Brook for GPUs: Stream Computing on Graphics Hardware

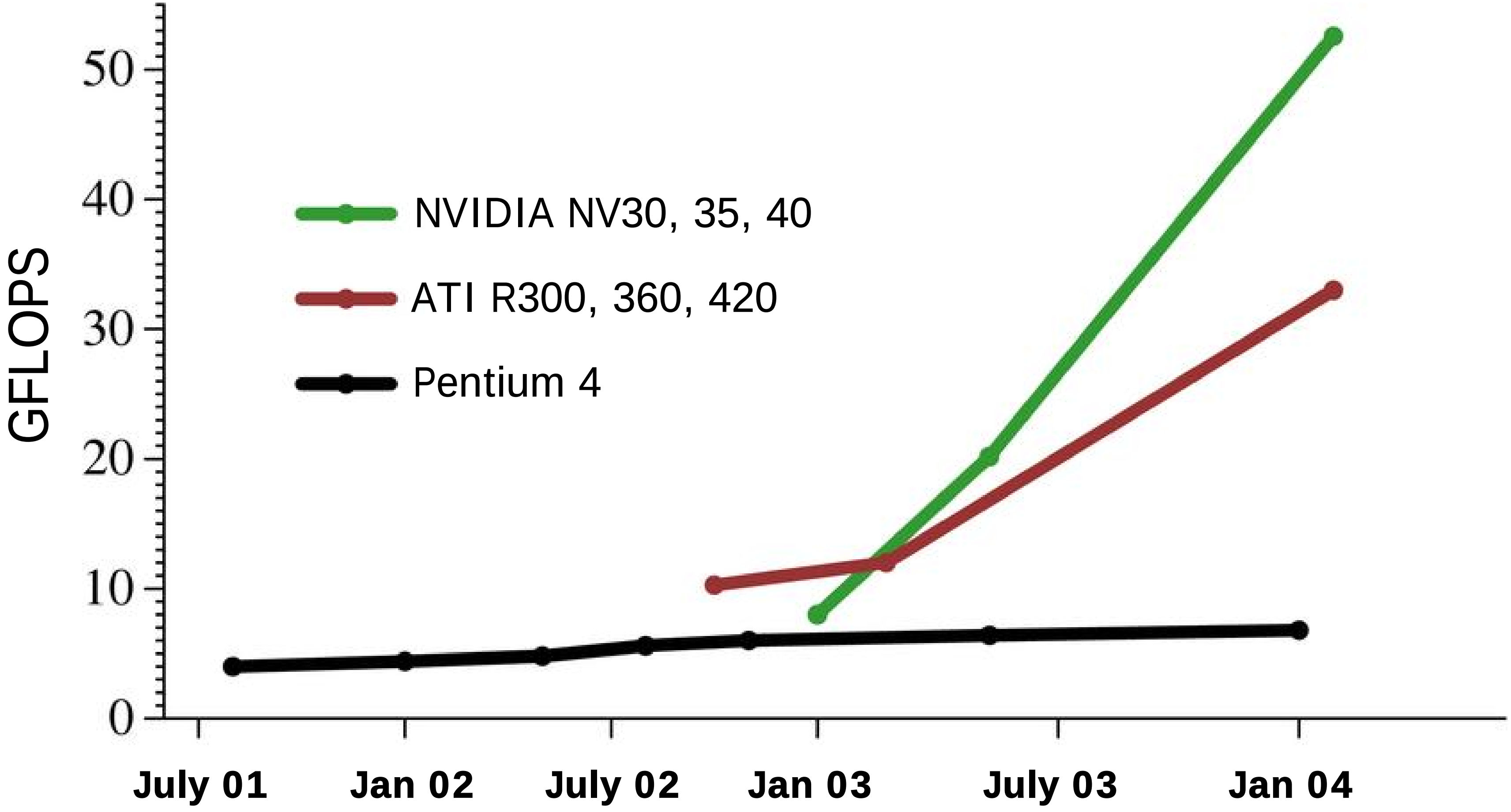
Ian Buck, Tim Foley, Daniel Horn, Jeremy Sugerman, Kayvon Fatahalian, Mike Houston, and Pat Hanrahan

