Brook for GPUs

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

Merrimac April 6th, 2004
GPU: stream processor for graphics

Pentium 4 SSE theoretical*
3GHz * 4 wide * .5 inst / cycle = 6 GFLOPS

GeForce FX 5900 (NV35) fragment shader observed:
MULR R0, R0, R0: 20 GFLOPS
equivalent to a 10 GHz P4
and getting faster: 3x improvement over NV30 (6 months)

*from Intel P4 Optimization Manual
Brook for gpus

- demonstrate gpu streaming coprocessor
  - explicit programming abstraction
Brook for gpus

- demonstrate gpu streaming coprocessor
  - make programming gpus easier
    - hide texture/pbuffer data management
    - hide graphics based constructs in CG/HLSL
    - hide rendering passes
    - virtualize resources
Brook for gpus

- demonstrate gpu streaming coprocessor
  - make programming gpus easier
    - hide texture/pbuffer data management
    - hide graphics based constructs in CG/HLSL
    - hide rendering passes
    - virtualize resources

- **performance!**
  - ... on applications that matter
Brook for gpus

- demonstrate gpu streaming coprocessor
  - make programming gpus easier
    - hide texture/pbuffer data management
    - hide graphics based constructs in CG/HLSL
    - hide rendering passes
    - virtualize resources
  - performance!
    - ... on applications that matter
  - highlight gpu areas for improvement
    - features required general purpose stream computing
system outline

- .br
  - Brook source files
- brcc
  - source to source compiler
- brt
  - Brook run-time library
Brook language

streams

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

float3 positions<200>;
float3 velocityfield<100,100,100>;
Brook language

kernels

- kernels
  - functions applied to streams
    - similar to for_all construct

```c
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];
```
Brook language

kernels

• kernels arguments
  – input/output streams
  – constant parameters
  – gather streams

```c
kernel void foo (float a<>, float b<>,
    float t, float array[],
    out float result<>) {
    result = array[a] + t*b;  
}

float a<100>;
float b<100>;
float c<100>;
float array<25>

foo(a,b,3.2f,array,c);
```

gpu

bonus
Brook language

kernels

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

```c
kernel void foo (float a<>, float b<>,
    float t, float array[],
    iter float n<>,
    out float result<>) {
    result = array[a] + t*b + n;
}
```

```c
float a<100>;
float b<100>;
float c<100>;
float array<25>
iter float n<100> = iter(0, 10);
```

foo(a,b,3.2f,array,n,c);
Brook language kernels

- Ray Triangle Intersection

```c
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);
    }
}
```
Brook language

reductions

- reductions
  - compute single value from a stream

```
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];
```
Brook language

reductions

• multi-dimension reductions
  – stream “shape” differences resolved by reduce function
Brook language

reductions

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

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

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

sum(a,r);
```
**Brook language reductions**

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

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

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

sum(a,r);
```

```cpp
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];
```
Brook language
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];
```
Brook language

stream repeat & stride

- kernel arguments of different shape
  - resolved by repeat and stride
*Brook language*

**stream repeat & stride**

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

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

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

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

stream repeat & stride

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

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

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

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])
```

April 6th, 2004
Brook language

matrix vector multiply

```c
#include <brook.h>

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);
```

April 6th, 2004
Brook language

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);

April 6th, 2004
brcc compiler
infrastructure
brcc compiler

infrastructure

- based on ctool

- parser
  - build code tree
  - extend C grammar to accept Brook

- convert
  - tree transformations

- codegen
  - generate cg & hlsl code
  - call cgcc, fxc
  - generate stub function
brcc compiler

kernel compilation

```c
kernel void updatepos (float2 pos<>,
    float2 vel[100][100],
    float timestep,
    out float2 newpos<>) {
    newpos = pos + vel[pos]*timestep;
}
```

```c
float4 main (uniform float4 _workspace : register (c0),
    uniform sampler _tex_pos : register (s0),
    float2 _tex_pos_pos : TEXCOORD0,
    uniform sampler vel : register (s1),
    uniform float4 vel_scalebias : register (c1),
    uniform float timestep : register (c2)) : COLOR0 {
    float4 _OUT; float2 pos; float2 newpos;
    pos = tex2D(_tex_pos, _tex_pos_pos).xy;
    newpos = pos
        + tex2D(vel,(pos).xy*vel_scalebias.xy+vel_scalebias.zw).xy
        * timestep;
    _OUT.x = newpos.x; _OUT.y = newpos.y;
    _OUT.z = newpos.y; _OUT.w = newpos.y;
    return _OUT;
}
```
brcc compiler

