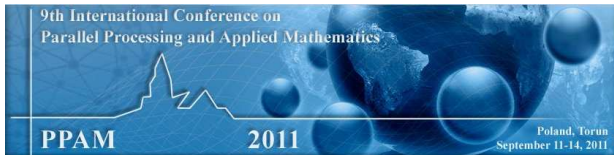


# Scientific Computing on GPUs: GPU Architecture Overview

Dominik Göddeke, Jakub Kurzak, Jan-Philipp Weiß,  
André Heidekrüger and Tim Schröder



PPAM 2011 Tutorial  
Toruń, Poland, September 11  
<http://gpgpu.org/ppam11>

# Overview

---

## Goals of this talk

- Introduce GPU hardware on an *abstract* level, as a *mental model*
- Establish intuition which workloads may benefit the GPU architecture to which extend
- Obtain architectural understanding required to write and optimise code for GPUs
- Explicit non-goal: computer architecture, electrical engineering

## Overview

- Three major ideas that make GPUs fast
  - Simplify and multiply
  - Share instruction stream: wide-SIMD
  - Hide latencies by interleaving
- Instantiations of these ideas: NVIDIA Fermi and AMD Cypress

# Acknowledgements

---

## Kayvon Fatahalian (Stanford / Carnegie Mellon University)

- Most of this talk is based on, and even taken directly from, his 'classic' talk  
*From Shader Code to a Teraflop: How GPU Shader Cores Work*  
presented at various SIGGRAPH conferences since 2008
- Used with kind permission

## See also

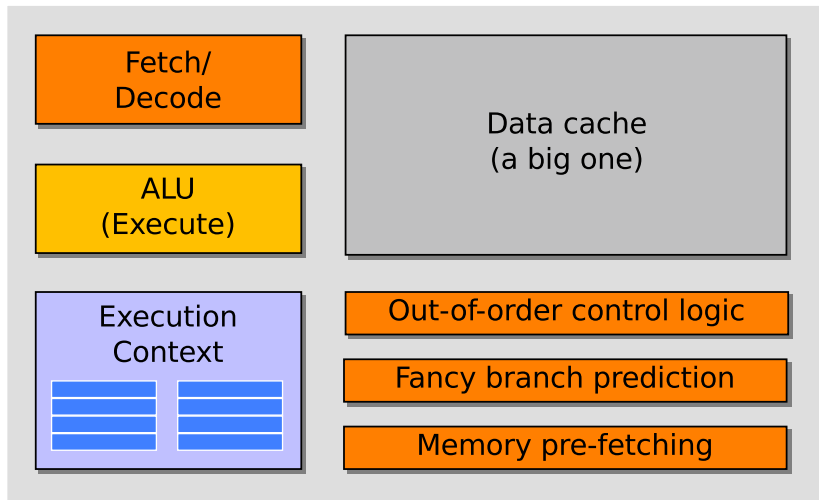
- <http://graphics.stanford.edu/~kayvonf/>
- K. Fatahalian and M. Houston: *A Closer Look at GPUs*, Communications of the ACM. Vol. 51, No. 10 (October 2008)
- M. Garland and D. B. Kirk: *Understanding throughput-oriented architectures*, Communications of the ACM. Vol. 53, No. 11 (November 2008)

# Throughput Computing

Three key design ideas

# CPU-Style Cores

---

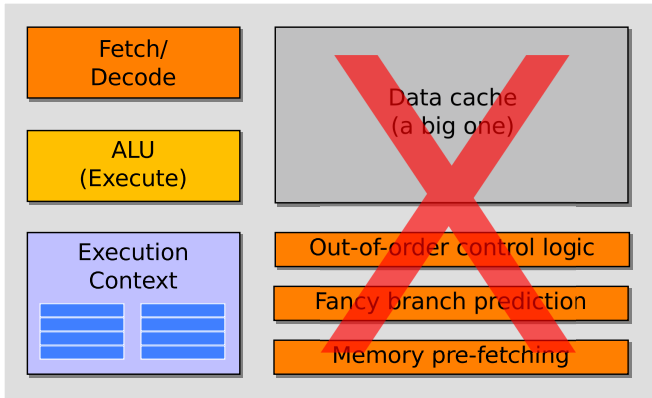


# Idea 1: Simplification

---

## Remove everything that makes a single instruction stream run fast

- Caches (50% of die area in typical CPUs!)
- Hard-wired logic: out-of-order execution, branch prediction, memory pre-fetching

