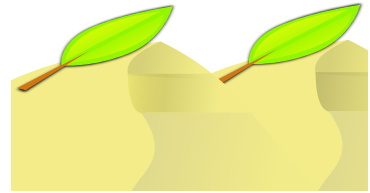


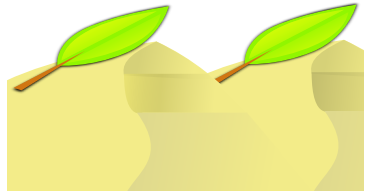
Massively Parallel Algebraic Multigrid Solvers

Markus Blatt

Dr. Markus Blatt
HPC-Simulation-Software & Services

Universität Stuttgart
September 18, 2012

Outline



Past Evolution of Parallel Computers

Helics I (2003)

- 256 nodes
- Dual AMD Athlon 1,4 GHz
- 5.9 GFLOPS peak/node
- 1GB main memory/node
- Myrinet 2 Gbit

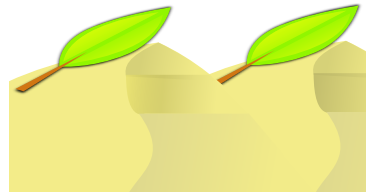
Helics II (2007)

- 156+4 nodes
- 2x Dual Core AMD Opteron 2220 2.8 GHz
- 18.8 GFLOPS peak/node
- 8 GB RAM/node (21.4 GB/s)
- Myricom 10Gbit

Blue Gene/P (2009)

- 73728 nodes
- PowerPC 450 Quad-core 850Mhz
- 13,6 GFLOPS peak/node
- 2 GB RAM/node (13.6 GB/s)

Observations in Parallel Computing

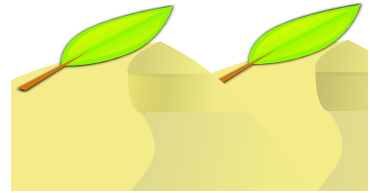


Software

- Solution of time dependent (nonlinear) equations with implicit time stepping schemes.
- Most time consuming: Solution of linear system.
- Peak GFLOPS out of reach!
- Methods are limited by memory bandwidth.

Hardware

- Costs for compute power drop fast (2002: 12 USD/MFLOP, 2011: .01 USD/MFLOP)
- Costs for main memory drop only slightly.
- Main memory not power efficient.



Current Hardware/Software Trends

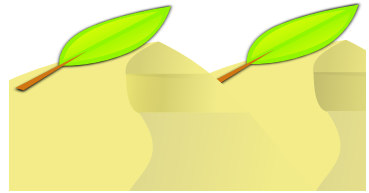
The hardware manufacturers solution (Green Computing)

- More cores per node
- Less main memory per core.
- SIMD (Blue Gene/Q, GPGPU)
- Increase GFLOPS per GB/s main memory speed
- Faster network interconnects.

How software has to cope

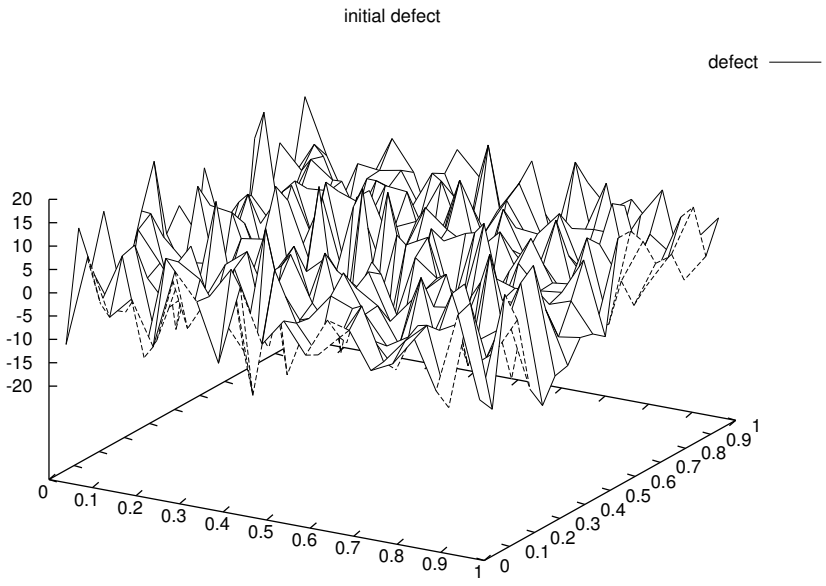
- Memory efficiency!
- Minimize communication!
- Favor less convergent iterative methods!
- Time to solution / scalability matters most!

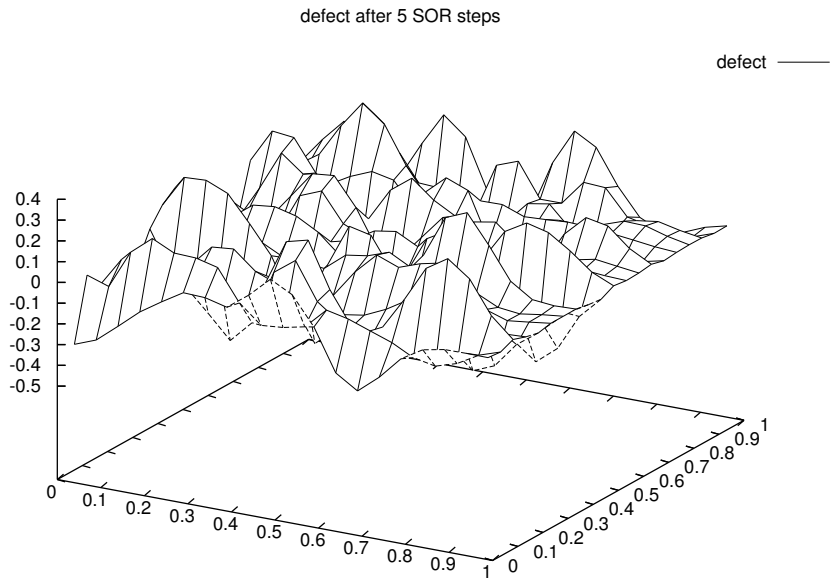
Software always two steps behind.

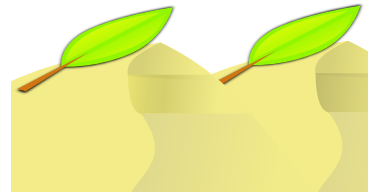


Problem with Stationary Iterative Methods

- Error reduction stagnates after some iterations
- Reduces only high frequency errors





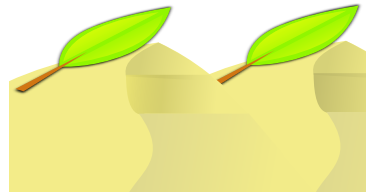


Remedy: Multigrid

- approximate smooth residual on a coarser grid and solve there.
- calculate a correction there
- interpolate correction to the fine grid and add it to the current guess

Algorithm

```
mgm(A_l, x_l, b_l){
  if(l == 0) x_0 = A_0^{-1} b_0; // direct solve
  else{
    x_l = S^{ν1}(x_l, b_l);
    b_{l-1} = R_{l-1}(b_l - A_l x_l);
    x_{l-1} = 0;
    for(int i = 0; i < γ, i++) mgm(A_{l-1}, x_{l-1}, b_{l-1});
    x_l = x_l + P_{l-1} x_{l-1};
    x_l = S^{ν2}(x_l, b_l);
  }
}
```

