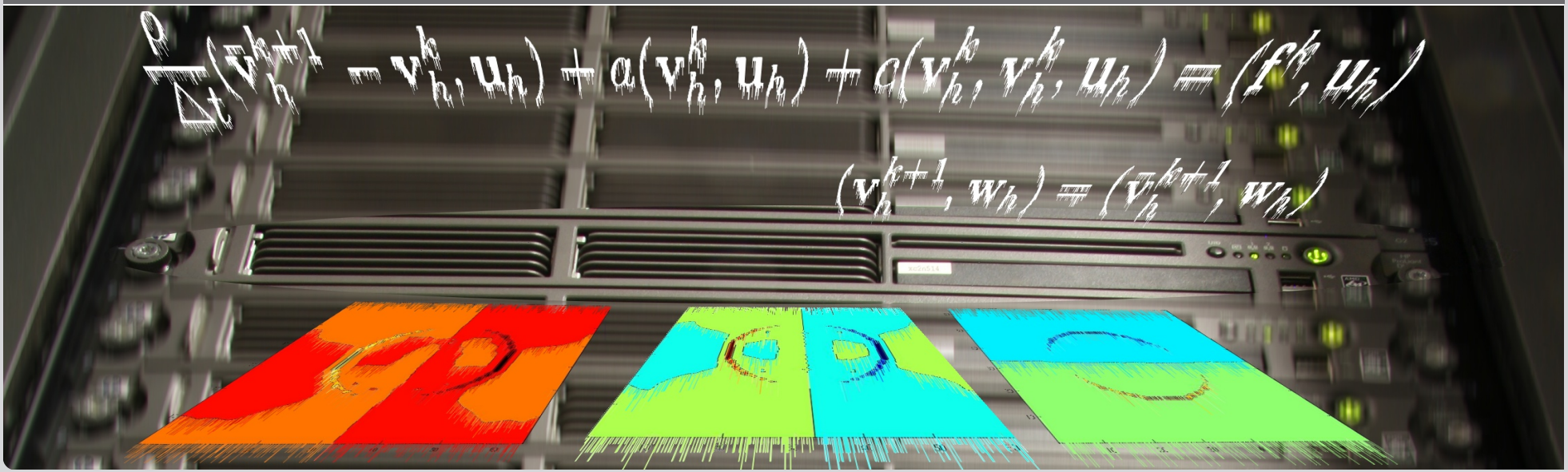
Karlsruhe Institute of Technology

LAtoolbox: A Multi-platform Sparse Linear Algebra Toolbox

Vincent Heuveline, Dimitar Lukarski, Jan-Philipp Weiss

NVIDIA GPU Technology Conference • San Jose, CA • May 17, 2012 • Session S0291

Engineering Mathematics and Computing Lab (EMCL)



The Shift towards Multicore/Manycore

Karlsruhe Institute of Technology

Paradigm shift

- The paradigm shift towards manycore is well on the way
- But still many problems have to be overcome
  - Scalable algorithms and implementations, ubiquitous parallelism, programmers' productivity, ...

- Many processors concepts have evolved (and some have gone)
  - x86, SPARC (2-12 cores)
  - NVIDIA / AMD GPUs (240 - 1536 cores)
  - Accelerated Processing Units
    (2-4 cores + 400 cores, hybrid CPU/GPU)
  - NVIDIA Denver (ARM-CPUs + GPUs on a chip)
  - Cell, ClearSpeed (8+1, 92 cores)
  - Intel SCC, MIC (48, 32/80 cores)
  - Tile64, Epiphany (64, 64 - 4096 cores)
- Manycore trend: much higher core count, less complex cores

Manycore – Revolution or Evolution?

Karlsruhe Institute of Technology

Manycore Devices

- High core count and incredible computing performance
- Fine-grained parallelism, stream processors + wide vector units
- Fast on-chip bandwidth, caches + local memory

Impact on the Software

- Without changing the software there will be no more speedups
- For performance benefits the mathematical models, algorithms, numerical schemes and programs need to be adapted, re-engineered and further developed
- New ways of parallel thinking required

The Programming Challenge

Karlsruhe Institute of Technology

Zoo of languages and programming approaches

OpenMP, CUDA, OpenCL, OpenACC, ArBB, TBB, pthreads, UPC/PGAS, Cilk Plus, and many many more

Who can handle them all? Which is the best approach?
We need some further abstractions!

Three main questions

- **Portability**: How portable is my program?
  (in terms of platforms and performance)
- **Flexibility**: Can I easily modify the software?
  (new modules, algorithms, platforms, functionality, ...)
- **Scalability**: How scalable is my algorithm/programming model?
  (ready for future platforms? ready for exascale?)

