Westfälische
Wilhelms-Universität
Münster

SPPEXA

The EXA-DUNE project

Dune vs. 1 000 000 000 000 000 000

wissen.leben
WWU Münster

Christian Engwer

joint work with P. Bastian, O. Ippisch, J. Fahlke, D. Göddeke, S. Müthing, D. Ribbrock, S. Turek

2.04.2016,
OPM Meeting

German Priority Programme 1648
Software for Exascale Computing

2006 Discussion in the German Research Foundation (DFG) on the necessity of a funding initiative for HPC software.

2010 Discussion with DFG's Executive Committee. Suggestion of a flexible, strategically initiated SPP.
Initiative out of Germany HPC community, referring to increasing activities on HPC software elsewhere (USA: NSF, DOE; Japan; China; G8)

2011 Submission of the proposal, international reviewing, and formal acceptance.

2012 13 full proposals accepted for funding SPPEXA 1.
Review of project sketches and full proposals.

2013 Official start of SPPEXA 1.

2014 Call for proposals SPPEXA 2, incl. international partners (France, Japan)

2015 16 full proposals accepted for funding SPPEXA 2 (7 collaborations with Japan / 3 collaborations with France).

2016 Official start of SPPEXA 2.

2020 Expected arrival of EXA-scale machines ;-)

German Priority Programme 1648
Software for Exascale Computing

2006 Discussion in the German Research Foundation (DFG) on the necessity of a funding initiative for HPC software.

2010 Discussion with DFG's Executive Committee. Suggestion of a flexible, strategically initiated SPP.
Initiative out of Germany HPC community, referring to increasing activities on HPC software elsewhere (USA: NSF, DOE; Japan; China; G8)

2011 Submission of the proposal, international reviewing, and formal acceptance.

2012 13 full proposals accepted for funding SPPEXA 1.
Review of project sketches and full proposals.

2013 Official start of SPPEXA 1.

2014 Call for proposals SPPEXA 2, incl. international partners (France, Japan)

2015 16 full proposals accepted for funding SPPEXA 2 (7 collaborations with Japan / 3 collaborations with France).

2016 Official start of SPPEXA 2.

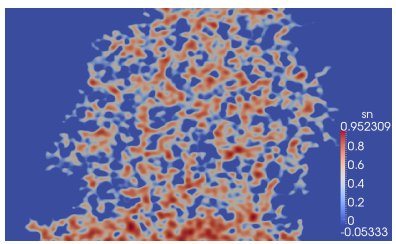
2020 Expected arrival of EXA-scale machines ;-)

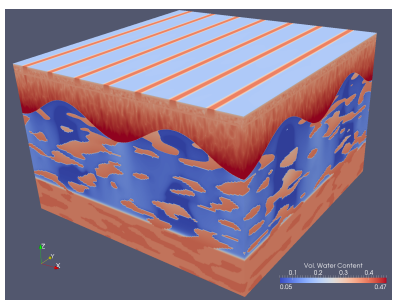
6 years HPC related project funding

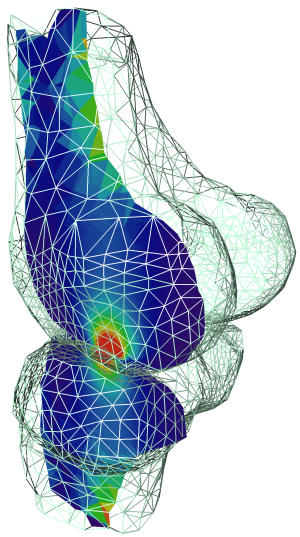
EXA-DUNE

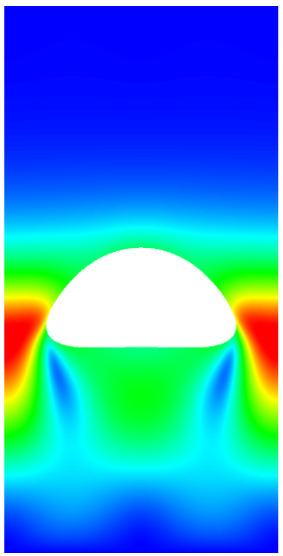
DUNE's Framework approach to software development

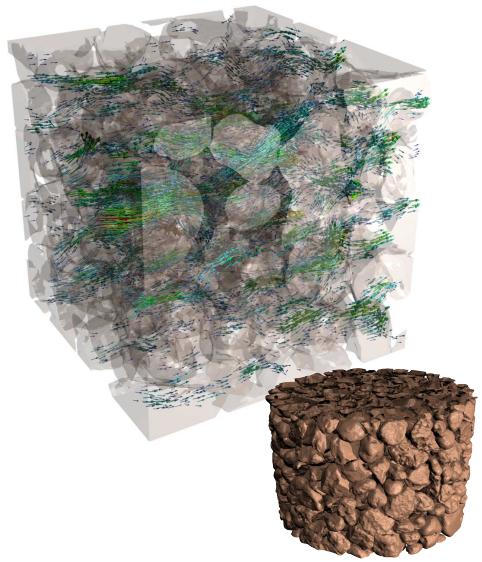
- Integrated toolbox of simulation components
- Existing body of complex applications
- Good performance + scalability for traditional MPI model







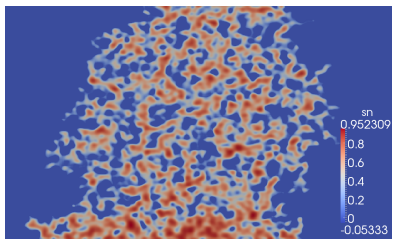


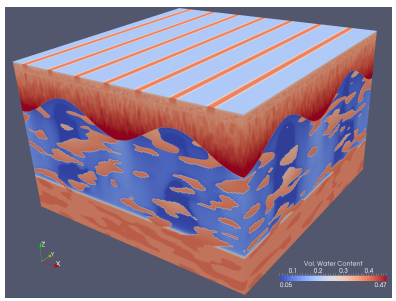


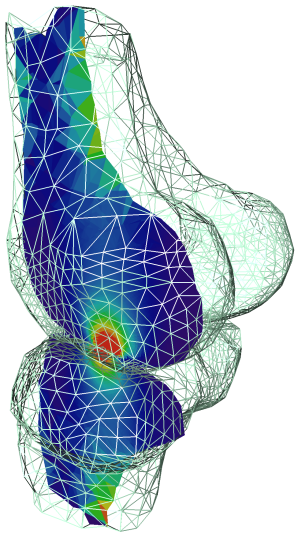
EXA-DUNE

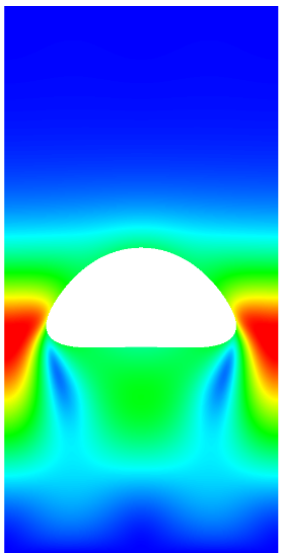
Challenges:

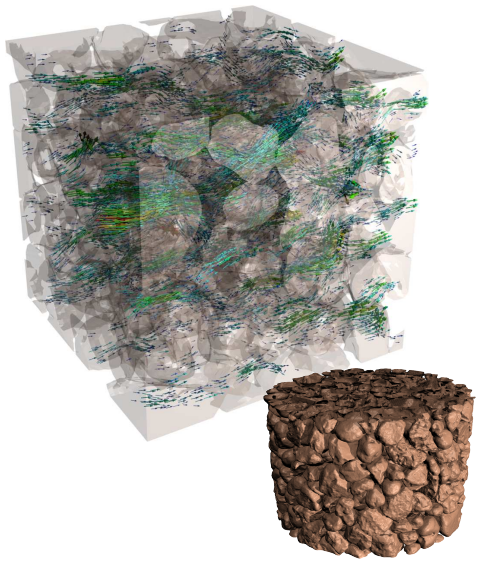
- Standard low order algorithms do not scale any more
- Incorporate new algorithms, hardware paradigms
- Integrate changes across simulation stages (Ahmdahl's Law)
- Provide "reasonable" upgrade path for existing applications







