CS8803SC

Software and Hardware Cooperative Computing

GPGPU

Prof. Hyesoon Kim

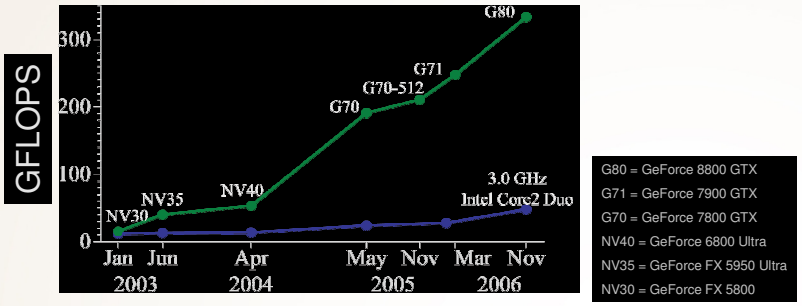
School of Computer Science

Georgia Institute of Technology

Georgia Institute of Technology

Why GPU?

- A quiet revolution and potential build-up
  - Calculation: 367 GFLOPS vs. 32 GFLOPS
  - Memory Bandwidth: 86.4 GB/s vs. 8.4 GB/s
- Until recently, programmed through graphics API



  - GPU in every PC and workstation – massive volume and potential impact

Georgia Tech College of Computing School of Computer Science

Computational Power

- *Why are GPUs getting faster so fast?*
  - Arithmetic intensity: the specialized nature of GPUs makes it easier to use additional transistors for computation not cache
  - Economics: multi-billion dollar video game market is a pressure cooker that drives innovation
- Architecture design decisions:
  - General CPU : cache, branch handling units, OOO support etc.
  - Graphics processor: most transistors are ALUs

Georgia Tech College of Computing School of Computer Science

www.gpgpu.org/s2004/slides/luebke.Introduction.ppt

GPGPU?

- http://www.gpgpu.org
- GPGPU stands for *General-Purpose computation on GPUs*

- General Purpose computation using GPU in applications other than 3D graphics
  - GPU accelerates critical paths of applications
- Data parallel algorithms leverage GPU attributes
  - Large data arrays, streaming throughput
  - Fine-grain SIMD parallelism
  - Low-latency floating point (FP) computation
- Applications
  - Game effects physics, image processing
  - Physical modeling, computational engineering, matrix algebra, convolution, correlation, sorting

