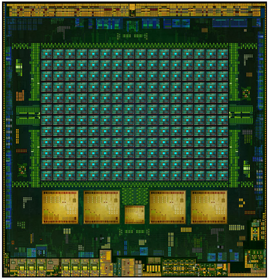
OpenGL ES



Jens Ogniewski
Information Coding Group
Linköping University

- Introduction / Motivation
- OpenGL ES - Differences
- Good Practice

Introduction / Motivation

- **NVIDIA: every GPU architecture we develop will be mobile-first**

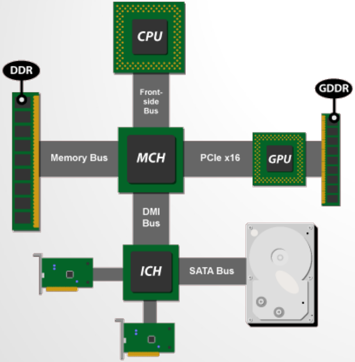
Introduction / Motivation

- **NVIDIA: every GPU architecture we develop will be mobile-first**
- **PCs (incl. laptops) sales declining, smartphones/tablets fast rising**
  - Will smartphones/tablets replace PCs?

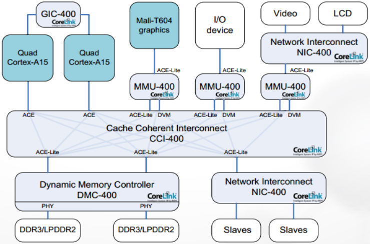
Introduction / Motivation

- **NVIDIA: every GPU architecture we develop will be mobile-first**
- **PCs (incl. laptops) sales declining, smartphones/tablets fast rising**
  - Will smartphones/tablets replace PCs?
- **Driving factors: cheaper, smaller, longer runtime**
  - due to different architecture (so called System-on-a-Chip (SOC))

Introduction / Motivation



PC



SOC

Introduction / Motivation

PC

- Distributed memory and caches

SOC

- Shared memory, small caches (if at all)

Introduction / Motivation

PC

- Distributed memory and caches
- Several broad, often star-organized busses

SOC

- Shared memory, small caches (if at all)
- One central bus, limited size

Introduction / Motivation

PC	SOC
▶ Distributed memory and caches	▶ Shared memory, small caches (if at all)
▶ Several broad, often star-organized busses	▶ One central bus, limited size
▶ Optimized for Performance	▶ Optimized for Efficiency

Introduction / Motivation

- **ARM**
  - THE SOC CPU
  - Licensing rather than selling

Introduction / Motivation

- **ARM**
  - THE SOC CPU
  - Licensing rather than selling
  - GPU core: Mali
    Only middle-class performance
    Cheap? Good for high resolution?

Introduction / Motivation

- **ARM**
  - THE SOC CPU
  - Licensing rather than selling
  - GPU core: Mali
    Only middle-class performance
    Cheap? Good for high resolution?
- **Imagination Technologies: PowerVR**
  - Traditionally best architecture
  - GPU used in the Dreamcast
  - Also licensing core, not selling chips

Introduction / Motivation

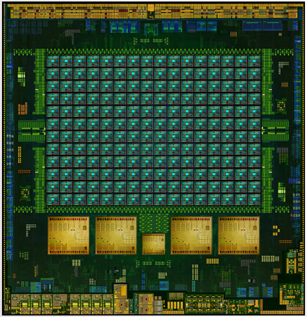
- **Qualcomm: Snapdragon**
  - Chip manufacturer
  - ARM CPUs, Own GPU: Adreno (former ATI)
  - Currently fastest (due to fast memory access?)