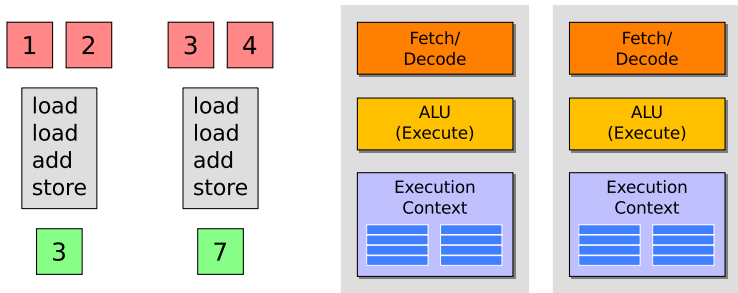


# Consequence: Use Many Simple Cores in Parallel

---

## Invest saved transistors into more copies of the simple core

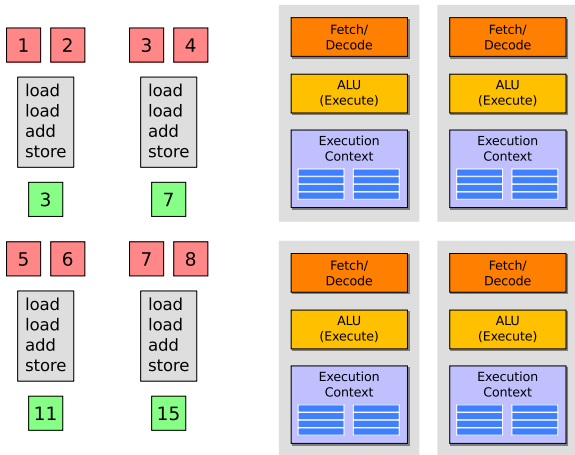
- 'More copies': More than a conventional multicore CPU could afford



# Consequence: Use Many Simple Cores in Parallel

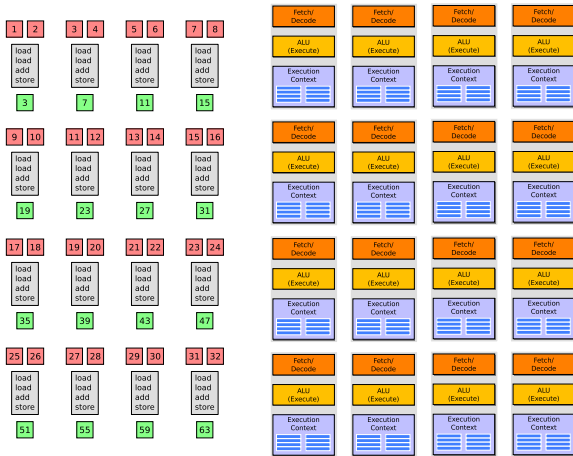
---

Invest saved transistors into more copies of the simple core



# Consequence: Use Many Simple Cores in Parallel

Invest saved transistors into more copies of the simple core



# Instruction Stream Sharing

---

## Observations

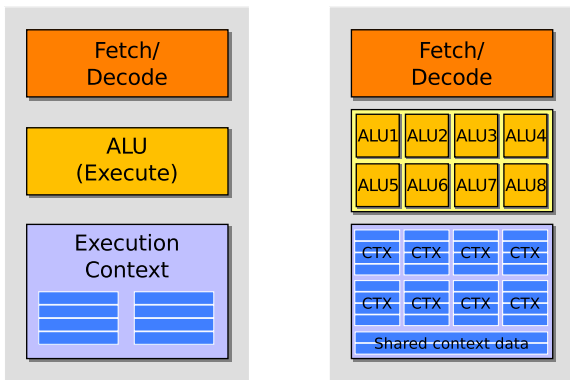
- The same operation, e.g. some lighting effect or geometry transformation, is typically performed on many different data items
- GPGPU: Many operations exhibit the same ample 'natural' parallelism: vector addition, sparse matrix vector multiply, ...
- This is called *data parallelism*
- Why run many *identical* instructions streams simultaneously?