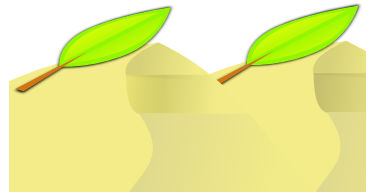


Remedy: Multigrid

- approximate smooth residual on a coarser grid and solve there.
- calculate a correction there
- interpolate correction to the fine grid and add it to the current guess

Algorithm

```
mgm(A_l, x_l, b_l){
   if(l == 0) x_0 = A_0^{-1} b_0; // direct solve
   else{
      x_l = S^{ν1}(x_l, b_l);
      b_{l-1} = R_{l-1}(b_l - A_l x_l);
      x_{l-1} = 0;
      for(int i = 0; i < γ, i++) mgm(A_{l-1}, x_{l-1}, b_{l-1});
      x_l = x_l + P_{l-1} x_{l-1};
      x_l = S^{ν2}(x_l, b_l);
   }
}
```



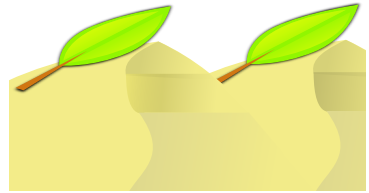
Drawbacks of Geometric Multigrid

- Hierarchy of grid $\Omega_0 \subset \Omega_1 \subset \ldots \subset \Omega_l$ needed
- Coarsening depends on problem, e.g. semi coarsening for

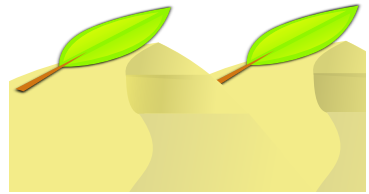
$$\nabla \cdot \begin{pmatrix} 1 & 0 \\ 0 & 10^{-3} \end{pmatrix} \nabla u = f$$

- Coarsening and according prolongation and restriction hard for unstructured meshes

Algebraic Multigrid



- Use algebraic nature of the problem to define MG Components
- Only the fine grid is needed
- No grid hierarchy exists but a hierarchy of matrices
- Solver decoupled from grid
- Coarsening adapts to problem and grid automatically
- Setup phase needed for creating coarse matrices



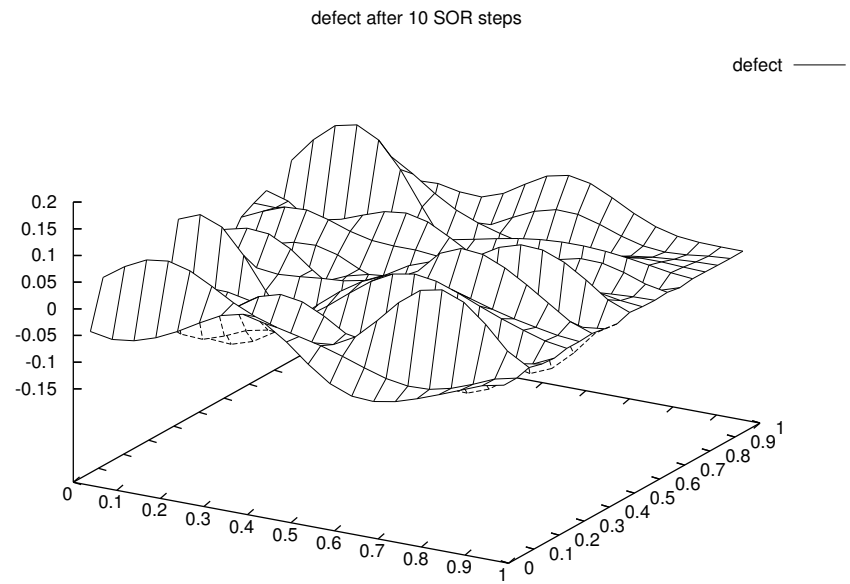
Algebraically Smooth Error

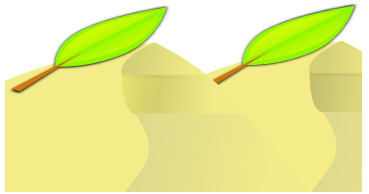
- The error e is called *algebraically smooth* if it is not sufficiently reduced by applying the smoothing operator S, i. e. $Se \approx e$
- Smoothing is best along strong connections(j, i):

$$\frac{\bar{a_{ij}}\,\bar{a_{ji}}}{a_{ii}\,a_{jj}} > \alpha\gamma_i \text{ , with } \gamma_i = \max_{j\neq i} \frac{\bar{a_{ij}}\,\bar{a_{ji}}}{a_{ii}\,a_{jj}} \text{ , } \bar{a_{ij}} = \max\{0, -a_{ij}\}$$

$$\frac{a_{ij}\,a_{ji}}{a_{ii}\,a_{jj}} > \alpha \max_{j\neq i} \frac{a_{ij}^2}{a_{ii}\,a_{jj}}$$

- Algebraically smooth error is not necessarily geometrically smooth:

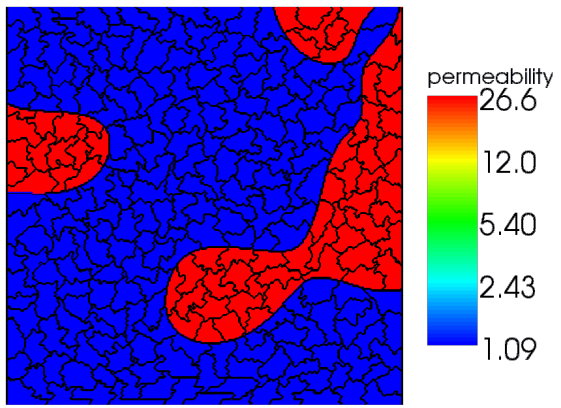




Aggregation AMG

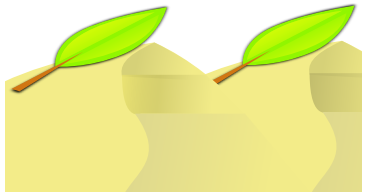
Simple, non-smoothed version

- Piecewise constant prolongators P_l.
- Heuristic and greedy aggregation algorithm.
- $A_{l-1} = P_l^T A_l P_l$
- Proposed by Raw, Vanek et al., Braess
- Preconditioner for Krylov methods.



Observations

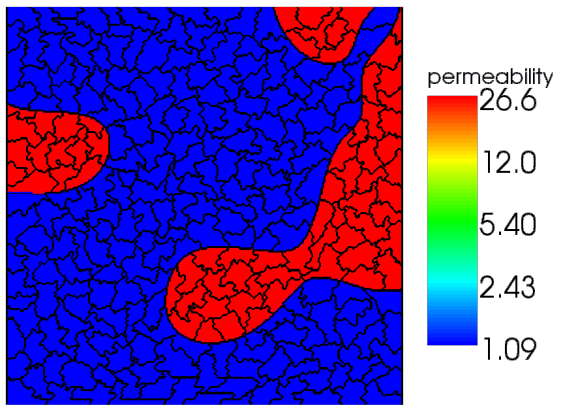
- Reasonable coarse grid operator for systems.
- Preserves FV discretization.
- Very memory efficient.
- Fast and scalable V-cycle.



Aggregation AMG

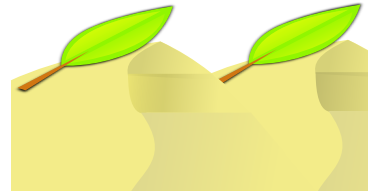
Simple, non-smoothed version

- Piecewise constant prolongators P_l.
- Heuristic and greedy aggregation algorithm.
- $A_{l-1} = P_l^T A_l P_l$
- Proposed by Raw, Vanek et al., Braess
- Preconditioner for Krylov methods.