Introduction / Motivation

- **Qualcomm: Snapdragon**
    - Chip manufacturer
    - ARM CPUs, Own GPU: Adreno (former ATI)
    - Currently fastest (due to fast memory access?)
- **NVIDIA: Tegra**
    - Chip including ARM CPUs and own GPU
    - good runtime, not so good performance
    - Until now: only non-unified architecture

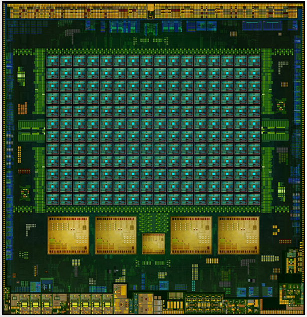
Introduction / Motivation

K1



Introduction / Motivation

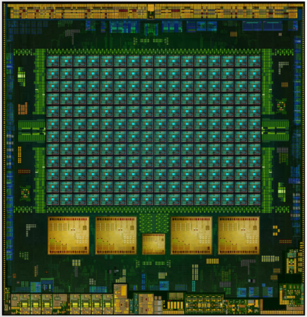
K1



	Performance (GFLOPS)	Memory Bandwidth (GB/s)
K1	365	17

Introduction / Motivation

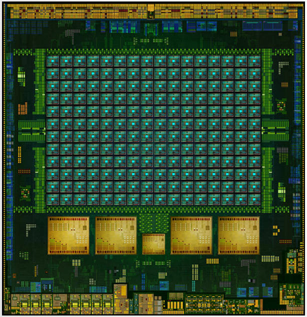
K1



	Performance (GFLOPS)	Memory Bandwidth (GB/s)
K1	365	17
GeForce GTX 780 TI	5000	336

Introduction / Motivation

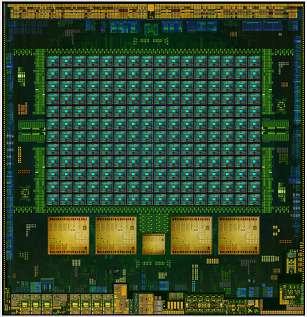
K1



	Performance (GFLOPS)	Memory Bandwidth (GB/s)
K1	365	17
GeForce GTX 780 TI	5000	336
Speedup (Graphics card)	x13.7	x19.8

Introduction / Motivation

K1



	Performance (GFLOPS)	Memory Bandwidth (GB/s)
K1	365	17
GeForce GTX 780 TI	5000	336
Speedup (Graphics card)	x13.7	x19.8
Compares to graphic-card	6 years old	11 years old

Introduction / Motivation

- SOCs are here to stay

Introduction / Motivation

- SOCs are here to stay
- Gap to PC won't become significantly smaller

Introduction / Motivation

- SOCs are here to stay
- Gap to PC won't become significantly smaller
- Gap between performance and memory bandwidth will become larger

OpenGL ES Versions

- **1.x: Fixed Pipeline**
  - not compatible to 2.0/3.0

OpenGL ES Versions

- **1.x: Fixed Pipeline**
  - not compatible to 2.0/3.0
- **2.0: Streamlined OpenGL**
  - Removed obscure methods
  - Optimize existing methods for low pow performance hardware
  - Introduce new specialized methods and data structures

OpenGL ES Versions

- **1.x: Fixed Pipeline**
  - not compatible to 2.0/3.0
- **2.0: Streamlined OpenGL**
  - Removed obscure methods
  - Optimize existing methods for low pow performance hardware
  - Introduce new specialized methods and data structures
- **3.0: “Simple” Extension to 2.0**
  - more flexible than 2.0
  - fully compatible

Differences

- **Only 2 shaders**
  - Vertex & Fragment
  - No Tesselation or Geometry Shaders

Differences

- **Only 2 shaders**
  - Vertex & Fragment
  - No Tesselation or Geometry Shaders
- **Removed memory-intensive operations and data structures**
  - Limited Anti-Aliasing
  - 2.0: Textures: only byte data types, only 2D
    3.0: also float data types, also 3D textures
  - Good support for texture compression

Differences

- **Memory Access**
  - Vertex buffer: standard in 3.0, optional in 2.0 (available in older iOS)
  - Same memory: can pass pointers from CPU to GPU

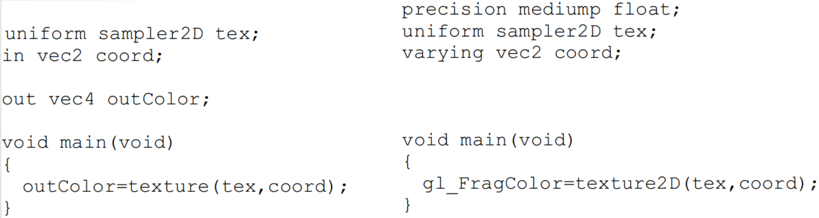
Differences

- **Memory Access**
  - Vertex buffer: standard in 3.0, optional in 2.0 (available in older iOS)
  - Same memory: can pass pointers from CPU to GPU
- **Need to declare precision for shader variables**
  - Select lesser precision for better performance
  - Recommendation: high precision in vertex shader, medium in fragment shader
  - Caution: if the same uniform variable is used in both vertex and fragment shader it has to have the same precision in both