The Algorithmic Challenge

Karlsruhe Institute of Technology

We have to rethink our methods and approaches!

Typical way of parallel thinking (?)

- Cluster-like problem decomposition with many fat nodes
  - Substructure problem into huge blocks (of same size)
  - Process blocks in parallel
  - Minimize communication between blocks
  - But typically: sequential processing on the block-level

New way of parallel thinking

- Decompose problem into blocks (of varying size)
  - There should be only a lower bound on the block size
- Process blocks in sequential order (or in parallel)
- **But: use scalable parallelism on the block-level !**

The Algorithmic Challenge

Karlsruhe Institute of Technology

We have to rethink our methods and approaches!

Typical way of parallel thinking (?)

- Cluster-like problem decomposition with many fat nodes
  - Substructure problem into huge blocks (of same size)
  - Process blocks in parallel
  - Minimize communication between blocks
  - But typically: sequential processing on the block-level

New way of parallel thinking

- Decompose problem into blocks (of varying size)
  - There should be only a lower bound on the block size
- Process blocks in sequential order (or in parallel)
- **But: use scalable parallelism on the block-level !**

Two Solution Approaches

We would like to show you two things:

I: How to build scalable and efficient algorithms

- How to re-arrange algorithms for scalable parallelism
- How to make them ready for GPUs and multicore CPUs
- How to keep mathematical quality and efficiency

II: How to build portable software (this talk!)

- Modular building of solver schemes
  - Single code base for GPUs, CPUs and other accelerators
  - Seamless integration of accelerators by unified interfaces
  - Choice of the platform can be taken at run time
- Highly optimized implementations
  - Transparent to the programmer/user

Two Solution Approaches

We would like to show you two things:

I: How to build scalable and efficient algorithms

We have demonstrated new algorithms for parallel preconditioners on GPUs in Session S0289, Wednesday, May 16, 9:00am. Please check these slides for further details.

II: How to build portable software (this talk!)

- Modular building of solver schemes
  - Single code base for GPUs, CPUs and other accelerators
  - Seamless integration of accelerators by unified interfaces
  - Choice of the platform can be taken at run time
- Highly optimized implementations
  - Transparent to the programmer/user

Two Solution Approaches

We would like to show you two things:

I: How to build scalable and efficient algorithms

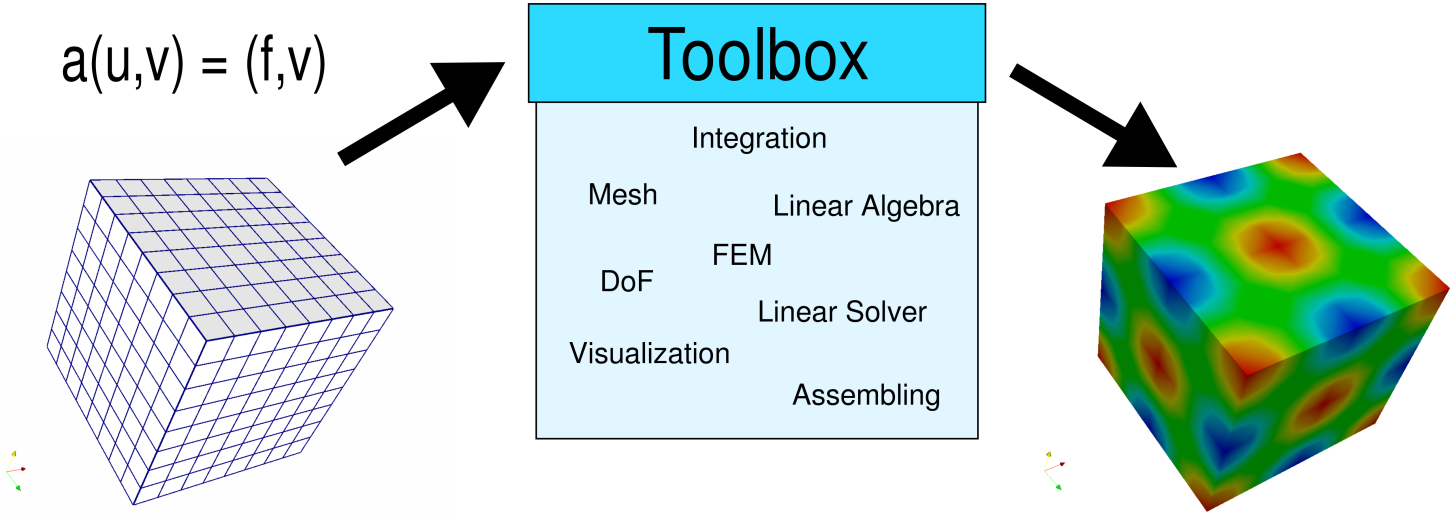
We have demonstrated new algorithms for parallel preconditioners on GPUs in Session S0289, Wednesday, May 16, 9:00am. Please check these slides for further details.

