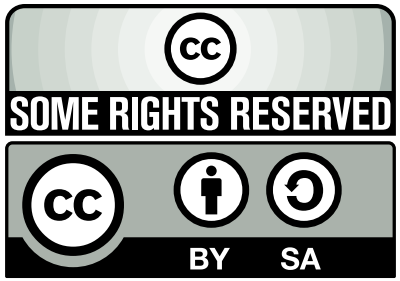
GPGPU and Stream Computing

Julian Fietkau

University of Hamburg

June 30th, 2011

Things to clear up beforehand...



Sources for the numbered figures are in the →**list of figures**.
Non- numbered pictures and illustrations are from the
OpenClipArt Project or are based on content from there.

Download these slides and give feedback:

http://www.julian-fietkau.de/gpgpu_and_stream_computing

Agenda

Flynn's Taxonomy

SISD	MISD
SIMD	MIMD

Why Does It Exist?

- How long can Moore's law hold true? $\rightarrow$ parallelism as a possible answer to computational demands
- "swiss army knife" (generally optimal solution) for parallel programming has not been found
- idea: exploit consumer-grade graphics hardware

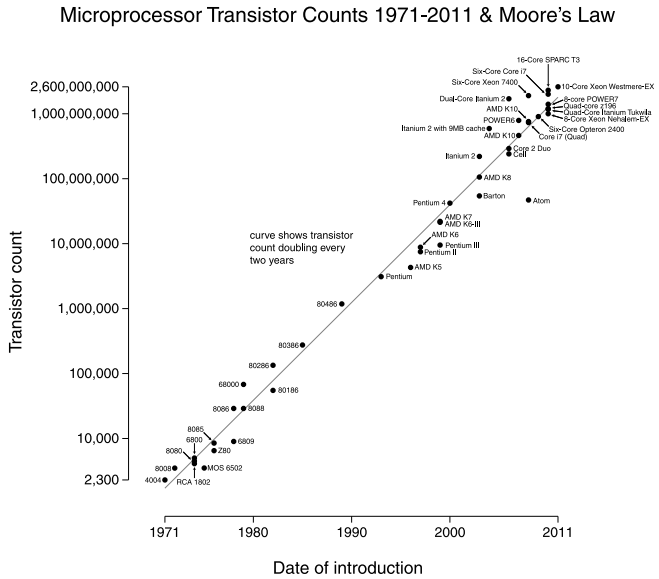
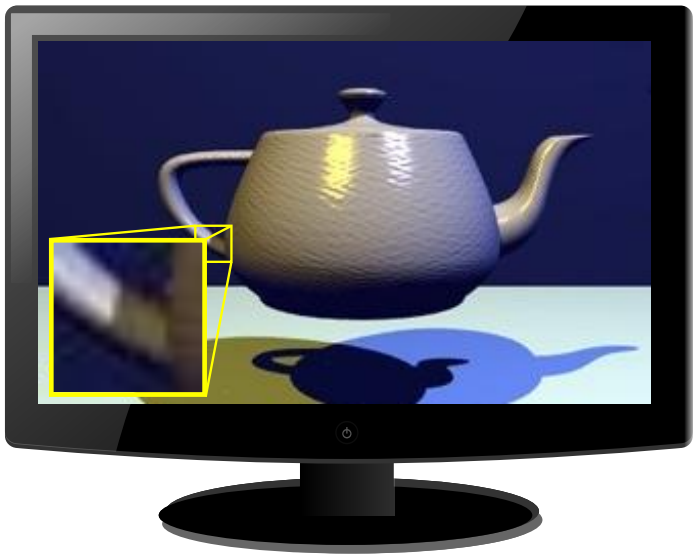


Figure 1: Moore's law – 2011

About Graphics Hardware

- games need to display increasingly realistic objects/scenes in real time
- need to calculate a lot of vertices and a lot of pixels very quickly → Pixel/Vertex Shaders, later Unified Shader Model
- consumer market ensures that graphics adapters remain (relatively) cheap
- **G**eneral **P**urpose computation on **G**raphics **P**rocessing **U**nits



Stream Computing

- idea: operate on a "stream" of data passing through different "kernels"
- related to SIMD
- mitigates some of the difficulties of parallelism on von Neumann architectures as well as simple SIMD implementations like SSE or AltiVec
- first came up in the 70ies, didn't gain much traction as "pure" implementations, but hybrid architectures survived

Stream Computing Example

```
Input:  u, v, w;

x = u - (v + w);
y = u * (v + w);

Output:  x, y;
```