Observations

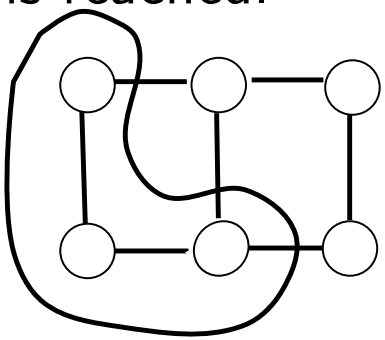
- Reasonable coarse grid operator for systems.
- Preserves FV discretization.
- Very memory efficient.
- Fast and scalable V-cycle.



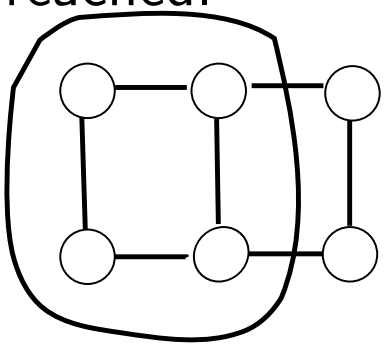
Greedy Aggregation Algorithm

While there unaggregated (non-isolated nodes), build aggregate:

1. Add neighbours with the most strong connections until minimum size is reached.

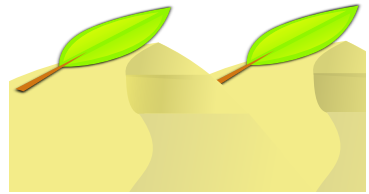


2. Rounding step: Add vertices that have more strong connections to the aggregate than to other vertices/aggregates until maximum size is reached.



Keep maximum travel distance within an aggregate below threshold.

Parallelization Approach



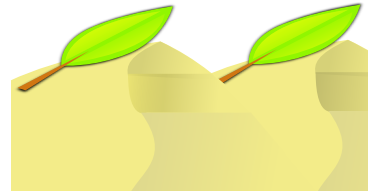
Goals

- Reuse efficient sequential data structures and algorithms for aggregation and matrix-vector-products.
- Agglomerate data onto fewer processes on coarser levels.
- Smoother is hybrid Gauss-Seidel.

Approach

- Separate decomposition and communication information from data structures.
- Use simple and portable index identification for data items.
- Data structures need to be augmented to contain ghost items.

Parallelization Approach



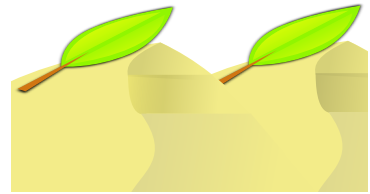
Goals

- Reuse efficient sequential data structures and algorithms for aggregation and matrix-vector-products.
- Agglomerate data onto fewer processes on coarser levels.
- Smoother is hybrid Gauss-Seidel.

Approach

- Separate decomposition and communication information from data structures.
- Use simple and portable index identification for data items.
- Data structures need to be augmented to contain ghost items.

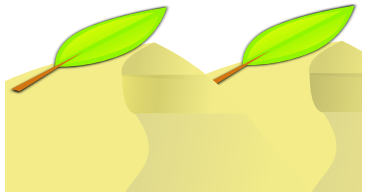
Index Sets



Index Set

- Distributed overlapping index set $I = \bigcup_0^{P-1} I_p$
- Process p manages mapping $I_p \longrightarrow [0, n_p)$.
- Might only store information about the mapping for shared indices.

Index Sets

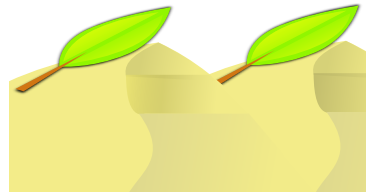


Index Set

- Distributed overlapping index set $I = \bigcup_0^{P-1} I_p$
- Process p manages mapping $I_p \longrightarrow [0, n_p)$.
- Might only store information about the mapping for shared indices.

Global Index

- Identifies a position (index) globally.
- Arbitrary and not consecutive (to support adaptivity).
- Persistent.
- On JUGENE this is not an int to get rid off the 32 bit limit!



Index Sets

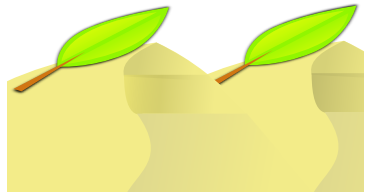
Index Set

- Distributed overlapping index set $I = \bigcup_0^{P-1} I_p$
- Process p manages mapping $I_p \longrightarrow [0, n_p)$.
- Might only store information about the mapping for shared indices.

Local Index

- Addresses a position in the local container.
- Convertible to an integral type.
- Consecutive index starting from 0.
- Non-persistent.
- Provides an attribute to identity ghost region.

Remote Information and Communication



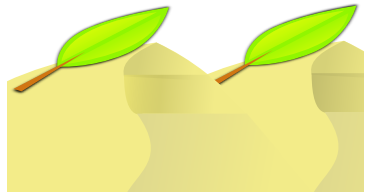
Remote Index Information

- For each process q the process p knows all common global indices together with the attribute on q.

Communication

- Target and source partition of the index is chosen using attribute flags, e.g from `ghost` to `owner` and `ghost`.
- If there is remote index information of q available on p, then p send all the data in one message.
- Communication takes place collectively at the same time.

Parallel Matrix Representation



- Let I_i is a nonoverlapping decomposition of our index set I.
- $\widetilde{I}_i$ is the augmented index set such set for all $k \in I_i$ with $|a_{kj}| + |a_{jk}| \neq 0$ also $k \in \widetilde{I}_i$ holds.
- Then the locally stored matrix looks like

$$\begin{array}{l} \widetilde{I}_i \left\{ \right. \\ I_i \left\{ \right. \end{array} \begin{array}{|c|c|} \hline A_{ii} & * \\ \hline 0 & I \\ \hline \end{array}$$

- Therefore Av can be computed locally for the entries associated with I_i if v is known for $\widetilde{I}_i$
- A communication step ensures consistent ghost values.
- Matrix can be used for hybrid preconditioners.

