GPUs in HPC – Introduction and Overview

Dominik Göddeke

Institut für Angewandte Mathematik (LS3)
TU Dortmund

dominik.goeddeke@math.tu-dortmund.de
http://www.mathematik.tu-dortmund.de/~goeddeke

27. Nutzertreffen Compute Service
TU Dortmund, 26. Mai 2011

technische universität dortmund   fakultät für mathematik

The Big Picture

Paradigm shift in hardware: parallelism and heterogeneity

- In a single chip: multicores $\rightarrow$ manycores (GPUs), ...
- In a workstation, cluster node: NUMA, accelerators, ...
- In a big cluster: NUMA, different node types ...

An inevitable development...

- Memory wall: data movement cost gets prohibitively expensive
- Memory wall: bandwidth $\sim$ number of sockets, not number of cores
- Power wall: cooling? atomic power plant next to each big machine?
- ILP wall: maximum resource utilisation?
- Memory wall + power wall + ILP wall = brick wall

Observation

- Already status quo for HPC systems, even without accelerators

GPUs: Myth, Marketing and Reality

Raw marketing numbers

- > 2 TFLOP/s peak floating point performance
- Lots of papers claim $> 100\times$ speedup

Looking more closely

- Single or double precision? same 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 5–30x speedup is realistic (and worth the effort)

GPUs: Myth, Marketing and Reality

Raw marketing numbers

- $> 100\times$ performance/\$ and performance/Watt

Looking more closely

- Strongly depending on the specific application
- CPU vs. GPU or plain vs. enhanced cluster node?
- GPUs have their own memory, hard to quantify
- How idle are CPU cores when GPUs compute?

Reality

- No hard numbers available for a wide range of 'typical' applications
- Generally better than conventional (commodity based) clusters
- Dedicated systems (BlueGene, NEC etc.) unclear

GPUs and the HPC Mainstream

Petascale era

- Three of the four fastest TOP500 systems contain GPUs
- Small-scale installations (64-128 GPUs) quite prevalent
- Available in every workstation (develop locally, scale out later)

Exascale era

- Next factor 1000 will stem from strong scaling in each node
- GPU-type hardware is *one* out of two identified avenues ($\rightarrow$ IESP)

Use GPUs now

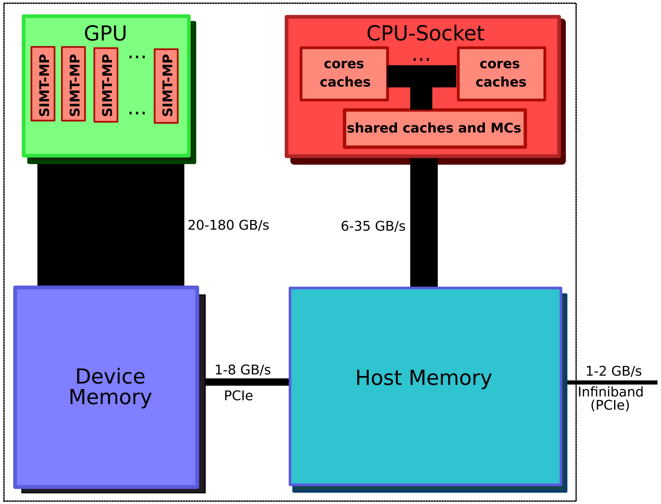
- Prototypes for exascale hardware
- Prototype for programming model (*much* more important)
- Exascale hardware will (?) scale down to the workstation level

GPU Hardware

and

GPU Programming

GPUs and the Memory Wall Problem



CPU Architecture

CPUs are general purpose architectures

- Optimised for latency of an individual task
- Several cores, each with lots of hardwired functionality
  - Branch prediction, context stacks, ...
  - SSE units and generic FPU $\rightarrow$ only 1–3% of the die area do the actual math!
- Cache hierarchies
  - Amount to approx. 50% of the die area
  - Currently three levels
  - Small private and large shared caches
- Fewer memory controllers than cores
- Fast direct link (QPI) between CPUs in the same node

GPU Architecture

GPUs are optimised for throughput of many similar tasks

- Note: NVIDIA speak of ‘CUDA cores’ is pure marketing
- Several cores (currently around 14)
  - One instruction unit
  - 48 ALUs execute the same instruction in each cycle $\rightarrow$ wide-SIMD
- Global hardware scheduler assigns ‘blocks of threads’ to cores (32–1024 threads per block)
- Blocks are virtualised cores
- Blocks from different ‘contexts’ can be scheduled

GPU Architecture

Memory subsystem

- 6–10 partitions/controllers, round-robin assignment
- Small shared cache (768 kB currently), tiny L1 cache per core
- Strict rules for memory access patterns of neighbouring threads
- But: 48 kB ‘scratchpad memory’ per core
  - Can be used as a user-controlled cache
  - Common use case: Load data into this memory, compute, write out

Most important difference to CPUs

- Stalls in one block (due to memory accesses) $\rightarrow$ scheduler switches to another block without context overhead
- Memory latency is completely hidden and bandwidth fully saturated
- All transfers are bulk transfers (granularity 16 threads)

GPU Architecture

Control flow

- Branch divergence: serialisation inside block (granularity 32 threads)
- Threads within each block may cooperate via shared memory
- Blocks cannot cooperate (implicit synchronisation at kernel scope)
- Kernels are launched asynchroneously
- Recently: Up to four smaller kernels active simultaneously