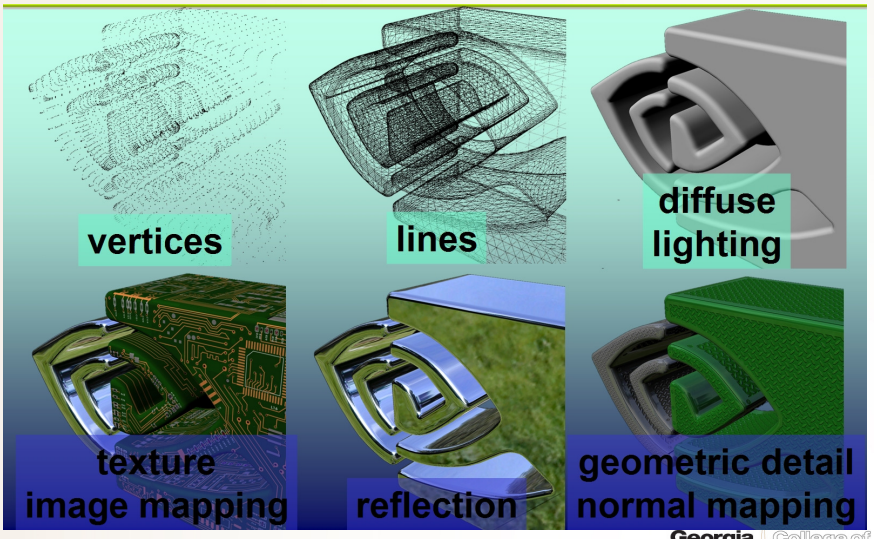
GPGPU

Georgia Tech College of Computing School of Computer Science

- Background on Graphics

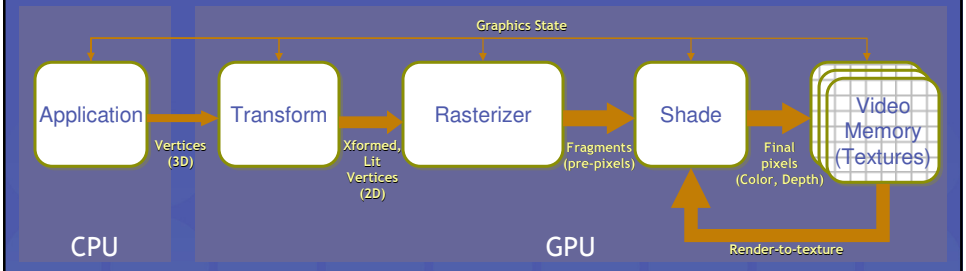
Georgia Tech College of Computing School of Computer Science

Describing an Object



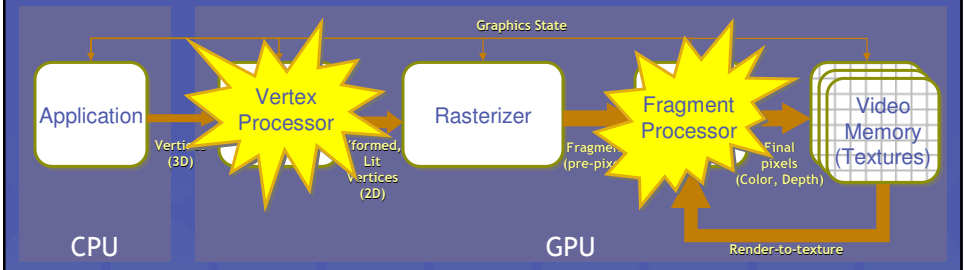
Georgia Tech College of Computing School of Computer Science

GPU Fundamentals: The Graphics Pipeline



- A simplified graphics pipeline
  - Note that pipe widths vary
- Many caches, FIFOs, and so on not shown

GPU Fundamentals: The *Modern* Graphics Pipeline



- Programmable vertex processor!
- Programmable pixel processor!

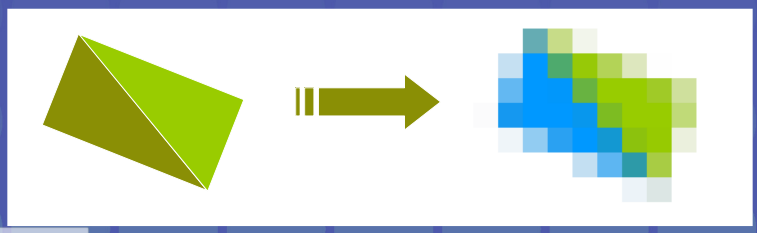
GPU Pipeline: Transform

- Vertex Processor (multiple operate in parallel)
  - Transform from “world space” to “image space”
  - Compute per-vertex lighting
  - Rotate, translate, and scale the entire scene to correctly place it relative to the camera’s position, view direction, and field of *view*.

GPGPU

GPU Pipeline: Rasterizer

- Rasterizer
  - Convert geometric rep. (vertex) to image rep. (fragment)
    - Fragment = image fragment
      - Pixel + associated data: color, depth, stencil, etc.
  - Interpolate per-vertex quantities across pixels