Illustration Parallel Setup

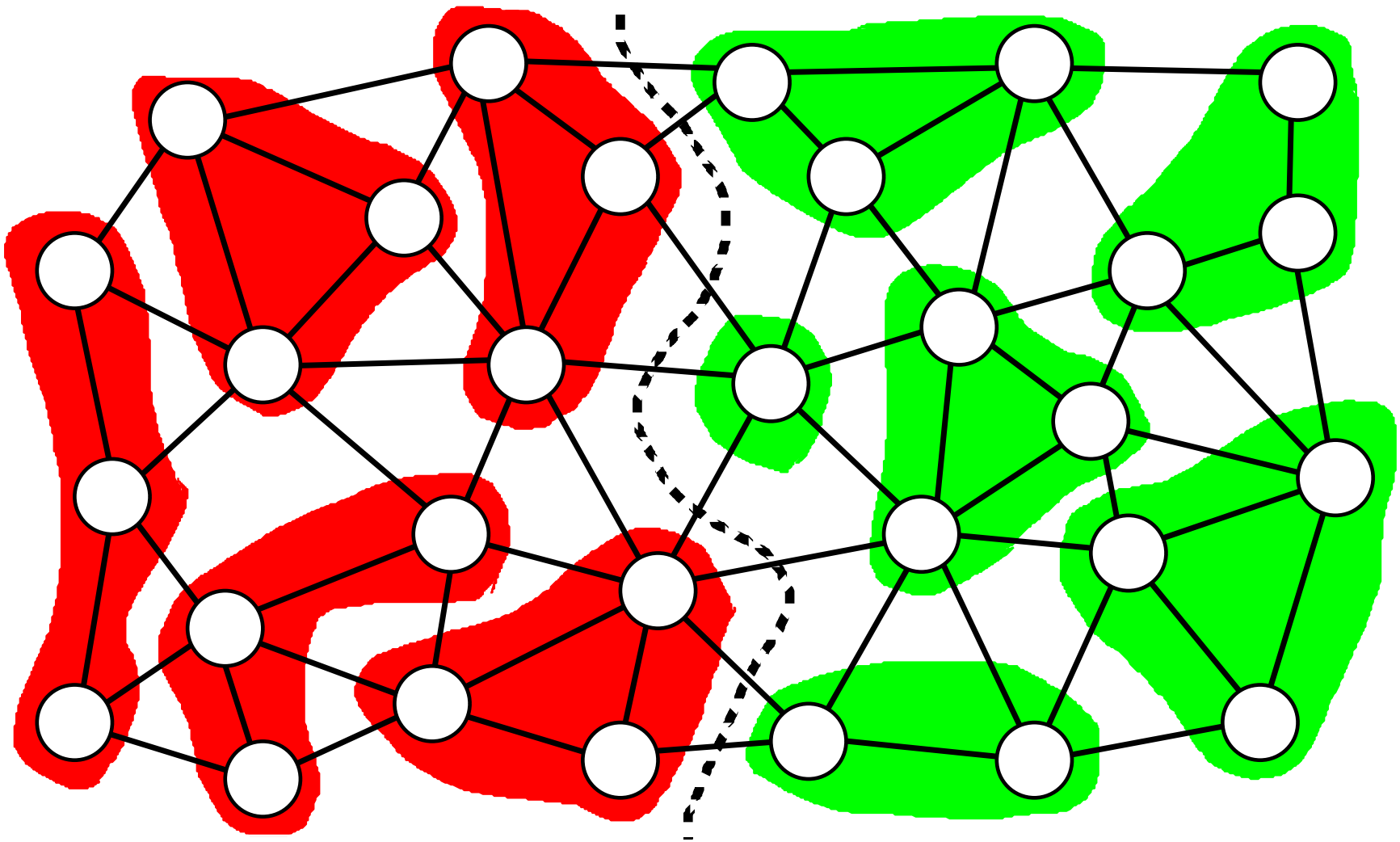
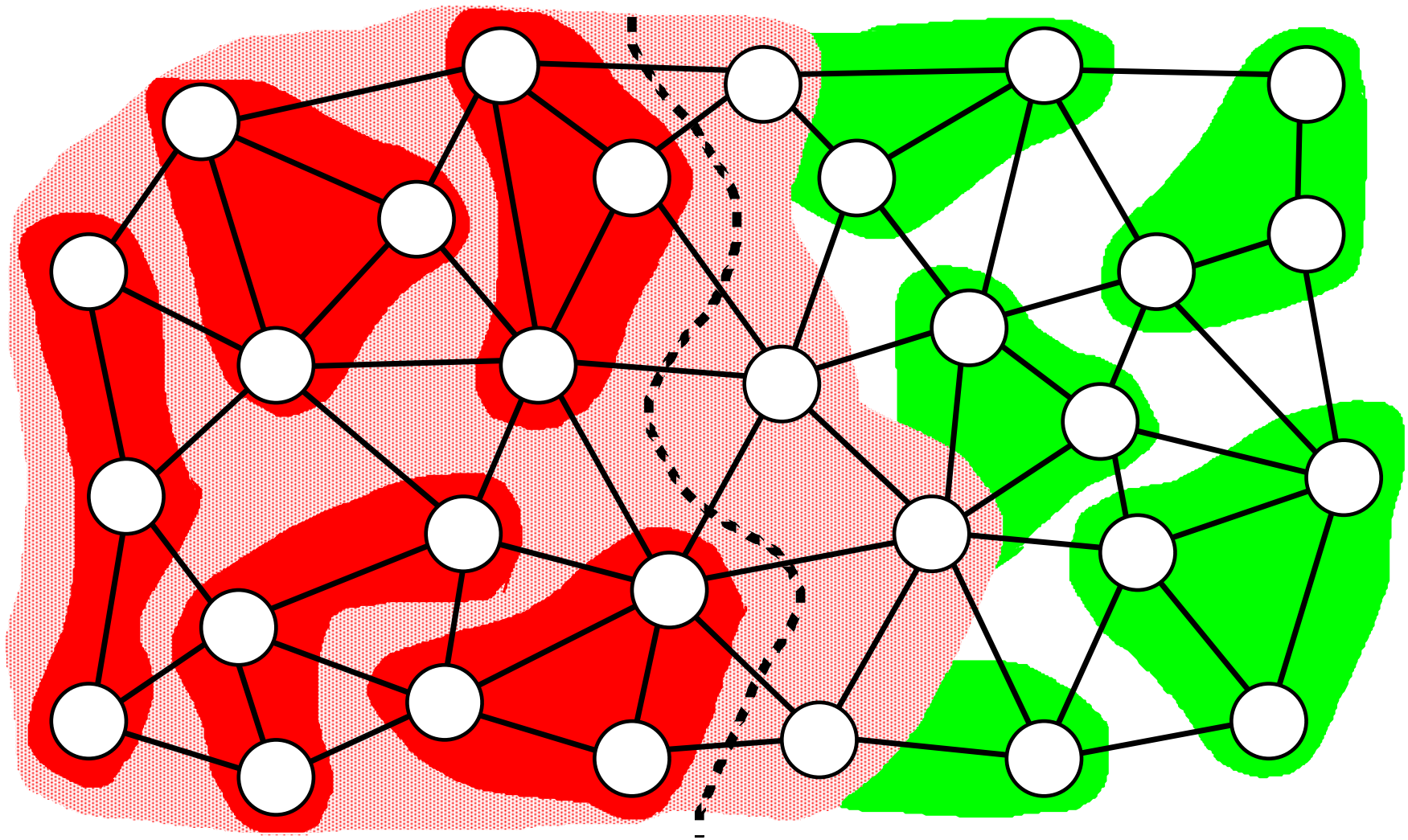
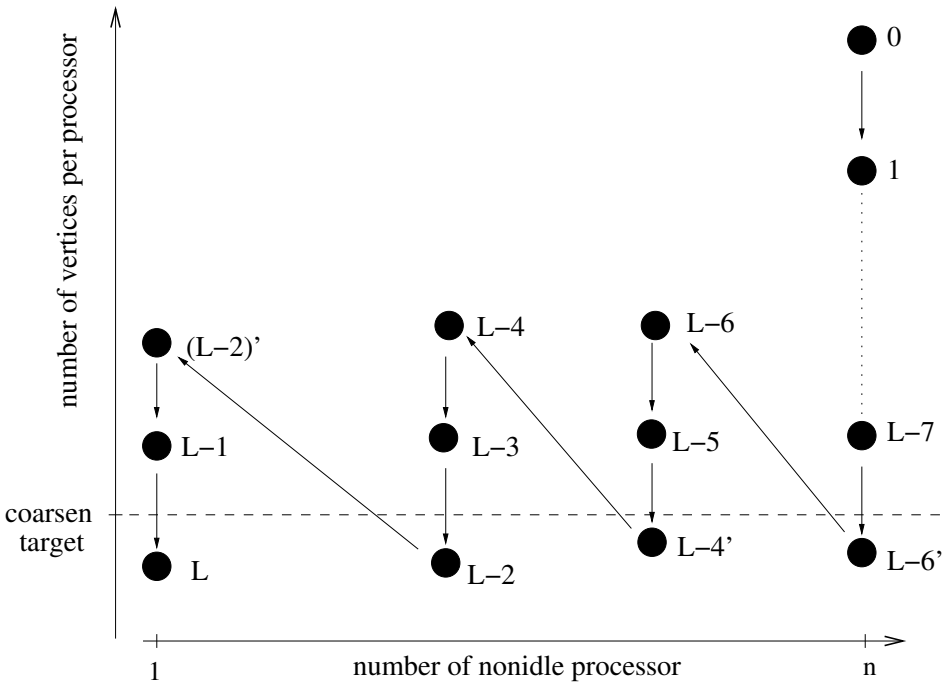


