CS/CoE 1541 Introduction to Computer Architecture

Graphics and Computing GPUs

Sangyeun Cho

Dept. of Computer Science
University of Pittsburgh

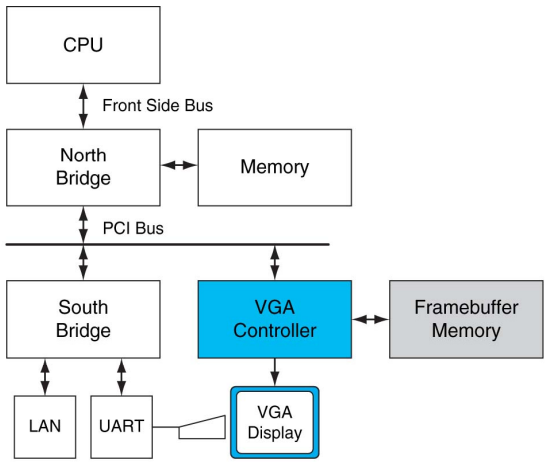
Some terms

- GPU = graphics processing unit
  - Integrates 2D/3D graphics, images, and video that enable window-based OSes, GUIs, video games, visual imaging applications, and video
- Visual computing
  - A mix of graphics processing and computing that lets you visually interact with computed objects via graphics, images, and video
- Heterogeneous system
  - A system combining different processor types; a PC is a heterogeneous CPU-GPU system

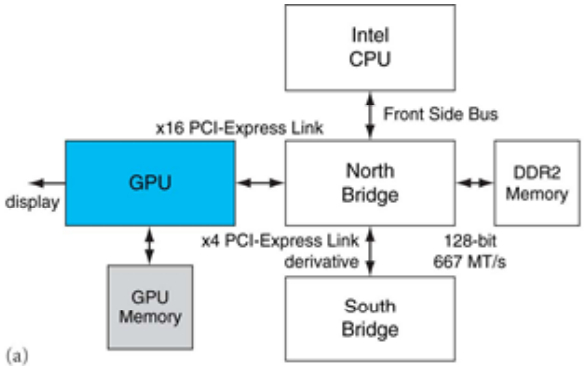
GPU evolution

- VGA in early 90's
  - A memory controller and display generator connected to some (video) RAM
- By 1997, VGA controllers were incorporating some 3D acceleration functions
- In 2000, a single chip graphics processor incorporated almost every detail of the traditional high-end workstation graphics pipeline (1st generation GPUs)
- More recently, processor instructions and memory hardware were added to support general-purpose programming languages
  - Hardware has evolved to include double-precision floating-point operations and massive parallel programmable processors