Computer Science Department

Stanford University

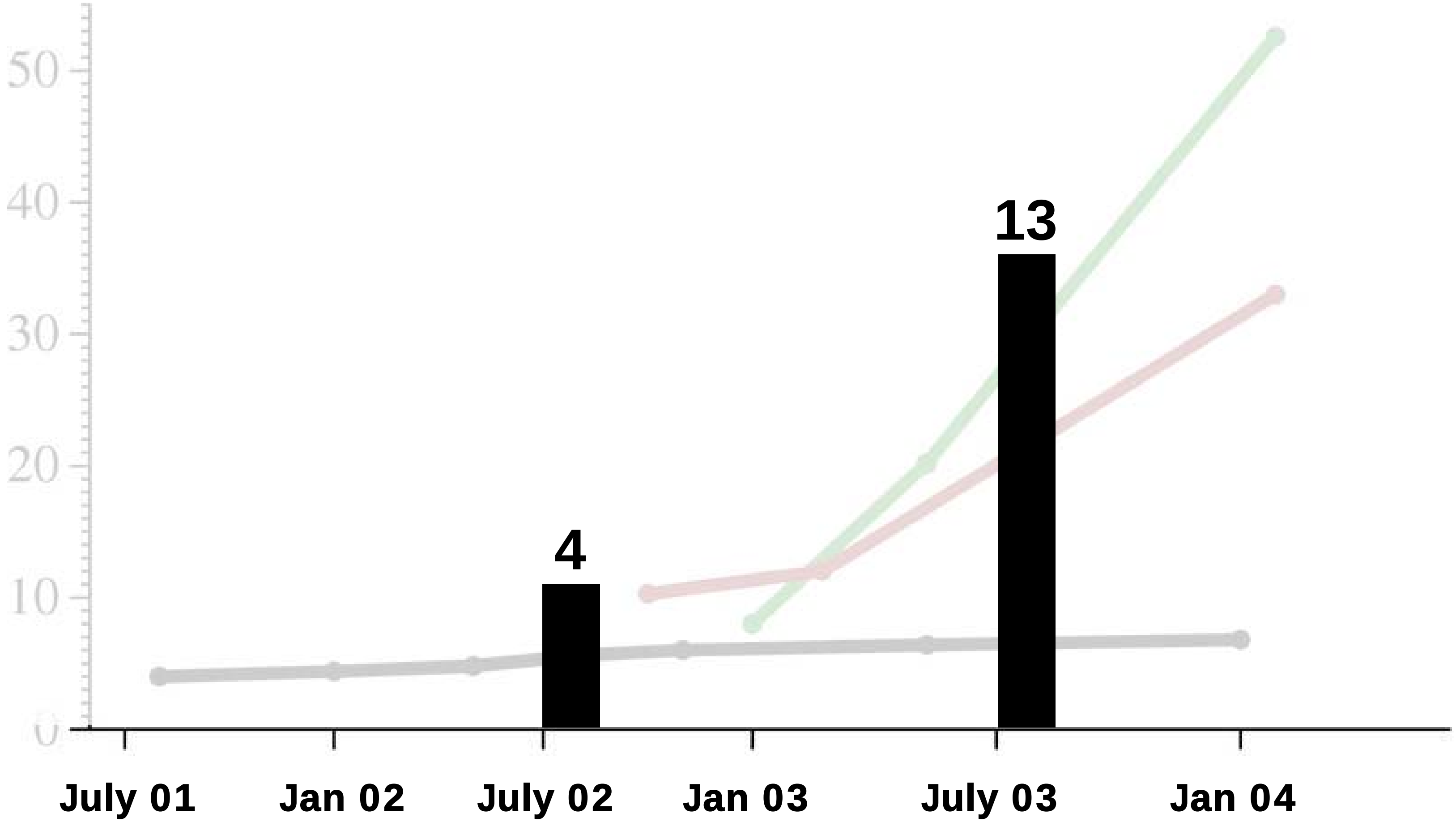
recent trends

multiplies per second

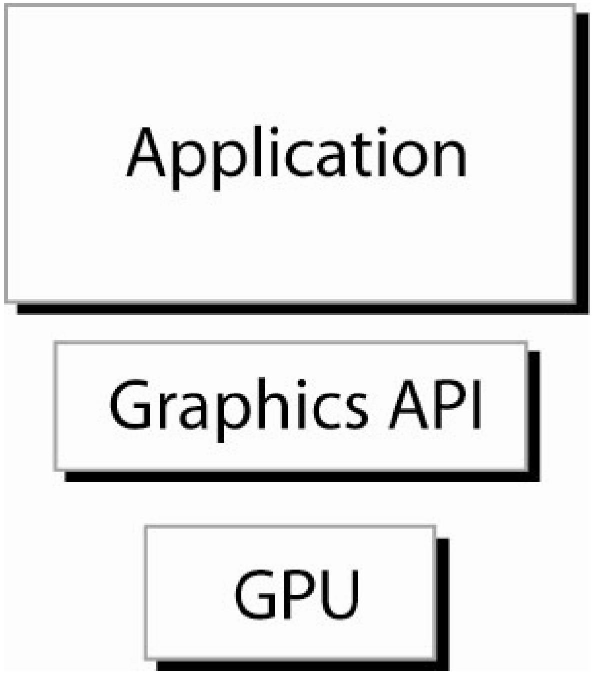


recent trends

GPU-based SIGGRAPH/Graphics Hardware papers



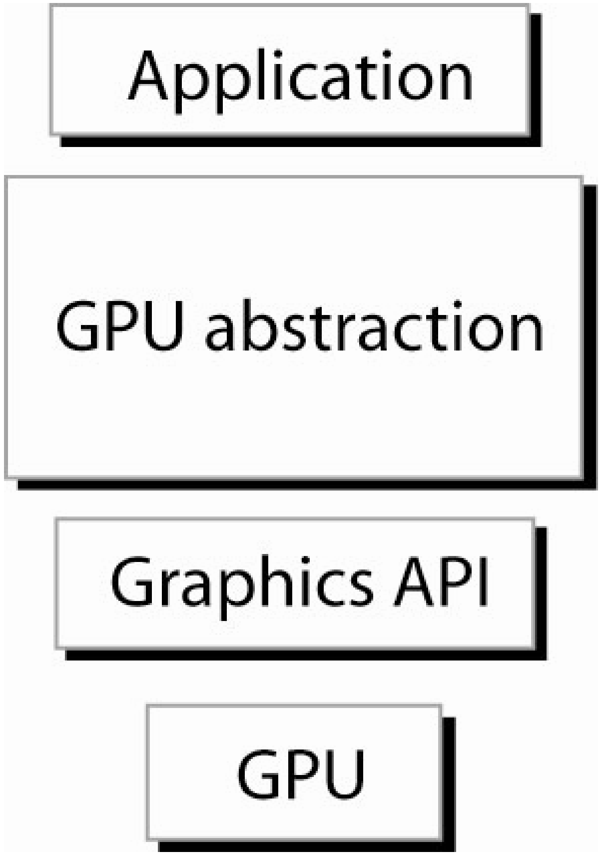
domain specific solutions



map directly to graphics primitives

requires extensive knowledge of GPU programming

building an abstraction



general GPU computing question

- can we simplify GPU programming?
- what is the correct abstraction for GPU-based computing?
- what is the scope of problems that can be implemented efficiently on the GPU?

contributions

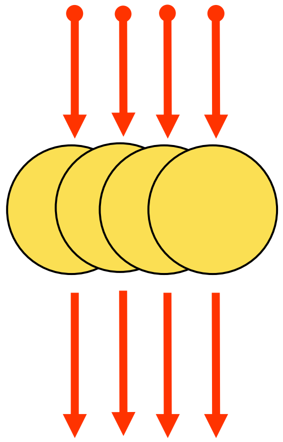
- Brook stream programming environment for GPU-based computing
  - language, compiler, and runtime system
- virtualizing or extending GPU resources
- analysis of when GPUs outperform CPUs

GPU programming model

each fragment shaded independently

– no dependencies between fragments
  - temporary registers are zeroed
  - no static variables
  - no read-modify-write textures

– multiple “pixel pipes”

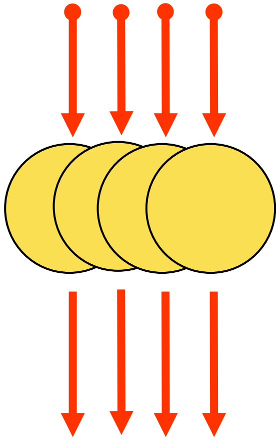


GPU = data parallel

each fragment shaded independently

– no dependencies between fragments
  - temporary registers are zeroed
  - no static variables
  - no read-modify-write textures

– multiple “pixel pipes”