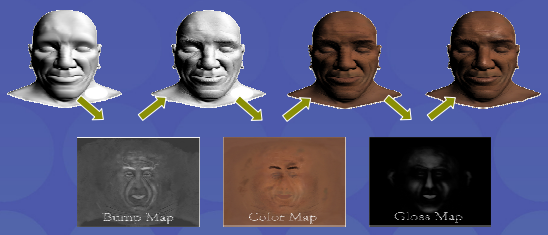


GPGPU

GPU Pipeline: Shade

SIGGRAPH2004

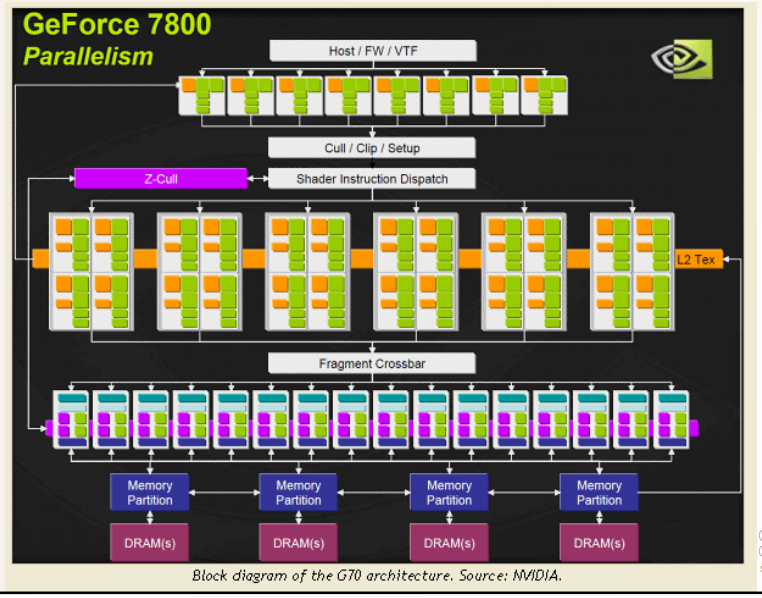
- Fragment Processors (multiple in parallel)
  - Compute a color for each pixel
  - Optionally read colors from textures (images)



A **fragment** is a computer graphics term for all of the data necessary needed to generate a pixel in the frame buffer. This may include, but is not limited to: raster position depth interpolated attributes (color, texture coordinates , etc.)

GPGPU

NVIDIA GeForce 7800 Pipeline

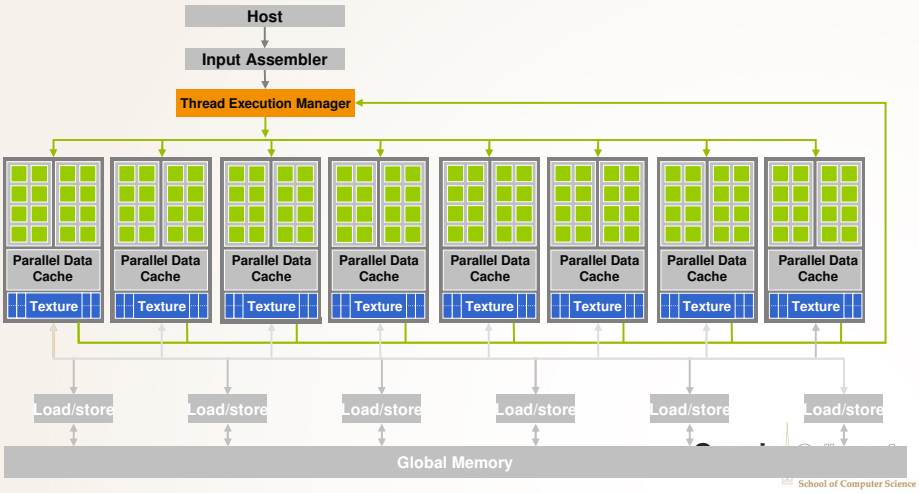


Block diagram of the G70 architecture. Source: NVIDIA.

College of Computing
School of Computer Science

GeForce 8800

16 highly threaded SM's, >128 FPU's, 367 GFLOPS,
768 MB DRAM, 86.4 GB/S Mem BW, 4GB/S BW to CPU



GPGPU Programming

- Traditional GPGPU
  - Use a pixel processor, vortex processor, texture cache ..
  - Copies from the frame buffer to a texture
  - Uses a texture as the frame buffer
- With CUDA
  - Highly parallel threads
  - SIMD programming with MPI style

What Kinds of Computation Map Well to GPUs?

