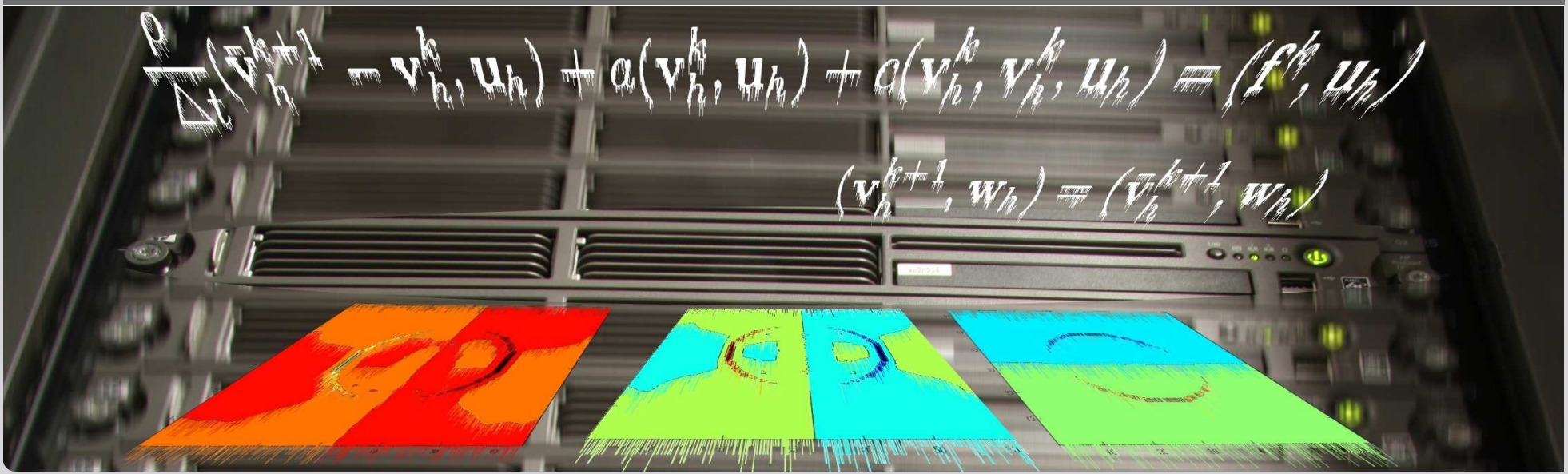
Karlsruhe Institute of Technology

Scalable Multi-Coloring Preconditioning for Multi-core CPUs and GPUs

Vincent Heuveline[1], Dimitar Lukarski[1,2], Jan-Philipp Weiss[1,2]

UCHPC'10 Workshop ● Ischia, Italy ● August 30, 2010 ● Euro-Par 2010

Engineering Mathematics and Computing Lab (EMCL)[1] / SRG New Frontiers in High Performance Computing[2]



Emerging Multi-/Many-core Technologies

Karlsruhe Institute of Technology

Sea change in hardware technologies and programming paradigms

- Exponentially increasing core counts
- Multi-level and fine-grained parallelism
- Deeply nested hierarchical memory sub-systems
- Heterogeneous platforms

Programming Challenges

MPI, OpenMP, CUDA, OpenCL, Ct, IBM Cell SDK, PGAS, ...

Urgent Questions:

- Portability, Flexibility, Scalability!
- How to adapt algorithms and numerical schemes?
- How to develop hardware-aware methodologies?

Linear Solvers and Preconditioners

Karlsruhe Institute of Technology

We want to solve $Ax = b$ on a node-level parallel system:

Most iterative linear solvers can be performed in parallel

Krylov subspace methods: CG, GMRES, ...
Splitting methods: Jacobi, Richardson, ...
Projection methods: Chebyshev, ...

