics Lab
Stanford University

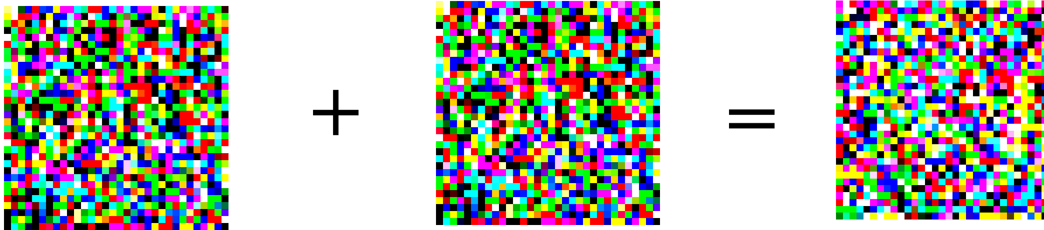
Without High Level Languages

- Adding Two Vectors

A[1024] + B[1024]

- Load A into texture0
- Load B into texture1
- Load "ADD" shader
- Draw 32x32 quad
- Read back results

```
char addshader[] =
"!!ARBfp1.0\n"
"TEMP R0;\n"
"TEMP R1;\n"
"TEX R0, fragment.texcoord[0],"
         texture[0], RECT;\n"
"TEX R1, fragment.texcoord[0],"
         texture[1], RECT;\n"
"ADD result.color, R0, R1;\n"
"END\n";
```



Cg/HLSL Helps

```
void sum(float2 Pos : TEXCOORD0,
          uniform sampler texA : register(s0),
          uniform sampler texB : register(s1),
           out float output : COLOR0) {
  float a = texRECT(texA, Pos);
  float b = texRECT(texB, Pos);
  output = a+b;
}
```

- But…
  - Still graphics centric (texture, colors, etc.)

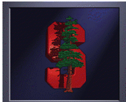
Graphics Glue…

- Creating Floating Point Render targets…
- Floating Point Textures…
- Issue Geometry…
- Reading Pixel data back to the card…

Ugh…

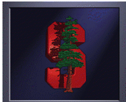
GPGPU Languages

- Why do 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
- Sh
  - University of Waterloo
  - http://libsh.sourceforge.net
  - http://www.cgl.uwaterloo.ca
- Brook
  - Stanford University
  - http://brook.sourceforge.net
  - http://graphics.stanford.edu/projects/brookgpu



Sh Features

- Implemented as C++ library
  - Use C++ modularity, type, and scope constructs
  - Use C++ to metaprogram shaders and kernels
  - Use C++ to sequence stream operations
- Operations can run on
  - GPU in JIT compiled mode
  - CPU in immediate mode
  - CPU in JIT compiled mode
- Can be used
  - To define shaders
  - To define stream kernels
- No glue code
  - To set up a parameter, just declare it and use it
  - To set up a texture, just declare it and use it
- Memory management
  - Automatically uses pbuffers and/or uberbuffers
  - Arrays simulated with textures Textures can encapsulate interpretation code
  - Programs can encapsulate texture data
- Program manipulation
  - Introspection
  - Uniform/varying conversion
  - Program specialization
  - Composition & concatenation
  - Interface adaptation
- Free and Open Source
- http://libsh.sourceforge.net



Sh Fragment Shader

```
fsh = SH_BEGIN_PROGRAM("gpu:fragment") {
  ShInputNormal3f nv;          // normal (VCS)
  ShInputVector3f lv;          // light-vector (VCS)
  ShInputVector3f vv;          // view vector (VCS)
  ShInputColor3f ec;           // irradiance
  ShInputTexCoord2f u;         // texture coordinate

  ShOutputColor3f fc;          // fragment color

  vv = normalize(vv);
  lv = normalize(lv);
  nv = normalize(nv);
  ShVector3f hv = normalize(lv + vv);
  fc = kd(u) * ec;
  fc += ks(u) * pow(pos(hv|nv), spec_exp);
} SH_END;
```

Streams and Channels

- `ShChannel<element_type>`
  - Sequence of elements of given type
- `ShStream`
  - Sequence of channels
  - Combine channels with &:

    ```
    ShStream s = a & b & c;
    ```
  - *Refers* to channels, does *not* copy
  - Single channel also a stream
- Apply programs to streams with `<<`

```
ShStream t = (x & y & z);
s = p << t;
(a & b & c) = p << (x & y & z);
```

Stream Processing: Particles

// SETUP (define particle state update kernel)

```
p = SH_BEGIN_PROGRAM("gpu:stream") {
  ShInOutPoint3f Ph, Pt;
  ShInOutVector3f V;
  ShInputVector3f A;
  ShInputAttrib1f delta;
  Pt = Ph;
  A = cond(abs(Ph(1)) < 0.05,
    ShVector3f(0.,0.,0.), A);
  V += A * delta;
  V = cond((V|V) < 1.,
    ShVector3f(0., 0., 0.), V);
  Ph += (V + 0.5*A)*delta;
  ShAttrib1f mu(0.1), eps(0.3);
  for (i = 0; i < num_spheres; i++) {
    ShPoint3f C = spheres[i].center;
    ShAttrib1f r = spheres[i].radius;
    ShVector3f PhC = Ph - C;
    ShVector3f N = normalize(PhC);
    ShPoint3f S = C + N*r;
    ShAttrib1f collide =
      ((PhC|PhC) < r*r)*((V|N) < 0);
    Ph = cond(collide,
      Ph - 2.0*((Ph - S)|N)*N, Ph);
    ShVector3f Vn = (V|N)*N;
    ShVector3f Vt = V - Vn;
    V = cond(collide,
      (1.0 - mu)*Vt - eps*Vn, V);
  }
  ShAttrib1f under = Ph(1) < 0.;
  Ph = cond(under,
    Ph * ShAttrib3f(1.,0.,1.), Ph);
  ShVector3f Vn =
    V * ShAttrib3f(0.,1.,0.);
  ShVector3f Vt = V - Vn;
  V = cond(under,
    (1.0 - mu)*Vt - eps*Vn, V);
  Ph(1) = cond(min(under,(V|V)<0.1),
    ShPoint1f(0.), Ph(1));
  ShVector3f dt = Pt - Ph;
  Pt = cond((dt|dt) < 0.02, Pt +
    ShVector3f(0.0, 0.02, 0.0), Pt);
} SH_END;
```

// define state stream

```
ShStream state =
  (pos & pos_tail & vel);
```

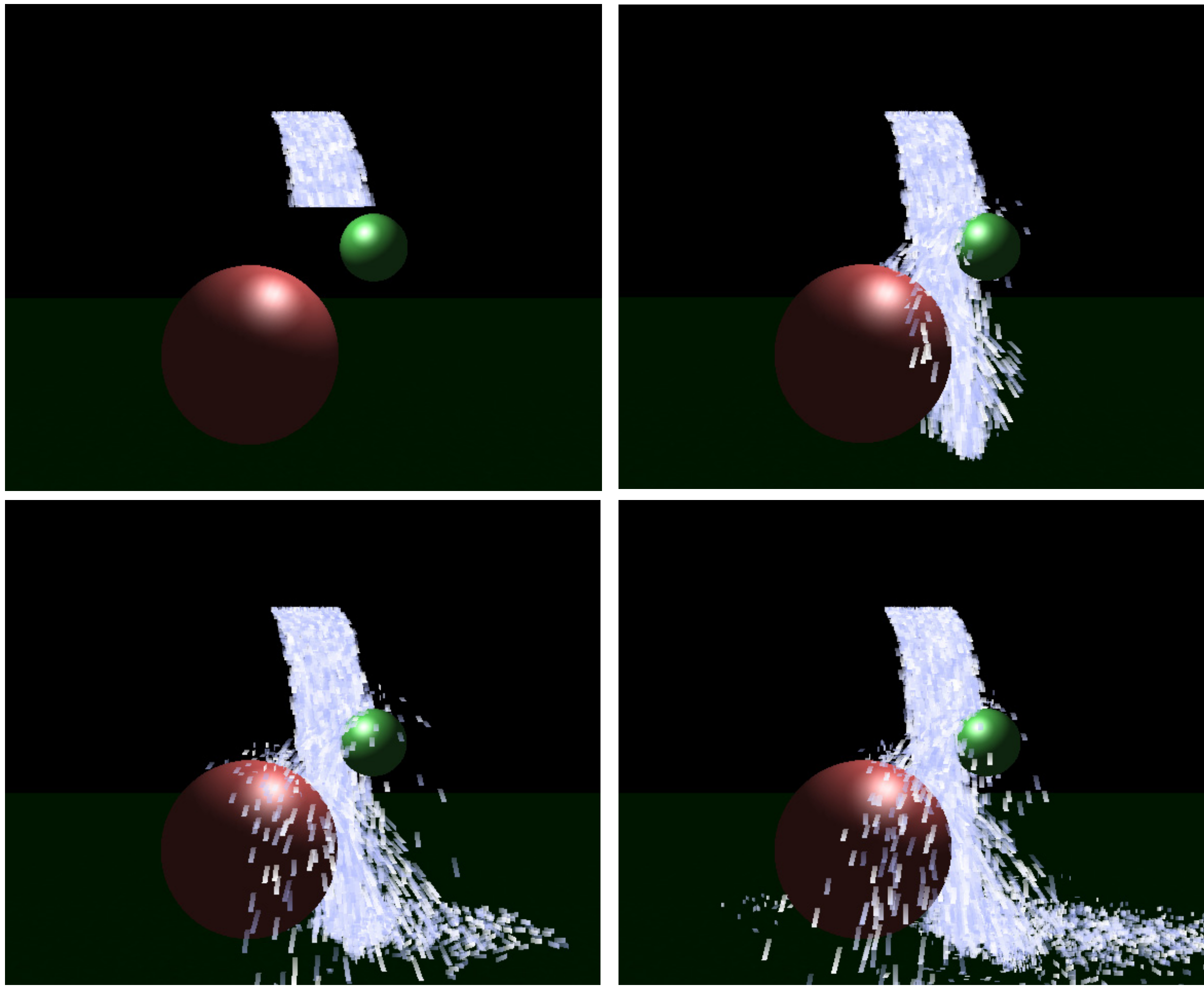
// curry p with state and parameters

```
ShProgram update =
  p << state << gravity << delta;

...
```

