


Digital Signal Processors
Winter Term 2014/15

Graphics Processing Units - GPUs

 **ALPEN-ADRIA
UNIVERSITÄT**
KLAGENFURT | WIEN GRAZ

FAKULTÄT FÜR TECHNISCHE WISSENSCHAFTEN
Institute for Networked and Embedded
Systems

Sergei N. Bauer
sergei.bauer@k-ai.at

Graphics Processing Unit – History 1

- Before 1980s no dedicated graphics HW in PCs
- All processing was done in SW by CPU
- 1970s: Earliest HW in Arcade system boards
 - Fujitsu MB 14241 (60 sprites, RGB)
 - *Gun Fight* (1975), *Space Invaders* (1978)
- 1980s: foundations for the future
 - NEC μ PD7220 GDC (1981)
 - Drawing lines, circles and arcs to bm display
 - First IC containing many ten-thousand transistors (LSI)
 - Intel iSBX 275 (1982)
 - industrial systems, multibus standard
 - Commodore Amiga GPU (1985)
 - Co-processor, prim. inst. set without CPU
 - Unusual for the time (all drawing in CPU)



Graphics Processing Unit – History 2

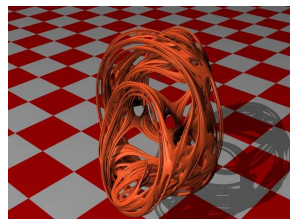
- 1980s: continued
 - Texas Instruments TMS34010
 - First uC with on-chip graphics
 - Both general purpose but graphics oriented inst. Set
 - Led to „TIGA“ (Windows accelerator cards)
- 1990s: expand and accelerate
 - S3 Graphics 86C911 (1991)
 - 2D acceleration (many imitators)
 - API evolve: WinG (*Win 3.x*), DirectDraw (*Win95*)
 - OpenGL perf. issues
 - Nvidia GeForce 256 (1999)
 - Pixel shader
 - Vertex shader



3

Graphics Processing Unit – History 3

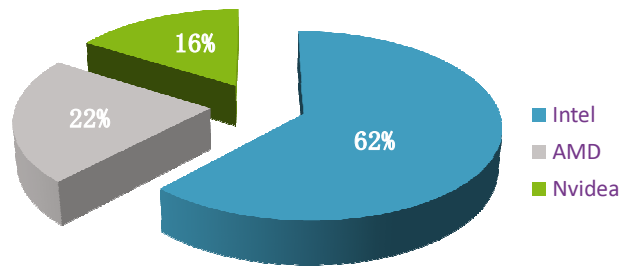
- 2000 - 2006:
 - OpenGL advances (model DirectX)
 - Add shading to capabilities (Nvidia)
 - Floating point math, as flexible as CPUs
 - Faster for image-array operations
- 2006 - present:
 - Parallel GPUs
 - GPGPU (general purpose computing)
 - OpenCL supported by Intel, AMD, Nvidia (*Heterogeneous PUs*)