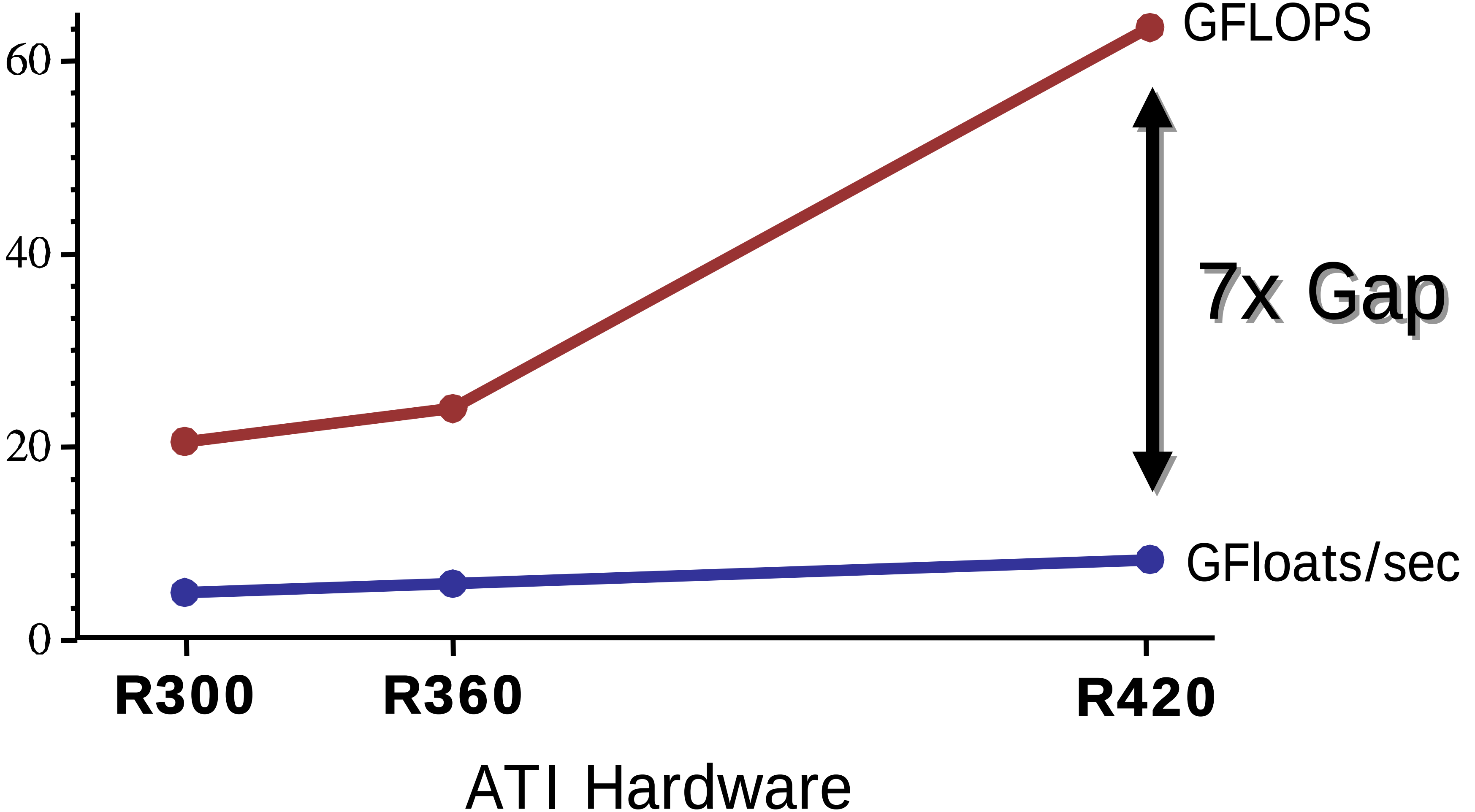
data parallelism

– support ALU heavy architectures

– hide memory latency

[Torborg and Kajiya 96, Anderson et al. 97, Igehy et al. 98]

compute vs. bandwidth



compute vs. bandwidth

arithmetic intensity =

compute-to-bandwidth ratio

graphics pipeline

– vextex
  - BW: 1 vertex = 32 bytes;
  - OP: 100-500 f32-ops / vertex
– fragment
  - BW: 1 fragment = 10 bytes
  - OP: 300-1000 i8-ops/fragment

Brook language

stream programming model

- enforce data parallel computing
  - streams
- encourage arithmetic intensity
  - kernels

design goals

- general purpose computing
  - GPU = general streaming-coprocessor
- GPU-based computing for the masses
  - no graphics experience required
  - eliminating annoying GPU limitations
- performance
- platform independent
  - ATI & NVIDIA
  - DirectX & OpenGL
  - Windows & Linux

Other languages

- Cg / HLSL / OpenGL Shading Language
  - + C-like language for expressing shader computation
  - – graphics execution model
  - – requires graphics API for data management and shader execution
- Sh [McCool et al. '04]
  - + functional approach for specifying shaders
  - • evolved from a shading language
- Connection Machine C*
- StreamIt, StreamC & KernelC, Ptolemy

Brook language

C with streams

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

```
Ray r<200>;
float3 velocityfield<100,100,100>;
```

  - data parallelism
    - provides data to operate on in parallel

kernels

- 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

- kernels arguments
  - input/output streams

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

kernels

- kernels arguments
  - input/output streams
  - gather streams

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

kernels

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

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

kernels

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

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

kernels

why not allow direct array operators?

A + B * C

– arithmetic intensity
  - temporaries kept local to computation

– explicit communication
  - kernel arguments

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 );
    qvec = cross( tvec, edge1 );
    candidatehit.data.z = dot( ray.d, qvec );
    candidatehit.data.x = dot( edge2, qvec );
    candidatehit.data.xyz *= inv_det;
    candidatehit.data.w = idx;
  } else {
    candidatehit.data = float4(0,0,0,-1);
  }
}
```

reductions

- reductions
  - compute single value from a stream

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

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);
```

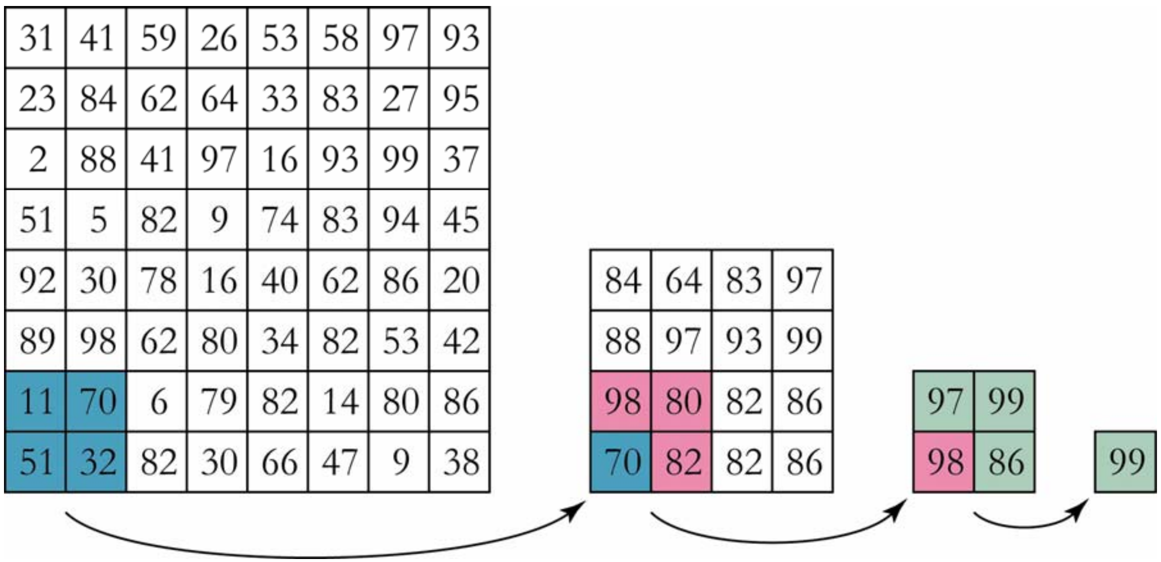
```
r = a[0];
for (int i=1; i<100; i++)
  r += a[i];
```

