Hardware oriented numerics

for real-time CFD applications on parallel hardware architectures

Markus Geveler, Dirk Ribbrock, Dominik Göddeke, Stefan Turek

Technische Universität Dortmund

October 15, 2009

Overview

1. HPC issues
   - Hardware
   - Software
2. The HONEI-approach
   - Basic design principles
   - Application-specific kernels
3. Evaluation: Real-time CFD
   - Test application
   - FSI Extensions
   - Results

Overview

1. HPC issues
   - Hardware
   - Software
2. The HONEI-approach
   - Basic design principles
   - Application-specific kernels
3. Evaluation: Real-time CFD
   - Test application
   - FSI Extensions
   - Results

The paradigm shift

Turning point:

The performance of serial codes does no longer improve automatically, as it did in the last century!

- Optimising circuits and deeper processor pipelines is limited by power and heat considerations (*Power Wall*)
- *Frequency scaling* is limited by *process shrinking*
- *Instruction-level parallelism* (dynamically and at compile time) is hard to extract, superscalar-, out-of-order-, speculative execution, VLIWs (EPICs),... have reached their limits (*ILP Wall*)

Multicore architectures

The solution (?):

Higher level parallelism

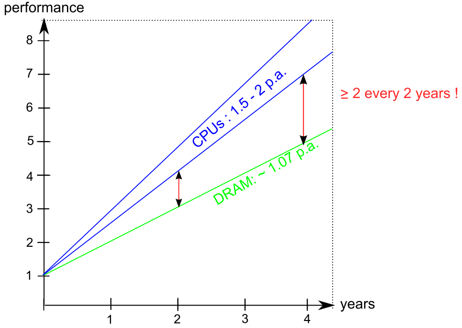
doubling the amount of transistors $\Rightarrow$ doubling the amount of *cores*

multicore-architectures have been established in the mass market

manycore-architectures too (GPUs) or: are about to come (Larrabee)

But:

Memory performance (bandwidth to off-chip memory) continues to increase at a significant slower rate!

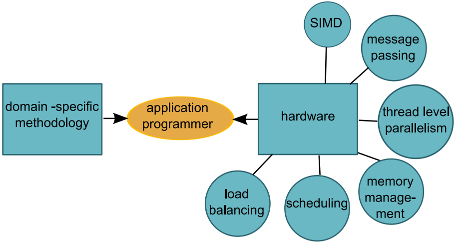


The challenge

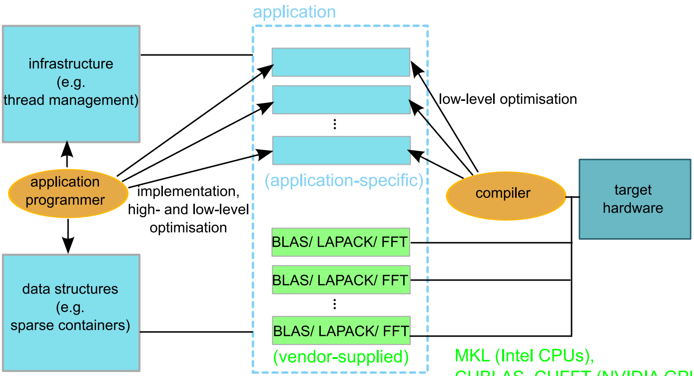
Application programmers have to deal with...

...both, implementing specific functionality and the paradigm shift of the underlying hardware:

- high-level optimisation on the application level
- hardware-oriented implementation to exploit parallel hardware



The usual way



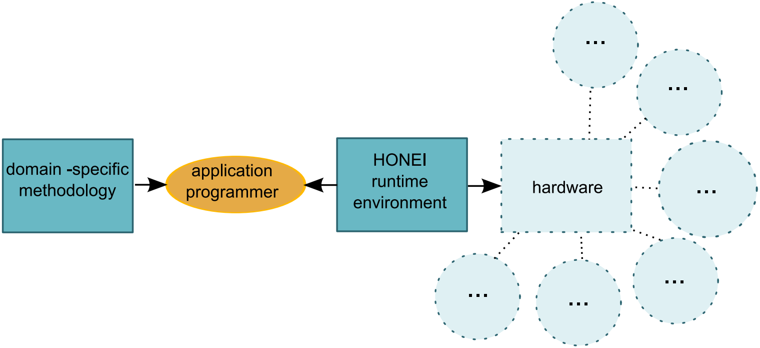
MKL (Intel CPUs),
CUBLAS, CUFFT (NVIDIA GPUs),
ACML (AMD/ATI multicore + GPUs)
CBE BLAS/LAPACK/FFT (IBM for Cell)

Overview

technische universität
dortmund

Primary design-goal:

Abstract from target-hardware!



