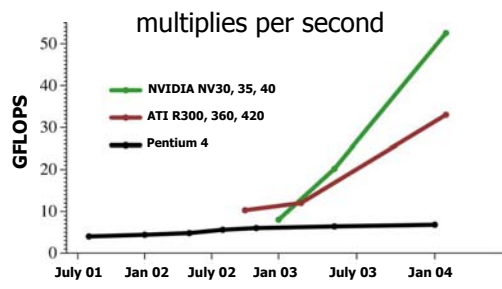


Ian Buck, Tim Foley, Daniel Horn, Jeremy Sugerman, Kayvon Fatahalian, Mike Houston, and Pat Hanrahan

## Motivation



## Design

- General purpose computing  
GPU = general stream coprocessor
- Friendly programming environment  
Programmer need not know GL  
Virtualize or abstract GPU resources
- Cross platform  
ATI & NVIDIA  
Linux and Windows  
DX and OpenGL

## In the News

- Open source  
<http://brook.sourceforge.net>
- Over 6,300 downloads in 8 months
- 163K page hits
- GPGPU SIGGRAPH Course
- In the news  
IEEE Computer  
<http://slashdot.org>  
<http://opengl.org>  
<http://ggpu.org>

## Brook Language

### 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

```
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);
```

### 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);
```

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

```
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);
```

### Gather Streams

- Indirect addressing of stream data allowed inside kernels.
- Argument passed with array syntax
  - `float array[]`
- Maps to depended texture lookup

### Iterator Streams

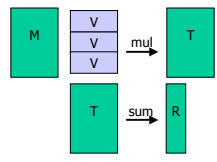
- Special stream type preinitialized with sequential values (1,2,3,4,...).
- `iter float s<100> = iter(0.0f, 100.0f);`
  - s initialized with 0.0, 1.0, 2.0, ..., 99.0

## Using Stream Shape: Matrix Multiply

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

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

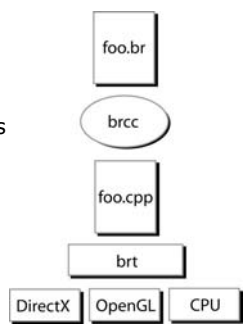
```
float matrix<20,10>;
float vector<1, 10>;
float tempmv<20,10>;
float result<20, 1>;
mul(matrix,vector,tempmv);
sum(tempmv,result);
```



## Running Brook

### BRCC Compiler

- Based on `ctool`  
– <http://ctool.sourceforge.net>
- Leverage vendor shader compilers
  - Microsoft: `fxc`
  - NVIDIA: `cgc`
- Converting kernels into shaders
  - stream fetch and store
  - gather operations
  - register mapping
  - stub function



### Kernel Code

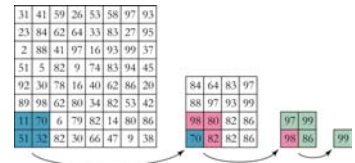
```
kernel void updatepos (float2 pos<>,
                    float2 vel[100][100],
                    float timestep,
                    out float2 newpos<>) {
    newpos = pos + vel[pos]*timestep;
}
```

### Compiled GPU Code

```
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;
}
```

## Brook GPU Runtime

- Kernel execution
  - issue screen size quad
  - texture coords provide stream positions
- Reductions
  - multi-pass method



## Virtualization

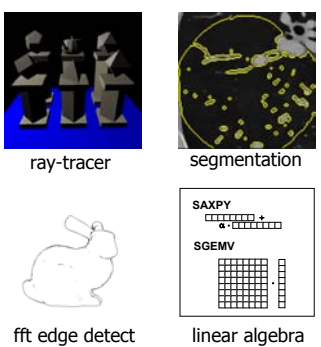
- Stream size and dimensions
  - packing streams into 2D segmented memory space
  - compiler inserts address translation code

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

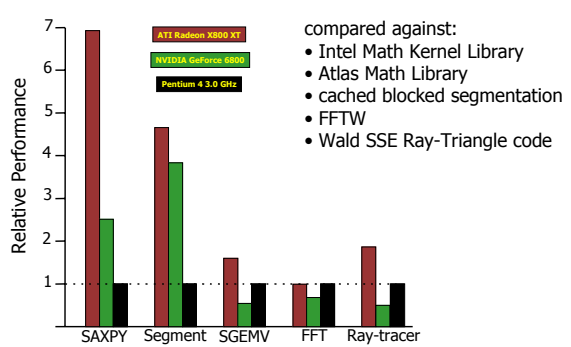


- Extending kernel outputs
  - duplicate kernels, let `cgc` or `fxc` do dead code elimination
  - better solution: "Efficient Partitioning of Fragment Shaders for Multiple-Output Hardware" Graphics Hardware 2004  
Tim Foley, Mike Houston, and Pat Hanrahan

## Applications



## GPU Results



The GPU implementation of Brook is supported by DARPA. Additional support has been provided by ATI, IBM, NVIDIA and SONY. The Brook programming language has been developed with support from Department of Energy, NNSA, under the ASC Alliances program (contract LLL-03-1491), the DARPA Smart Memories Project (contract MDA904-96-R-8855), and the DARPA Polymorphic Computing Architectures Project (contract F29601-00-2-0085). Additional support is provided by the NVIDIA Fellowship, Rambus Stanford Graduate fellowship, and Stanford School of Engineering fellowship programs.