II: How to build portable software (this talk!)

- Modular building of solver schemes
  - Single code base for GPUs, CPUs and other accelerators
  - Seamless integration of accelerators by unified interfaces
  - Choice of the platform can be taken at run time
- Highly optimized implementations
  - Transparent to the programmer/user

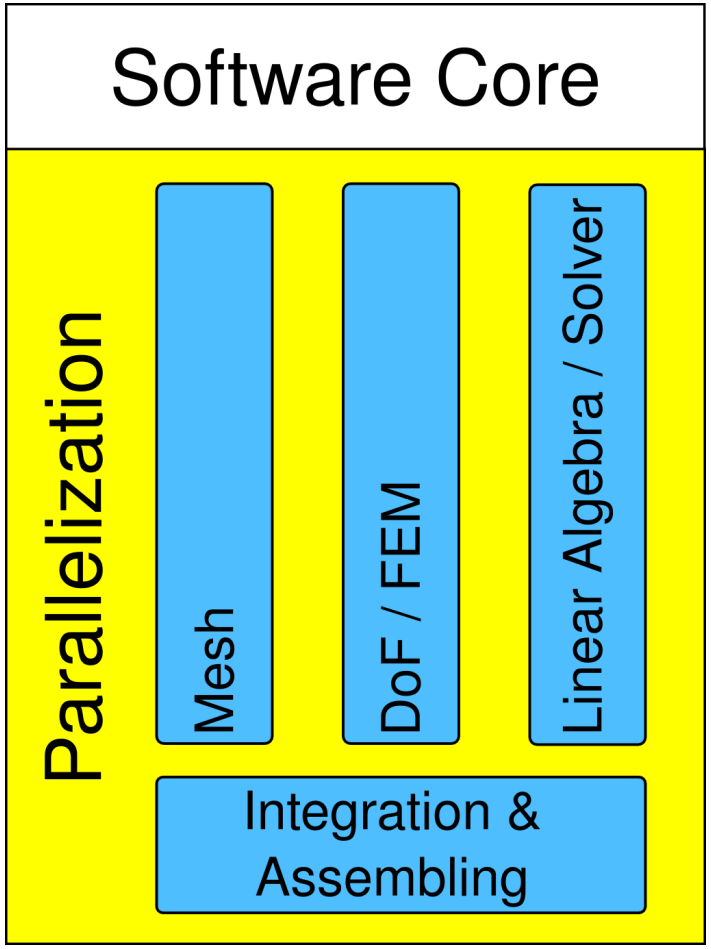
HiFlow^3 - Parallel Finite Element Software

- **Application Fields**: Numerical simulation in computational fluid dynamics, meteorology, medical engineering and energy research
- **Performance**: Scalability proven on huge clusters and GPU-accelerated systems
- **Flexibility**: Generic, modular and extensible software for multi-purpose simulation based on object-orientation in C++



HiFlow^3 - Modules

- **Mesh**: Distributed and flexible mesh handling
- **FEM**: Finite element spaces
- **DoF**: Management of degrees of freedom
- **Integration routines**: Numerical integration and assembly routines
- **Linear algebra/solvers**: Multi-platform linear algebra and hardware-aware computing