Basic structure: shared libraries

Frontends:

- unified dense and sparse (sparse-banded) containers: matrices and vectors
- linear algebra operations
- iterative solvers
- domain-specific operations (CFD)

Backends:

- x86 SIMD (handcrafted SSE2 using intrinsics)
- x86 multicore (pthreads)
- distributed memory clusters (MPI)
- NVIDIA GPUs (CUDA 2.2)
- Cell BE (libspe2)

Basic features

technische universität
dortmund

Building applications upon the provided operations/solvers

- generic programming ⇒ clean interfaces:

```
ScaledSum<tags::CPU::SSE>::value(a, x, y)
```

  ⇒ implementational details kept away from the programmer
- solvers/operations can be plugged together:

```
ConjugateGradients<tags::CPU,
methods::JAC,...>::value(...)
```

- fixed + mixed precision implementations
- applications run automatically with all the supported backends
- backend combination (example):

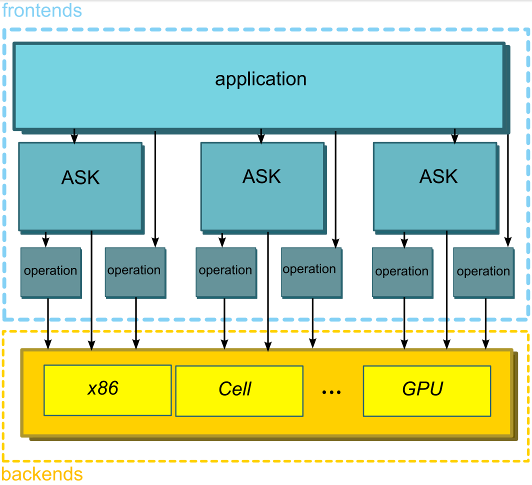
```
ScaledSum<tags::Multicore::SSE>::value(a, x, y)
```

- multiple device support e.g. `tags::CPU::SSE` + `tags::GPU::CUDA`
- straight-forward CPU implementation for numerical comparison (`tags::CPU`)

Advanced features

Application specific functionality not covered by the frontends

HONEI supports the development of *application-specific kernels* (ASKs)



Advanced features

technische universität dortmund

General support

- unittest + benchmark frameworks, visualisation
- build system to create SPE kernels for Cell BE, CUDA-kernels
- thread-management, MPI

Common runtime environment: Cell BE

- support for RTTI and exception-handling, memory transfers
- RPC system to call SPE programs from the PPE
- templates to facilitate development of new callable SPE functions and registering them with the RPC system
- automatic job scheduling

Common runtime environment: NVIDIA GPUs

- implicit memory transfers by a transparent *memory arbiter*

HONEI: summary

technische universität dortmund

Two use-cases

- applications directly built on top of the provided operations are hardware-accelerated automatically
- development of application-specific kernels is eased by a CRE, which deals with hardware-specific details

Currently supported functionality (frontends):

- `libhoneila`: generic containers and linear algebra operations ('BLAS-like')
- `libhoneimath`: iterative solvers (Jacobi, CG, MG, iterative refinement, ...), numerical operations (Gaussian Quadrature, ...)
- `libhoneiswe`, `libhoneilbm`: datastructures, solvers and operations for CFD problems

Overview

Real-time CFD

Goal

Render fluid volumes at *interactive rate* for use in a 3D environment (e.g. computer games)

Quality criteria

- real-time
- qualitative correctness: correct behaviour
- quantitative correctness: numerical quality less important
- stability

Method

- use 2D methods to approximate the volume by its surface
- make it look like 3D
- enable interaction