## Idea 2: SIMD Processing

- GPU design assumption: Expect *only* data-parallel workloads and exploit this to maximum extent in the chip design
- Amortise cost/complexity of managing an instruction stream across many ALUs to reduce overhead (ALUs are very cheap!)
- Increase ratio of peak useful flops vs. total transistors

# Instruction Stream Sharing

---



**16 8-wide cores → 128 operations in parallel**

- Still possible: 16 independent (but 8-wide each) instruction streams

# Problems with SIMD

---

## SIMD processing implies SIMD memory access

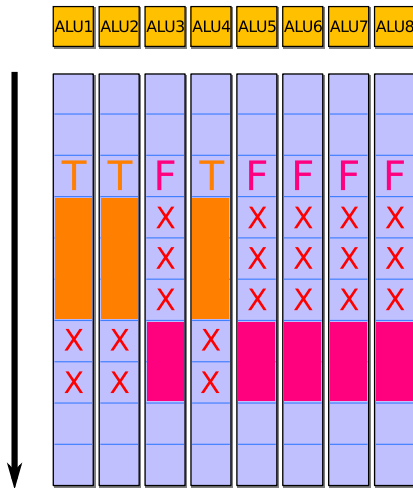
- Details in the talks by Tim later (Session 3)

## Branch divergence

- Resolution: Masking of ALUs = Serialisation of conditionals
- Implementation in hardware incurs small overhead
- Example (assuming identical workload in if and else branch):
  - Phase 1: 3 of 8 ALUs treat if-branch, 5 are idle
  - Phase 2: 5 ALUs treat else branch, 3 are idle
  - Cost: 2x compared to no branch, plus small HW overhead
- Consequence: Not all ALUs do useful work, 1/8 of peak performance in the worst case
- But: Common practice to force *few* ALUs to idle, e.g. for non-multiple-of-8 workloads (empty else branch)

# Branch Serialisation

---



# SIMD Processing $\neq$ SIMD Instructions

---

## Option 1: Explicit vector instructions

- Conventional x86 SSE and AVX
- PowerPC AltiVec
- Intel Knights Ferry/Corner ('Many Integrated Core', Larrabee)
- Typical SIMD width: 2–8

## Option 2: Scalar instructions, implicit HW vectorisation

- More programmer-friendly at the expense of more *implicit* knowledge
- HW determines instruction stream sharing across ALUs
- Amount of sharing deliberately hidden from software
- Contemporary GPU designs (slides+=5)
  - NVIDIA GPUs: SIMT warps
  - AMD GPUs: VLIW wavefronts
- Typical SIMD width: 16–64

# Improving Throughput

---

## Stalls: Delays due to dependencies in the instruction stream

- Example: ADD operands depend on completed LOAD instruction
- Latency: Accessing data from memory easily takes 1 000+ cycles
- Simplification: Fancy caches and logic that helps to hide stalls have been removed in idea # 1. Bad idea?