Illustration Parallel Setup

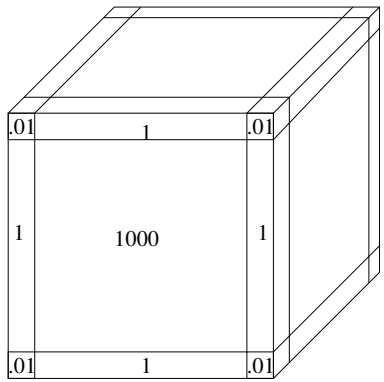


Data Agglomeration on Coarse Levels



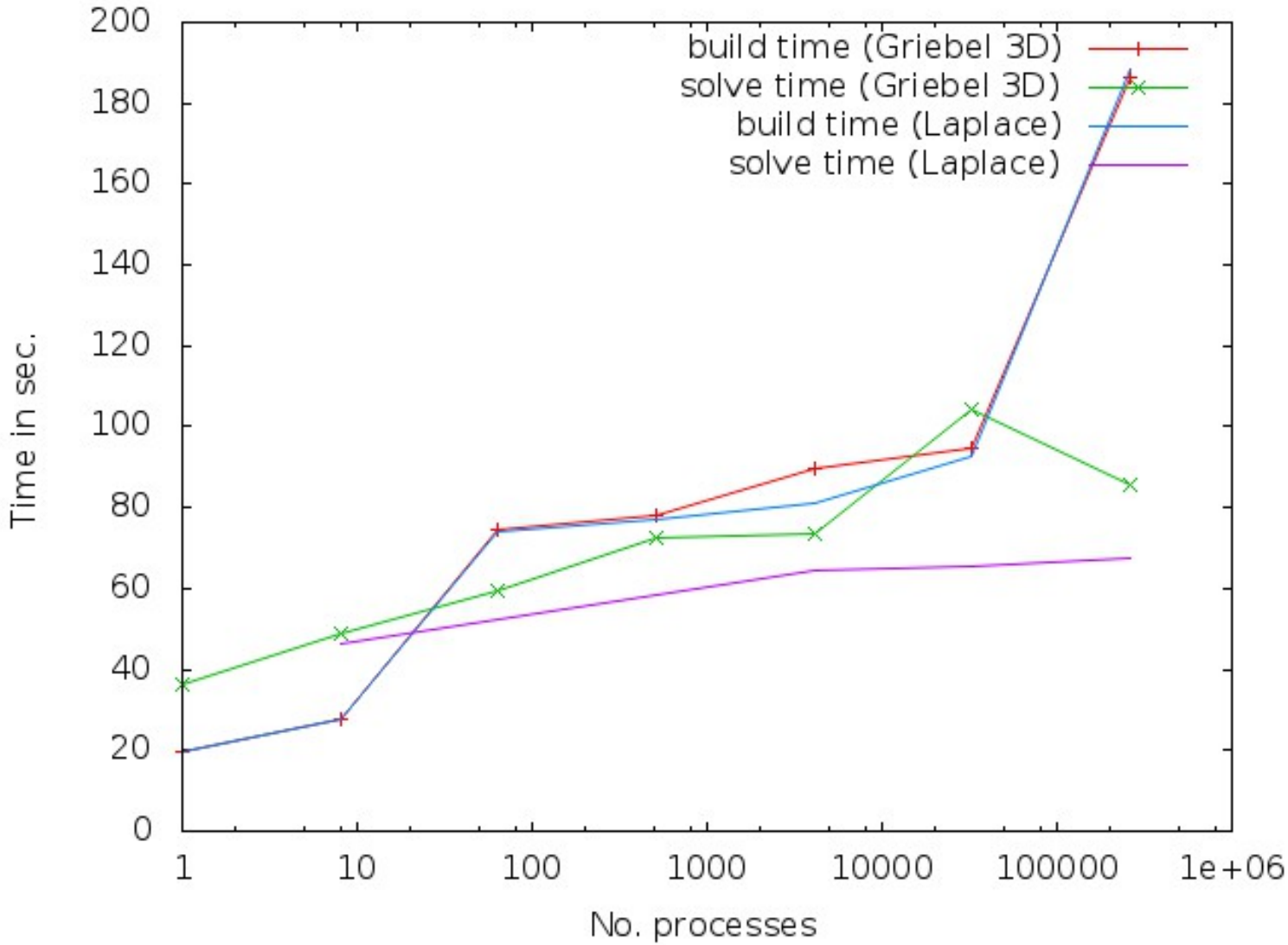
- Repartition the data onto fewer processes.
- Use METIS on the graph of the communication pattern. (ParMETIS cannot handle the full machine!)

Test Problems

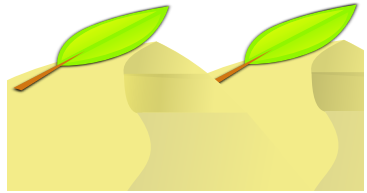


- Problem with discontinuous diffusion [Griebel et. al. 2007]
  - $\nabla K \cdot \nabla u = 0$
  - Dirichlet boundary condition
  - 80^3 unknowns per process ($1.34 \cdot 10^{11}$ unknowns for biggest problem)
- Laplace
- Discretization: Cell-centered finite volumes
- Machine: JUGENE (IBM Blue Gene/System P)

Weak Scalability Results



Weak Scalability Results



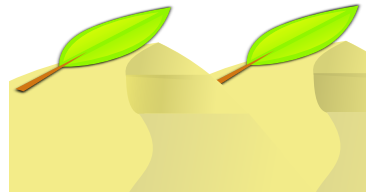
	Laplace		Griebel	
# procs	# It.	T/It.	# It	T/It.
1	8	3.99	9	4.03
8	10	4.64	10	4.89
64	10	4.93	12	4.96
512	12	5.02	14	5.19
4096	13	4.96	14	5.24
32768	13	5.04	20	5.21
262144	13	5.20	16	5.21