reductions

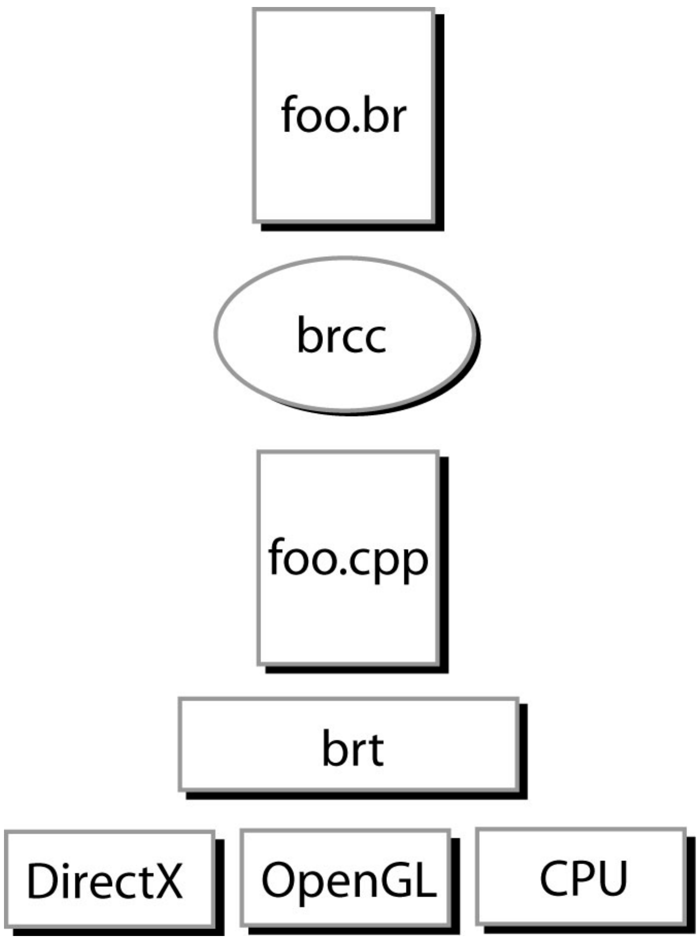
- reductions
  - associative operations only

    (a+b)+c = a+(b+c)

    - sum, multiply, max, min, OR, AND, XOR
    - matrix multiply
  - permits parallel execution



system outline



brcc

source to source compiler

- generate CG & HLSL code
- CGC and FXC for shader assembly
- virtualization

brt

Brook run-time library

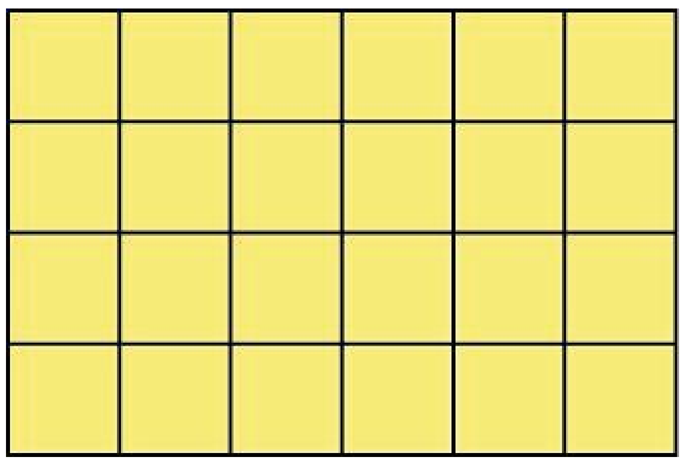
- stream texture management
- kernel shader execution

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

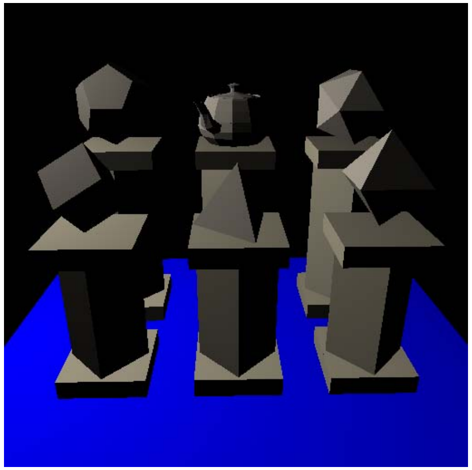
extending kernel outputs

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

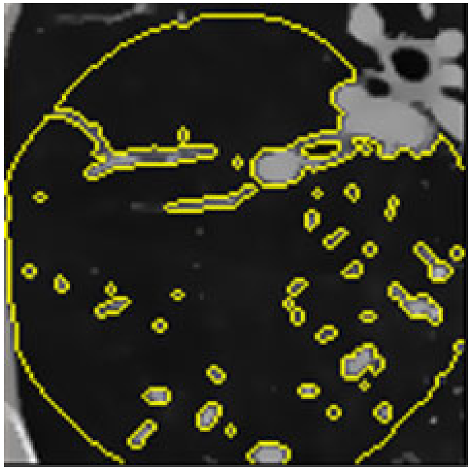
"Efficient Partitioning of Fragment Shaders for Multiple-Output Hardware”
Tim Foley, Mike Houston, and Pat Hanrahan