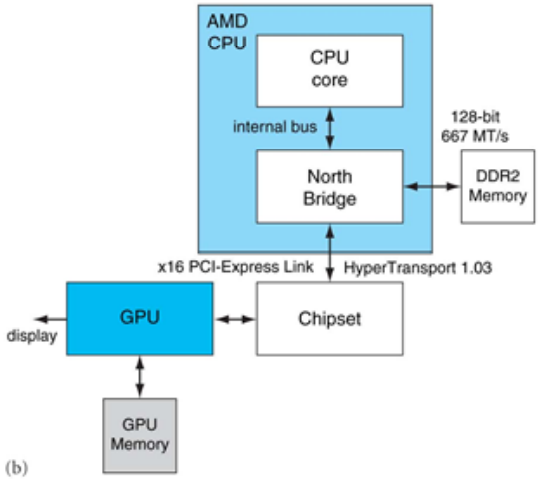
Historical PC architecture



Contemporary PC architecture



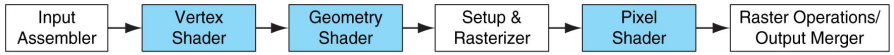
Contemporary PC architecture



More terms

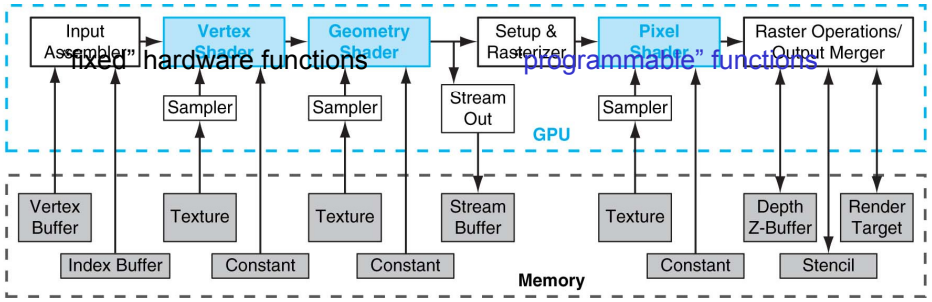
- OpenGL
  - A standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics
- DirectX
  - (Microsoft) A collection of APIs for handling tasks related to multimedia, especially game programming and video
- CUDA (compute unified device architecture)
  - (nVIDIA) A scalable parallel programming model and language based on C/C++; it is a parallel programming platform for GPUs and multicore CPUs

Graphics "logical" pipeline



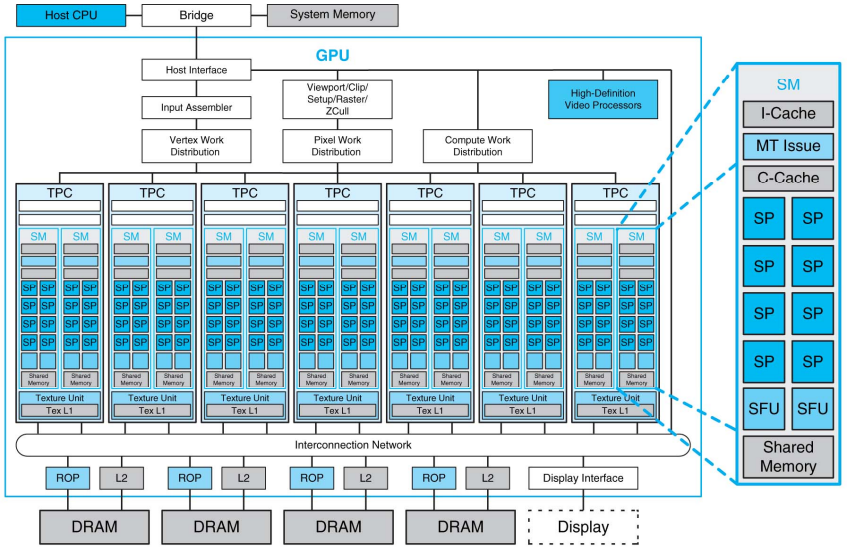
- Input assembler collects vertices and primitives
- Vertex shader executes per-vertex processing, e.g., transforming the vertex 3D position into a screen position, lighting the vertex to determine its color
- Geometry shader executes per-primitive processing
- Setup/rasterizer generates pixel fragments that are covered by a geometric primitive
- Pixel shader performs per-fragment processing, e.g., interpolating per-fragment parameters, texturing, and coloring; it makes extensive use of sampled and filtered lookups into large 1D, 2D, or 3D arrays called textures
- Raster operations processing stage performs Z-buffer depth testing and stencil testing