Parallel Groundwater Simulation

Figure: Cut through the ground beneath an acre

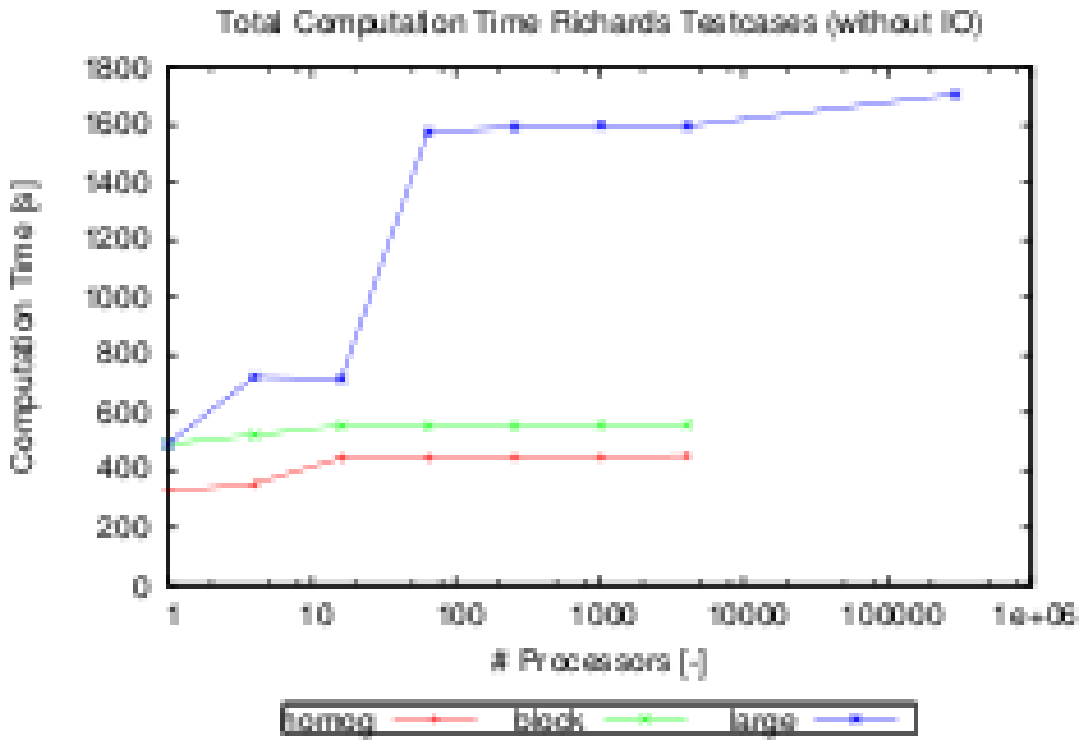
- Highly discontinuous permeability of the ground.
- 3D simulations with high resolution.
- Efficient and robust parallel iterative solvers.

$$-\nabla \cdot (K(x)\nabla u) = f \text{ in } \Omega \tag{1}$$

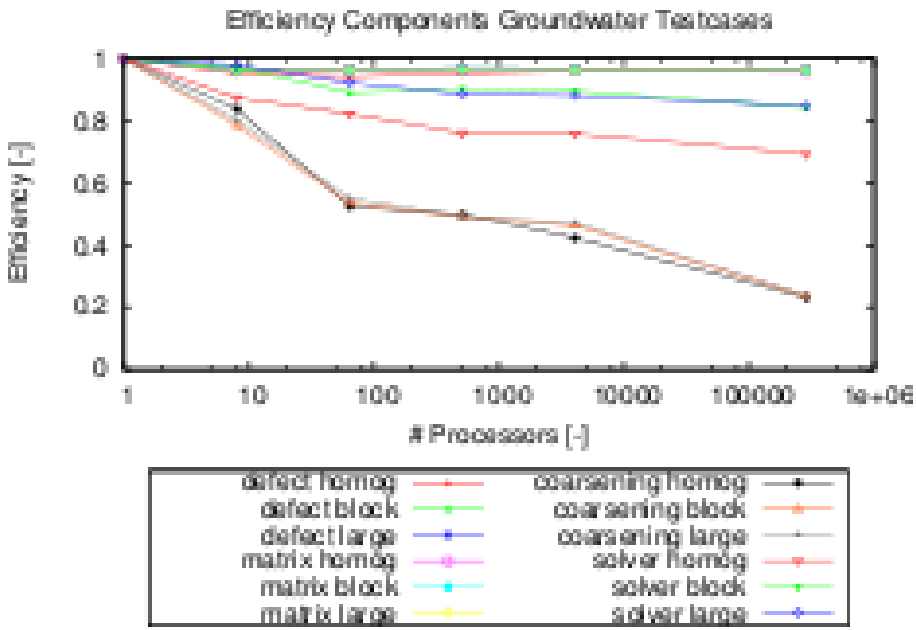


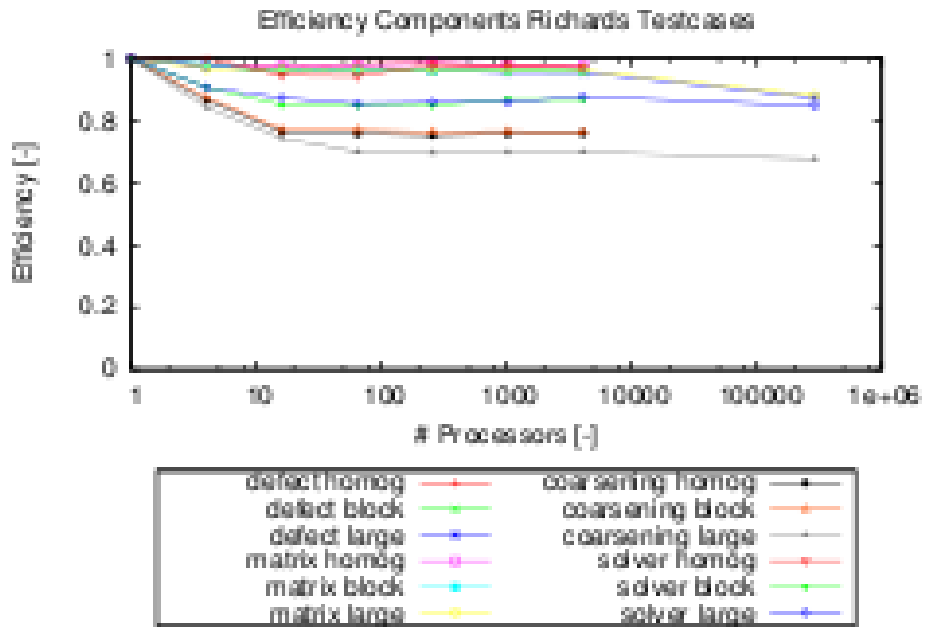
Weak Scalability Results II

- Richards equation.
- $64X64X128$ unknowns per process.
- $1.25 \cdot 10^{11}$ unknowns on the full JUGENE.
- One time step in simulation.