4

Present GPU Market Allocation

Global Market Share 3rd Quartal 2013

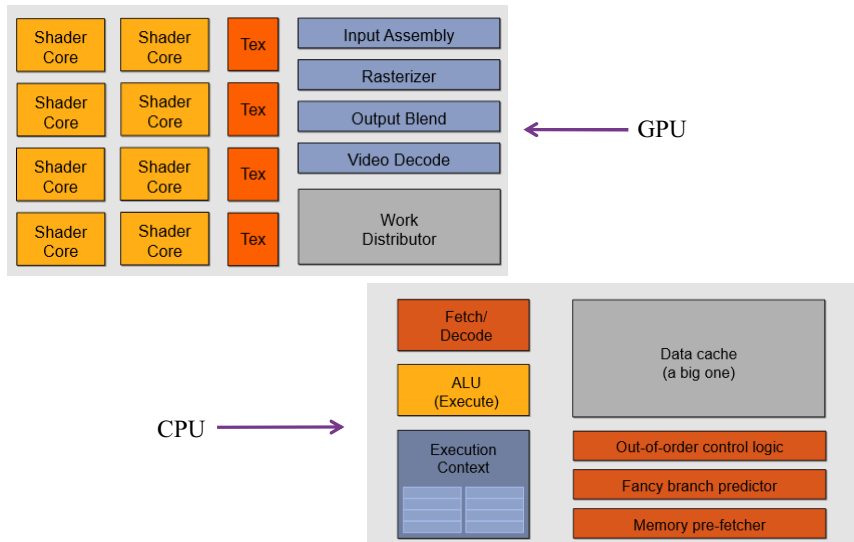


- large Intel market share due to on-Chip solutions

5


CPU vs. GPU

- Heterogeneous multi-processor chip (tuned for graphics)



Compiling Graphics Code

- Picture split into „fragments“
- Fragments processed independantly (no explicit parallelism)


1 unshaded fragment input record 

Shader code example:

```
sampler mySamp;
Texture2D<float3> myTex;
float3 lightDir;

float4 diffuseShader(float3 norm, float2 uv)
{
    float3 kd;
    kd = myTex.Sample(mySamp, uv);
    kd *= clamp(dot(lightDir, norm), 0.0, 1.0);
    return float4(kd, 1.0);
}
```

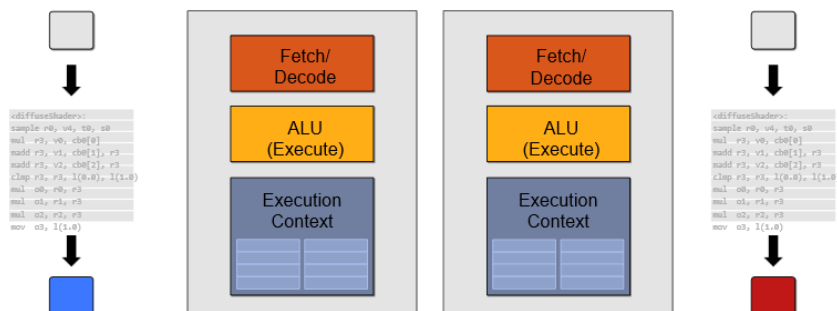
```
<diffuseShader>:
sample r0, v4, t0, s0
mul r3, v0, cb0[0]
madd r3, v1, cb0[1], r3
madd r3, v2, cb0[2], r3
clmp r3, r3, 1(0.0), 1(1.0)
mul o0, r0, r3
mul o1, r1, r3
mul o2, r2, r3
mov o3, 1(1.0)
```