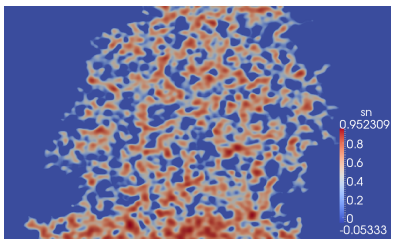


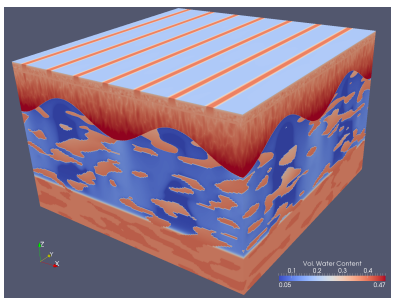


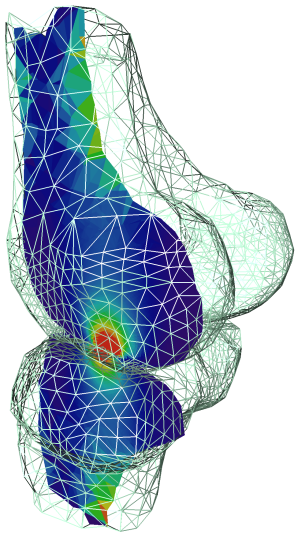
EXA-DUNE

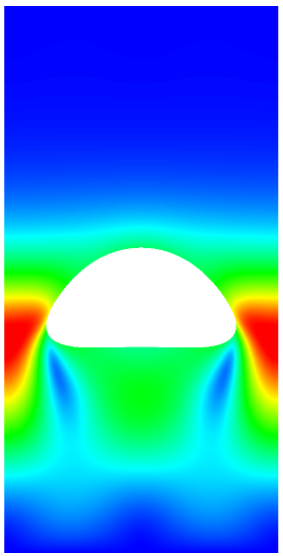
DUNE + FEAST = Flexibility + Performance

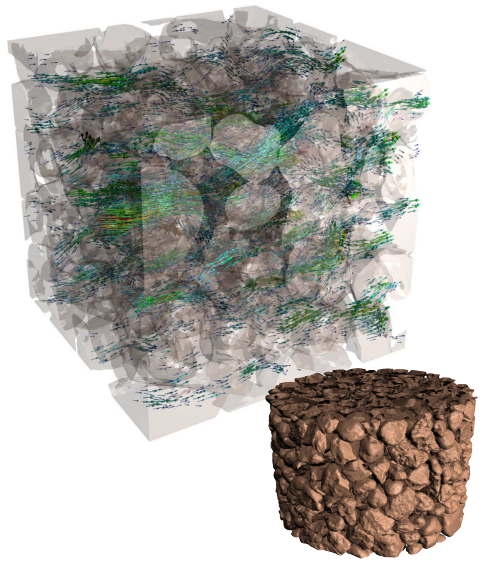
- General Software Frameworks
  - → co-designed to specific hardware platforms is not sufficient
- Hardware-Oriented Numerics
  - → design/choose algorithms with hardware in mind







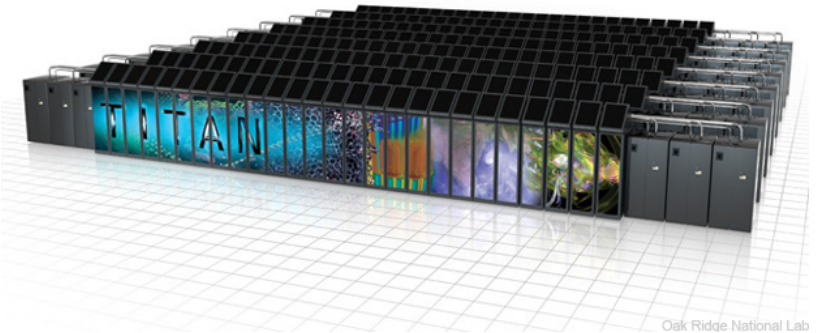




wissen.leben WWU Münster

Hardware Challenges

- Multiple Levels of concurrency
- MPI-parallel, Multi-core, SIMD
- Memory-wall



Hardware Challenges

- Multiple Levels of concurrency
- MPI-parallel, Multi-core, SIMD
- Memory-wall

Example Intel Xeon E5-2698v3 (Haswell)

- Advertised peak performance: 486.4 GFlop/s

Hardware Challenges

- Multiple Levels of concurrency
- MPI-parallel, Multi-core, SIMD
- Memory-wall

Example Intel Xeon E5-2698v3 (Haswell)

- Advertised peak performance: 486.4 GFlop/s
- 16 cores $\rightarrow$ single Core: 30.4 GFlop/s

Hardware Challenges

- Multiple Levels of concurrency
- MPI-parallel, Multi-core, SIMD
- Memory-wall

Example Intel Xeon E5-2698v3 (Haswell)

- Advertised peak performance: 486.4 GFlop/s
- 16 cores $\rightarrow$ single Core: 30.4 GFlop/s
- AVX2+FMA $\rightarrow$ without FMA: 15.2 GFlop/s

Hardware Challenges

- Multiple Levels of concurrency
- MPI-parallel, Multi-core, SIMD
- Memory-wall

Example Intel Xeon E5-2698v3 (Haswell)

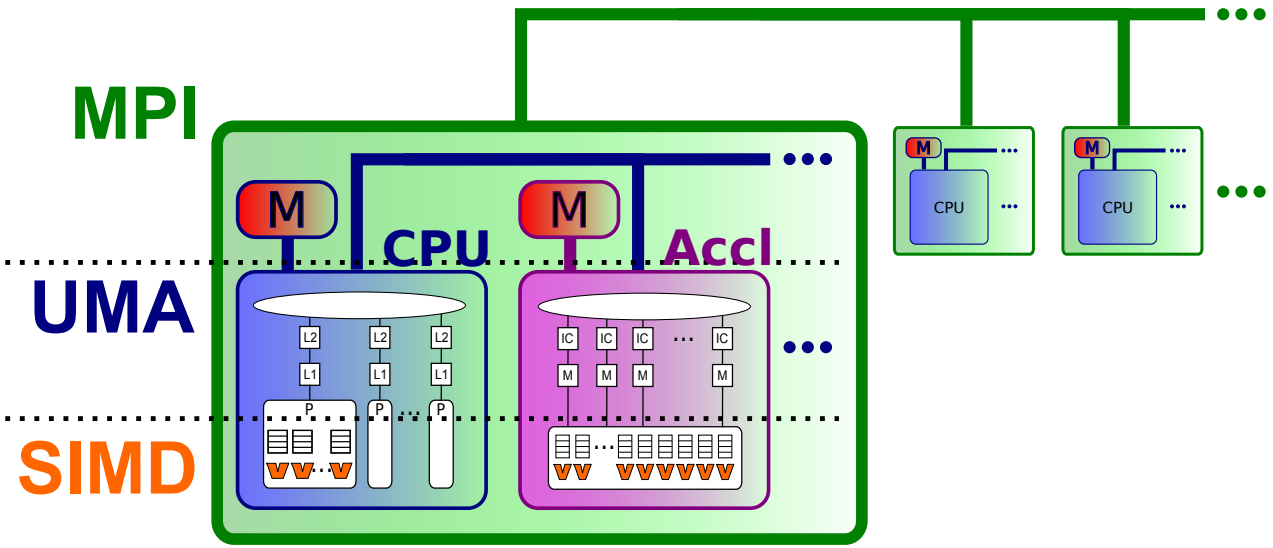
- Advertised peak performance: 486.4 GFlop/s
- 16 cores → single Core: 30.4 GFlop/s
- AVX2+FMA → without FMA: 15.2 GFlop/s
- 4× SIMD → without AVX: 3.8 GFlop/s