Graphics "logical" pipeline



Various objects and buffers are allocated in the GPU memory hierarchy

Basic unified GPU architecture



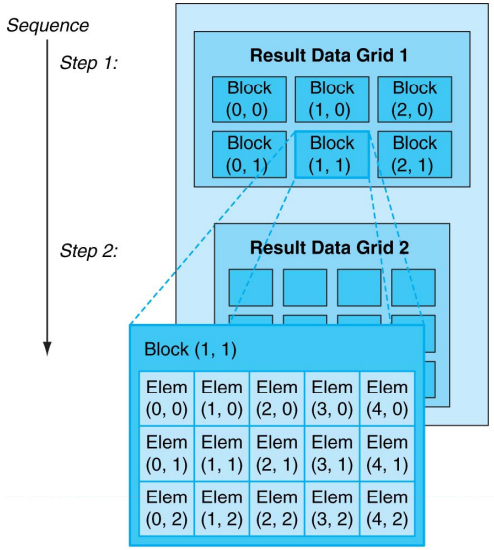
Pixel shader example

```
// called for each pixel thread
void reflection(
  float2      texCoord            : TEXCOORD0,
  float3      reflection_dir      : TEXCOORD1,
  out float4  color               : COLOR,
  uniform float shiny,
  uniform sampler2D surfaceMap,
  uniform samplerCUBE envMap)
{
  // fetch the surface color from a texture
  float4 surfaceColor = tex2D(surfaceMap, texCoord);
  // fetch reflected color by sampling a cube map
  float4 reflectedColor = texCUBE(envMap, reflection_dir);
  // output is weighted average of the two colors
  color = lerp(surfaceColor, reflectedColor, shiny);
}
```

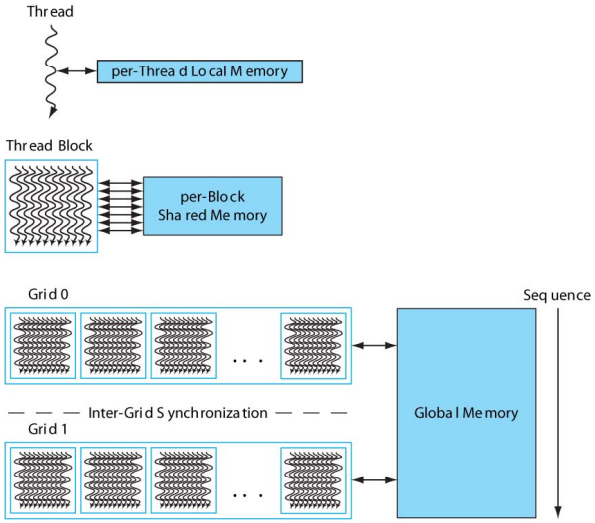

CUDA

- Developed by nVIDIA in 2007
- An data-parallel extension to the C/C++ languages for scalable parallel programming of manycore GPUs and multicore CPUs
- CUDA provides three key abstractions–a hierarchy of thread groups, shared memories, and barrier synchronization
- The programmer or compiler decomposes large computing problems into many small problems that can be solved in parallel

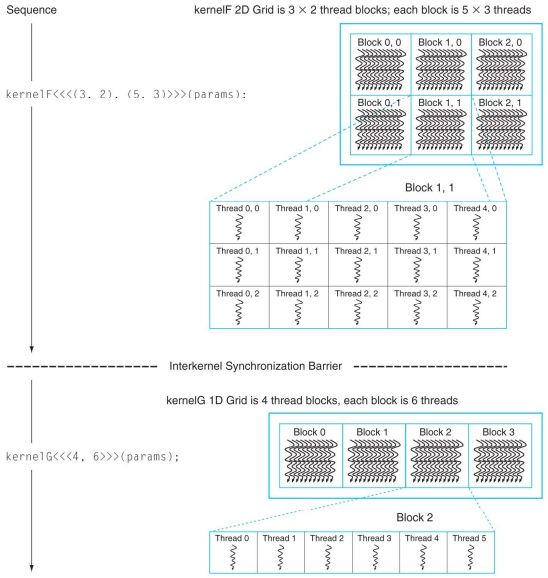
Decomposition of result data



Nested levels and memory



Core count independence



Restrictions

- Threads and thread blocks may only be created by invoking a parallel kernel, not from within a parallel kernel
- Thread blocks must be independent (no scheduling/ordering requirement)
  - The above two restrictions allow an efficient hardware management and scheduling of threads and thread blocks
- Recursive function calls are not allowed
- CUDA programs must copy data and results between host memory and device memory
  - DMA block transfer minimizes the overhead of CPU-GPU data transfer
  - Compute intensive problems amortize the data transfer overheads