1 shaded fragment output record 

7

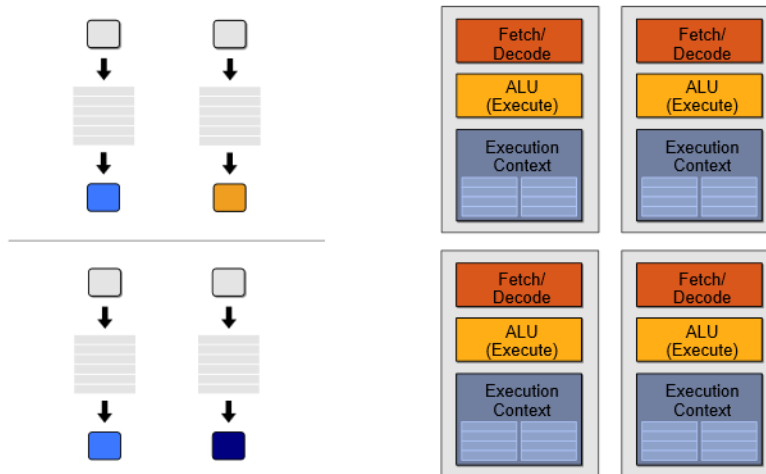
Idea: Utilization of Two Cores

- Two fragments processed parallel to each other



8

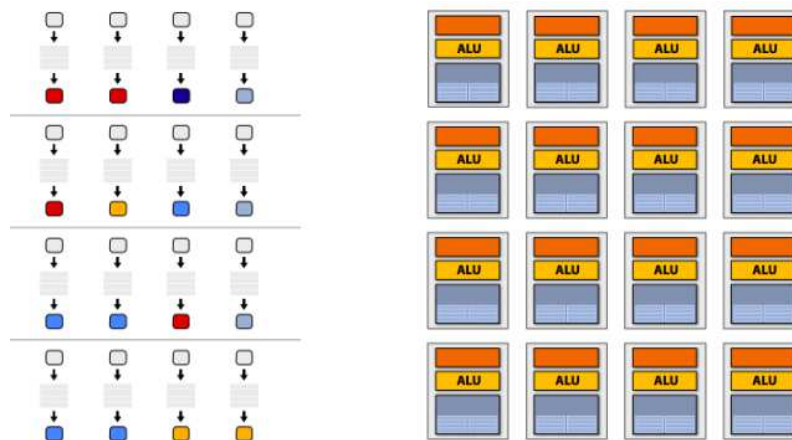
Idea: Utilization of Four Cores



9

Idea: Utilization of Sixteen Cores

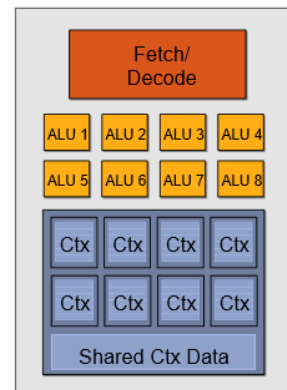
- 16 cores => 16 simultaneous inst. streams



10

Idea: Add ALUs

- Improving the cost to complexity ratio by using one instruction stream across many ALUs
- SIMD approach:
 - Single Instruction
 - Multiple Data



11

Idea: Modify the Shader

```
<diffuseShader>:
sample r0, v4, t0, s0
mul r3, v0, cb0[0]
madd r3, v1, cb0[1], r3
madd r3, v2, cb0[2], r3
clmp r3, r3, 1(0.0), 1(1.0)
mul o0, r0, r3
mul o1, r1, r3
mul o2, r2, r3
mov o3, 1(1.0)
```



```
<VEC8_diffuseShader>:
VEC8_sample vec_r0, vec_v4, t0, vec_s0
VEC8_mul vec_r3, vec_v0, cb0[0]
VEC8_madd vec_r3, vec_v1, cb0[1], vec_r3
VEC8_madd vec_r3, vec_v2, cb0[2], vec_r3
VEC8_clmp vec_r3, vec_r3, 1(0.0), 1(1.0)
VEC8_mul vec_o0, vec_r0, vec_r3
VEC8_mul vec_o1, vec_r1, vec_r3
VEC8_mul vec_o2, vec_r2, vec_r3
VEC8_mov o3, 1(1.0)
```

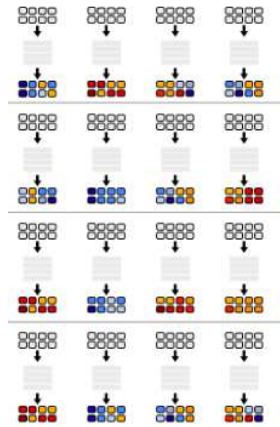
- Processes one fragment
- Scalar operations
- Scalar registers