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

applications

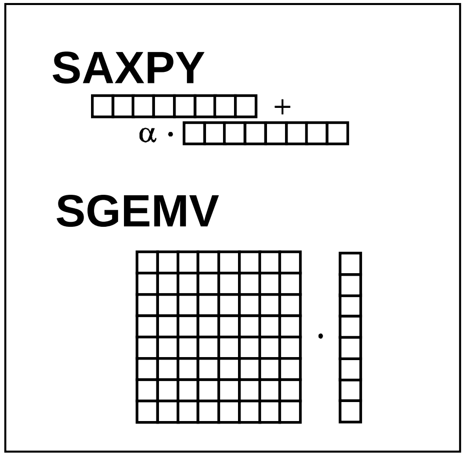


ray-tracer



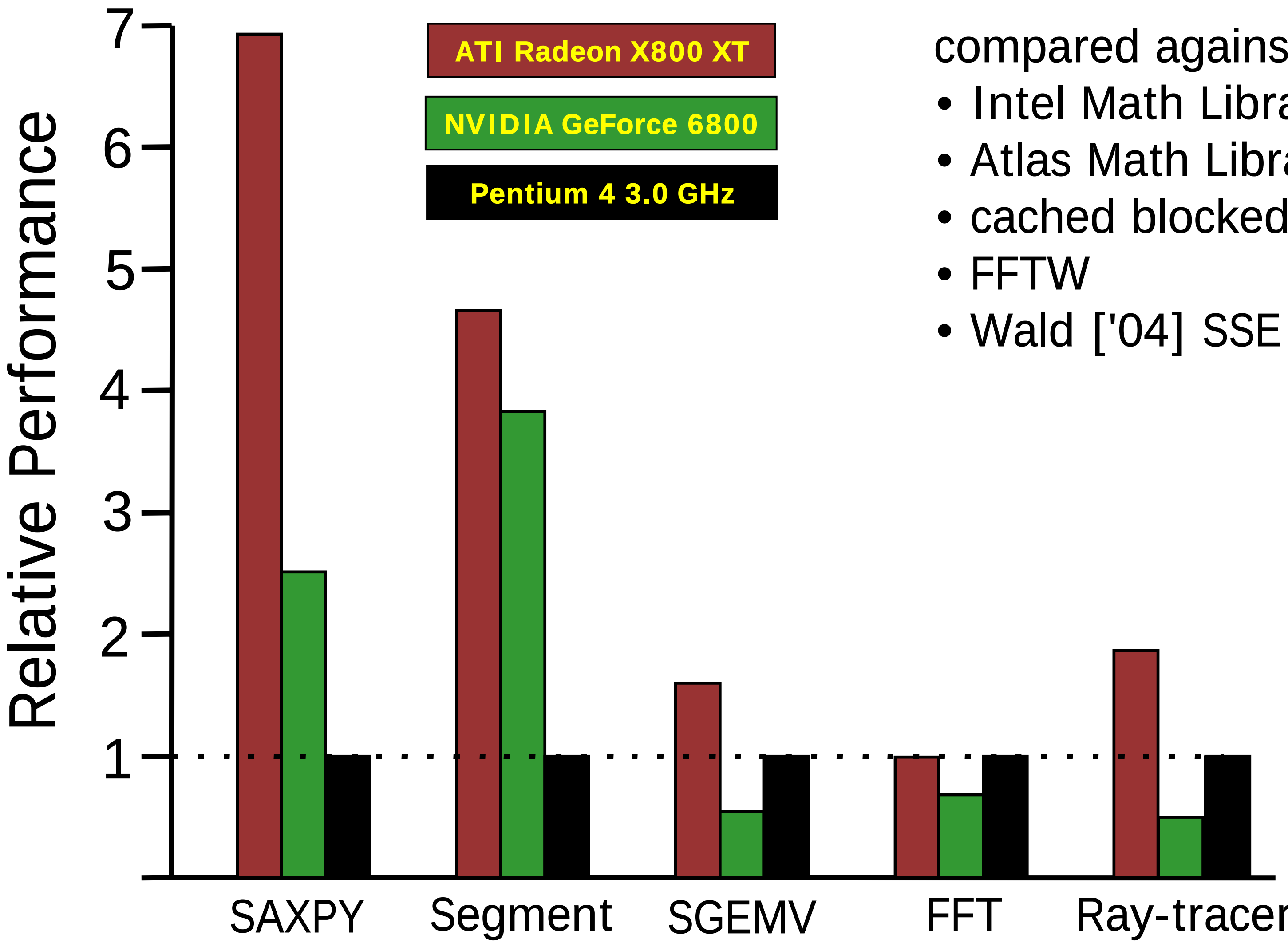
segmentation

fft edge detect



linear algebra

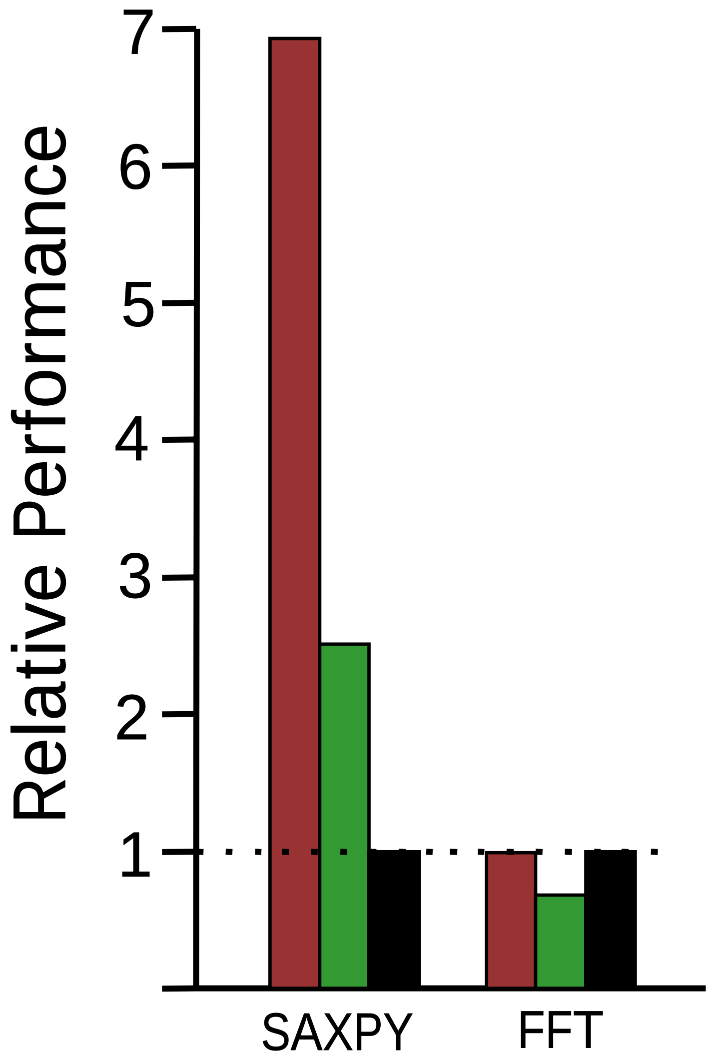
evaluation



compared against:

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

evaluation



GPU wins when...

- limited data reuse
  - ✓ SAXPY
  - ✗ FFT

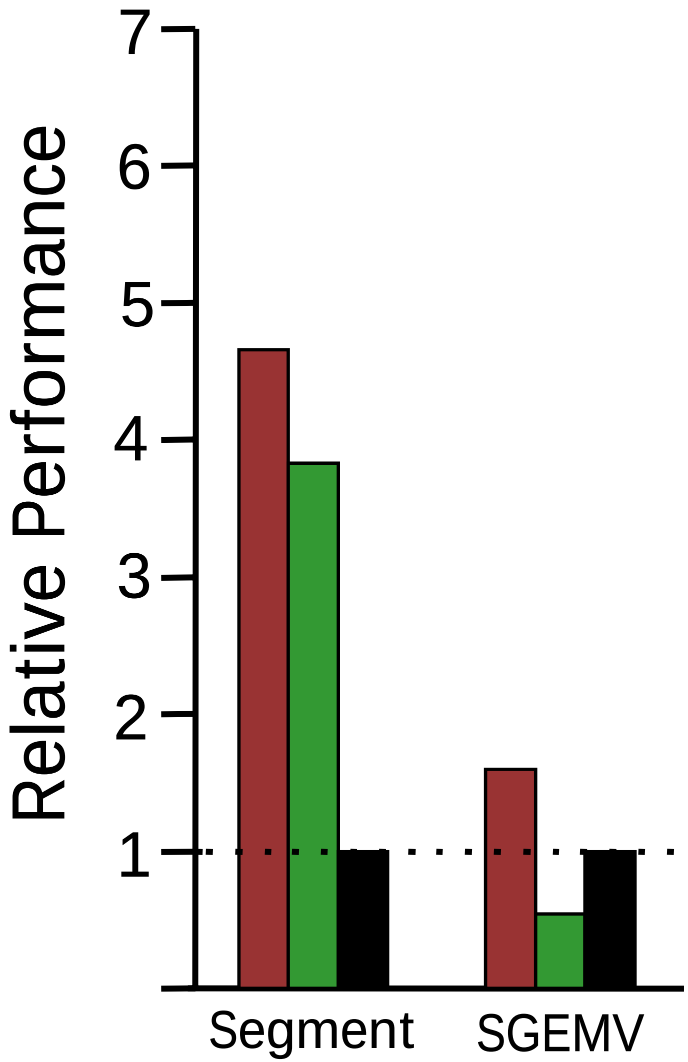
Pentium 4 3.0 GHz

44 GB/sec peak cache bandwidth

NVIDIA GeForce 6800 Ultra

36 GB/sec peak memory bandwidth

evaluation



GPU wins when...