Shallow Water waves

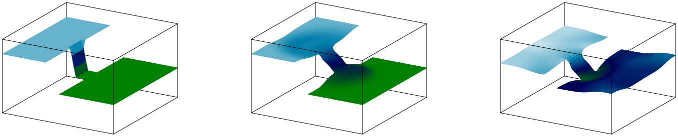
technische universität dortmund

2D Shallow Water Equations

$$\frac{\partial}{\partial t}Q + \frac{\partial}{\partial x}F(Q) + \frac{\partial}{\partial y}G(Q) = 0$$

$$F(Q) = \begin{pmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \\ huv \end{pmatrix}, \qquad G(Q) = \begin{pmatrix} hu \\ huv \\ hv^2 + \frac{1}{2}gh^2 \end{pmatrix},$$

$$Q = (h, h \cdot u, h \cdot v)^T$$



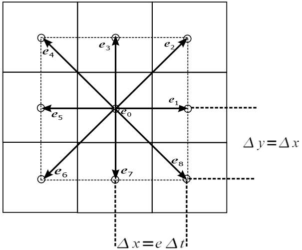
(a) (b) (c)

Numerical method

The Lattice-Boltzmann method: discrete phase-space

- representation of fluids in general by *particle populations*
- particle movement (*streaming*) is restricted to trajectories defined by the *lattice*

$$\mathbf{e}_\alpha = \begin{cases} (0,0) & \alpha = 0 \\ e[\cos\frac{(\alpha-1)\pi}{4}, \sin\frac{(\alpha-1)\pi}{4}] & \alpha = 1,3,5,7 \\ \sqrt{2}e[\cos\frac{(\alpha-1)\pi}{4}, \sin\frac{(\alpha-1)\pi}{4}] & \alpha = 2,4,6,8 \end{cases}$$



LBM for SWE

The Lattice-Boltzmann method: LBGK approximation of the collision-operator

- particle behaviour is determined by streaming and *collision* on the microscopic level
- fluid behaviour is determined by change of particle distributions on the lattice and local equilibria

$$f_\alpha(\mathbf{x} + \mathbf{e}_\alpha \Delta t, t + \Delta t) = f_\alpha(\mathbf{x}, t) - \frac{1}{\tau}(f_\alpha - f_\alpha^{eq})$$

The Lattice-Boltzmann method: local equilibrium distribution functions for the SWE

- the f^{eq} determine the governing flow equations in the LBM
- replacement of well-known f^{eq} for Navier-Stokes

$\Rightarrow$ adaption of SWE source-terms for bed-topography possible ($\rightarrow$ '$2\frac{1}{2}$D')

$$f_\alpha^{eq} = \begin{cases} h - \frac{5gh^2}{6e^2} - \frac{2h}{3e^2} u_i u_i & \alpha = 0 \\ \frac{gh^2}{6e^2} + \frac{h}{3e^2} e_{\alpha i} u_i + \frac{h}{2e^4} e_{\alpha j} u_i u_j - \frac{h}{6e^2} u_i u_i & \alpha = 1, 3, 5, 7 \\ \frac{gh^2}{24e^2} + \frac{h}{12e^2} e_{\alpha i} u_i + \frac{h}{8e^4} e_{\alpha j} u_i u_j - \frac{h}{24e^2} u_i u_i & \alpha = 2, 4, 6, 8 \end{cases}$$

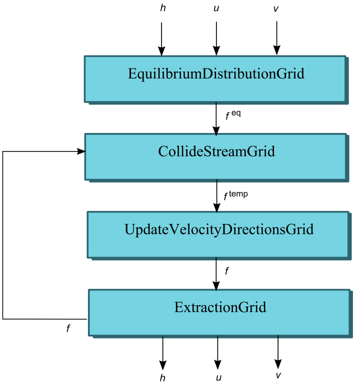
technische universität dortmund

The Lattice-Boltzmann method: extraction of physical quantities

- fluid depth is the sum of all particles residing at a lattice cite
- macroscopic velocity is the sum of all particles at a cite weighted by the corresponding component of the lattice-velocity, divided by 'macroscopic mass' (aka fluid depth)