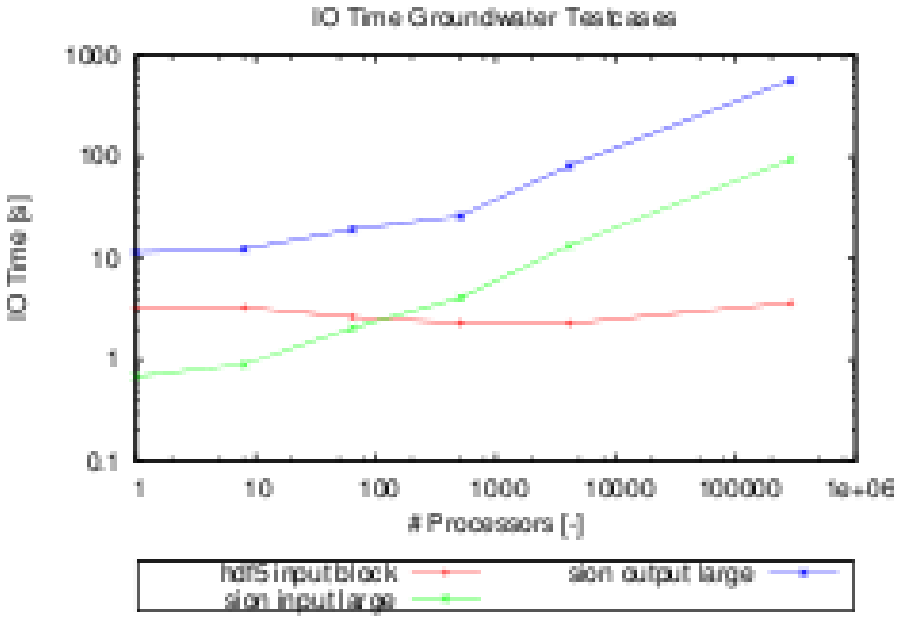


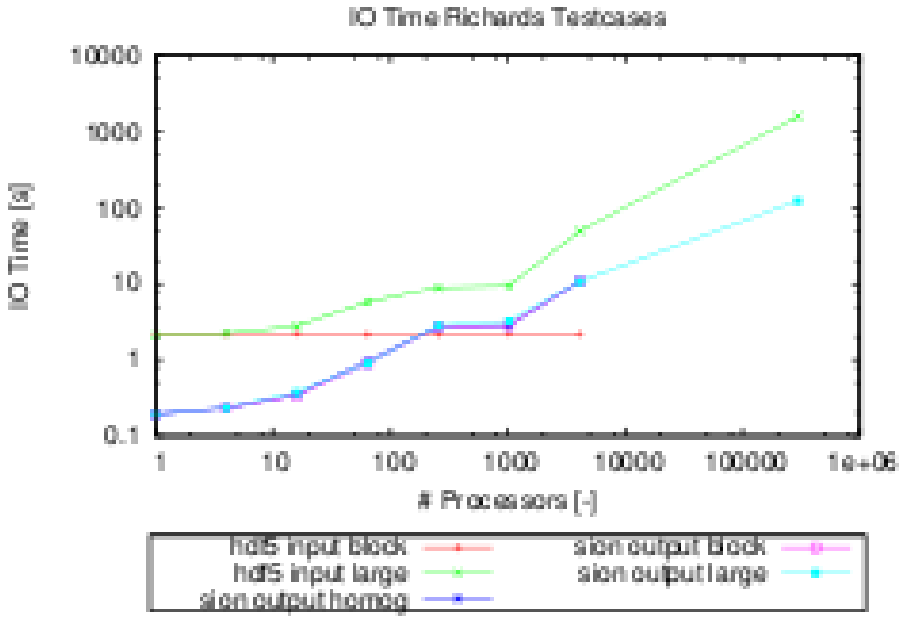
Efficiency Solver Components





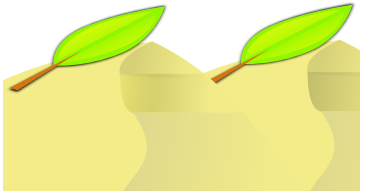
Efficiency IO



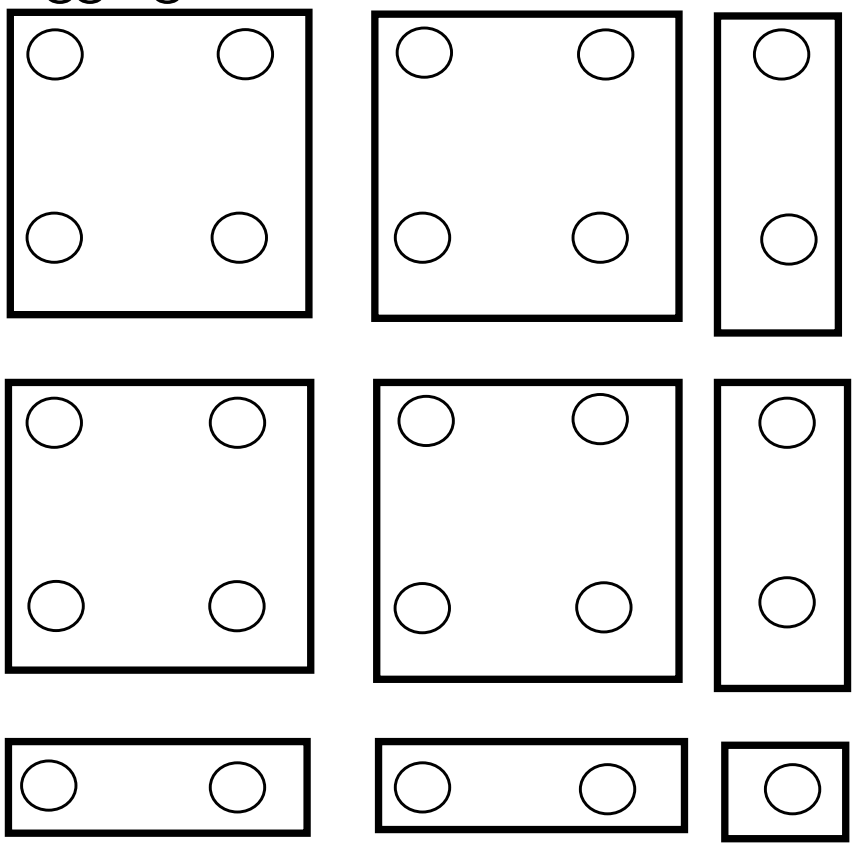


- Highly tuned by Olaf Ippisch
- SionLib used for IO (much faster than HDF5)
- Still: IO remains a bottleneck!

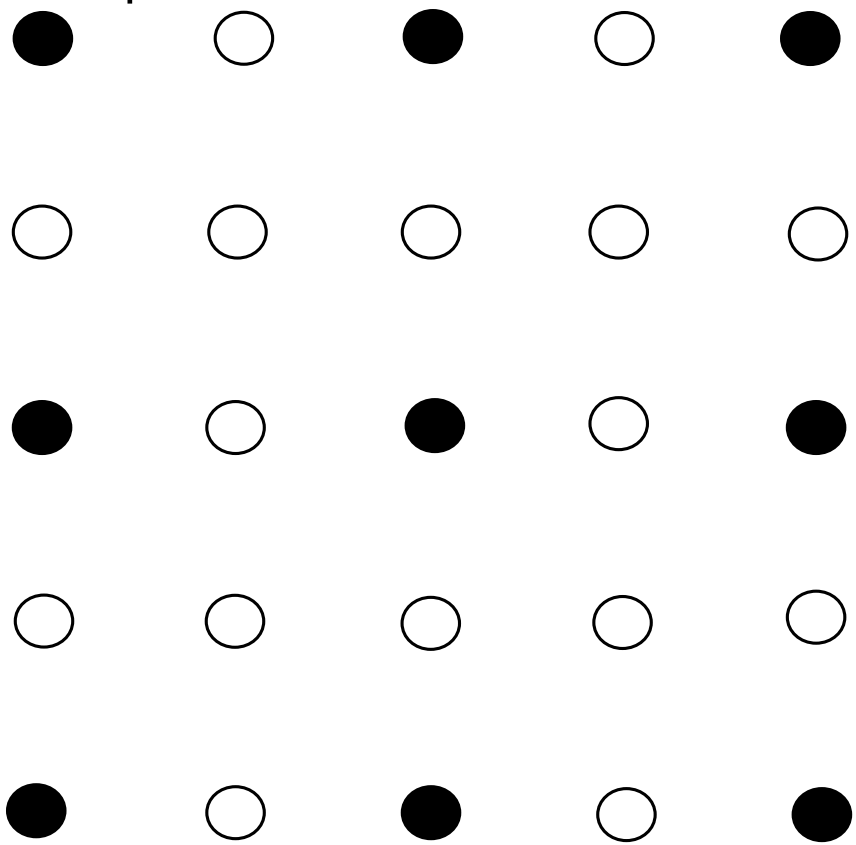
AMG methods



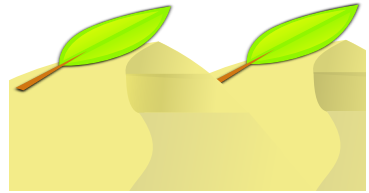
Aggregation AMG



Interpolation AMG

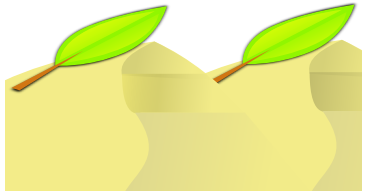


AMG for Atmospheric Physics



- 1/6 of cube sphere grid
- Horizontal partitioning only
- # processors relates to problem size
- Computed on Hector Cray XE6 supercomputer
  2816 nodes of 2 x AMD Opteron 16-core Interlagos 2.3 GHz
- Research done by Eike Müller and Rob Scheichl (Uni Bath)
- Alternative: BommerAMG of Hypre