Features of the LAtoolbox

Karlsruhe Institute of Technology

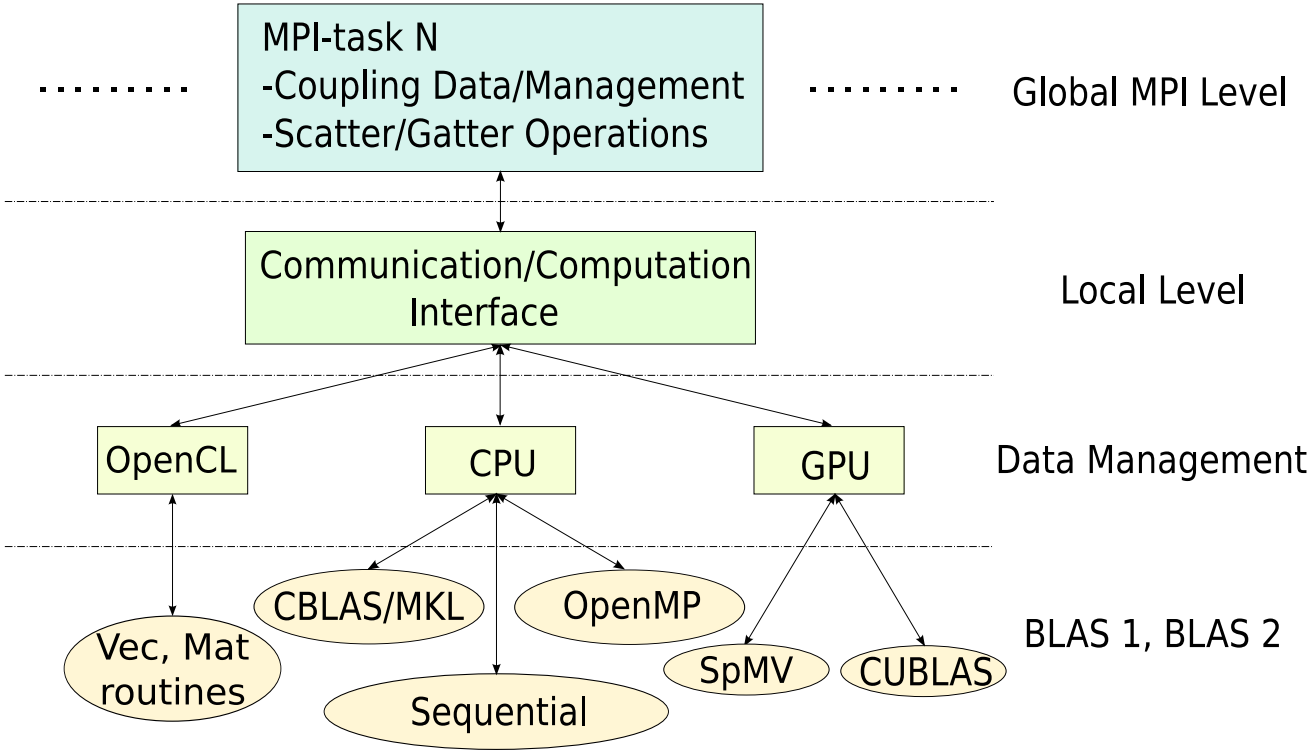
Communication and computation layer of HiFlow^3

- Multi-platform parallelization (GPUs, multicore-CPUs, accelerators, ...)
- Unified interfaces across platforms for all linear algebra operations and solvers
- Abstract data structures
- Platform-adapted library implementations chosen at run time
- No hardware knowledge required for building applications
- Minimal communication costs for intra-node data exchange

Solvers and preconditioners

- Iterative linear solvers: CG, (F)GMRES, geometric multigrid
- Newton-like solvers for nonlinear problems
- Parallel preconditioners for GPUs and CPUs

LAtoolbox - A Two Level Library



Parallelism is introduced on two levels:

- Coarse-grained parallelism by means of distributed grids and distributed data structures
- Fine-grained parallelism by means of platform-optimized linear algebra back-ends

Data Distribution and Communication

for Sparse Matrix-Vector Multiplication (SpMV)