All underlying routines are parallelizable:

- Vector operations (BLAS 1): scalar product, norm and vector updates are data parallel routines
- Sparse matrix-vector multiplications (sparse BLAS 2) are data parallel routines with irregular memory access patterns

Linear Solvers and Preconditioners

Karlsruhe Institute of Technology

The preconditioner influences the condition number of the linear system and decreases the number of iterations

Goal: Provide an efficient, flexible, and scalable preconditioner suitable for multi-core CPUs, GPUs, and other coprocessors

In each step of the solver an additional linear system has to be solved

$$Mz = r$$

In our test scenario we are using a Conjugate Gradient (CG) solver and a Symmetric Gauss-Seidel (SGS) preconditioner of type

$$M = (D + L)D^{-1}(D + R),$$

where $A = D + L + R$ with L lower-triangular, R upper-triangular, and D diagonal.

Solving the Preconditioning Equation

Karlsruhe Institute of Technology

- Sequential scheme
  - Gauss Elimination / Incomplete LU
    - **Pros:** *very easy to implement*
    - **Cons:** *not parallel; PCIe bottleneck (i.e. not suitable for GPUs)*
- Parallel schemes
  - Jacobi preconditioner
    - **Pros:** *very simple and very easy to implement*
    - **Cons:** *does often not improve the condition number of the system*
  - Block-Jacobi-type preconditioner
    - **Pros:** *simple and easy to implement*
    - **Cons:** *small sequential task, not scalable (decoupling the system)*
  - Algebraic Multigrid
    - **Pros:** *good improvements, scalable*
    - **Cons:** *complex, mostly sequential setup step*
  - Multi-coloring reordering
    - **Pros:** *fast, scalable, better cache utilization*
    - **Cons:** *requires a pre-processing step*

Multi-coloring Algorithm

Karlsruhe Institute of Technology

The goal is to color (label) the nodes of the sparse matrix (graph) in a way that there are no two adjacent nodes having the same color and the number of colors is as small as possible.

for i=1,...,N Set Color(i)=0; (where N=#nodes)
for i=1,...,N Set Color(i)=min(k>0:k!=Color(j) for j $\in$ Adj(i));

where $\mathrm{Adj}(i) = \{j \neq i | a_{i,j} \neq 0\}$ are the adjacents to node i.

- Parallel approach: block decomposition
- Diagonal blocks of size $b_k \times b_k$ are diagonal matrices with multi-coloring!
- Degrees of parallelism are $b_k = N/B$, where N is the number of unknowns in the system and B is the number of colors

Solving the Block-decomposed System

- Crucial point: Inversion of matrices D_i on block-diagonal
- Main goal of multi-coloring: obtain only diagonal elements in the block-diagonal matrices D_i
- SpMV dominates the algorithm: good scalability and high degree of parallelism
- The number of SpMV operations is $B(B-1)$
- The algorithm is bandwidth-bound !

$$x_i := D_i^{-1}(r_i - \sum_{j=1}^{i-1} L_{i,j}x_j) \text{ for } i = 1,\ldots,B$$

$$y_i := D_i^{-1}x_i \text{ for } i = 1,\ldots,B$$

$$z_i := D_i^{-1}(y_i - \sum_{j=1}^{B-i} R_{i,j}z_{i+j}) \text{ for } i = B,\ldots,1$$

D_1	R_{11}	R_{12}	R_{13}
L_{21}	D_2	R_{21}	R_{22}
L_{31}	L_{32}	D_3	R_{31}
L_{41}	L_{41}	L_{41}	D_4

Hardware Configurations

Karlsruhe Institute of Technology

Host				Device			
CPU	MEM [GB]	BW [GB/s]	H2D [GB/s]	GPU	MEM [GB]	BW [GB/s]	D2H [GB/s]
2x Intel Xeon 4c (E5450) 8 cores	16	8c: 6.14 1c: 2.62	Pa: 1.92 Pi: 5.44	Tesla T10 S1070	4x4	BT: 71.8 daxpy: 83.1 ddot: 83.3	Pa: 1.55 Pi: 3.77
1x Intel Core2 2c (6600) 2 cores	2	2c: 3.28 1c: 3.08	Pa: 1.76 Pi: 2.57	GTX 480	1.5	BT: 108.6 daxpy: 135.0 ddot: 146.7	Pa: 1.38 Pi: 1.82
1x Intel Core i7 4c (920) 4 cores	6	4c: 12.07 1c: 5.11	Pa: 5.08 Pi: 5.64	GTX 280	1.0	BT: 111.5 daxpy: 124.3 ddot: 94.8	Pa: 2.75 Pi: 5.31