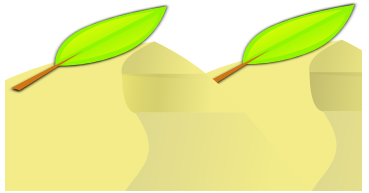
Scaling results Atmospherical Physics I



Time per Iteration

# procs	# dofs	DUNE AMG	BoomerAMG
16	$8.3 \cdot 10^6$	0.63	0.73
64	$2.4 \cdot 10^7$	0.64	0.73
256	$1.3 \cdot 10^8$	0.65	0.75
1024	$5.4 \cdot 10^8$	0.67	0.75
4096	$2.1 \cdot 10^9$	0.66	0.75
16384	$8.6 \cdot 10^9$	0.68	0.86
65536	$3.4 \cdot 10^{10}$	0.68	2.24

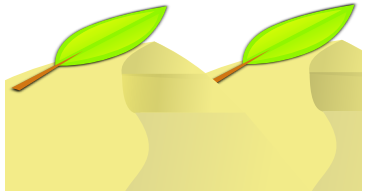
Scaling results Atmospherical Physics II



Number of Iterations (Residual Reduction $\|r\|/\|r-0\| \leq 10^{-5}$)

# procs	# dofs	DUNE AMG	BoomerAMG
16	$8.3 \cdot 10^6$	11	12
64	$2.4 \cdot 10^7$	11	13
256	$1.3 \cdot 10^8$	11	12
1024	$5.4 \cdot 10^8$	11	12
4096	$2.1 \cdot 10^9$	13	12
16384	$8.6 \cdot 10^9$	12	11
65536	$3.4 \cdot 10^{10}$	11	9

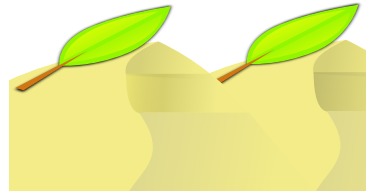
Scaling results Atmospherical Physics III



Solve time + AMG setup time (in seconds)

# procs	# dofs	DUNE AMG	BoomerAMG
16	$8.3 \cdot 10^6$	6.92+4.13	8.72+2.59
64	$2.4 \cdot 10^7$	7.01+4.92	9.52+2.74
256	$1.3 \cdot 10^8$	7.18+4.88	8.98+2.82
1024	$5.4 \cdot 10^8$	7.32+5.89	9.04+3.18
4096	$2.1 \cdot 10^9$	8.64+6.32	8.99+3.56
16384	$8.6 \cdot 10^9$	8.16+8.06	9.43+5.75
65536	$3.4 \cdot 10^{11}$	7.49+10.92	20.20+7.09

Memory Requirements



Aggregation Approach

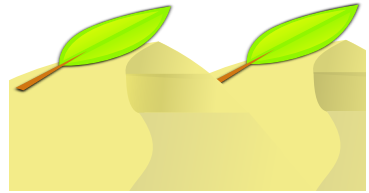
- Matrix hierachy: ≈ 1.4 times the memory of the fine level matrix
- Vector hierarchies: ≈ 2.2 times the memory of the fine level vector
- Aggregate information is stored in another Vector hierarchy.

Interpolation AMG (e.g. SAMG)

- Matriy hierachies > 1.9 times the memory of the fine level matrix

For an anisotropic upscaling code SAMG needs nearly three times the memory!

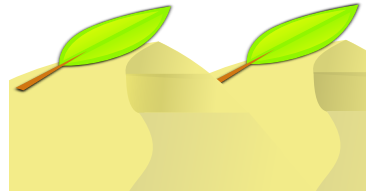
Increasing Robustness



Possibilities

- Play around with scaling factor ω $A_{l+1} = \omega P^T A_l P$ e.g 1.6 for isotropic and 1.1 for anisotropic problems
- Use Hybrid block Gauss-Seidel smoother (SeqOverlappingSchwarz) where the blocks are defined by the aggregates.
- Use a Krylov-cycle (KAMG) that recombines coarse grid corrections (experimental).

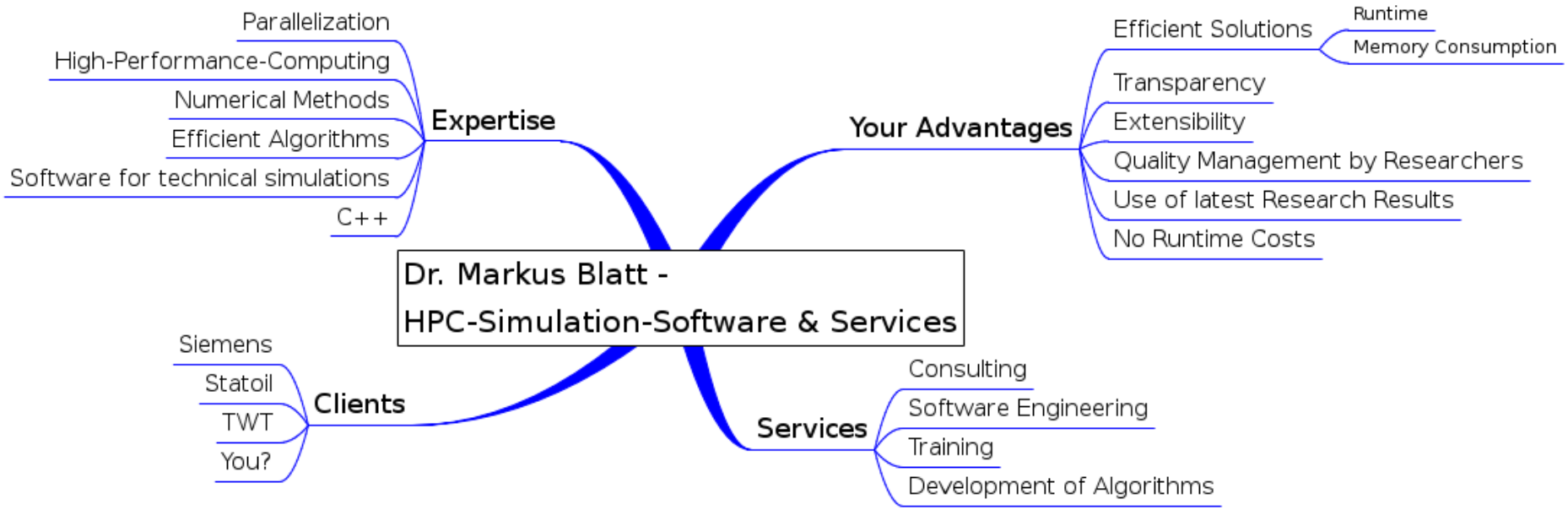
Summary



- Aggregation AMG is a highly scalable solver.
- Very memory efficient: $1.34 \cdot 10^{11}$ unknowns on IBM Blue Gene/P
- Quite robust!
- Next steps: improve setup phase
- Available as part of dune-istl `www.dune-project.org` (GPL with runtime exception)

What can we do for you?

Efficient simulation software made to measure.



Hans-Bunte-Str. 8-10, 69123 Heidelberg, Germany
http://www.dr-blatt.de