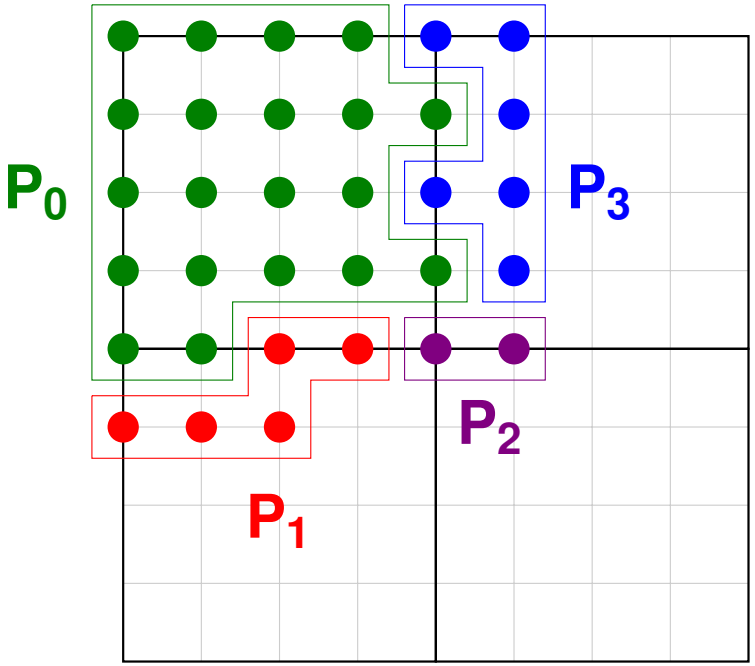
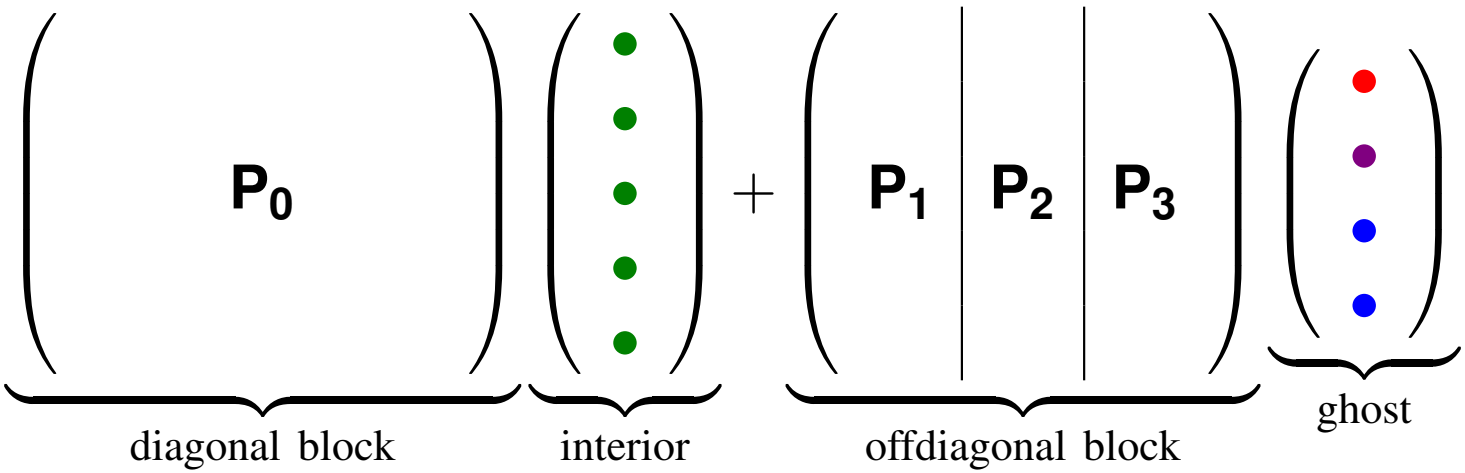


Figure: Domain partitioning: Degrees of Freedom (DoF) of process $\mathbf{P_0}$ are marked in green (interior DoF in diagonal block); the remaining DoF represent inter-process couplings for process $\mathbf{P_0}$ (ghost DoF in off-diagonal block)

Data Distribution and Communication

for Sparse Matrix-Vector Multiplication (cont.)



Sparse Matrix-Vector Multiplication:

- start asynchronous communication
- exchange ghost values
- $y_{\text{int}} = A_{\text{diag}}\, x_{\text{int}}$
- synchronize communication
- $y_{\text{int}} = y_{\text{int}} + A_{\text{offdiag}}\, x_{\text{ghost}}$

Advanced Features

Karlsruhe Institute of Technology

Advanced Features of the LAtoolbox

- No irregular memory access over the PCIe bus by blocking techniques
- Raw access to the data is possible with explicit declaration of device-specific classes (e.g. for nested loops, irregular memory access, etc.)
- Possible mixture of different types of platforms in the cluster (e.g. nodes with and without GPUs)

ImpLAtoolbox - Sample Source Code 1/2

Karlsruhe Institute of Technology

```
lVector<double> *x, *y ;

// initialize a vector on a specific platform
x = init_vector<double>(size,"vec x", platf, impl);

// clone y as x
y = x->CloneWithoutContent();

// Usage of BLAS 1
y->CopyFrom(*x);          // y = x
x->Scale(5.0);            // x = x * 5.0
y->Axpy(*x, 3.3 );        // y = y + 3.3*x
```

ImpLAtoolbox - Sample Source Code 2/2

```
lMatrix<double> *mat, *mat_c;

// init empty matrix on a specific platform
mat = init_matrix<double>(0,0,0,"A",platf,impl,CSR);

// init CPU matrix
mat_c = init_matrix<double>(0,0,0,"A",CPU,OpenMP,CSR)
mat_c->ReadFile("matrix.mt");

// Copy the sparse structure
mat->CopyStructureFrom(*mat_c);
// Copy only the values of the matrix
mat->CopyFrom(*mat_c);

mat->VectorMult(*y, x); // x = mat*y
```