Table: CPU and GPU system configuration: Pa/Pi = Pageable/Pinned memory, H2D = host-to-device, D2H = device-to-host, 1c/2c/4c/8c = 1/2/4/8 core(s)

Test Matrices

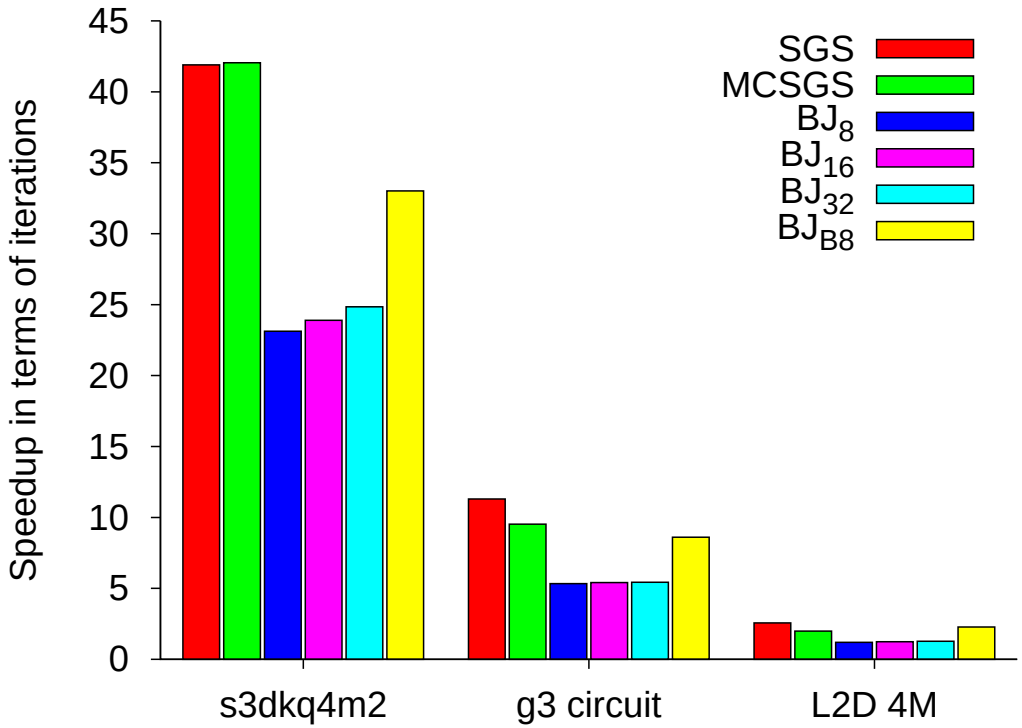
Name	Description of the problem	#rows	#non-zeros	#colors	#block-SpMV in MCSGS
g3 circuit	Circuit simulation	1585478	7660826	4	12
L2D 4M	FEM - Q1 Laplace 2D	4000000	19992000	2	2
s3dkq4m2	FEM - Cylindrical shells	90449	4820891	24	552