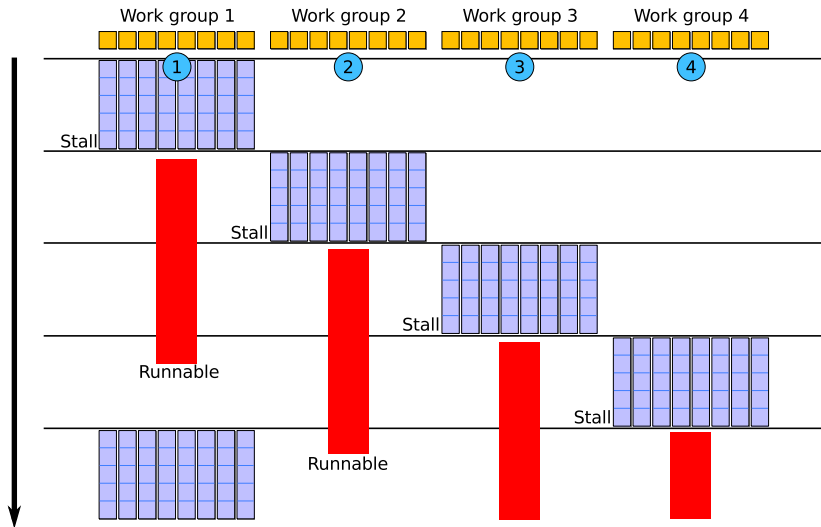
## But

- GPUs assume LOTS of independent work (independent SIMD 'groups')

## Idea 3: Interleave processing of many work groups on a single core

- Switch to instruction stream of another (non-stalled = ready) SIMD group in case currently active group stalls
- GPUs manage this in HW, overhead-free (!)
- Ideally, latency is fully hidden, throughput is maximised

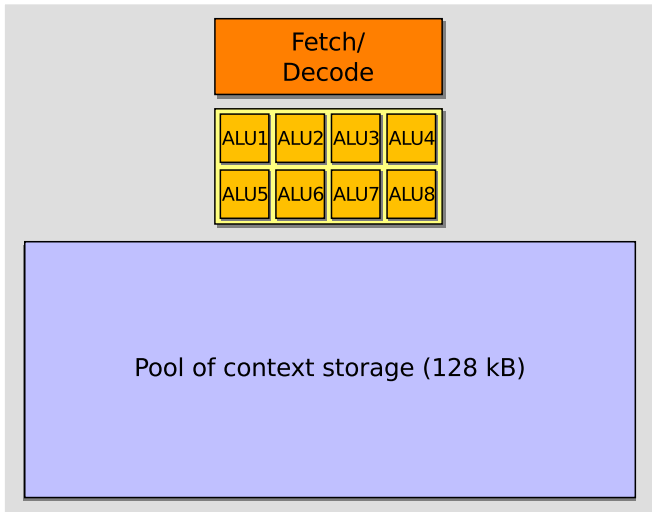
# Latency Hiding



# Storing Contexts

---

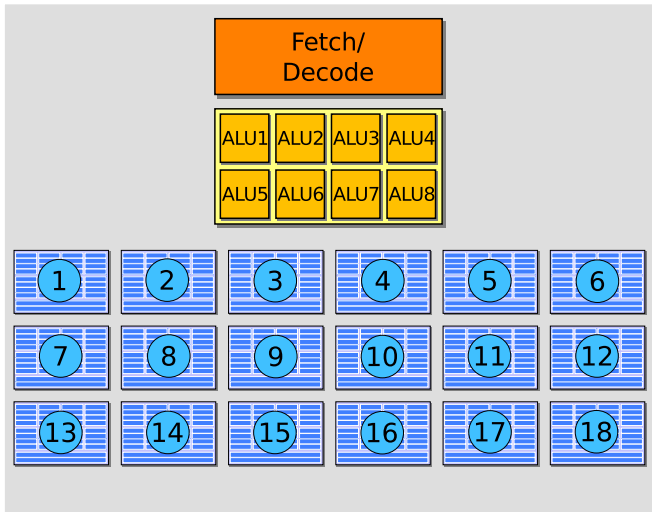
## Register file and shared context storage