Example: Geometric Multigrid Solver

Karlsruhe Institute of Technology

Model problem: $-\Delta u = f \text{ in } \Omega := (0,1)^2 \setminus (0,0.5)^2$

Solve Poisson problem on the L-shaped domain

- Zero Dirichlet boundary conditions, i. e. $u = 0 \text{ on } \partial\Omega$
- Locally refined and statically adapted mesh
- Mixed finite elements: Q1 elements on quadrilaterals and P1 elements on triangles (avoiding hanging nodes)
- Problem size: 3,211,425 DoF

Matrix-based geometric multigrid solver

- Start with zero initial guess
- Relative residual of 10^{-6} taken as stopping criterion
- Various parallel smoothers: Multi-colored Gauss-Seidel, power(q)-pattern enhanced ILU(p), FSAI

Solution in the L-shaped domain

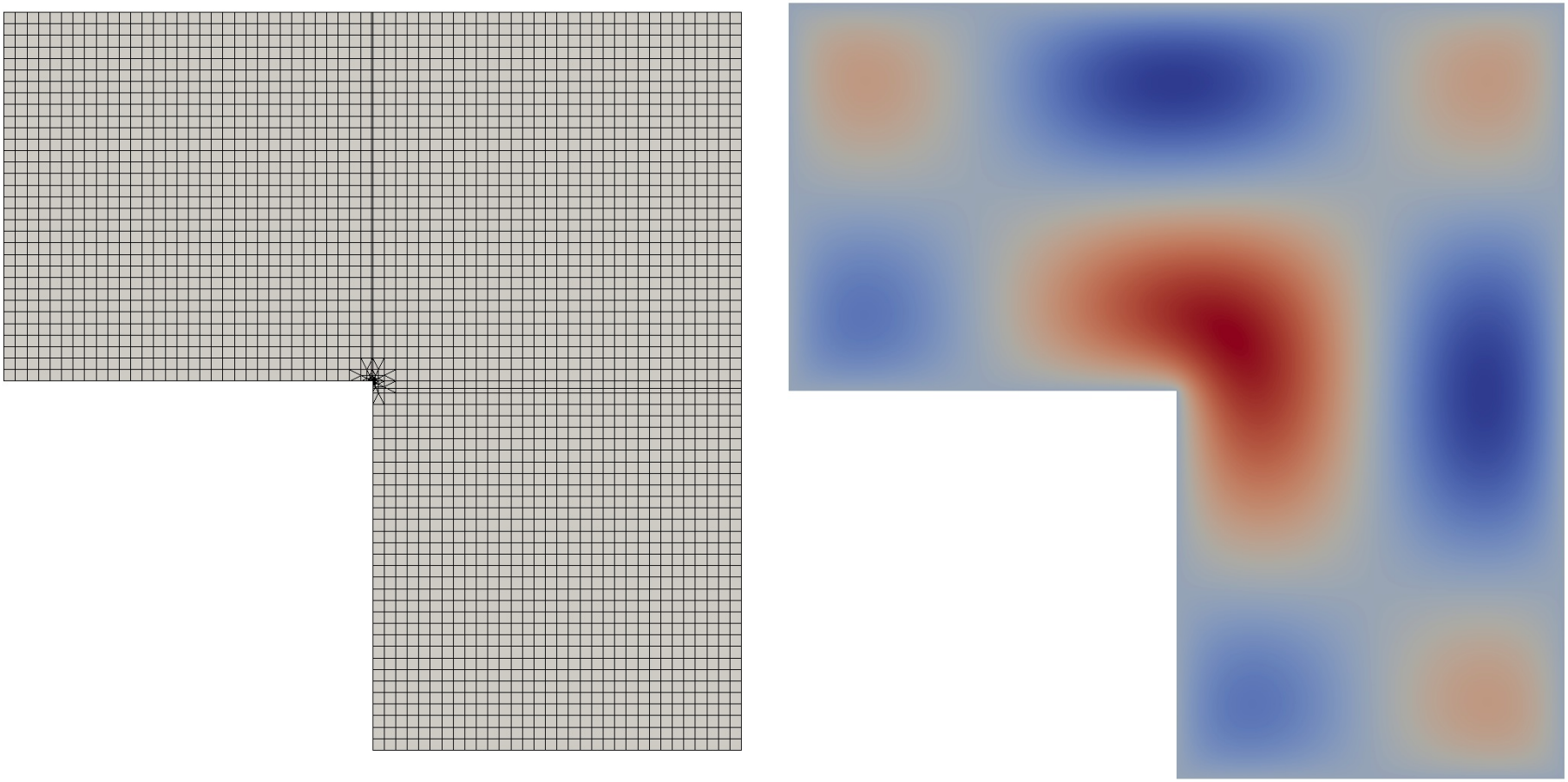
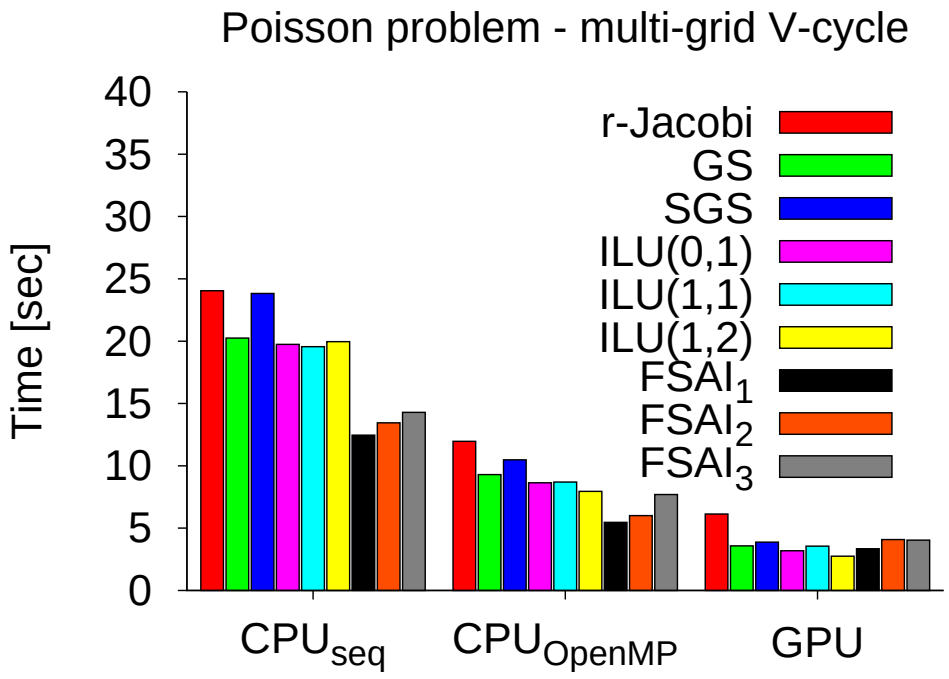