Table: Description and properties of test matrices

Impact of Preconditioning

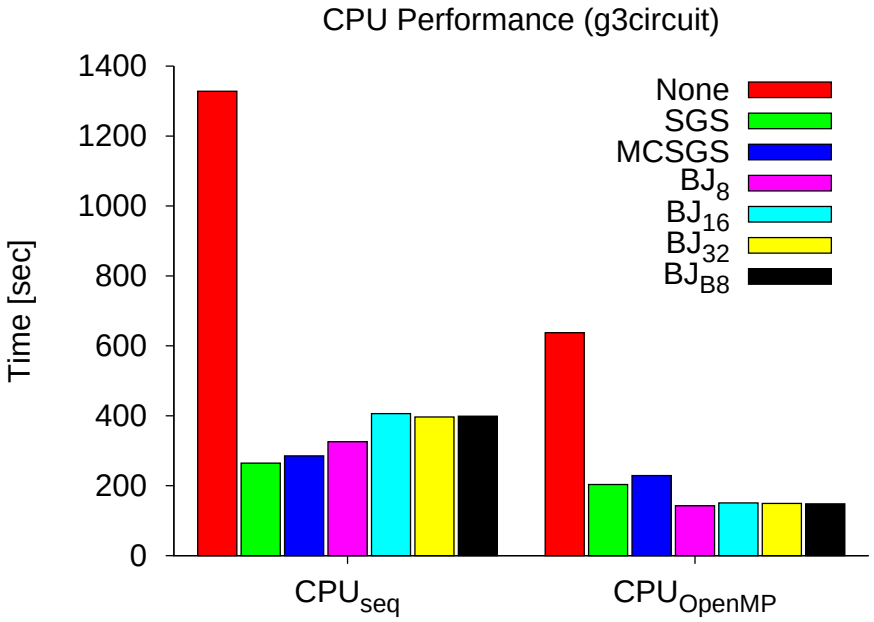
Karlsruhe Institute of Technology

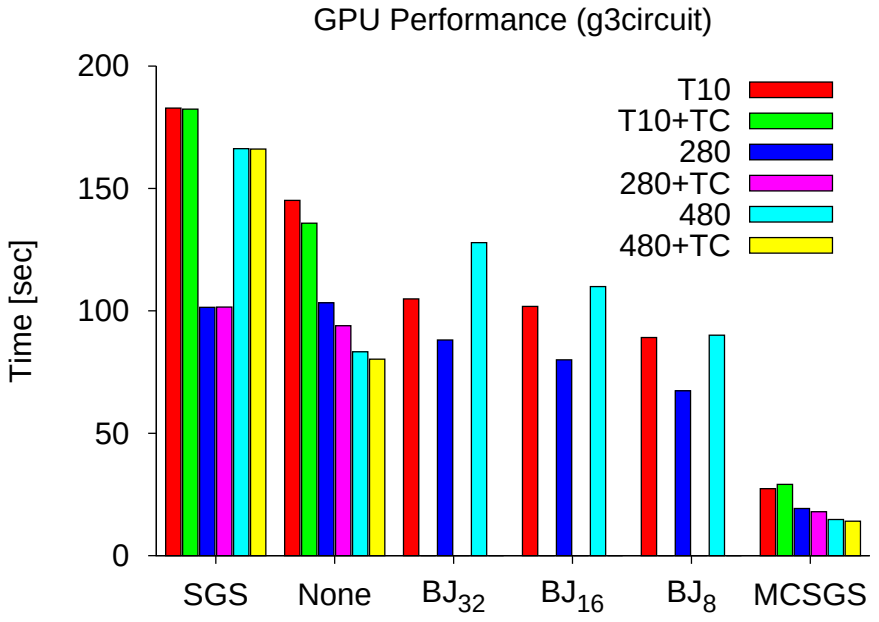
Reduction of number of iterations:



Speedup by preconditioning: ratio of necessary number of iterations of the unpreconditioned system to the necessary number of iterations of the preconditioned system