- Processes eight fragments
- Vector operations
- Scalar registers

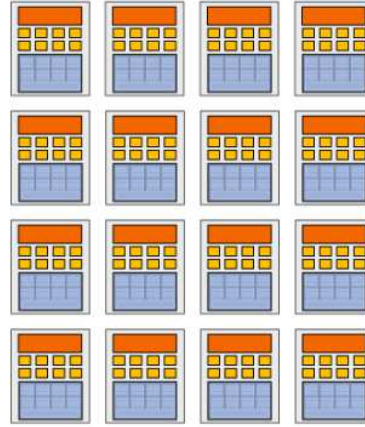
12

Massive Improvement Processing Tiny Tasks

- 128 (!) fragments in parallel



- 16 cores utilizing 128 ALUs

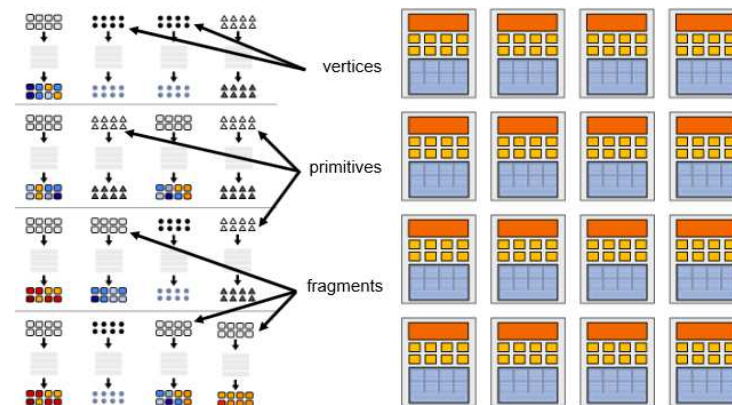


- 16 simultaneous instruction sets

13

Massive Improvement Processing Tiny Tasks

- 128 vertices/fragments/primitives (i.e.: OpenCL work items)

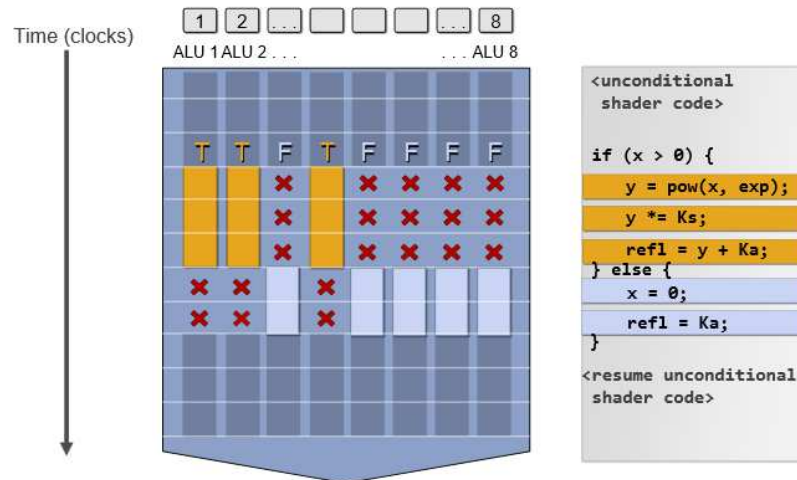


- Primitives - simplest instructions of a prog. lang
- Vertices – intersection of each optical surface with the optical axis

14

Branch Management

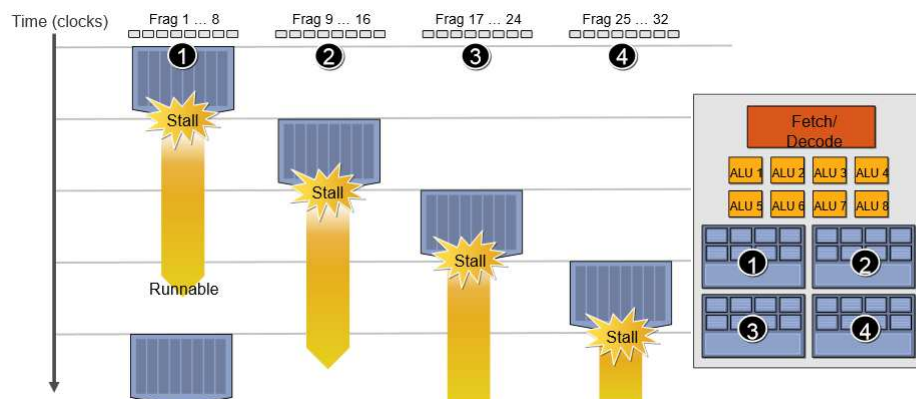
- Not all ALUs permanently doing work
- In the Worst case only 1 of eight is processing