// IN INNER LOOP
// execute state update (input to update is compiled in)

```
state = update;
```

Stream Processing: Particles



Scout: A Hardware-Accelerated System for Quantitatively Driven Visualization and Analysis

Patrick S. McCormick, Jeff Inman, James P. Ahrens

Los Alamos National Laboratory

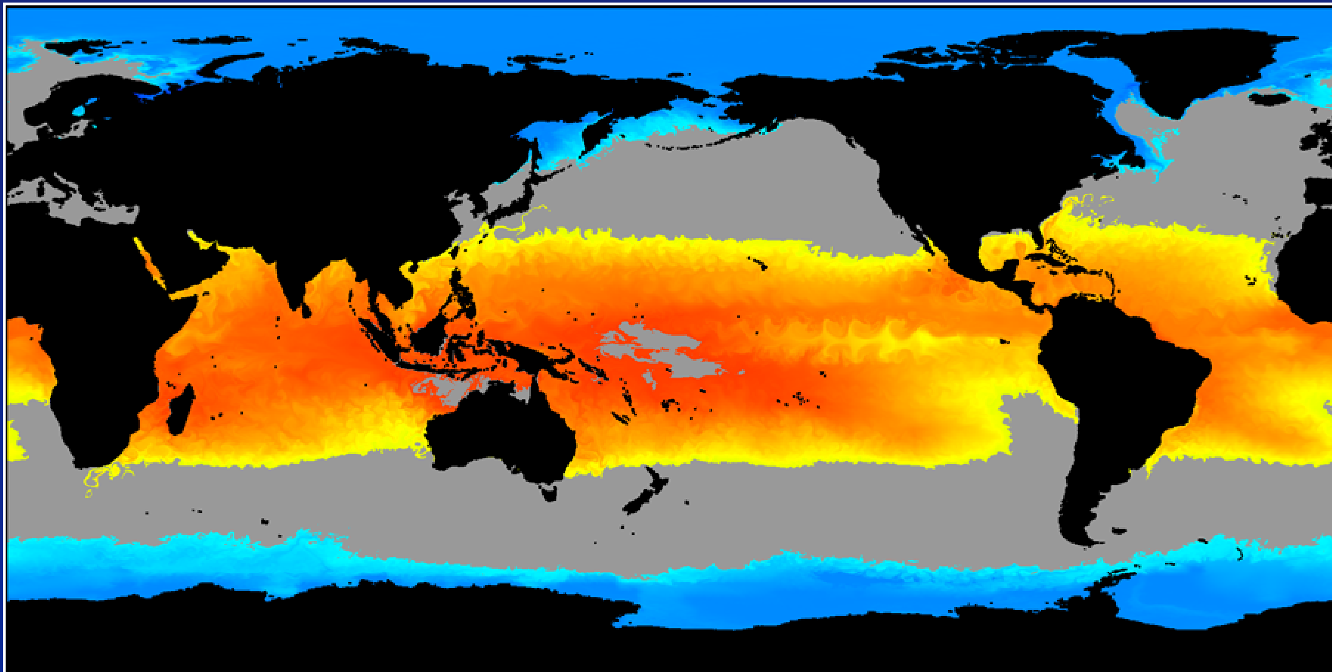
Charles Hansen and Greg Roth

University of Utah

LA-UR-04-7045

Example

```
where (land == 1)
  image = 0;  // render land as black
else where (pt < 2.375 || (pt >= 21 && pt < 29.5))
  image = hsva(240 - 240 * norm(pt), 1, 1, 1);
else
  image = 0.6; // outside range colored  gray
```



User Interface

- Data sets automatically in scope
- "On the fly" compilation
- Time series support

```
#DBG_MACHINE_CODE
float here = mlite[i][j][k];
float e_diff = here - mlite[i -1][j][k];
float s_diff = here - mlite[i][j -1][k];
float d_diff = here - mlite[i][j][k -1];
float4 grad = vector(e_diff, s_diff, d_diff, 0);
float gradmag = pow(dot3(grad, grad), 0.5);
grad = grad / gradmag;
float hlo = 0;
float hhi = 0.5;
// LIGHTING
float4 lightpos = vector(0.57735,-0.57735, 0.57735, 0);
float lit = max(dot3(grad, lightpos),0) * 0.8 + 0.5;
if (k > 50 && (here > hlo && here < hhi)
    && (gradmag > 0.15 && gradmag < 0.25))
  image = hsva(240 -((here -hlo)/(hhi -hlo))*240, lit, lit, 0.1);
else
  image = 0;
```

Brook

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