→ classic, non-parallel code bound by 3.8 GFlop/s

→ you loose 99% Performance

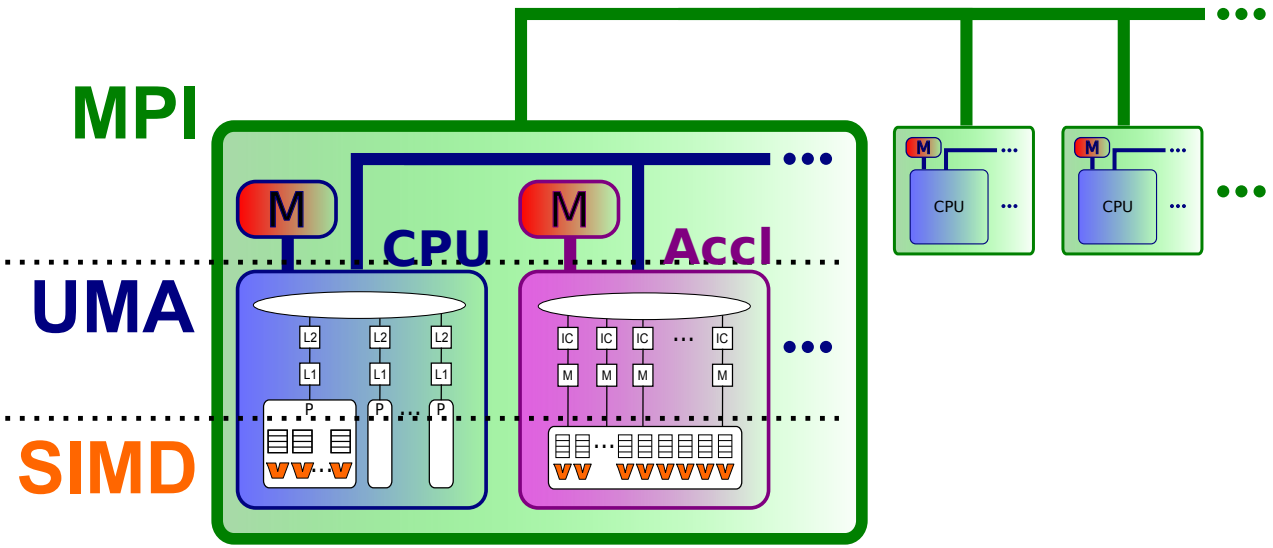
Outline

Hybrid Parallelism



- Coarse-grained: MPI between heterogeneous nodes
- Medium-grained: multicore-CPUs, GPUs, MICs, APUs, ...
- Fine-grained: vectorization, GPU 'threads', ...

Hybrid Parallelism



- **Coarse-grained**: MPI between heterogeneous nodes
- **Medium-grained**: multicore-CPUs, GPUs, MICs, APUs, . . . TBB
- **Fine-grained**: vectorization, GPU ‘threads’, . . . ???

Algorithm Design
Hardware-Oriented Numerics

Challenge: Combine flexibility and hardware efficiency

- Existing codes no longer run faster automatically
- Standard low order algorithms do not scale any more
- *Much more than a pure implementational issue*

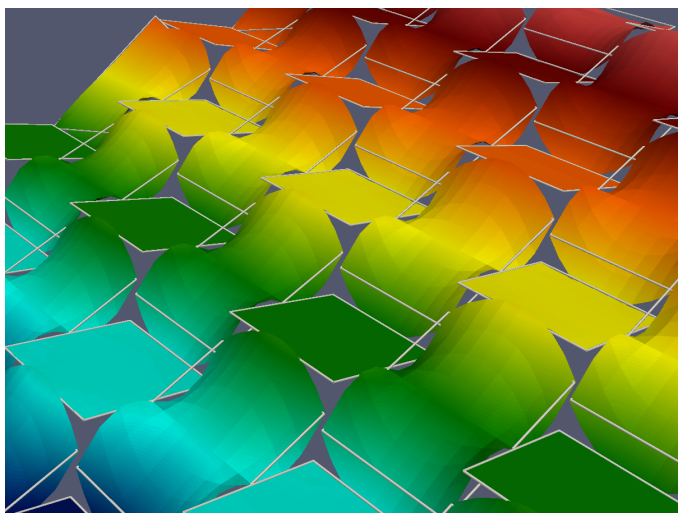
Algorithm Design
Hardware-Oriented Numerics

Challenge: Combine flexibility and hardware efficiency

Solution: Locally structured, globally unstructured data
$\rightarrow$ *increased algorithmic intensity and vectorization*

Concepts:

- higher order:
  Discontinuous Galerkin, Reduced Basis
- virtual local refinement:
  global unstructured mesh, local tensor-prduct mesh
- multiple right-hand-sides:
  horizontal SIMD-vectorization



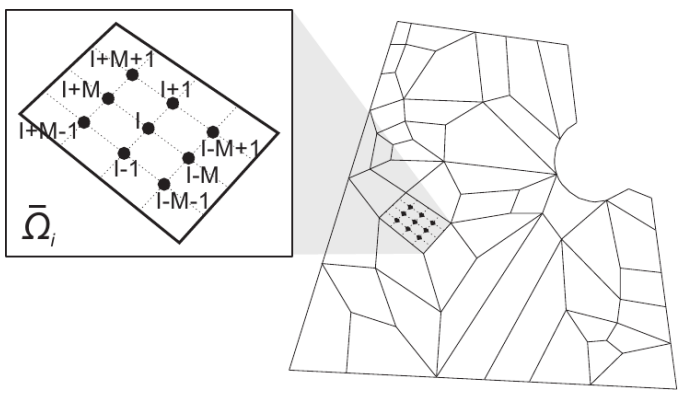
Algorithm Design
Hardware-Oriented Numerics

Challenge: Combine flexibility and hardware efficiency

Solution: Locally structured, globally unstructured data
$\rightarrow$ *increased algorithmic intensity and vectorization*

Concepts:

- higher order:
  Discontinuous Galerkin, Reduced Basis
- virtual local refinement:
  global unstructured mesh, local tensor-prduct mesh
- multiple right-hand-sides:
  horizontal SIMD-vectorization



Algorithm Design
Hardware-Oriented Numerics

Challenge: Combine flexibility and hardware efficiency

Solution: Locally structured, globally unstructured data
$\rightarrow$ *increased algorithmic intensity and vectorization*

Concepts:

- higher order:
  Discontinuous Galerkin, Reduced Basis
- virtual local refinement:
  global unstructured mesh, local tensor-prduct mesh
- multiple right-hand-sides:
  horizontal SIMD-vectorization

$$A \cdot \begin{pmatrix} x_{0,0} & x_{1,0} & \dots & x_{N,0} \\ x_{0,1} & x_{1,1} & \dots & x_{N,1} \\ x_{0,2} & x_{1,2} & \dots & x_{N,2} \\ & \vdots & & \vdots \\ x_{0,M} & x_{1,M} & \dots & x_{N,M} \end{pmatrix} = \begin{pmatrix} b_{0,0} & b_{1,0} & \dots & b_{N,0} \\ b_{0,1} & b_{1,1} & \dots & b_{N,1} \\ b_{0,2} & b_{1,2} & \dots & b_{N,2} \\ & \vdots & & \vdots \\ b_{0,M} & b_{1,M} & \dots & b_{N,M} \end{pmatrix}$$

Features in EXA-DUNE

dune-common

- Multi-Threading support in DUNE (via TBB)

dune-grid

- Hybrid parallelization of DUNE grids (on-top of the grid interface)
- Virtual-refinement of unstructured DUNE grids

dune-istl

- Hybrid parallelization
- Performance portable Matrix format
- (vertically) vectorized preconditioners (incl. CUDA versions)
- (horizontally) vectorized solvers (multiple RHS)

dune-pdelab

- Sum-Factorization DG
- Special-purpose high-performance assemblers

Features in EXA-DUNE

dune-common

- Multi-Threading support in DUNE (via TBB)

dune-grid

- Hybrid parallelization of DUNE grids (on-top of the grid interface)
- Virtual-refinement of unstructured DUNE grids

dune-istl

- Hybrid parallelization
- Performance portable Matrix format
- (vertically) vectorized preconditioners (incl. CUDA versions)
- (horizontally) vectorized solvers (multiple RHS)

dune-pdelab

- Sum-Factorization DG
- Special-purpose high-performance assemblers

SELL-C-σ as Cross-Platform Matrix Format

Kreutzer, Hager, Wellein, Fehske, Bishop '13

Block-sorted and chunked ELL (SELL-C-σ) with suitable tuning offers competitive performance across modern architectures (CPU, GPGPU, Xeon Phi)