- Computing graphics ☺
- Two key attributes:
  - Data parallelism
  - Independence
- Arithmetic Intensity
  - Arithmetic intensity = operations/works transferred

Georgia Tech College of Computing School of Computer Science

Data Streams & Kernels

- Streams
  - Collection of records requiring similar computation
    - Vertex positions, Voxels, FEM cells, etc.
  - Provide data parallelism
- Kernels
  - Functions applied to each element in stream
    - transforms, PDE, …
  - No dependencies between stream elements
    - Encourage high Arithmetic Intensity

Georgia Tech College of Computing School of Computer Science

CPU-GPU Analogies

CPU		GPU
Inner loops	=	Kernels
Stream / Data Array	=	Texture
Memory Read	=	Texture Sample

Georgia Tech College of Computing School of Computer Science

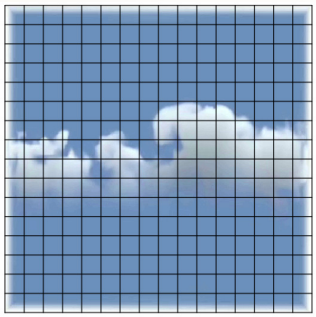
Importance of Data Parallelism

- GPUs are designed for graphics
  - Highly parallel tasks
- GPUs process *independent* vertices & fragments
  - Temporary registers are zeroed
  - No shared or static data
  - No read-modify-write buffers
- Data-parallel processing
  - GPUs architecture is ALU-heavy
    - Multiple vertex & pixel pipelines, multiple ALUs per pipe
  - Hide memory latency (with more computation)

Georgia Tech College of Computing School of Computer Science

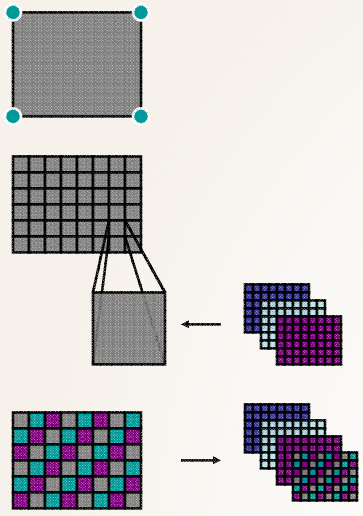
Example: Simulation Grid

- Common GPGPU computation style
  - Textures represent computational grids = streams
- Many computations map to grids
  - Matrix algebra
  - Image & Volume processing
  - Physical simulation
  - Global Illumination
    - ray tracing, photon mapping, radiosity
- Non-grid streams can be mapped to grids



Georgia Tech College of Computing School of Computer Science

Programming a GPU for Graphics

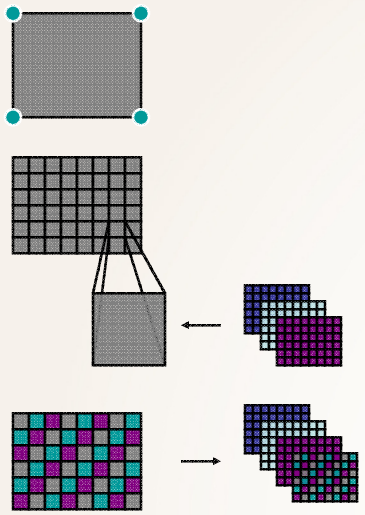


- Application specifies geometry → rasterized
- Each fragment is shaded w/ SIMD program
- Shading can use values from texture memory
- Image can be used as texture on future passes

Georgia Tech College of Computing School of Computer Science

Owens & Luebke

Programming a GPU for GP Programs



- Draw a screen-sized quad → stream
- Run a SIMD kernel over each fragment
- "Gather" is permitted from texture memory
- Resulting buffer can be treated as texture on next pass

Georgia Tech College of Computing School of Computer Science

Owens & Luebke

Kernels

CPU

advect

GPU

```
for (int j = 1; j < height - 1; ++j)
{
  for (int i = 1; i < width - 1; ++i)
  {
    // get velocity at this cell
    Vec2f v = grid(x, y);

    // trace backwards along velocity field
    float x = (i - (v.x * timestep / dx));
    float y = (j - (v.y * timestep / dy));

    grid(x,y) = grid.bilerp(x, y);
  }
}
```