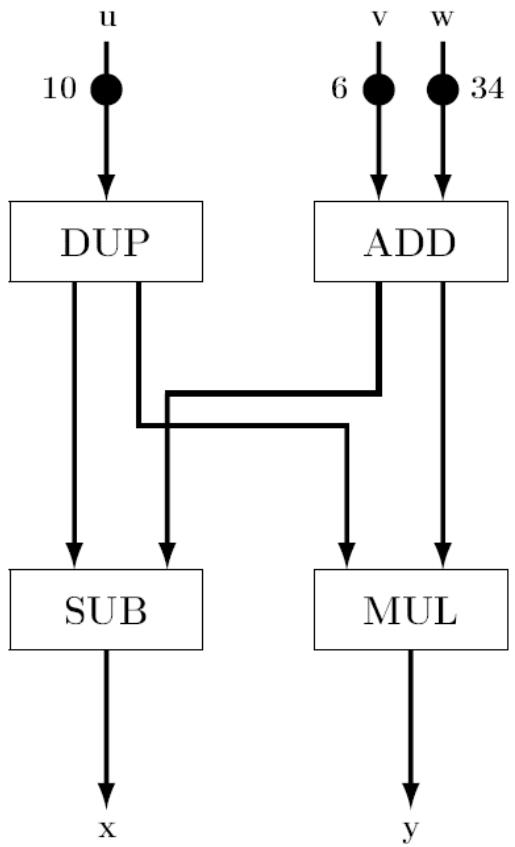


Figure 2: Stream Computing Example

Common Ideas

modern streaming programming languages...

- ...are verbose about different usage scenarios for memory
- ...help with partitioning problem spaces in a multitude of ways
- ...are not afraid to introduce limitations to faciliate optimization