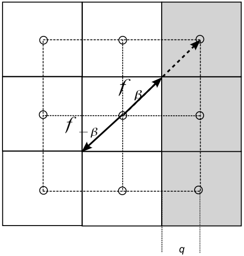
$$h(\mathbf{x}, t) = \sum_{\alpha} f_{\alpha}(\mathbf{x}, t)$$

$$u_i(\mathbf{x}, t) = \frac{1}{h(\mathbf{x}, t)} \sum_{\alpha} e_{\alpha i} f_{\alpha}(\mathbf{x}, t)$$



boundary conditions

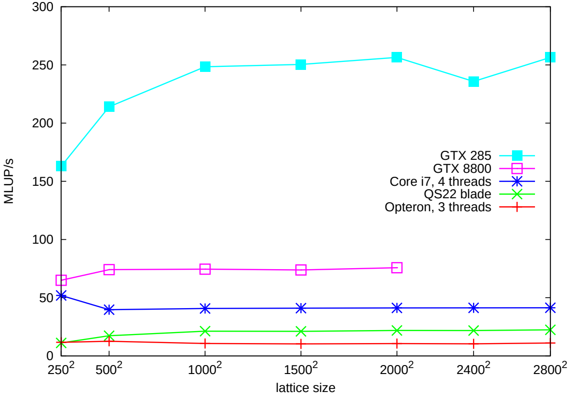
simple *bounce-back-scheme*



Results

Architecture comparison, backend evaluation

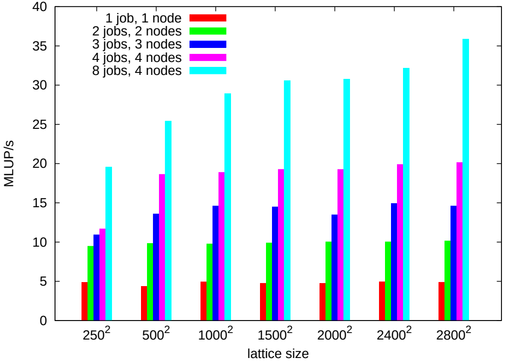
- Intel Core i7 920, AMD Opteron 2214 using `tags::Multicore::SSE`
- QS22 blade using `tags::Cell`, NVIDIA GeForce GPUs



Results

Opteron cluster, MPI, strong scalability

4 nodes, Opteron 2214 ↓



LiDO, weak scalability, n=600

#lattices	time
n^2	9.5
$2n^2$	9.7
$4n^2$	9.7
$8n^2$	9.8
$16n^2$	9.9

technische universität dortmund

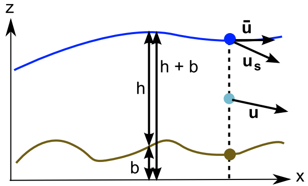
Inhomogeneous SWE, source terms, LBM force terms

Needed external forces: *bed-slope*, *bed-friction*

$$\frac{\partial}{\partial t}Q + \frac{\partial}{\partial x}F(Q) + \frac{\partial}{\partial y}G(Q) = S^b(Q).$$

$$F_i = -g(h\frac{\partial b}{\partial x_i} + n_b^2 h^{-\frac{1}{3}} u_i \sqrt{u_j u_j})$$

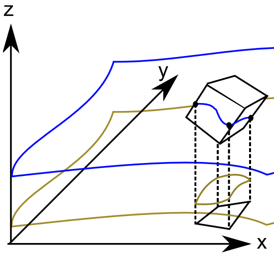
$$f_\alpha(\mathbf{x} + \mathbf{e}_\alpha \Delta t, t + \Delta t) = f_\alpha(\mathbf{x}, t) - \frac{1}{\tau}(f_\alpha - f_\alpha^{eq}) + \frac{\Delta t}{6e^2} e_{\alpha i} F_i$$

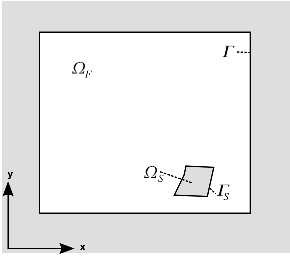


Interaction with the scene II

Representation of floating solids

- 3D solids become 2D regions
- lattice cites are flagged *solid* or *fluid* dynamically
- boundary conditions are locally adjusted to *moving boundaries* (BFL rule)
- new fluid cites are initialised by extrapolation methods