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

outperforming the CPU

considering GPU transfer costs: T_r

- computational intensity: γ

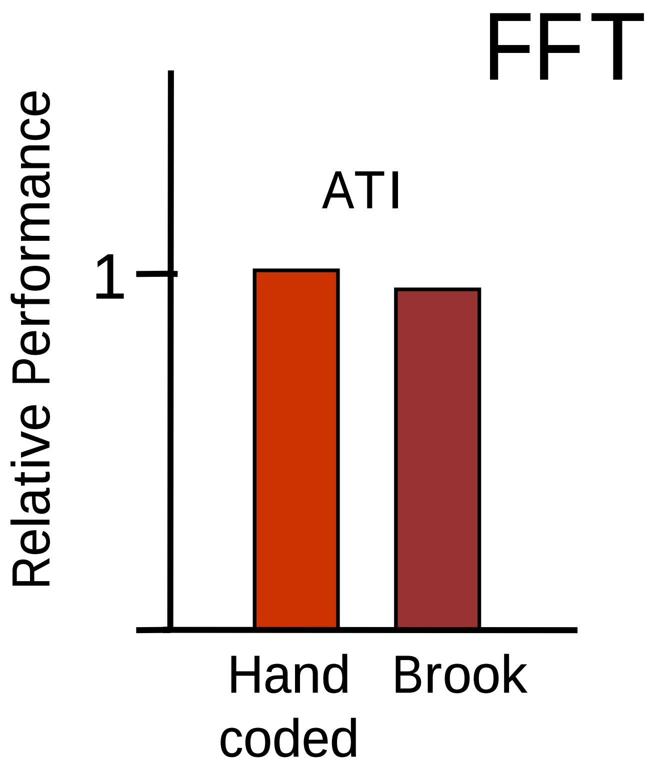
$$\gamma \equiv K_{gpu} / T_r$$

work per word transferred

considering CPU cost to issuing a kernel

efficiency

Brook version within 80% of hand-coded GPU version

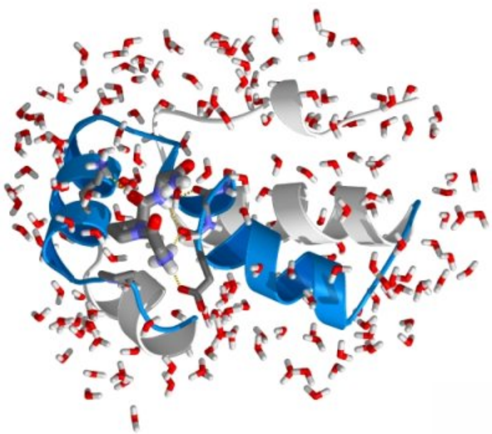


summary

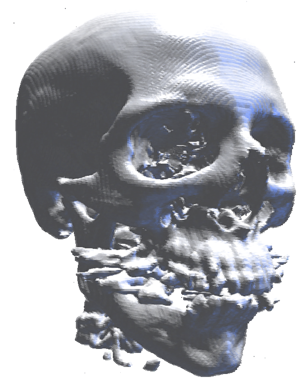
- GPUs are faster than CPUs
  - and getting faster
- why?
  - data parallelism
  - arithmetic intensity
- what is the right programming model?
  - Brook
  - stream computing

summary

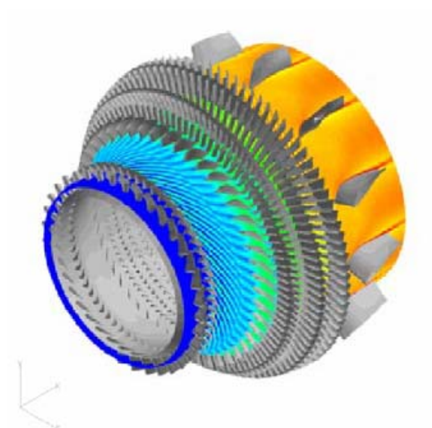
GPU-based computing for the masses



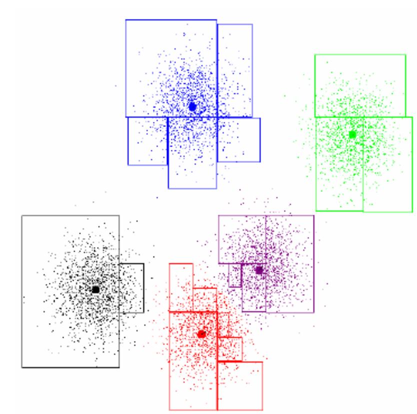
bioinfomatics



rendering



simulation



statistics

acknowledgements

- paper
  - Bill Mark (UT-Austin)
  - Nick Triantos, Tim Purcell (NVIDIA)
  - Mark Segal (ATI)
  - Kurt Akeley
  - Reviewers
- language
  - Stanford Merrimac Group
  - Reservoir Labs
- sponsors
  - DARPA contract MDA904-98-R-S855, F29601-00-2-0085
  - DOE ASC contract LLL-B341491
  - NVIDIA, ATI, IBM, Sony
  - Rambus Stanford Graduate Fellowship
  - Stanford School of Engineering Fellowship

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!
- interested in collaborating?

fly-fishing fly images from The English Fly Fishing Shop