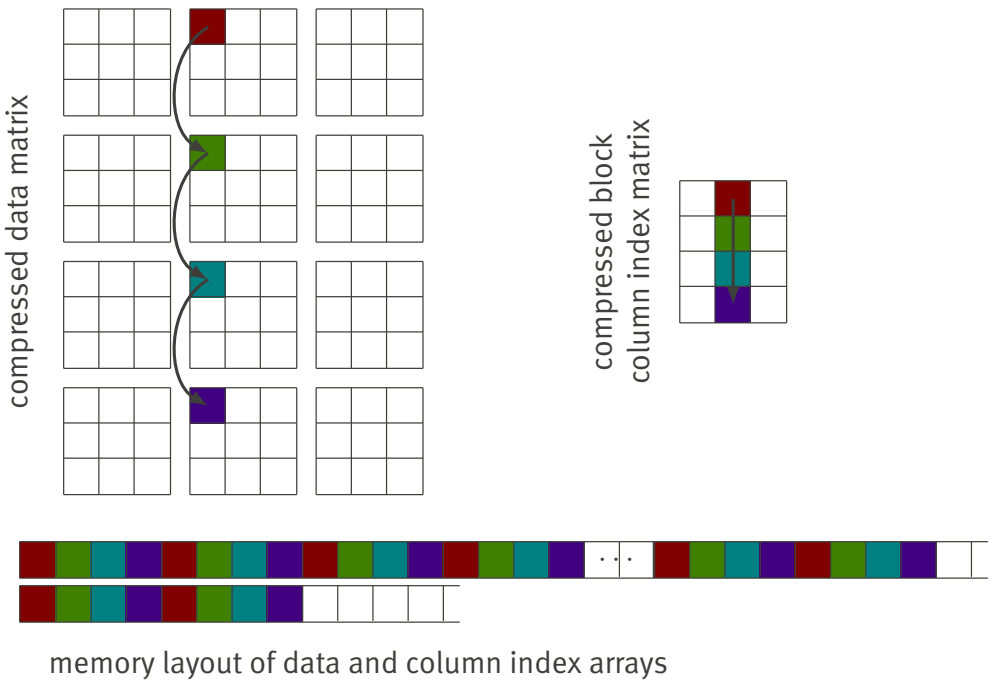
- Adopt SELL-C-σ as unified matrix format in DUNE *(including assembly phase $\rightarrow$ dune-pdelab)*
- Differentiate by memory domain (host, Xeon Phi, GPU)
- Memory domain and C via extended C++ allocator interface

Blocked SELL Format for DG discretizations

Choi, Singh, Vuduc '10

Interleaved storage of blocks from C block rows

- Move SIMD from scalar level to block level
- Vectorize algorithms by operating on *multiple independent blocks* simultaneously
- Tradeoff between vectorization and cache usage at larger block sizes

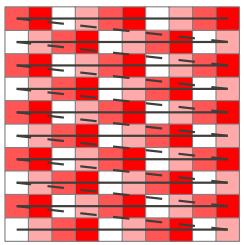


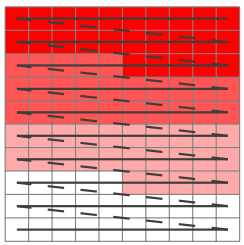
[Muething et.al. 2013]

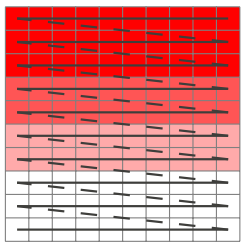
Hybrid-parallel DUNE Grid

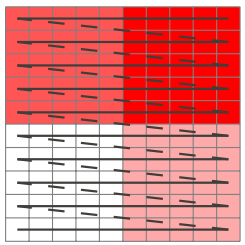
Strategies for multi-threaded assembly of matrices and vectors

- Thread parallelization on-top of the DUNE grid interface
- Evaluation of partitioning strategies:


strided


ranged


sliced


tensor

- ... and data access strategies:
  - batched: Batched writeback with global lock.
  - elock: One lock per mesh entity
  - coloring: partitions of the same color do not “touch”.

  other strategies not considered here:
  - global locking → as bad as expected.
  - race-free schemes → in general not possible not possible.

Evaluation

Evaluation of partitioning strategies

- Best partitioning strategy: ranges of cells
  → memory efficient and fast

... and data access strategies

- Best strategy: entity-wise locking (no benefit from coloring)

k	E_{10}^{CPU}	E_{20}^{CPU}	E_{60}^{PHI}	E_{120}^{PHI}	E_{240}^{PHI}
0	62%	42%	75%	43%	21%
1	62%	42%	84%	57%	30%
2	72%	46%	90%	69%	38%
3	79%	50%	92%	72%	41%
4	87%	49%			
5	88%	51%			

Runtimes per dof, degree k, jacobian, sliced partitioning, entity-wise locking

Evaluation: 60-core Xeon PHIs and 10-core CPU's

- Full benefit of Xeon PHI will require vectorization of user code
  → vectorization: non-trivial, work in progress

Evaluation

Evaluation of partitioning strategies

- Best partitioning strategy: ranges of cells
  → memory efficient and fast

... and data access strategies

- Best strategy: entity-wise locking (no benefit from coloring)

k	E_{10}^{CPU}	E_{20}^{CPU}	E_{60}^{PHI}	E_{120}^{PHI}	E_{240}^{PHI}
0	62%	42%	75%	43%	21%
1	62%	42%	84%	57%	30%
2	72%	46%	90%	69%	38%
3	79%	50%	92%	72%	41%
4	87%	49%			
5	88%	51%			

Runtimes per dof, degree k, jacobian, sliced partitioning, entity-wise locking

Evaluation: 60-core Xeon PHIs and 10-core CPU's

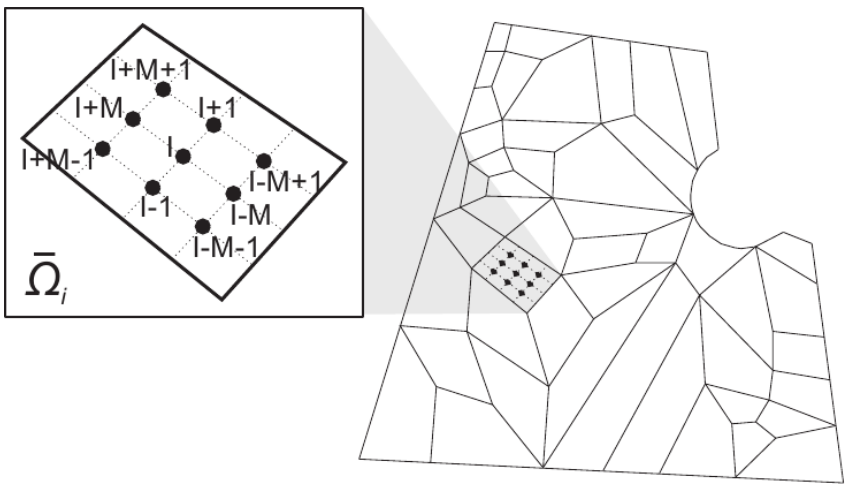
- Full benefit of Xeon PHI will require vectorization of user code
  → vectorization: non-trivial, work in progress

wissen.leben WWU Münster

Vectorizing over Elements

User provides local Operator to compute local Stiffness matrix

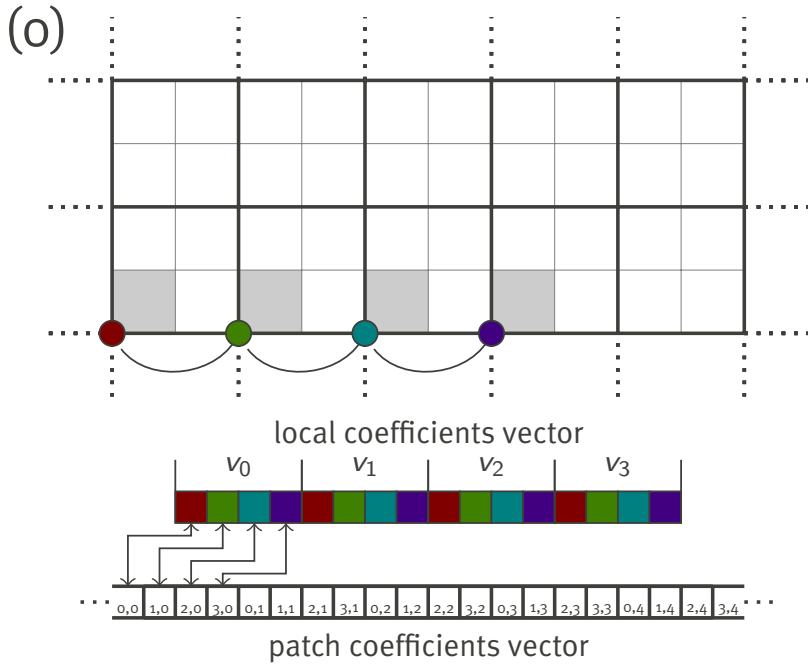
- Patch Grid
- reduce costs of unstructured meshes
  - extract subset of mesh
  - store as flat grid
  - store in consecutive arrays, without pointers
- add structured refinement
  - exploit local structure
  - improve data locality