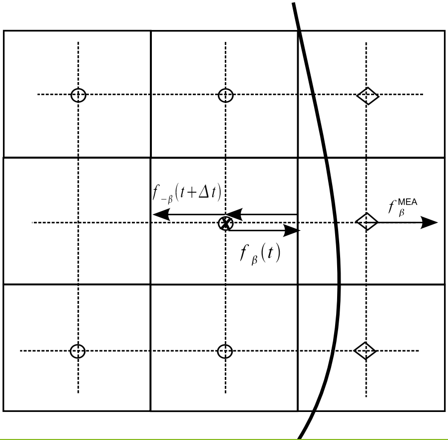




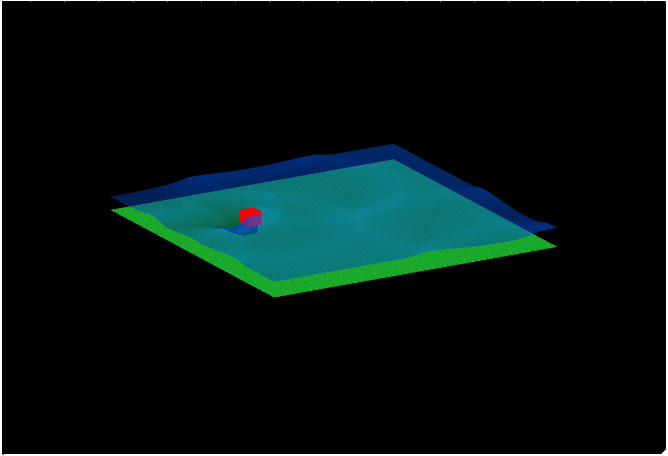
technische universität dortmund

And finally: *dry-states*, solid reaction

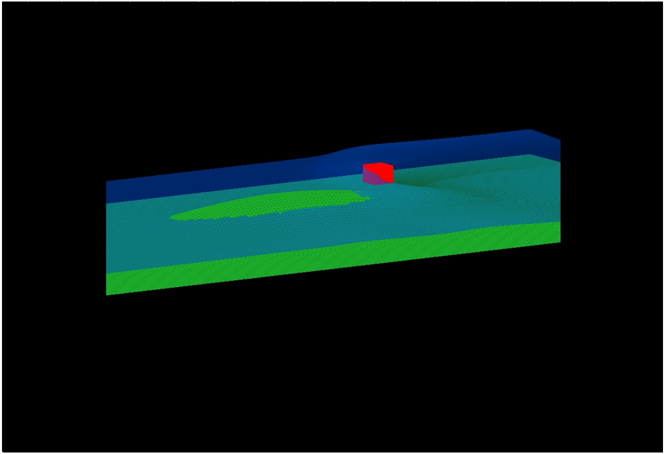
- stabilise critical flows
- solid reaction via microscopic momentum exchange



technische universität dortmund

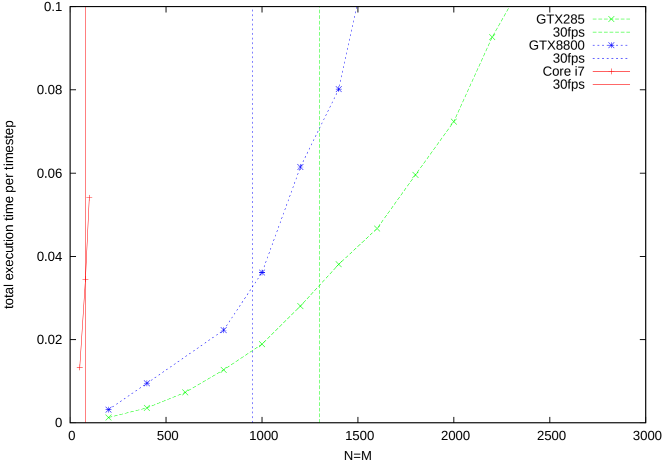


Results



Example: real-time performance on the GPU

GTX 285, GeForce 8800



Conclusion

Modern hardware changes towards parallelism and heterogeneity
⇒ dealing with hardware specific details is unavoidable but undesireable

Making codes run on alternative designs is relatively easy, exploiting the performance capabilities is not

HONEI abstracts the hardware and is able to redirect the programmers attention to domain specific methodology and is designed for

- testability
- flexibility
- extendability, maintainability

www.honei.org

Supported by DFG, projects TU 102/22-1, TU 102/22-2
and BMBF, *HPC Software für skalierbare Parallelrechner: SKALB project 01IH08003D*.