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;
}
```

Kernels

- Ray triangle intersection

```
kernel void krnIntersectTriangle(Ray ray<>, Triangle tris[],
                                 RayState oldraystate<>,
                                 GridTrilist trilist[],
                                 out Hit candidatehit<>) {
  float idx, det, inv_det;
  float3 edge1, edge2, pvec, tvec, qvec;
  if(oldraystate.state.y > 0) {
    idx = trilist[oldraystate.state.w].trinum;
    edge1 = tris[idx].v1 - tris[idx].v0;
    edge2 = tris[idx].v2 - tris[idx].v0;
    pvec = cross(ray.d, edge2);
    det = dot(edge1, pvec);
    inv_det = 1.0f/det;
    tvec = ray.o - tris[idx].v0;
    candidatehit.data.y = dot( tvec, pvec ) * inv_det;
    qvec = cross( tvec, edge1 );
    candidatehit.data.z = dot( ray.d, qvec ) * inv_det;
    candidatehit.data.x = dot( edge2, qvec ) * inv_det;
    candidatehit.data.w = idx;
  } else {
    candidatehit.data = float4(0,0,0,-1);
  }
}
```

Reductions

- Compute single value from a stream
  - associative operations only

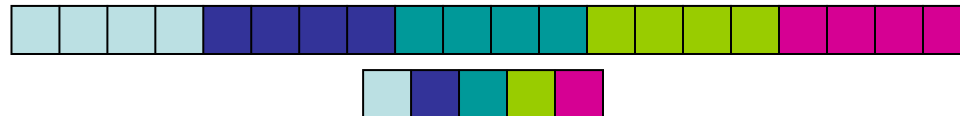
```
reduce void sum (float a<>,
                 reduce float r<>)
  r += a;
}


float a<100>;
float r;

sum(a,r);
```

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

Reductions

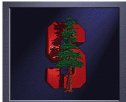
- Multi-dimension reductions
  - stream "shape" differences resolved by reduce function