Vectorizing over Elements

User provides local Operator to compute local Stiffness matrix

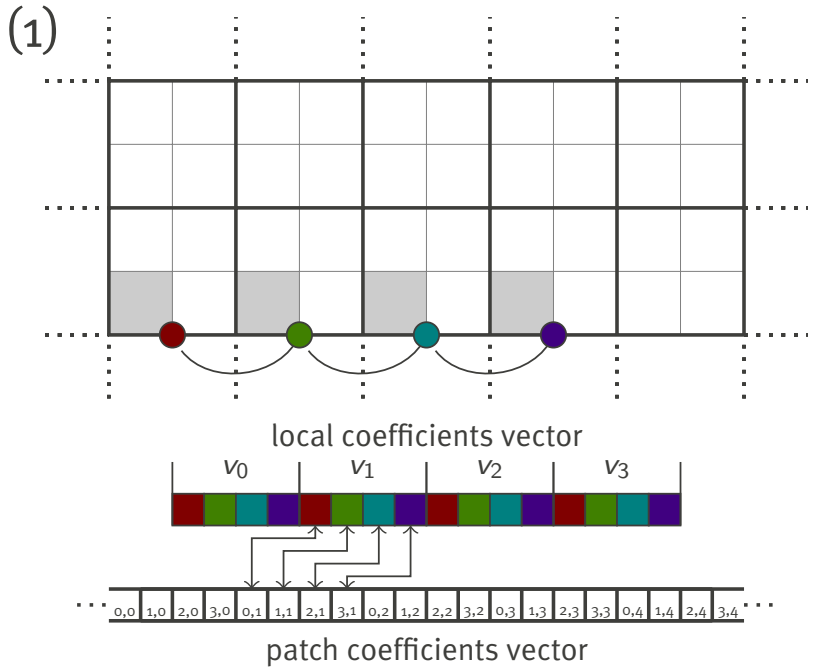
- Patch Grid
- reduce costs of unstructured meshes
  - extract subset of mesh
  - store as flat grid
  - store in consecutive arrays, without pointers
- add structured refinement
  - exploit local structure
  - improve data locality



Vectorizing over Elements

User provides local Operator to compute local Stiffness matrix

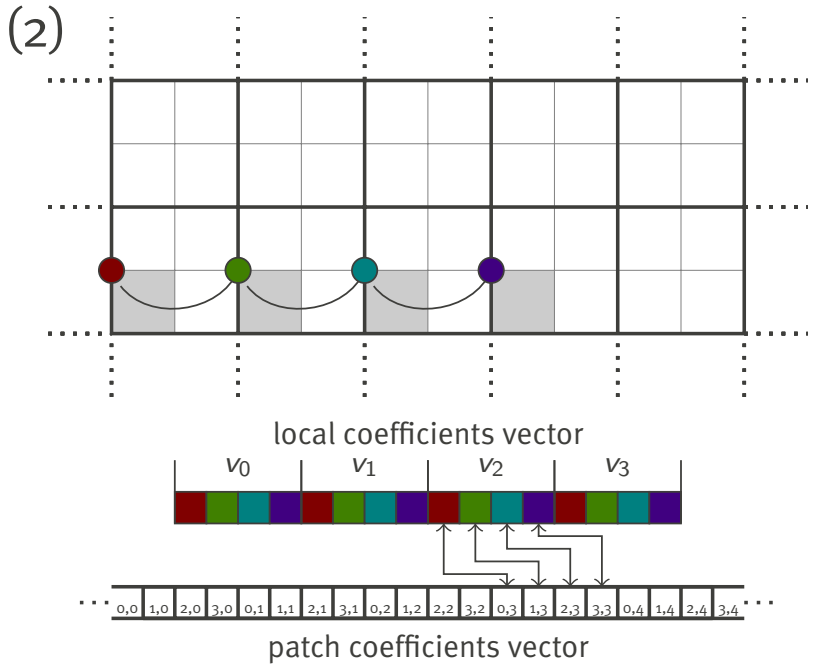
- Patch Grid
- reduce costs of unstructured meshes
  - extract subset of mesh
  - store as flat grid
  - store in consecutive arrays, without pointers
- add structured refinement
  - exploit local structure
  - improve data locality



Vectorizing over Elements

User provides local Operator to compute local Stiffness matrix

- Patch Grid
- reduce costs of unstructured meshes
  - extract subset of mesh
  - store as flat grid
  - store in consecutive arrays, without pointers
- add structured refinement
  - exploit local structure
  - improve data locality

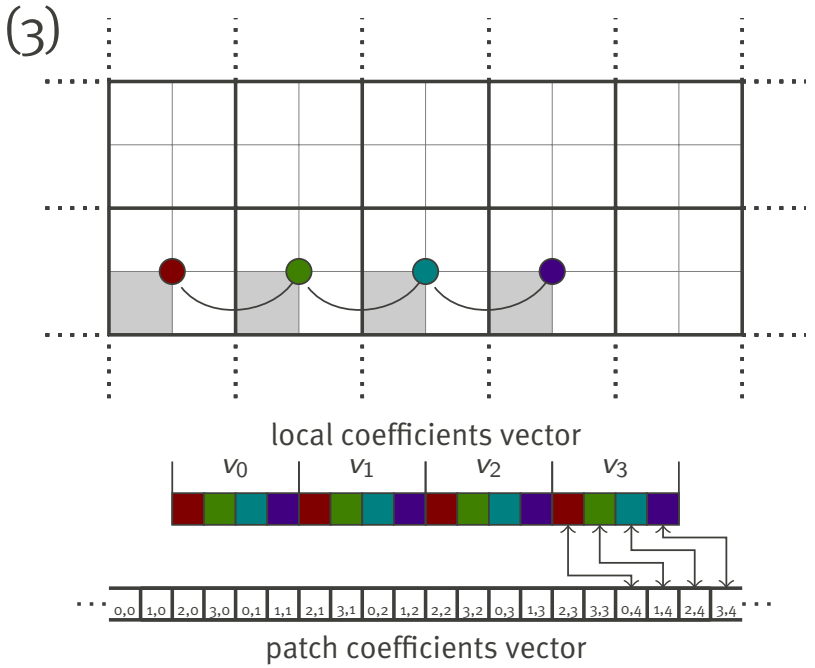


wissen.leben WWU Münster

Vectorizing over Elements

User provides local Operator to compute local Stiffness matrix

- Patch Grid
- reduce costs of unstructured meshes
  - extract subset of mesh
  - store as flat grid
  - store in consecutive arrays, without pointers
- add structured refinement
  - exploit local structure
  - improve data locality