OpenCL™

- **Open C**omputing **L**anguage, free standard by Khronos™ Group

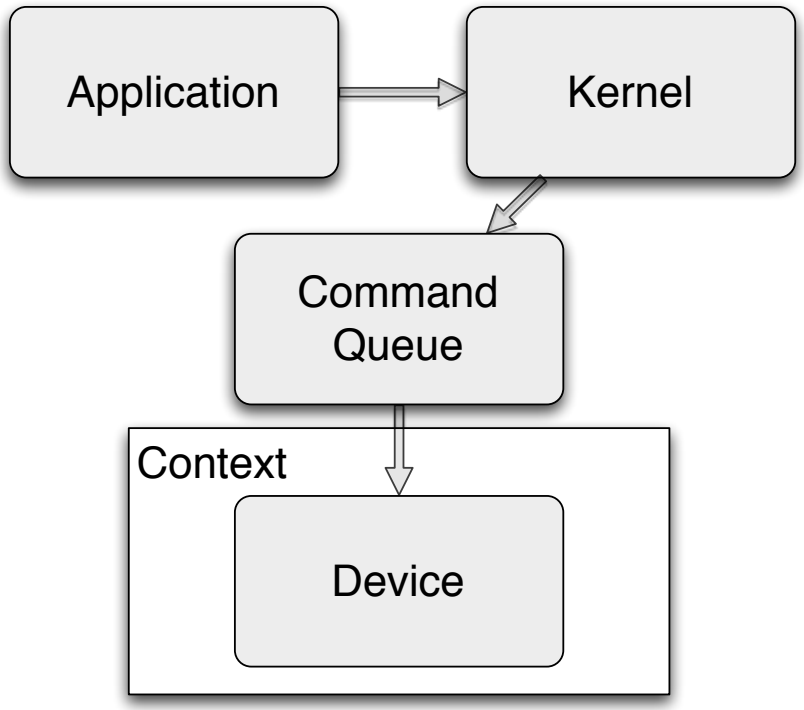


Figure 3: OpenCL™ Application Model

OpenCL™ in Detail

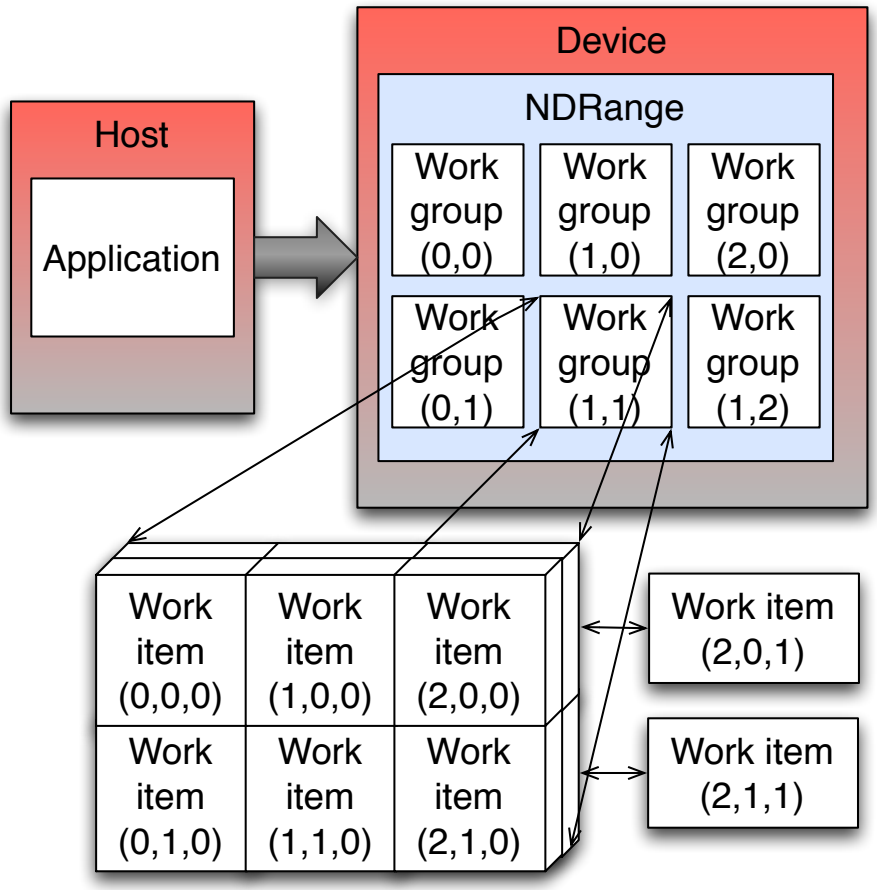


Figure 4: OpenCL™ Problem Partitioning

CUDA

- NVIDIA's custom framework for high-level GPGPU
  - (it's actually older than OpenCL though)
- same basic idea, but specific to NVIDIA GPUs
- conceptually only minor differences between CUDA and OpenCL
  - biggest one: CUDA is compiled at application compile time while OpenCL is (typically) compiled at application run time
  - also, annoying nomenclature differences (e.g. *shared* vs. *local* vs. *private* memory)

Others

There are several more stream processing languages, some of them long in development. Notable:

- *Brook* (and *Brook+*)
- *Cilk*, compare also *Intel Array Building Blocks*

Intermediary Languages

Problem

The actual binary code that runs on devices needs to "know" about exact numbers for cores, memory, registers etc., information that is generally not known at compile time.

$\rightarrow$ compilation to an **intermediary language** like NVIDIA's *PTX* and AMD's *IL*, low-level and assembly-like yet abstracting some hardware limitations

PTX and AMD IL

PTX example

```
        .reg     .b32 r1, r2;
        .global  .f32 array[N];

start: mov.b32   r1, %tid.x;
       shl.b32   r1, r1, 2;          // shift thread id by 2 bits
       ld.global.b32 r2, array[r1]; // thread[tid] gets array[tid]
       add.f32   r2, r2, 0.5;        // add 1/2
```

AMD IL example

```
sample_resource(0)_sampler(0) r0.x, v0.xy00
mov r2.x, r0.xxxx
dcl_output_generic o0
ret
```

Programmability

- as they're mostly custom versions of C, GPGPU languages are rather simple to pick up for someone with C experience
- OpenCL™ and CUDA both look slightly boilerplate-y for small tasks
  - hypothesis: they might not be *designed* for small tasks
- disadvantage of the cutting edge: toolchain maturity might be lacking
- watch out for vendor dependencies!

Efficiency

- hard to find actual data
- optimizations and proficiency might skew the results
- conceptual similarities indicate that implementations would also be similar
- CUDA can get a (constant) head start vs. OpenCL™ due to being precompiled
- **CUDA might generally perform faster**, sometimes significantly, than OpenCL (but take this with a grain of salt)

Things to Come

The future remains notoriously hard to predict.

- at the moment, we see increased interest in specialized GPGPU boards (cf. *NVIDIA Tesla* and *AMD FireStream*)
- OpenCL promotes device flexibility at the cost of efficiency – no way to know if this strategy will win
- Intel pushes for integrated solutions with more processing power (cf. *Sandy Bridge*, *Ivy Bridge*)

Conclusion

- GPGPU is a viable way to to massively parallel work even on a home PC
- will be further developed and refined, knowledge may be valuable

Weblinks

AMD Developer Central: Introduction to OpenCL™ Programming

http://developer.amd.com/zones/openclzone/...-may-2010.aspx

GPGPU: OpenCL™ (Università di Catania)

http://www.dmi.unict.it/~bilotta/gpgpu/notes/11-opencl.html

NVIDIA: PTX ISA Version 2.1

http://developer.download.nvidia.com/compute/.../ptx_isa_2.1.pdf

AMD: High Level Programming for GPGPU

http://coachk.cs.ucf.edu/courses/CDA6938/s08/AMD_IL.pdf

List of figures