```
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

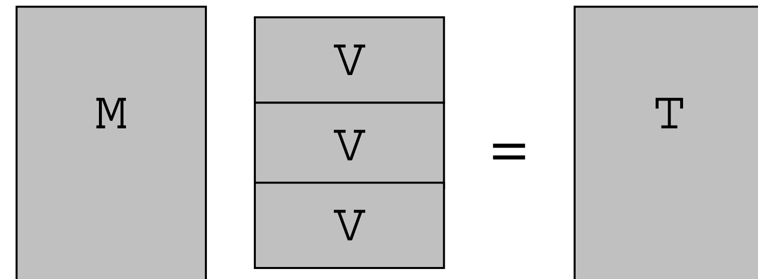
- Kernel arguments of different shape
  - resolved by repeat and stride

```
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])
```

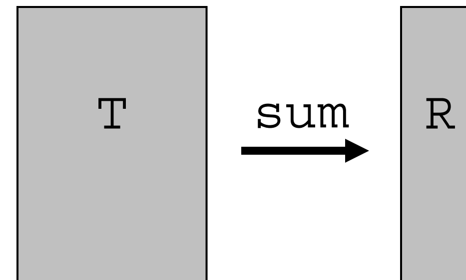
Matrix Vector Multiply

```
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);
```

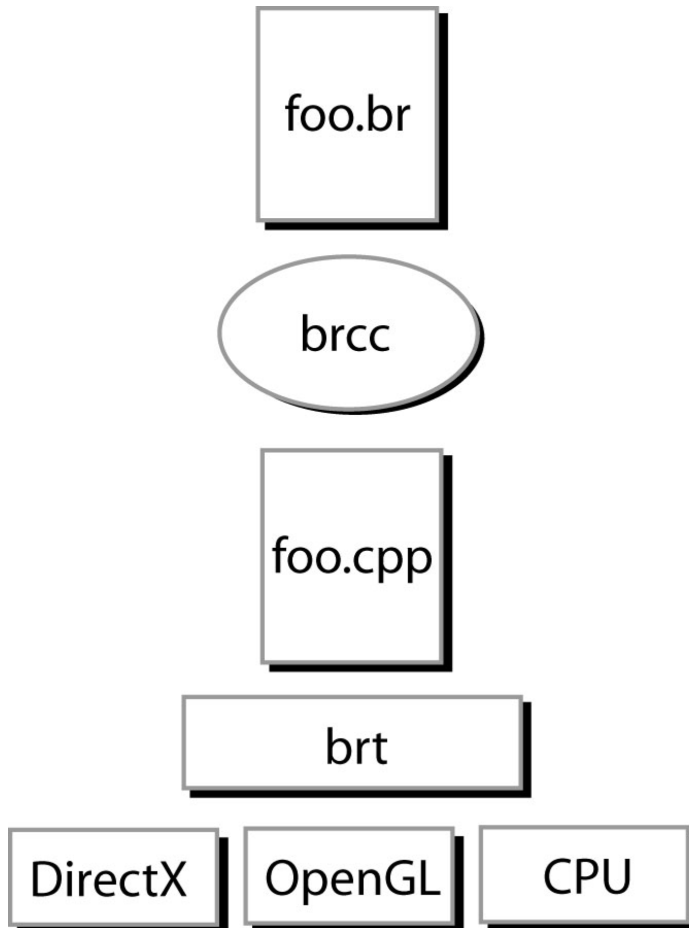


Matrix Vector Multiply

```
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);
```