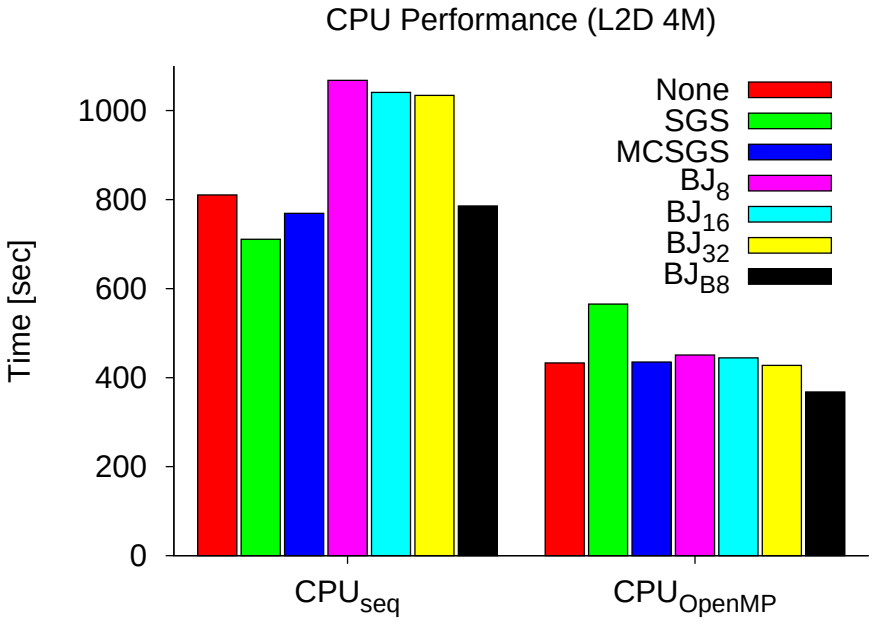
Problem 1: Circuit simulation

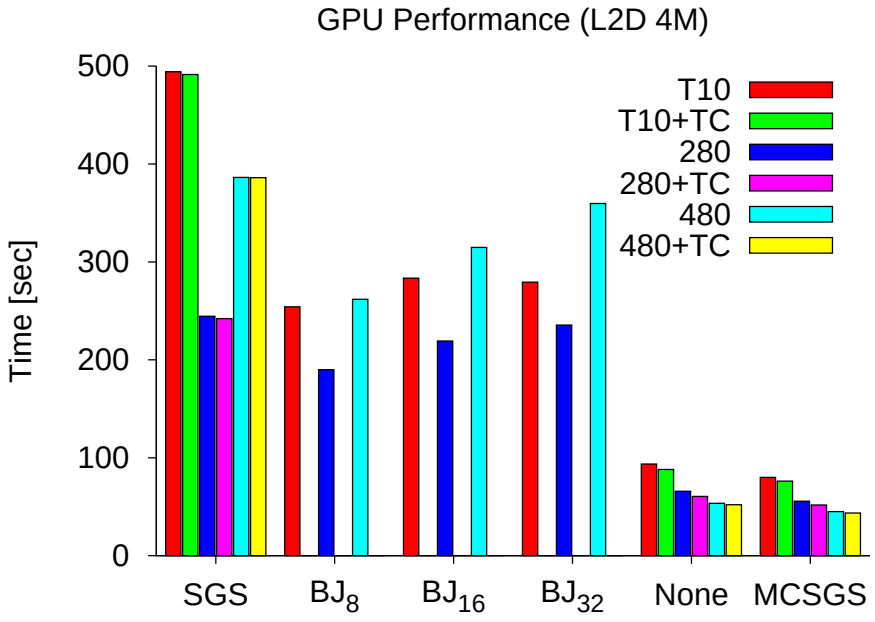




- Matrix color decomposition is imbalanced
- The solver behaves according to platform-specific bandwidth
- The best CPU performance is BJ since the cores are optimized for executing large sequential codes; on the GPU MCSGS is superior

