Scientific Computing with GPUs
Part 1: Welcome and introduction

Dominik Göddeke and Robert Strzodka

Faculty of Mathematics
Applied Mathematics and Numerics
TU Dortmund
Germany
dominik.goeddeke@math.tu-dortmund.de

NVIDIA Corporation
Emerging Applications
Zurich
Switzerland
rstrzodka@nvidia.com

PPAM 2013
Warsaw, Poland
September 8, 2013
http://gpgpu.org/ppam2013
The big picture

**Hardware evolution**
- Memory wall: data movement cost prohibitively expensive
- Power wall: nuclear power plant next to each big machine, but also: data movement more energy-hungry than computation
- ILP wall: ‘automagic’ maximum resource utilisation
- Memory wall + power wall + ILP wall = brick wall

**Consequences**
- Hardware solution: parallelism and heterogeneity
- In a single chip: singlecore → multicore, manycore, ...
- In a workstation (cluster node): NUMA, CPUs, GPUs, MICs, ...
- In a big cluster: different nodes, communication characteristics, ...
- Affects all levels of scientific computing, from smartphones to supercomputers
Sequential execution is a myth

Some fun facts for the x86 architecture

- Memory accesses are parallel since i386 (cache lines)
- Computation is parallel since Pentium MMX (SIMD registers)
- ... and this happens in a single chip already (!)

Hardware today

- Smartphones and tablets
  - 4–8 ARM cores, 1–4 (recently programmable) GPU cores
- Laptops, workstations and cluster nodes
  - 1–4 CPU sockets with up to 16 cores each, NUMA
  - 0–4 accelerators (GPUs, MICs), discrete memory
- Clusters and big machines
  - Heterogeneous or homogeneous nodes
  - Distributed memory, interconnects
Challenges

Technical challenges

- No unified programming environment
- Different languages for different architectures

Algorithmic and methodological challenges

- Common case for textbook methods: sequential or coarse-grained distributed memory parallelism
- Data structures and data access patterns not suited for vectorisation
- Fine-grained scalability (SIMD width) often no concern

The ultimate goal

- Design algorithms and devise methodologies that address these challenges in a scalable way
Enter GPUs

Why GPUs?

- Hardware *and* programming model for parallel computing
- Performance stems from HW being able to maintain many more threads ‘in flight’ than x86: fine-grained parallelism
- Good potential for improvements in time and energy to solution

GPUs are now established in scientific computing

- GPU support in many commercial software packages
- Ready-to-use libraries in many fields
- Abstractions and wrappers (e.g., python, .NET)
- Mature toolchains for developers
Enter GPUs

Transition in the GPU community
- Using GPUs is easy now, getting things implemented efficiently can be challenging

Focus of this tutorial
- NVIDIA CUDA ecosystem
- Most mature of all environments
- Note: OpenCL is not performance-portable

Target audience
- Session 1: beginner-level, plug-in and minimally-invasive acceleration
- Sessions 2+3: advanced performance optimisation
3 ways to accelerate applications

- Libraries
  - “Drop-in” Acceleration

- OpenACC Directives
  - Easily Accelerate Applications

- Programming Languages
  - Maximum Performance

Copyright: NVIDIA conference tutorial material
Tutorial overview

**Session 1 (12:30–13:30)**
- Welcome and introduction
- Ready-to-use GPU libraries
- OpenACC: directive-based incremental approach

**Coffee break (13:30–14:00)**

**Session 2 (14:00–16:00)**
- Mental model of GPU hardware, CUDA primer
- Advanced performance optimisation

**Coffee break (16:00–16:30)**

**Session 3 (16:30–18:30)**
- Advanced performance optimisation
- Future development of the CUDA ecosystem