15

Dependency Issues 1

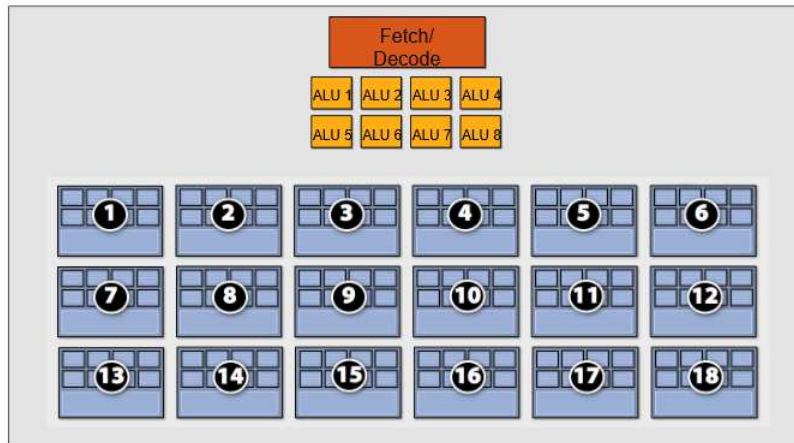
- If a core unable to run next instruction due to dependancy => STALL!
- Special caches and “fancy” logic of CPU removed now is a benefit (less stalls)
- Apply interleaving of processes to avoid stalls



16

Dependency Issues 2

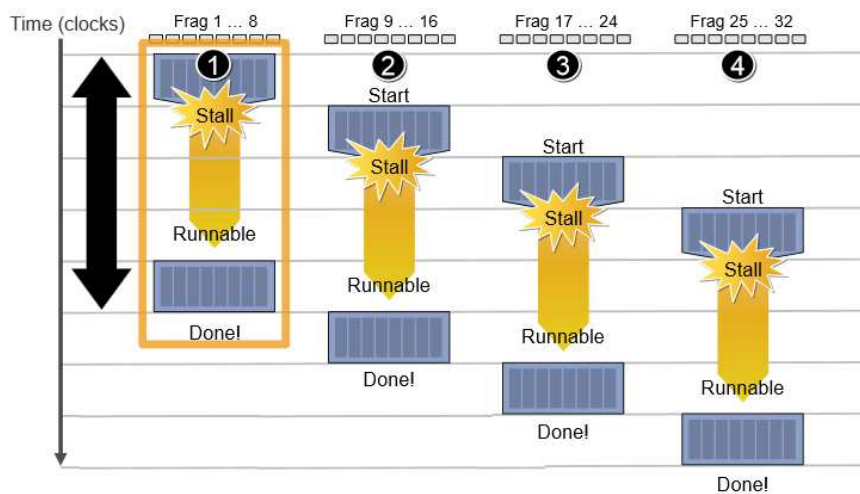
- Increase of latency hiding when context partitioned into smaller parts



17

Dependency Issues 3

- Increase runtime of one group increases the throughput of many groups
- Interleaving between contexts achieved by HW or SW or both (!)