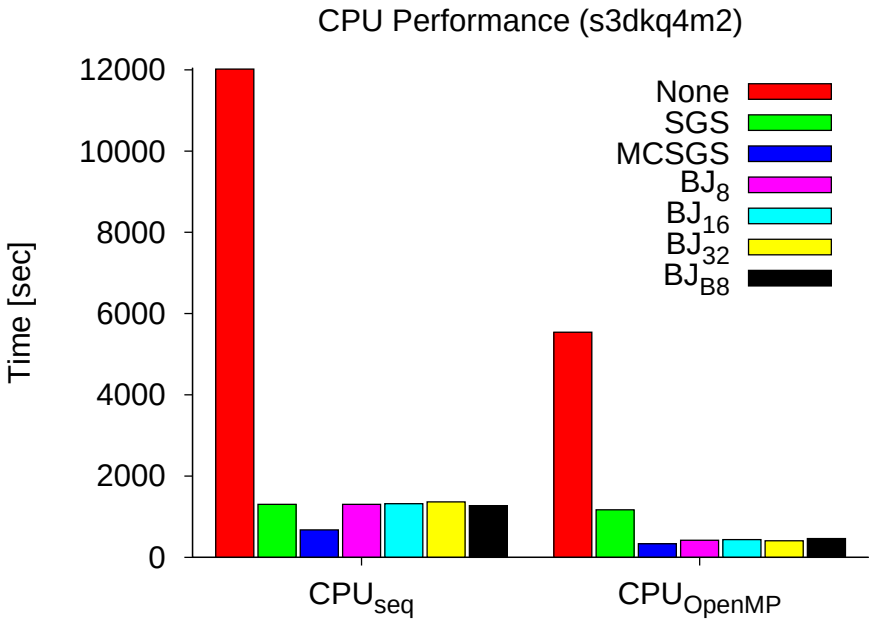
Problem 2: Laplace on regular grids

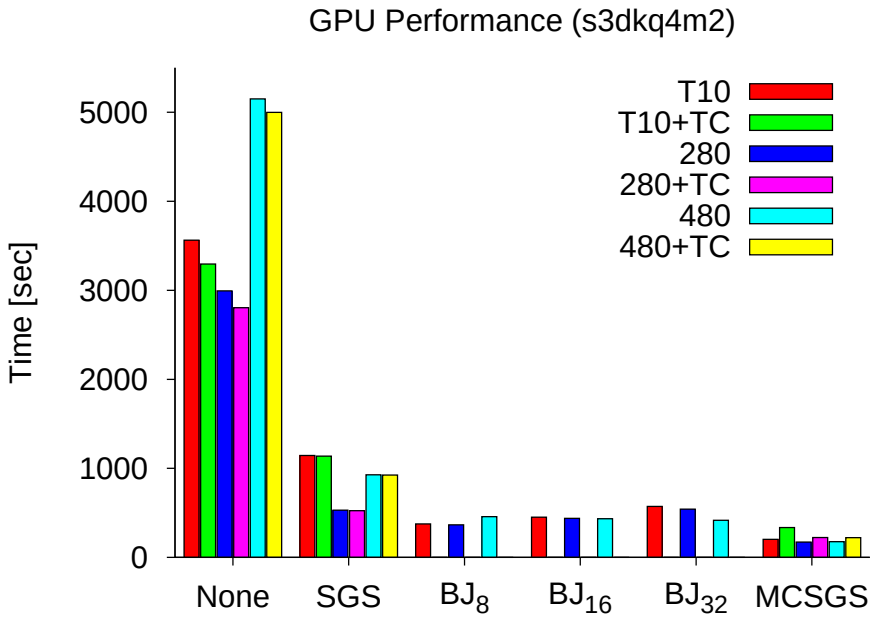




- The matrix color decomposition is balanced
- The solver behaves according to platform-specific bandwidth
- SGS - single core bandwidth utilization and PCIe for the GPU
- Small number of SpMV for MCSGS / texture caching on GPU improves performance

Problem 3: FEM - Cylindrical shells





- The matrix is comparably small: #rows = 90449
- Due to the small matrix size the function calls on the GPU (latency) have a significant impact on the total run time
- Very good cache utilization for MCSGS on the CPU
- Large number of SpMV for MCSGS / impact of texture caching on GPU

Conclusion

- The multi-coloring SGS scheme provides a parallel and scalable preconditioner
- Applicable on multi-core CPUs and GPUs
- Out-of-the box solution for general deployment
- Convincing performance for all test scenarios
- Improved cache utilization
- No bottlenecks due to PCIe connection
- No sequential parts

HiFlow^3

Our preconditioner is part of the ImpLAtoolbox of the HiFlow^3 open source parallel finite element package. It will be available on **http://www.hiflow3.org**

Contact and Acknowledgements

Further Information

- dimitar.lukarski@kit.edu
- http://srg-multicore.math.kit.edu
- http://www.emcl.kit.edu
- http://www.hiflow3.org

The work of the *Shared Research Group* is granted by Hewlett-Packard and the *Concept for the Future* of Karlsruhe Institute of Technology in the framework of the German Excellence Initiative.

Thank you for your attention!