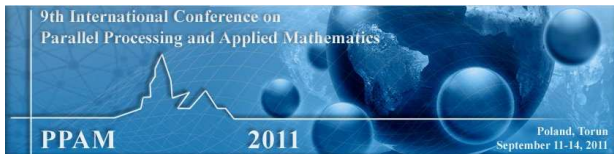


Scientific Computing on GPUs

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



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

The Big Picture

Hardware evolution

- Memory wall: Data movement cost prohibitively expensive
- Power wall: Nuclear power plant for each machine (in the cloud)?
- ILP wall: 'Automagic' maximum resource utilisation?
- Memory wall + power wall + ILP wall = brick wall

Inevitable paradigm shift: Parallelism and heterogeneity

- In a single chip: singlecore \rightarrow multicore, manycore, ...
- In a workstation (cluster node): NUMA, CPUs and GPUs, ...
- In a big cluster: different nodes, communication characteristics, ...

This is our problem as applied mathematicians

- Affects all machines we use, including workstations and laptops

Consequences for Numerics

Parallelism is inevitable

- Impossible to exploit ever increasing peak performance
- Sequential codes even run slower on newer hardware (!)

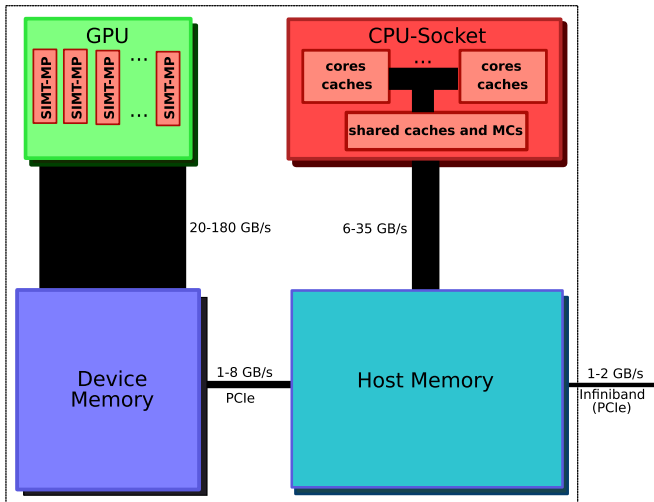
Challenges

- Technical: Compilers can't solve these problems, libraries are limited
- Numerical: Traditional methods often contrary to hardware trends
- Goal: Redesign existing numerical schemes (and invent new ones) to work well in the fine-grained parallel setting

GPUs ('manycore') are forerunners of this development

- 10 000s of simultaneously active threads
- Promises of significant speedups
- Focus of this tutorial

GPUs vs. CPUs



GPUs: Myth, Marketing and Reality

Raw marketing numbers (from industry and academia)

- > 2.5 TFLOP/s peak floating point performance
- Lots of papers claim $> 100\times$ speedup

Looking more closely

- Single or double precision? Same on both devices?
- Sequential CPU code vs. parallel GPU implementation?
- 'Standard operations' or many low-precision graphics constructs?

Reality

- GPUs are undoubtedly fast, but so are CPUs
- Quite often: CPU codes significantly less carefully tuned
- Anything between $5 - 30\times$ speedup is realistic (and worth the effort)

Using and Programming GPUs

Many possibilities

- OpenCL: open standard, platform- and vendor independent
- CUDA: NVIDIA-specific, more 'feature-rich' ecosystem
- Compiler support by PGI, rapidly and constantly growing body of libraries, commercial software, domain-specific environments, ...

Focus of this tutorial: OpenCL and CUDA

- Explicitly no pseudo-religious comparison, pragmatic approach
- Core languages semantically and syntactically almost identical
- Programming model and algorithmic way of thinking 100% identical
- Tuning strategies obviously hardware-dependent
- Here: OpenCL for language mechanics, both for examples

Tutorial Speakers

Organisers

- Dominik Göttsche, Institute of Applied Mathematics, TU Dortmund, Germany
- Jakub Kurzak, Innovative Computing Laboratory, University of Tennessee, Knoxville, USA
- Jan-Philipp Weiß, Engineering Mathematics and Computing Lab, Karlsruhe Institute of Technology, Germany

Industry speakers

- André Heidekrüger, AMD
- Tim Schröder, NVIDIA (unfortunately, he's ill today, so we'll do our best to improvise)

Tutorial Schedule: Session 1

11:00 am: Welcome and introduction

- This talk, 15 minutes
- Course material: <http://gpgpu.org/ppam11>

11:15 am: Ready-to-use GPU-accelerated mathematical libraries

- Jan-Philipp, 30 minutes
- Harness the power of GPUs quickly and effortlessly
- Focus on widely-used mathematical libraries and building blocks

11:45 am: GPU architecture

- Dominik, 30 minutes
- Build a mental model of how GPUs work in contrast to CPUs

12:15–1:00 pm: Coffee break

- Coffee being served downstairs

Session 2: Programming with OpenCL

1:00 pm: Introduction to OpenCL

- André, 45 minutes
- High-level introduction to the underlying abstract machine model, API design and core language features

1:45 pm: Practical OpenCL programming: Demo session

- Dominik, 30 minutes
- Walkthrough of simple code examples to get a feeling of how things work

2:15 pm: Short break, 15 minutes

Session 3: Performance Tuning and Advanced Programming

2:30 pm: The CUDA ecosystem

- Tim Jan-Philipp, 20 minutes
- Introduction to NVIDIA's GPU computing ecosystem

2:50 pm: Performance tuning for NVIDIA GPUs

- Tim Dominik, 25 minutes
- Focus on memory performance optimisations

3:15 pm: Performance tuning for AMD CPUs and GPUs

- André, 45 minutes
- Focus on different strategies for different architectures

4:00–4:30 pm: Coffee break

- Coffee being served downstairs

Session 4: Case Studies: Mathematical Building Blocks

4:30 pm: ASTRA – Automatic Stencil TuneR for Accelerators

- Jakub, 45 minutes
- Programming techniques and autotuning for dense linear algebra

5:15 pm: Sparse linear algebra and iterative solvers

- Jan-Philipp, 45 minutes
- Preconditioners and smoothers for Krylov subspace and multigrid solvers for PDEs discretised on structured and unstructured grids

6:00 pm: Summary and wrap-up

7:30 pm: Conference reception (downstairs)