Figure: Locally refined mesh for the L-shaped domain (left) and discrete solution of the Poisson problem for $f = 8\pi^2 \cos(2\pi x)\cos(2\pi y)$ (right).

Performance Data for the Multigrid Solver

Run time comparison for various smoothers and platforms



- Platform: 2x Xeon quadcore CPU + Tesla S1070 4G memory
- Best multigrid solver with ILU(1,2) takes 2.75sec on the GPU
- Significantly faster than the best preconditioned CG solver (with ILU(1,2)) on the GPU with 158sec

Example: CG Solver for a 3D problem

3D-Neumann-Laplace problem

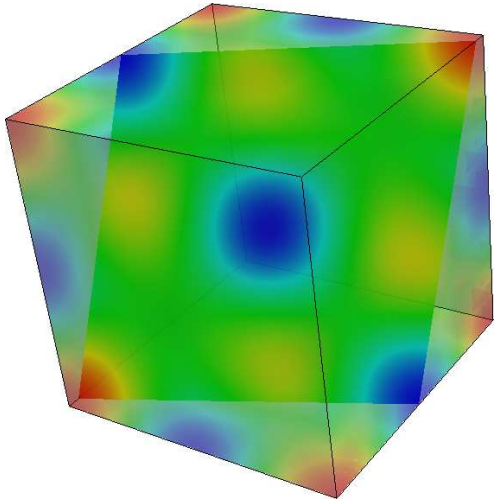
$$-\Delta \mathbf{u} = \mathbf{f}, \quad \text{in } \Omega = [0,1]^3$$
$$\frac{\partial \mathbf{u}}{\partial \mathbf{n}} = 0, \quad \text{on } \partial\Omega$$

Configuration of the conjugate gradient solver:

- Q1 finite elements
  with 2.1 M degrees of freedom
- CG solver with absolute tolerance 10^{-14}
- 510 iterations needed
- Double precision computations