System Outline



brcc

Source to source compiler

- Generate CG & HLSL code
- CGC and FXC for shader assembly
- Virtualization

brt

Brook run-time library

- Stream texture management
- Kernel shader execution

Running Brook

- Compiling .br files

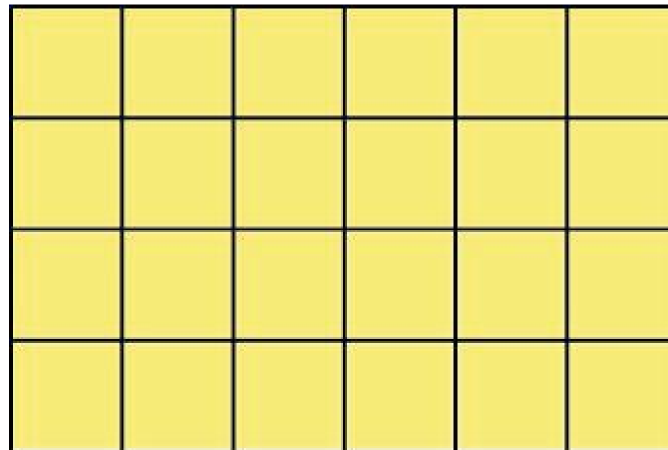
```
Brook CG Compiler
Version: 0.2  Built: Apr 24 2004, 18:11:59
brcc [-hvndktyAN] [-o prefix] [-w workspace] [-p shader ] foo.br
        -h              help (print this message)
        -v              verbose (print intermediate generated code)
        -n              no codegen (just parse and reemit the input)
        -d              debug (print cTool internal state)
        -k              keep generated fragment program (in foo.cg)
        -t              disable kernel call type checking
        -y              emit code for ATI 4-output hardware
        -A              enable address virtualization (experimental)
        -N              deny support for kernels calling other kernels
        -o prefix       prefix prepended to all output files
        -w workspace    workspace size (16 - 2048, default 1024)
        -p shader       cpu / ps20 / fp30 / cpumt (can specify multiple)
        -f compiler     favor a particular compiler (cgc / fxc / default)
```

Eliminating GPU Limitations

Treating texture as memory

- Limited texture size and dimension
- Compiler inserts address translation code

```
float matrix<8096,10,30,5>;
```



Eliminating GPU Limitations

Extending kernel outputs

- duplicate kernels, let **cgc** or **fxc** do dead code elimination
- better solution:

"Efficient Partitioning of Fragment Shaders for Multiple-Output Hardware"
Tim Foley, Mike Houston, and Pat Hanrahan

"Mio: Fast Multipass Partitioning via Priority-Based Instruction Scheduling"
Andrew T. Riffel, Aaron E. Lefohn, Kiril Vidimce, Mark Leone, and John D. Owens

Running Brook

- BRT_RUNTIME selects platform
  - CPU Backend:

    BRT_RUNTIME = cpu
  - OpenGL ARB Backend:

    BRT_RUNTIME = arb
  - DirectX9 Backend:

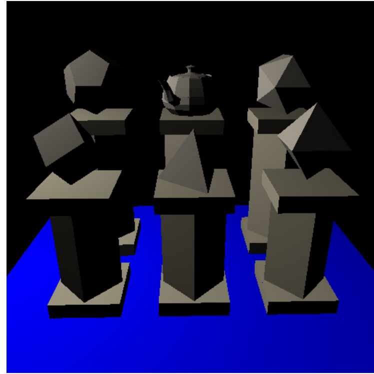
    BRT_RUNTIME = dx9

Runtime

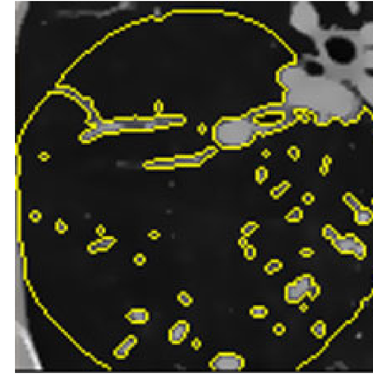
- Accessing stream data for graphics aps
  - Brook runtime api available in C++ code
  - autogenerated .hpp files for brook code

```
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

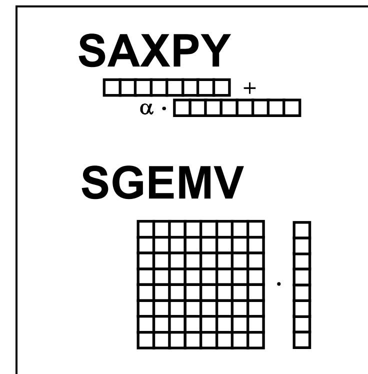


Ray-tracer



Segmentation

FFT edge detect



Linear algebra

Performance

ATI Radeon X800 XT
NVIDIA GeForce 6800
Pentium 4 3.0 GHz

Relative Performance

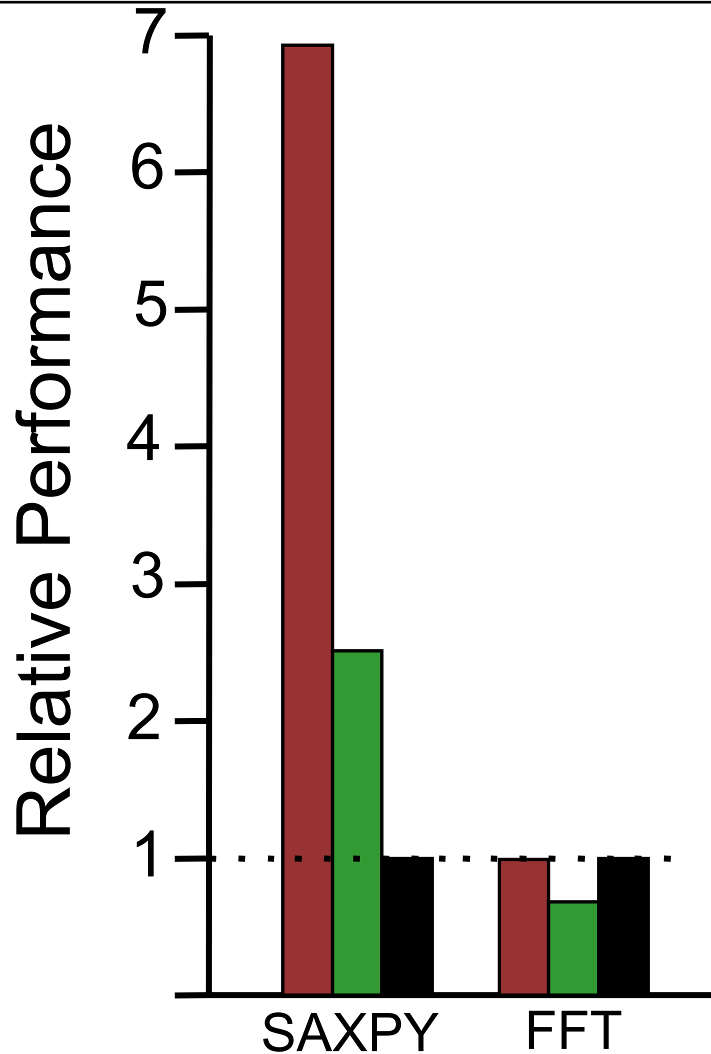
7
6
5
4
3
2
1

SAXPY Segment SGEMV FFT Ray-tracer

Compared against:

- Intel Math Library
- Atlas Math Library
- cached blocked segmentation
- FFTW
- Wald ['04] SSE Ray-Triangle

Understanding Performance



GPU wins when…

- limited data reuse
  - ✓ SAXPY
  - ✗ FFT