Differences

OpenGL 3.0

OpenGL ES



OpenGL 3.0

```
uniform sampler2D tex;
in vec2 coord;

out vec4 outColor;

void main(void)
{
  outColor=texture(tex,coord);
}
```

OpenGL ES

```
precision mediump float;
uniform sampler2D tex;
varying vec2 coord;


void main(void)
{
  gl_FragColor=texture2D(tex,coord);
}
```

Which version to use?

- **1.x:**
- **2.0:**
- **3.0:**

Which version to use?

- 1.x: Too old, limited
- 2.0:
- 3.0:

Which version to use?

- **1.x: Too old, limited**
- **2.0:**
- **3.0: Still too new?**
  - Android: since 4.3
  - iOS: since 7 (older phones and pads might not support it)
  - Blackberry: since 10.2

Which version to use?

- **1.x: Too old, limited**
- **2.0: Best bet, should offer enough flexibility for most purposes**
- **3.0: Still too new?**
  - Android: since 4.3
  - iOS: since 7 (older phones and pads might not support it)
  - Blackberry: since 10.2

Future version(s)

- **OpenGL ES completely a subset of OpenGL**
  - Even more so than today
  - Cross-platform development

Future version(s)

- **OpenGL ES completely a subset of OpenGL**
  - Even more so than today
  - Cross-platform development
  - High-end SOCs: can use normal OpenGL, but low-end not!
  - also: OpenGL ES might be faster

Future version(s)

- **OpenGL ES completely a subset of OpenGL**
  - Even more so than today
  - Cross-platform development
  - High-end SOCs: can use normal OpenGL, but low-end not!
  - also: OpenGL ES might be faster
- **Tesselation ?**

Best Practice

- **Procedural methods often better than precomputed**

Best Practice

- Procedural methods often better than precomputed
- Don't use memory intense data structures like framebuffers, 3D textures, etc. if not absolutely necessary

Best Practice

- Procedural methods often better than precomputed
- Don't use memory intense data structures like framebuffers, 3D textures, etc. if not absolutely necessary
- Don't use double, reduce precision or use fix-point if possible

Best Practice

- **Procedural methods often better than precomputed**
- **Don't use memory intense data structures like framebuffers, 3D textures, etc. if not absolutely necessary**
- **Don't use double, reduce precision or use fix-point if possible**
- **Use ifs only to avoid memory accesses or heavy computations**
    - Might however be less useful when core count increases

Best Practice

- **Avoid irregular access of textures or data arrays in the shader as well as loops based on variables**
  - Rule of thumb: uniforms can be ok, varyings sometimes, others are worse

Best Practice

- **Avoid irregular access of textures or data arrays in the shader as well as loops based on variables**
  - Rule of thumb: uniforms can be ok, varyings sometimes, others are worse
- **Use texture compression, it's (mostly) free!**
  - typically: 1:6 compression, 30 dB
  - supported in hardware
  - different standards, best bet: DXT
  - caution however if using textures for other things than images

Best Practice

- **Avoid irregular access of textures or data arrays in the shader as well as loops based on variables**
  - Rule of thumb: uniforms can be ok, varyings sometimes, others are worse
- **Use texture compression, it's (mostly) free!**
  - typically: 1:6 compression, 30 dB
  - supported in hardware
  - different standards, best bet: DXT
  - caution however if using textures for other things than images
- **Biggest bottlenecks: overdraw, texture accesses**

Conclusion

- **Smartphones/Tablets: big and still fast growing market**
  - Directly linked to SOC architecture
    => unlikely to change
  - Memory Access: expensive, will only get worse
- **OpenGL ES: streamlined OpenGL designed for these systems**
- **Biggest challenge: minimize memory footprint**
  - But be aware: you can break the rules...
  - ...just as long as you know what you are doing

Questions?

Thank you very much!

www.icg.isy.liu.se