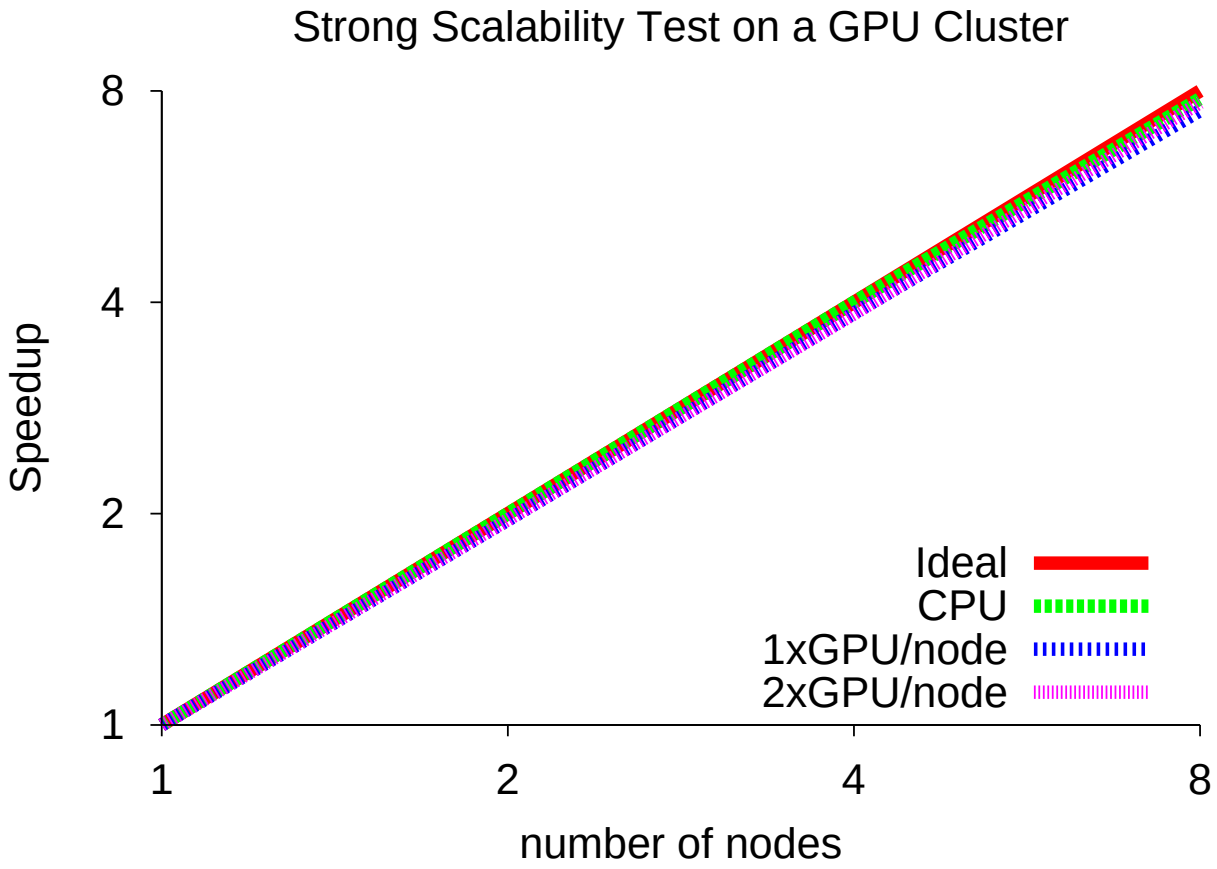
Solution on a cluster with 16 GPUs:

- 8 nodes with two NVIDIA
  Tesla M1060 GPU each, 4G memory
- 20 Gbit/s Infiniband (small DDR switch)



Strong Scalability Results

for the 3D Neumann problem on a 16-GPU-cluster



- Close to linear speedup for all GPU and CPU configurations
- Same source code for all setups

Summary

We have presented the easy-to-use Linear Algebra Toolbox

- Modular building of applications
- Same source code for all platforms
- No hardware knowledge required
- Can be compiled on all platforms
- Choice of the platform can be taken at run time

- Two-level parallelism with minimal communication costs
  - Distributed data across nodes
  - Linear algebra backends for GPUs, CPUs, and others
- We have demonstrated
  - Scalability – very good strong scalability for GPUs and CPUs
  - Portability – same source code for CPUs and GPUs
  - Flexibility – easy modifications of solvers and algorithms
  - Extensibility – add new backends for new platforms

Contact and Acknowledgements

Check our open-source software

The LAtoolbox is part of the HiFlow^3 open source parallel finite element package with a rich suite of solvers.
See **http://www.hiflow3.org**

Further information

- dimitar.lukarski@kit.edu
- jan-philipp.weiss@kit.edu
- http://www.emcl.kit.edu
- http://www.hiflow3.org

The Engineering Mathematics and Computing Lab (EMCL) at KIT is an NVIDIA CUDA Research Center with kind support by NVIDIA.