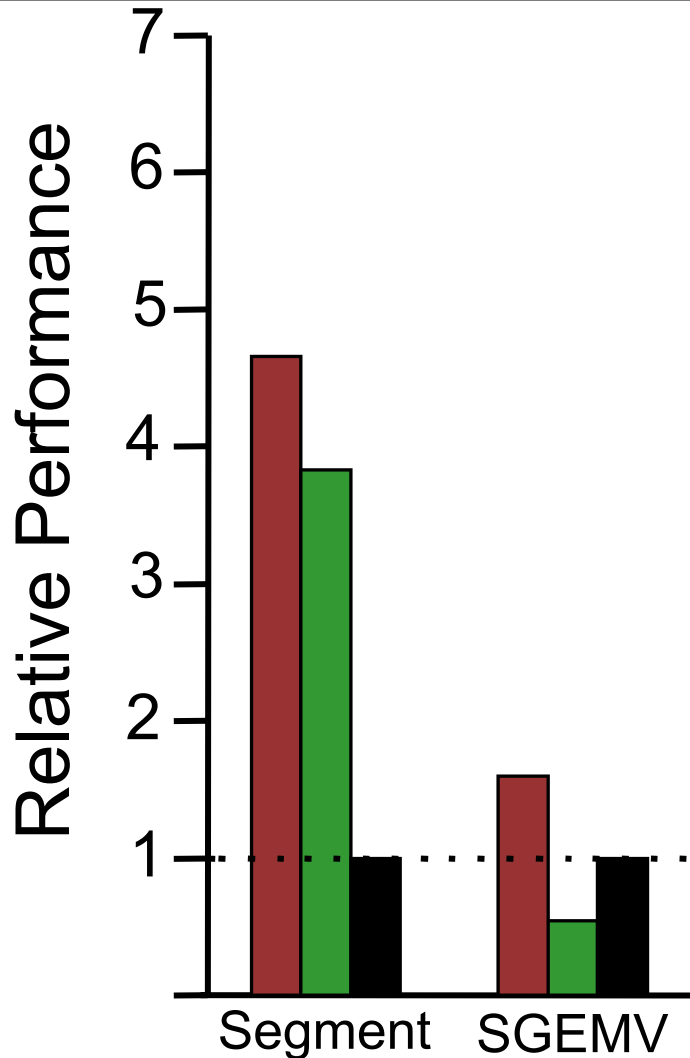
Pentium 4 3.0 GHz

44 GB/sec peak cache bandwidth

NVIDIA GeForce 6800 Ultra

36 GB/sec peak memory bandwidth

Understanding Performance

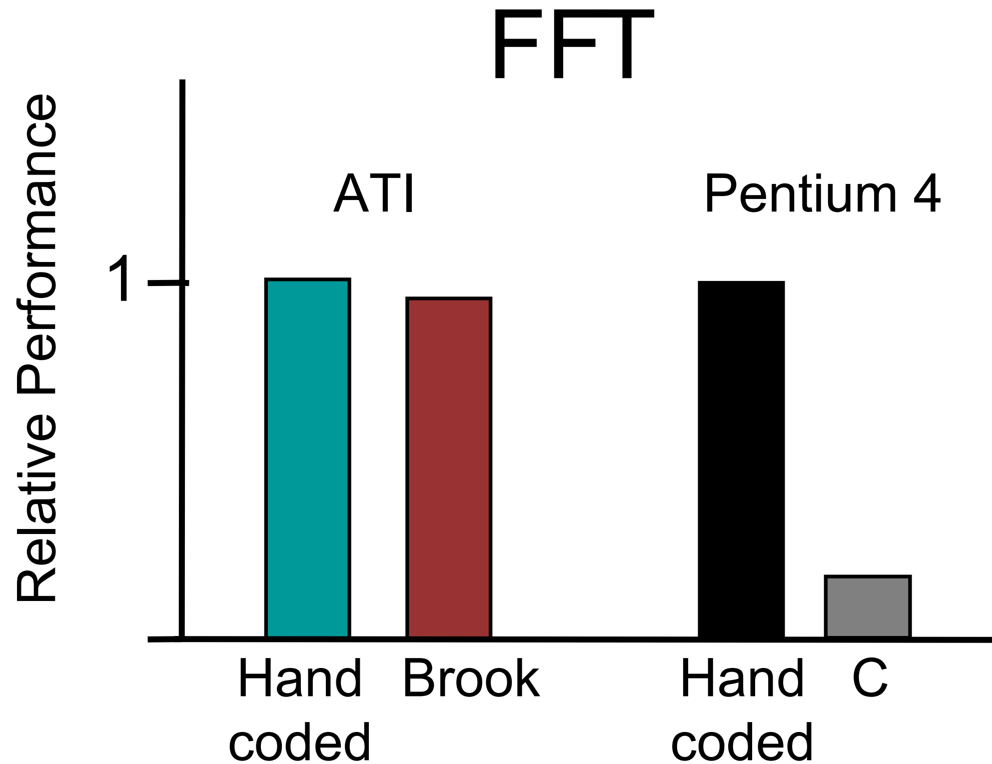


GPU wins when…

- Arithmetic intensity
  - ✓ Segment
    - 3.7 ops per word
  - ✗ SGEMV
    - 1/3 ops per word

Efficiency

Brook version within 80% of hand-coded GPU version



Brook for GPUs

- Release v0.3 available on Sourceforge
- Project Page
  - http://graphics.stanford.edu/projects/brook
- Source
  - http://www.sourceforge.net/projects/brook
- Over 6K downloads!

Brook for GPUs: Stream Computing on Graphics Hardware
SIGGRAPH 2004

Ian Buck, Tim Foley, Daniel Horn, Jeremy Sugerman, Kayvon Fatahalian, Mike Houston, Pat Hanrahan

Fly-fishing fly images from [The English Fly Fishing Shop]