# Storing Contexts

---

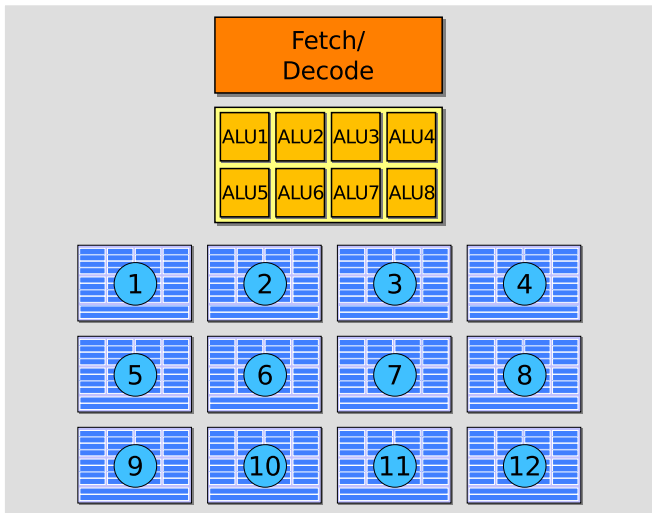
**18 small contexts: High latency hiding**



# Storing Contexts

---

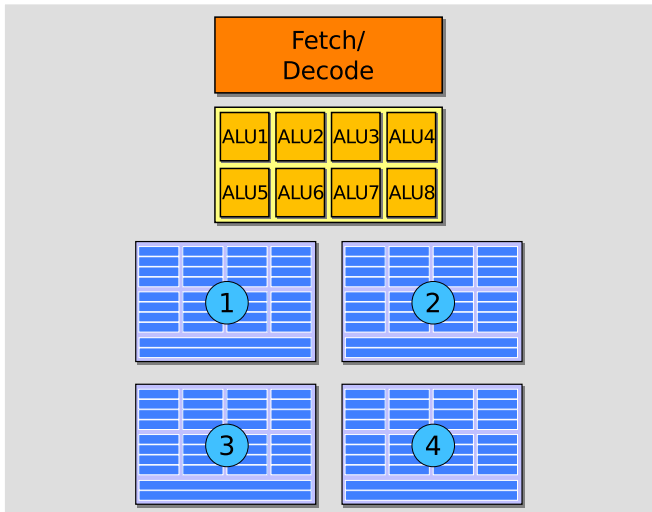
## 12 medium contexts



# Storing Contexts

---

## 4 large contexts: Low latency hiding



# Summary

---

## Three key ideas to maximise 'compute density'

- Use many simplified cores to run in parallel
- Pack cores full of ALUs
  - By sharing instruction stream across groups of work items (SIMD)
- Avoid latency stalls by interleaving execution of many *groups* and overhead-free HW context switch
- Benefit: *Scalable* design for all price/power/performance regimes

## Exposing the HW to programmers

- OpenCL / CUDA programming model 'virtualises' the concept of interleaved SIMD groups and introduces synchronisation hierarchy among groups

# Real GPUs

NVIDIA Fermi, AMD Cypress

# NVIDIA GTX 480 (Fermi)

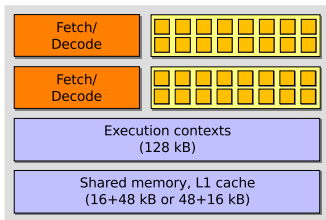
---

## NVIDIA speak

- 480 stream processors (CUDA cores)
- SIMT execution

## Generic speak

- 15 cores, 2 groups of 16 SIMD functional units per core



## Details of one core (streaming multiprocessor)

- Contains 32 functional units (CUDA cores)
- Two groups of 32 work items (warps) are selected in each clock
- Up to 48 warps are interleaved, up to 1536 contexts can be stored

# ATI Radeon HD 5870 (Cypress)

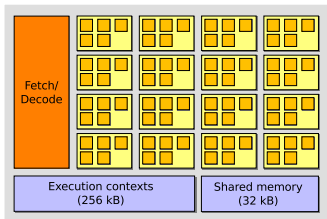
---

## AMD speak

- 1600 stream processors

## Generic speak

- 20 cores
- 16 'beefy' SIMD units per core
- 5-wide VLIW processing in each functional unit



## Details of one core (streaming multiprocessor)

- Groups of 64 (a wavefront) share an instruction stream of 5-wide VLIW instructions
- Four clocks to execute an instruction for entire group
- 20 of these SIMD engines on the full chip