18

Dependency Issues 4

- Interleaving between contexts achieved by HW or SW or both (!)
- AMD Radeon or Nvidia:
 - HW scheduling and on-chip storage for fragment states
- Intel Larrabee:
 - HW managing four x86 contexts with fine graining
 - SW scheduling interleaves many fragment groups on each HW context
 - L1-L2 cache hold fragment state

Example GPU:

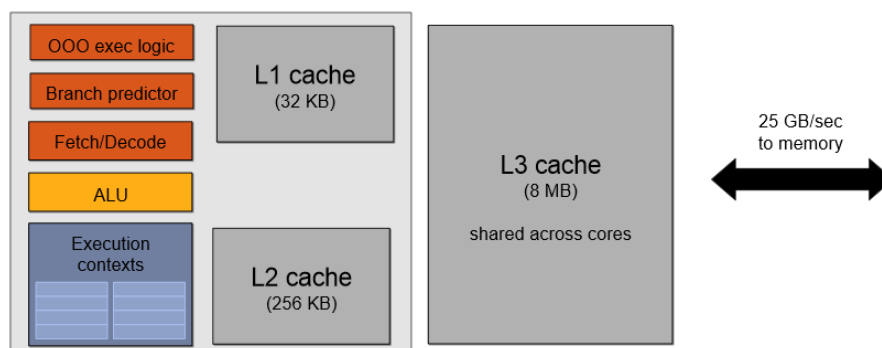
- 16 cores
 - 8 ALUs per core
 - 16 simul. inst. streams
 - 64 concurrent inst. Streams
 - 512 concurrent fragments
- This yields: 256 GFLOPS ($f = 1\text{GHz}$)



19

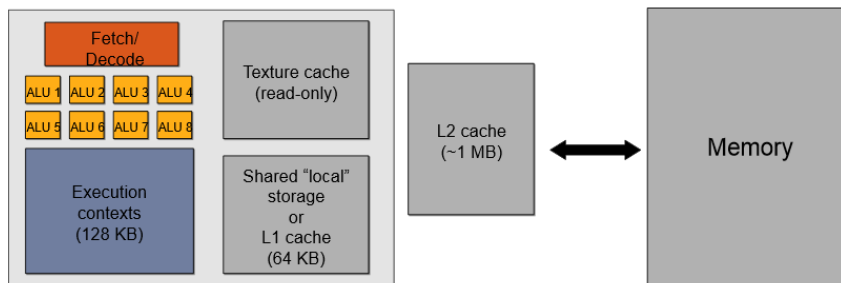
Memory Management in a CPU

- CPU core runs most efficiently when data “lives” in cache (reduce latency)



20

Modern Memory Management in a GPU



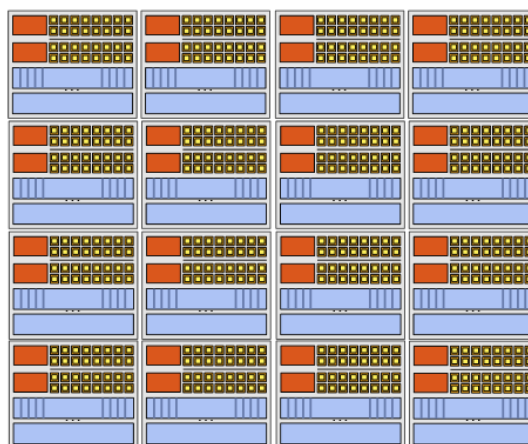
- On-chip storage takes load off memory system
- Requirement for high-bandwidth connection to memory
 - Over **20** (!) times the performance of a quad-core CPU
 - Repack/reorder/interleave memory to maximize the use of mem. Bus
 - Still this is only **6** times the bw. Available to a CPU

Solutions:

- Request data less often (do more calculations instead)
- Fetch data from memory less often (share and re-use data among fragments)

21

Nvidia GeForce GTX 580 SM



- 16 such units on the GTX 580 SM
- Resulting in 24.500 fragments or 24.000 CUDA threads