kernel compilation

```c
static const char __updatepos_ps20[] = "ps_2_0 ..... 
static const char __updatepos_fp30[] = "!!fp30 ..... 

void updatepos (const __BRTStream& pos,
    const __BRTStream& vel,
    const float  timestep,
    const __BRTStream& newpos) {
    static const void *__updatepos_fp[] = {
        "fp30", __updatepos_fp30,
        "ps20", __updatepos_ps20,
        "cpu", (void *) __updatepos_cpu,
        "combine", 0,
        NULL, NULL }
    static __BRTKernel k(__updatepos_fp);
    k->PushStream(pos);
    k->PushGatherStream(vel);
    k->PushConstant(timestep);
    k->PushOutput(newpos);
    k->Map();
}
```
brcc runtime

streams
brt runtime

streams

- streams

  separate texture per stream

  - $\textit{vel}$: texture 1
  - $\textit{pos}$: texture 2
brt runtime

kernels

- kernel execution
  - set stream texture as render target
  - bind inputs to texture units
  - issue screen size quad
    - texture coords provide stream positions

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

reductions

- reduction execution
  - multipass execution
  - associativity required
applications

ray-tracer
fft
segmentation
linear algebra:
  – BLAS, LINPACK, LAPACK
Brook performance

2-3x faster than CPU implementation
arithmetic intensity

\[ T_{gpu}(i, l_r, l_w) = l_r/R + i/K_{gpu} + l_w/W \]

\[ T_{cpu}(i) = i/K_{cpu} \]

\[ \alpha \equiv i/l \]
research directions

• virtualize gpu resources
  - texture size and formats
    • packing streams to fit in 2D segmented memory space

```cpp
float matrix<8096, 10, 30, 5>;
```
research directions

• virtualize gpu resources
  – multiple outputs
    • simple: let cgcc or fxc do dead code elimination

```cpp
kernel void foo (float3 a<>,
                float3 b<>, ..., 
                out float3 x<>,
                out float3 y<>)
```

```cpp
kernel void foo1(float3 a<>, 
                float3 b<>, ..., 
                out float3 x<>)
```

```cpp
kernel void foo2(float3 a<>, 
                float3 b<>, ..., 
                out float3 y<>)
```

• better: compute intermediates separately
research directions

• virtualize gpu resources
  – limited instructions per kernel

• Improve RDS algorithm for kernels
  – Graphics Hardware 2004 paper (Tim Foley)
  – RDS for hardware with multiple outputs.
research directions

• Brook v0.2 support
  – stream operators
    • stencil, group, repeat, stride, ...
      – All can be implemented with gathers and iterators
    • domain, merge
      – Adding domain operator:
        mykernel (a.domain(...), b.domain(...))
  – ScatterOp and GatherOp
    • Difficult to do efficiently
    • Point rendering
research directions

• Vout
  – Emulate with scan, search, and gather
  – Quite expensive for significant input/output ratios
  – GPUs Need Hardware Vout
    • Graphics Hardware Paper (Daniel Horn)
    • Marching Cubes
    • Collision Detection
    • Subdivision Surfaces
research directions

• The SRF Effect
  – Without GPU SRF or cache, P4 can win on blocked algorithms
    • Matrix Matrix
    • Matrix Vector
  – GPUs have small read-only caches
    • Fine for traditional stream aps
  – P4 operating out of cache will cream GPUs
  – Graphics Hardware Paper 2004 (Kayvon)
    • “Why are GPUs so slow?”
research directions

- Applications
  - Gromacs
- Clustered BrookGPU
  - 16 node cluster
  - Each node 3U half depth
  - 32 2.4GHz P4 Xeons
  - 16GB DDR
  - 1.2TB disk
  - Infiniband interconnect
  - Dual 2.4GHz P4 Xeons
  - Intel E7505 chipset
  - 1GB DDR
  - ATI Radeon 9800 Pro 256MB
  - Infiniband 4X
  - GigE
  - 80 GB IDE