C++

```
void advect(float2      uv   : WPOS,
            out float4  xNew : COLOR,

    uniform float       dt, // timestep
    uniform float       dx, // grid scale
    uniform samplerRECT u,  // velocity
    uniform samplerRECT x)  // state
{
  // trace backwards along velocity field
  float2 pos = ub - dt * f2texRECT(u, uv) / dx;

  xNew = f4texRECTbilerp(x, pos);
}
```

Cg

Kernel / loop body / algorithm step = Fragment Program

Georgia Tech College of Computing School of Computer Science

Owens & Luebke

CUDA Programming Model: A Highly Multithreaded Coprocessor

- The GPU is viewed as a compute device that:
  - Is a coprocessor to the CPU or host
  - Has its own DRAM (device memory)
  - Runs many threads in parallel
- Data-parallel portions of an application are executed on the device as kernels which run in parallel on many threads
- Differences between GPU and CPU threads
  - GPU threads are extremely lightweight
    - Very little creation overhead
  - GPU needs 1000s of threads for full efficiency
    - Multi-core CPU needs only a few

Georgia Tech College of Computing School of Computer Science

CUDA: Matrix Multiplication

```
__global__ void MatrixMulKernel(Matrix M, Matrix N, Matrix P)
{
    // 2D Thread ID
    int tx = threadIdx.x;
    int ty = threadIdx.y;

    // Pvalue is used to store the element of the matrix
    // that is computed by the thread
    float Pvalue = 0;
for (int k = 0; k < M.width; ++k)
    {
        float Melement = M.elements[ty * M.pitch + k];
        float Nelement = Nd.elements[k * N.pitch + tx];
        Pvalue += Melement * Nelement;
    }
    // Write the matrix to device memory;
    // each thread writes one element
    P.elements[ty * P.pitch + tx] = Pvalue;
}
}
```

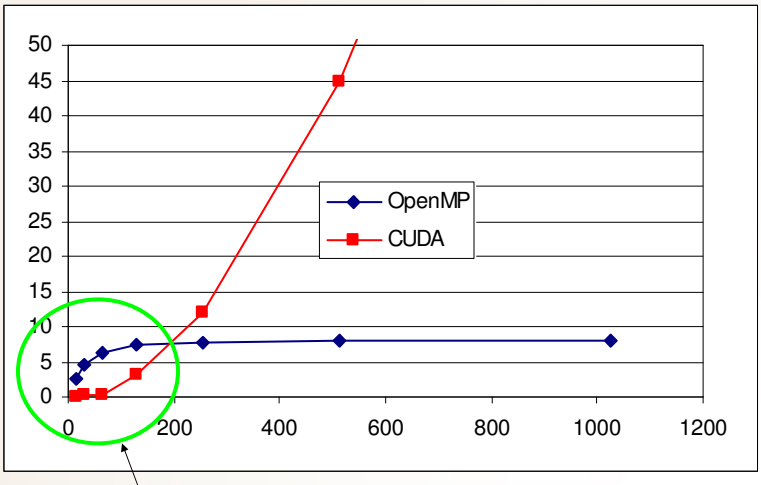
Georgia Tech College of Computing School of Computer Science

Limitations in GPGPU

- High latency between CPU-GPU
- Handling control flow graphs
- I/O access
- Bit operations
- Limited data structure (e.g., no link list)

- But then why are we looking at this?
  - Relatively short development time
  - Relatively cheap devices

Is GPU always Good?



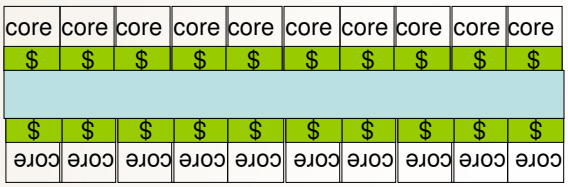
Not enough data parallelism
GPU overhead is higher than the benefit

- Parallel programming is difficult.
- GPGPU could be one solution to utilize parallel processors.

Georgia Tech College of Computing School of Computer Science

The Future of GPGPU?

- Architecture is a moving target.
- Programming environment is evolving.
- e.g.) Intel's Larrabee ('09 expected)



MIMD style
Can it provide enough performance to beat Nvidia??

Georgia Tech College of Computing School of Computer Science