GPU is a co-processor

- CPU orchestrates computations on the GPU, GPU can work independently until all queued-up work has been finished
- Blocking and non-blocking transfers (of independent data)
- Streaming computations: Read A back, work on B and copy input data for C simultaneously
- Try to minimise CPU-GPU synchronisation and data transfers

GPU Architecture Summary

GPUs are not

- Vector architectures (but: wide-SIMD with independence)
- Fully task-parallel (although slowly getting there, yet performance stems from data parallelism)
- Easy to program efficiently (getting something running is easy though)

GPUs are particularly bad at

- Pointer chasing through memory (serialisation of memory accesses)
- Codes with lots of fine-granular branches
- Codes with lots of synchronisation and huge sequential portions

Lots of research going on

- Rule of thumb: ‘structured’ cases pretty much solved, irregular and seemly inherently sequential ones are challenging

GPU Programming Environments

Ready-to-use environments

- Academia and industry are pushing things
- Matlab, Mathematica, Photoshop, Ansys, Paraview, ...

Ready-to-use libraries

- BLAS, LAPACK, FFT, SpMV, ...

Compiler support

- PGI 11.x: PGI accelerator compiler: OpenMP-like code instrumentation
- Some PGAS-like approaches here and there

GPU Programming Environments

CUDA and OpenCL

- CUDA: vendor lock-in, but *much* more mature, larger toolkit ecosystem, better support
- OpenCL: in its infancies, designed as an API to build APIs, but: not limited to GPUs
- Pragmatic suggestion: CUDA now, switch to CL eventually (kernels are converted by smart copy & paste)

Several academic environments for hybrid programming

- HMPP, StarPU, Quark, ...

Rule of thumb

- There is none, depends on the particular application how much manual work is necessary

Some Examples

Acknowledgements

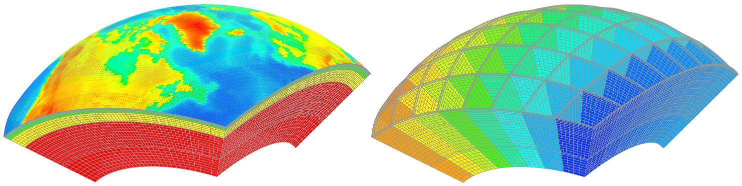
Joint work with a bunch of people over the past few years

- FEAST group at TU Dortmund: C. Becker, S. Buijssen, H. Wobker, S. Turek, M. Geveler, D. Ribbrock, P. Zajac, T. Rohkämper
- Robert Strzodka (Max Planck Institut Informatik)
- Pat McCormick and Jamal Mohd-Yusof (Los Alamos)
- Dimitri Komatitsch, Gordon Erlebacher and David Michea (Pau, Florida State and BGRM)

Geophysics: Seismic Wave Propagation

Porting of a complex MPI application entirely to GPU clusters

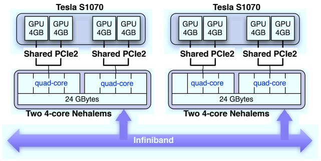
- Seismic wave propagation at the scale of the Earth
- Unstructured hexahedral mesh, high-order spectral element discretisation
- Bull cluster with 192 GPUs

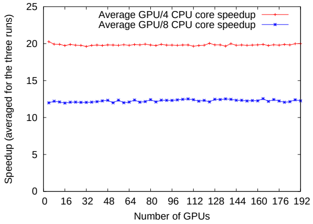


Geophysics: Seismic Wave Propagation

Highlights

- SEM-assembly on unstructured mesh
- Full overlap of async. MPI with CPU-GPU communication
- Perfect weak scaling
- 12–20 $\times$ faster

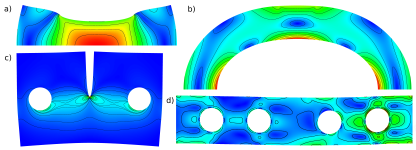


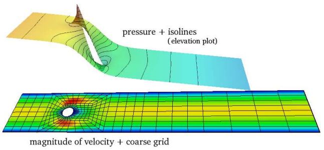


Numerics for PDEs

Accelerate portions of complex toolkit

- FEAST: Finite Element Analysis and Solution Tools
- Toolbox for large-scale simulations
- GPU acceleration of parts of the linear solver
- Example applications: linearised elasticity and stationary laminar flow





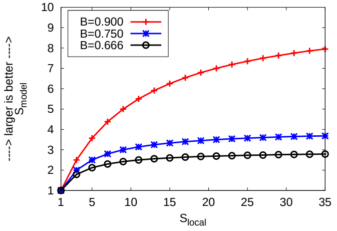
Numerics for PDEs

Theoretical model of expected speedup

- Integration of GPUs increases resources
- Correct model: strong scaling within each node
- Acceleration potential of the elasticity solver: $R_{\text{acc}} = 2/3$ (remaining time in MPI and the outer solver)
- $S_{\text{max}} = \frac{1}{1-R_{\text{acc}}}$ $\qquad S_{\text{model}} = \frac{1}{(1-R_{\text{acc}})+(R_{\text{acc}}/S_{\text{local}})}$

This example (Amdahl's Law)

Accelerable fraction R_{acc}	66%
Local speedup S_{local}	9x
Modeled speedup S_{model}	2.5x
Measured speedup S_{total}	2.6x
Upper bound S_{max}	3x



Summary

GPUs are becoming mainstream HPC

- But lots of open questions and research opportunities
- Challenge 1: ‘unstructured, irregular’ computations
- Challenge 2: Using both CPUs and GPUs efficiently

Further questions?

- I’m happy to discuss things
- Any application domains I omitted?
- Some particular software package that I didn’t mention?