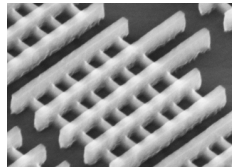
22

Heterogeneous Cores a Future Perspective

- Integration of CPU and GPU style cores closer together
 - Reduces memory overhead (copies/transfers)
 - Declines offload cost
 - Results in better Power management

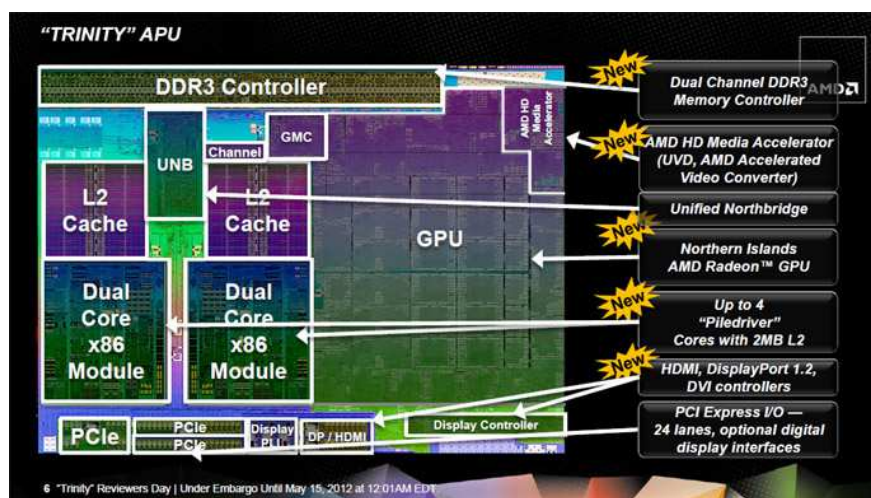


- Developments in this Direction:
 - AMD Fusion APUs
 - Nvidia Tegra X1
 - Intel Haswell and Broadwell
 - Qualcomm Snapdragon 810
 - Apple A7 and A8
 - ...