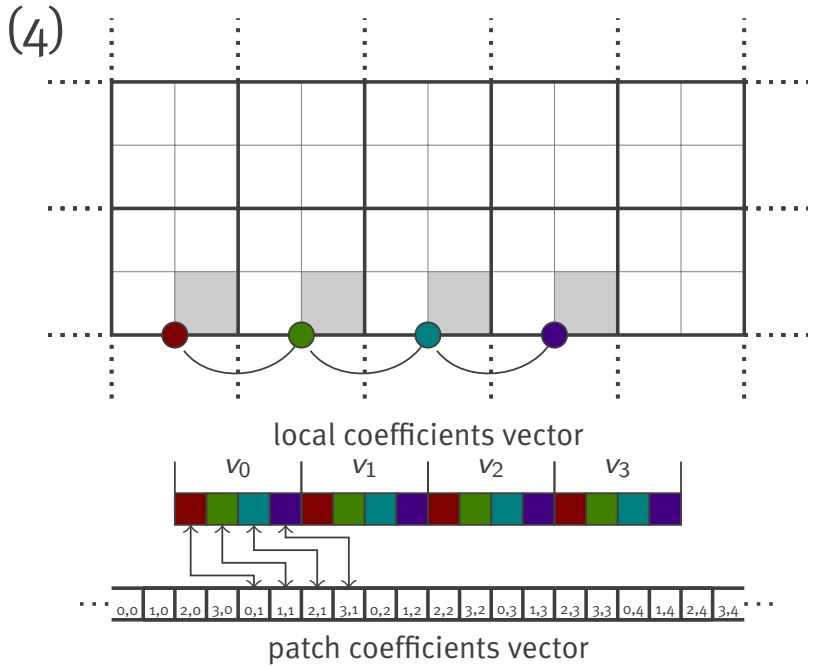


wissen.leben WWU Münster

Vectorizing over Elements

User provides local Operator to compute local Stiffness matrix

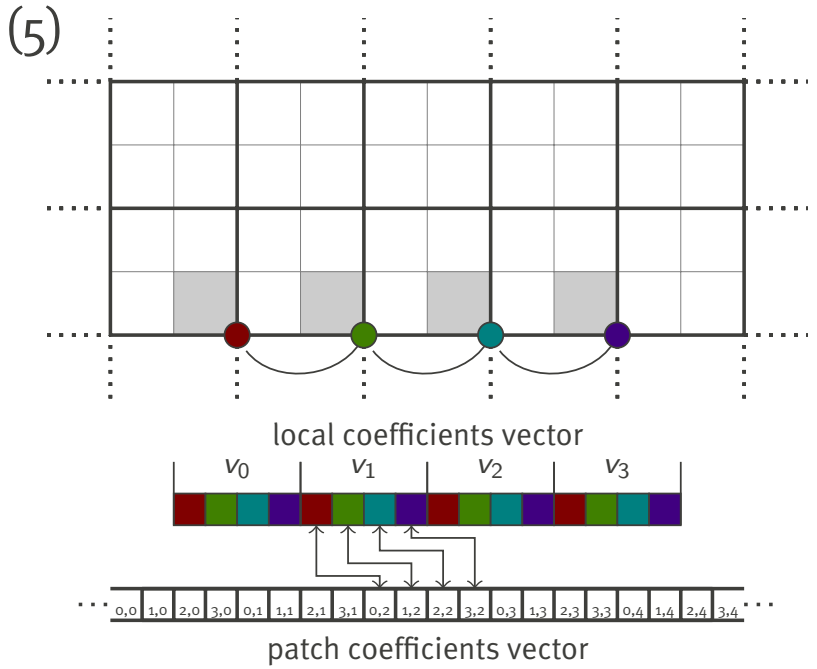
- Patch Grid
- reduce costs of unstructured meshes
  - extract subset of mesh
  - store as flat grid
  - store in consecutive arrays, without pointers
- add structured refinement
  - exploit local structure
  - improve data locality



Vectorizing over Elements

User provides local Operator to compute local Stiffness matrix

- Patch Grid
- reduce costs of unstructured meshes
  - extract subset of mesh
  - store as flat grid
  - store in consecutive arrays, without pointers
- add structured refinement
  - exploit local structure
  - improve data locality



wissen.leben WWU Münster

Vectorizing over Elements (2)

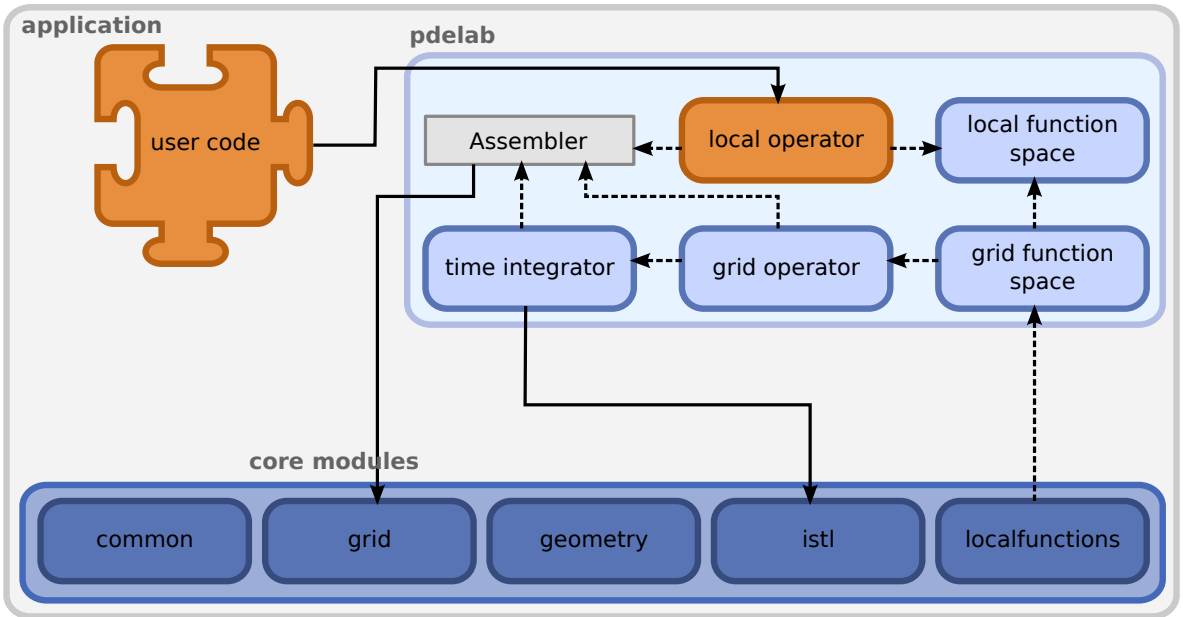
$$-\Delta u = 0 \qquad \text{on } \Omega + \text{BC}$$

- Conforming FEM, Lagrange, Q_1, three level virtual refinement
- 16-Core Intel Haswell E5-2698v3
  - Advertised 486.4 GFlop/s,
  - Peak w.o. FMA: 243.2 GFlop/s ($\rightarrow$ %avail)
  - Bandwidth: STREAM 2×40 GByte/s

SIMD	lanes	threads	GByte/s	%effBW	GFlop/s	%avail
-	1	16	26.73	33.4	9.758	4.0
-	1	32	34.32	42.9	12.53	5.2
AVX	4	16	62.89	78.8	22.96	9.4
AVX	4	32	73.01	91.3	26.65	11.0

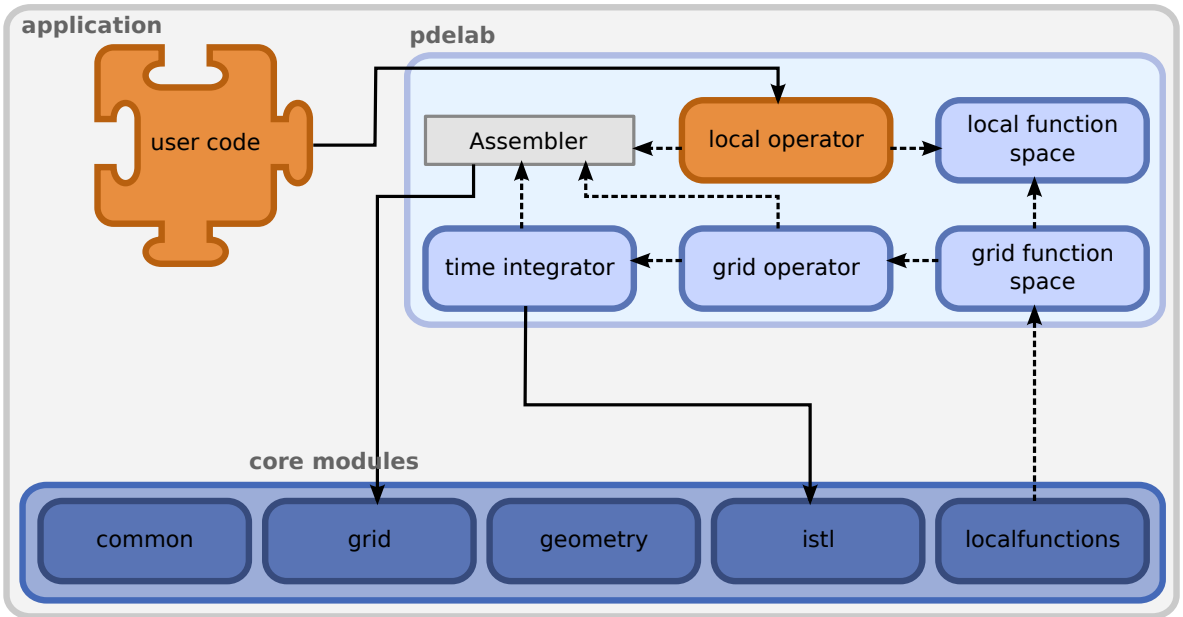
Interface Challenges

Vectorization of local operator (user code)