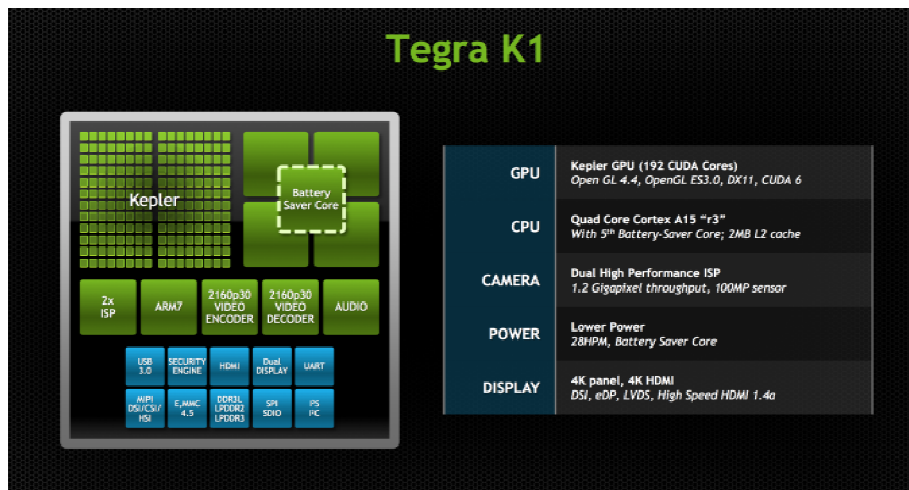
23

AMD Fusion APU



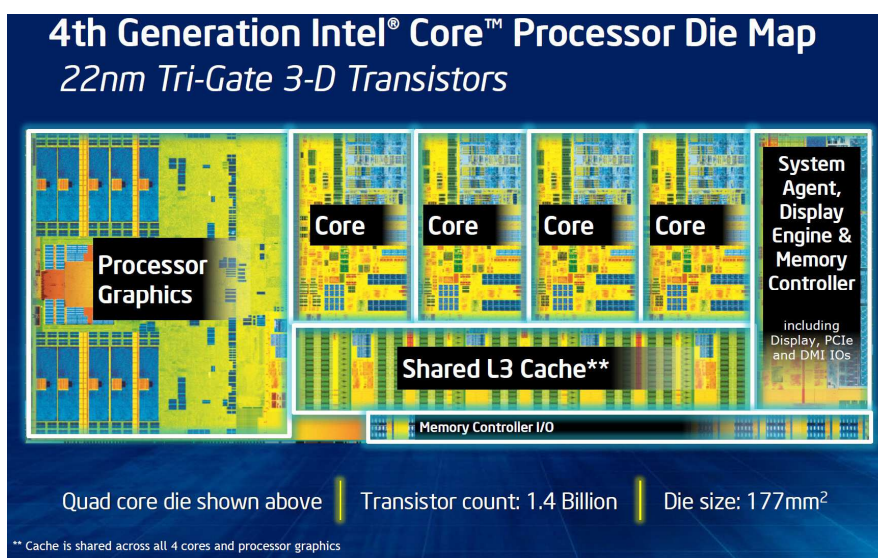
24

Nvidia Tegra K1

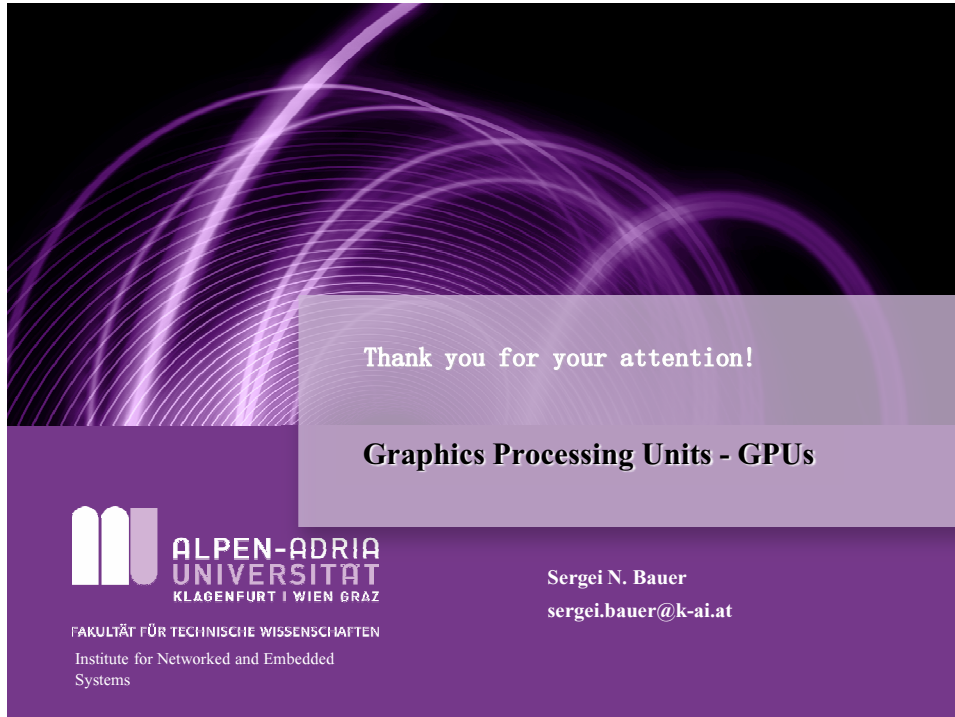


25

Intel Haswell




26



Thank you for your attention!

Graphics Processing Units - GPUs

 **ALPEN-ADRIA
UNIVERSITÄT**
KLAGENFURT | WIEN GRAZ

FAKULTÄT FÜR TECHNISCHE WISSENSCHAFTEN
Institute for Networked and Embedded
Systems

Sergei N. Bauer
sergei.bauer@k-ai.at