Interface Challenges

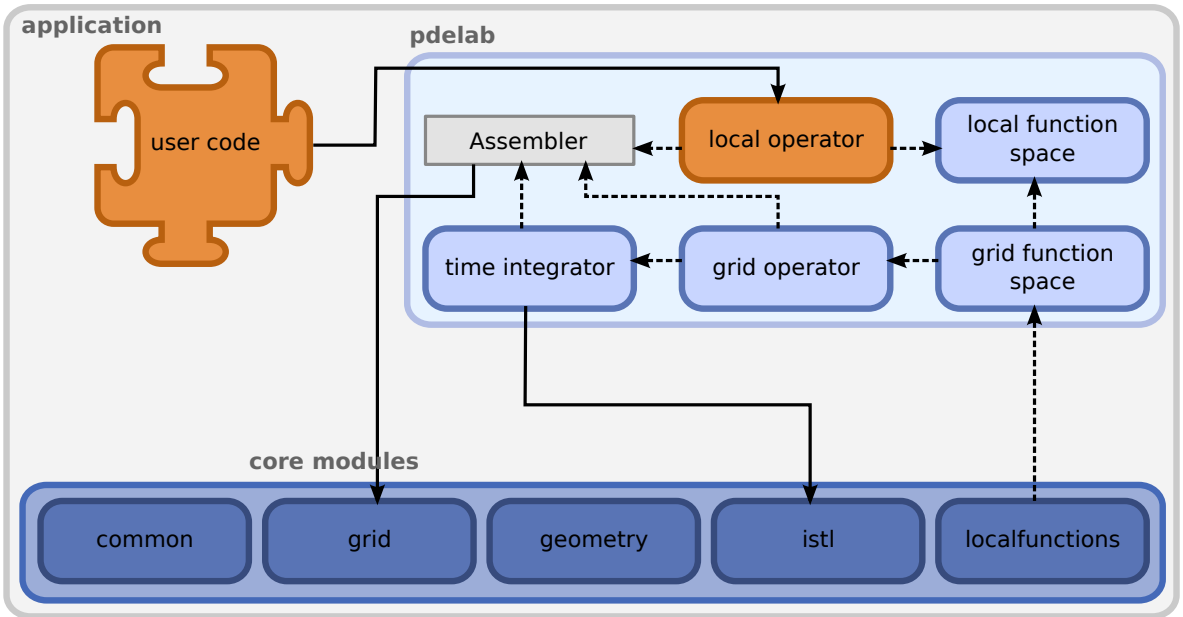
Vectorization of local operator (user code)



- Intrinsics (non-portable)
- Special language (needs special compiler)
- Autovectorizer (difficult to drive portably)

Interface Challenges

Vectorization of local operator (user code)



- Intrinsics (non-portable)
- Special language (needs special compiler)
- Autovectorizer (difficult to drive portably)

→ Vectorization library (e.g. Vc, Boost.SIMD, NGSolve)

Programming Approaches

- Intrinsics
  (non-portable)
- Special language
  (needs special compiler)
- Autovectorize
  (difficult to drive portably)

Programming Approaches

- Intrinsics
  (non-portable)
- Special language
  (needs special compiler)
- Autovectorize
  (difficult to drive portably)

Programming Approaches

- Intrinsics
  (non-portable)
- Special language
  (needs special compiler)
- Autovectorize
  (difficult to drive portably)
- vectorization library
  - Hide instrinsics beneath a portable interface
  - Implementations: e.g. Vc, Boost.SIMD, NGSolve

Features of Vc

Storage: `Vc::SimdArray<T,N>`

Abstracts a vector of type `T` with size `N`

SIMD Operator overloads: $+, -, \cdot, /$, etc.

```
template <typename T>
SimdArray<T,N> operator*(SimdArray<T,N> a, SimdArray<T,N> b)
{
    for (std::size_t i=0; i<N; i++) a[i] *= b[i];
    return a;
}
```

Features of Vc

Storage: `Vc::SimdArray<T,N>`

Abstracts a vector of type `T` with size `N`

SIMD Operator overloads: $+, -, \cdot, /$, etc.

```
template <typename T>
SimdArray<T,N> operator*(SimdArray<T,N> a, SimdArray<T,N> b)
{
    for (std::size_t i=0; i<N; i++) a[i] *= b[i];
    return a;
}
```

Limitations:

Not all operations are supported on SIMD data types
In particular

- No implicit cast from `float` $\rightarrow$ `double`
- No branching, e.g. `if(c) x=a else x=b`
  Alternative: `x = c?a:b;` $\rightarrow$ `x = cond(c,a,b);`

Horizontal Vectorization for multiple RHS

- Many applications require solves for many right-hand-sides
  - Multi-Scale FEM
  - Inverse problems
  - ...

Horizontal Vectorization for multiple RHS

- Many applications require solves for many right-hand-sides
  - Multi-Scale FEM
  - Inverse problems
  - ...
- This corresponds to

$$\text{foreach} \quad i \in [0, N] : \text{solve } Ax_i = b_i \qquad \rightarrow \qquad \text{solve } AX = B$$

with $X = (x_0, ..x_N)$, $B = (b_0, ..b_N)$

Horizontal Vectorization for multiple RHS

- Many applications require solves for many right-hand-sides
  - Multi-Scale FEM
  - Inverse problems
  - ...
- This corresponds to

$$\text{foreach} \quad i \in [0, N] : \text{solve } Ax_i = b_i \qquad \rightarrow \qquad \text{solve } AX = B$$

  with $X = (x_0, ..x_N)$, $B = (b_0, ..b_N)$
- Templated FEM-code allows to implement vectorized assembly and solvers, e.g.

```
template<typename T> Dune::BlockVector      and
template<class X> class BiCGSTABSolver : public InverseOperator<X,X>
template<class M, class X, class Y> class AssembledLinearOperator
```

Modifications to the linear Algebra code

```
Dune::BlockVector<double>
```

$$\rightarrow$$

```
Dune::BlockVector<Vc::SimdArray<double,N>>
```

$$X = \begin{pmatrix} x_{0,0} & x_{1,0} & x_{2,0} & \cdots & x_{N,0} \\ x_{0,1} & x_{1,1} & x_{2,1} & \cdots & x_{N,1} \\ x_{0,2} & x_{1,2} & x_{2,2} & \cdots & x_{N,2} \\ & \vdots & & & \vdots \\ x_{0,M} & x_{1,M} & x_{2,M} & \cdots & x_{N,M} \end{pmatrix}$$

Memory layout: Corresponds to $M \times N$ dense-matrix with row-major storage.

Modifications to the linear Algebra code

```
Dune::BlockVector<double>
```

$$\rightarrow$$

```
Dune::BlockVector<Vc::SimdArray<double,N>>
```

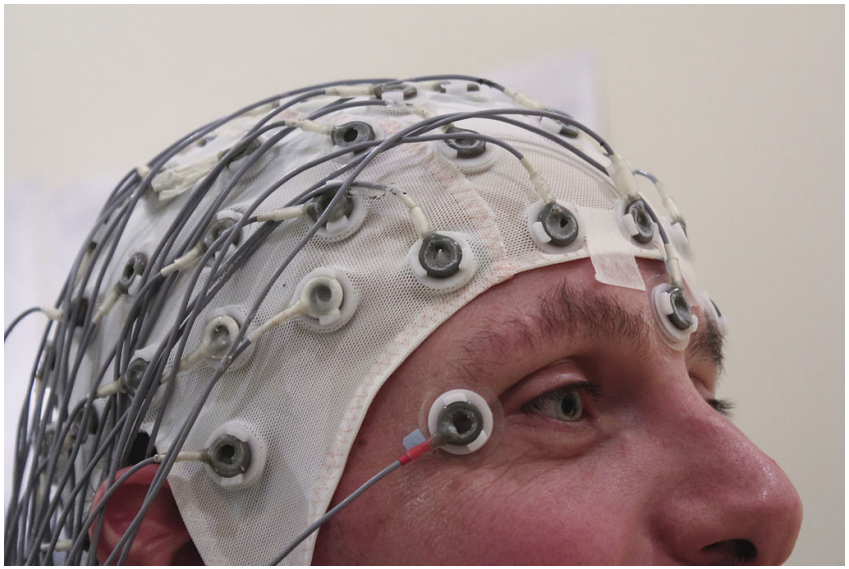
$$X = \begin{pmatrix} x_{0,0} & x_{1,0} & x_{2,0} & \cdots & x_{N,0} \\ x_{0,1} & x_{1,1} & x_{2,1} & \cdots & x_{N,1} \\ x_{0,2} & x_{1,2} & x_{2,2} & \cdots & x_{N,2} \\ & \vdots & & & \vdots \\ x_{0,M} & x_{1,M} & x_{2,M} & \cdots & x_{N,M} \end{pmatrix}$$

Memory layout: Corresponds to $M \times N$ dense-matrix with row-major storage.

And smaller SIMD-aware modifications to the Solvers

Application: EEG Source Reconstruction

Cooperation UKM (Münster), BESA GmbH (München)



Source: Wikipedia

- EEG measurements
- $>$ 200 electrodes
- measure surface potential
- governing equation

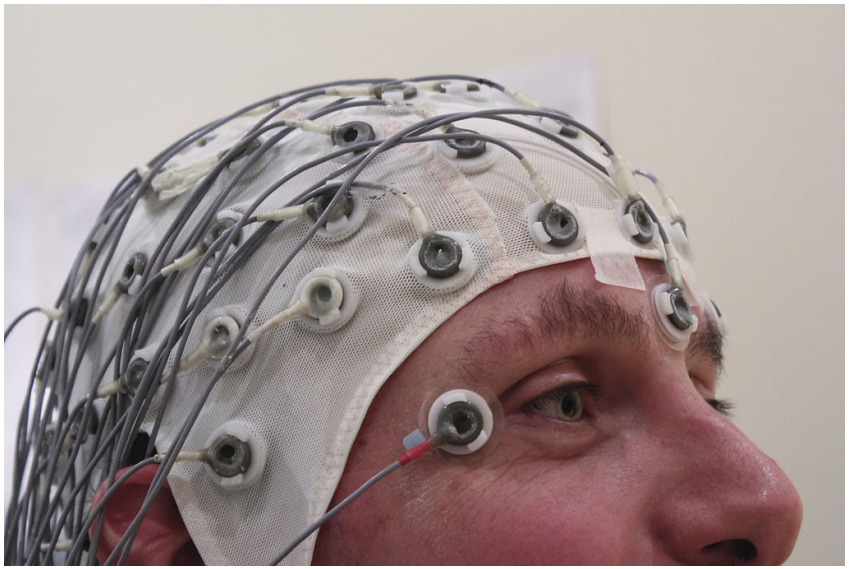
$$-\nabla \cdot K \nabla u = f \qquad \text{on } \Omega$$
$$\nabla u \cdot n = 0 \qquad \text{on } \partial\Omega$$

Goal: reconstruct brain activity

- Inverse modelling approach
- L_2 regulerization
- Fidelity term on potential at electrodes

Application: EEG Source Reconstruction

Cooperation UKM (Münster), BESA GmbH (München)



Source: Wikipedia

- EEG measurements
- $>$ 200 electrodes
- measure surface potential
- governing equation

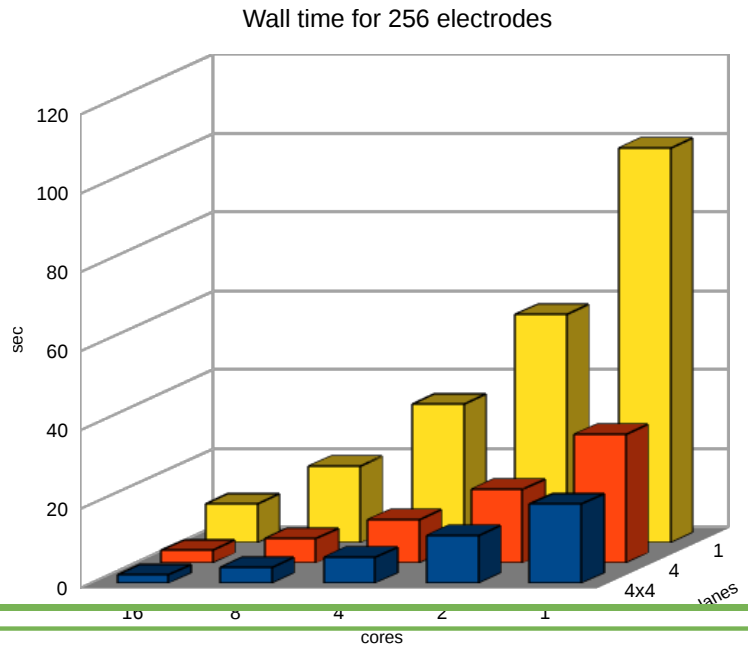
$$-\nabla \cdot K \nabla u = 0 \qquad \text{on } \Omega$$
$$\nabla u \cdot n = j \qquad \text{on } \partial\Omega$$

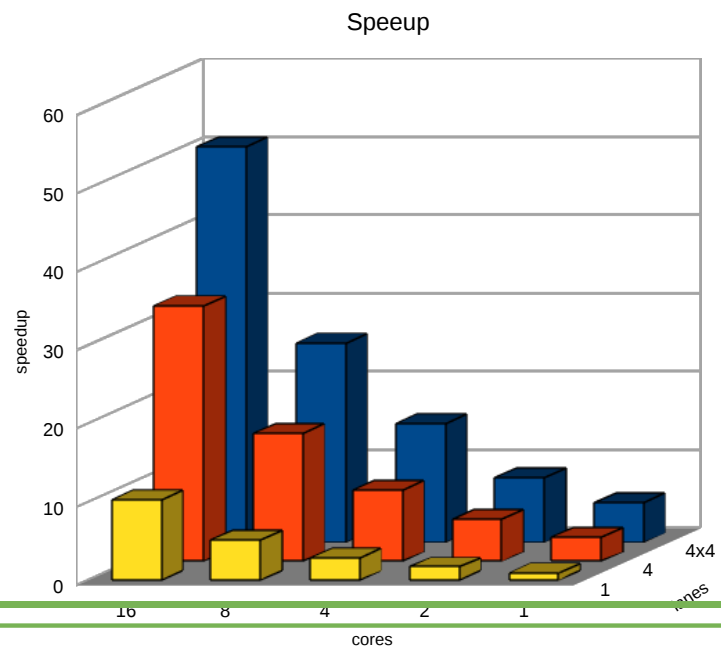
Goal: reconstruct brain activity

- Inverse modelling approach
- L_2 regulerization
- Fidelity term on potential at electrodes

Performance Measurements

- SIMD-vectorized AMG-CG solver
- Solve for 256 RHS, 300K Cells, 60K Vertices
- Timing for Haswell-EP (E5-2698v3, 16 cores, AVX2, 4 lanes)
  - 1-16 cores
  - no SIMD, AVX (4 lanes), AVX (4×4 lanes)
  - speedup 50 (max theoretical speedup 64)





Outlook

Transition from EXA-DUNE to mainline?!

dune-common

- Multi-Threading support in DUNE (via TBB)

dune-grid

- Hybrid parallelization of DUNE grids (on-top of the grid interface)
- Virtual-refinement of unstructured DUNE grids

dune-istl

- Hybrid parallelization
- Performance portable Matrix format
- (vertically) vectorized preconditioners (incl. CUDA versions)
- (horizontally) vectorized solvers (multiple RHS)

dune-pdelab

- Sum-Factorization DG
- Special-purpose high-performance assemblers

Outlook

Transition from EXA-DUNE to mainline?!

dune-common

X Multi-Threading support in DUNE (via TBB)

dune-grid

X Hybrid parallelization of DUNE grids (on-top of the grid interface)

~ Virtual-refinement of unstructured DUNE grids

dune-istl

X Hybrid parallelization

? Performance portable Matrix format

? (vertically) vectorized preconditioners (incl. CUDA versions)

\+ (horizontally) vectorized solvers (multiple RHS)

dune-pdelab

\+ Sum-Factorization DG

~ Special-purpose high-performance assemblers

Outlook II
second phase

- Push features upstream
- Task-parallel scheduling
- Asynchronicity $\rightarrow$ latency hiding, communication hiding, ...
- Resilience
- ...

Outlook II
second phase

- Push features upstream
- Task-parallel scheduling
- Asynchronicity $\rightarrow$ latency hiding, communication hiding, ...
- Resilience
- ...

Questions?!.