ISTL - Iterative Solvers Template Library

Markus Blatt, Peter Bastian

Interdisciplinary Center for Scientific Computing, University of Heidelberg

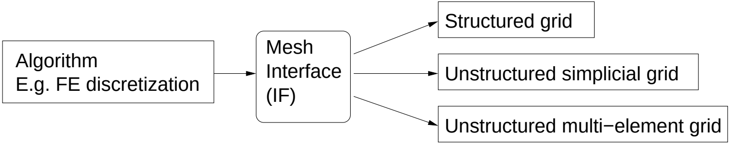
March 6-10, 2006

Outline

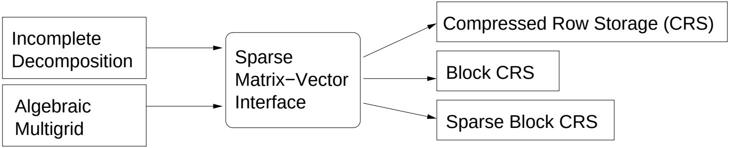
1. Motivation
2. Expressing Structure in FE Matrices
3. Performance
4. Solver Components
5. Agglomeration Algebraic Multigrid
6. Parallel Agglomeration Algebraic Multigrid
7. Conclusions

DUNE

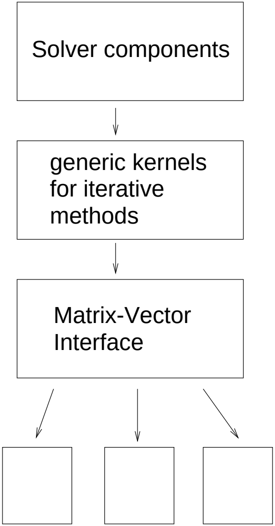
- **D**istributed **U**nified **N**umerics *E*nvironment (http://dune.uni-hd.de)
    - Seperate data structures and algorithm
    - Formulate algorithms based on interfaces
    - Provide different implementations of the interface
    - No lack performance due to generic programming
    - Parts of DUNE
        - Grid interface (not covered)



        - **I**terative **S**olvers **L**ibrary (ISTL)



Structure of ISTL



- There are already template libraries for linear algebra: MTL/ITL
- Existing libraries cannot efficiently use (small) structure of FE-Matrices
- Solver components: Based on operator concept, Krylov methods, (A)MG preconditioners
- Generic kernels: Triangular solves, Gauss-Seidel step, ILU decomposition
- Matrix-Vector Interface: Support recursively block structured matrices
- Various implementations of the interface are available

Example Definitions

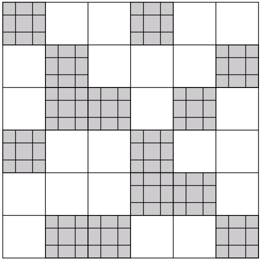
- A vector containing 20 blocks where each block contains two complex numbers using double for each component:

```
typedef FieldVector<complex<double>,2> MyBlock;
BlockVector<MyBlock> x(20);
x[3][1] = complex<double>(1,-1);
```

- A sparse matrix consisting of sparse matrices having scalar entries:

```
typedef FieldMatrix<double,1,1> DenseBlock;
typedef BCRSMatrix<DenseBlock> SparseBlock;
typedef BCRSMatrix<SparseBlock> Matrix;
Matrix A(10,10,40,Matrix::row_wise);
... // fill matrix
A[1][1][3][4][0][0] = 3.14;
```

Block Structure in FE Matrices

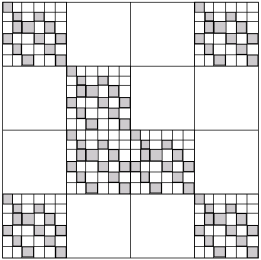


sparse block matrix

blocks are dense

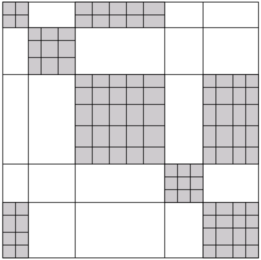
blocks have fixed size

DG fixed p



blocks are sparse

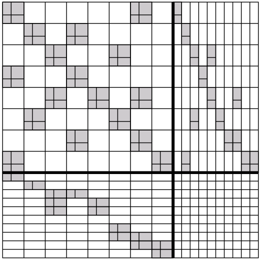
diffusion-reaction systems



blocks are dense

blocks have variable size

DG hp version



2x2 block matrix

each block is sparse

Taylor-Hood elements

Vector-Matrix Interface

- Vector
  - Is a one-dimensional container
  - Sequential access
  - Random access
  - Vector space operations: Addition, scaling
  - Scalar product
  - Various norms
  - Sizes
- Matrix
  - Is a two-dimensional container
  - Sequential access using iterators
  - Random access
  - Organization is row-wise
  - Mappings $y = y + Ax; y = y + A^T x; y = y + A^H x;$
  - Solve, inverse, left multiplication
  - Various norms
  - Sizes

Performance I

- Pentium 4 Mobile 2.4 GHz: Stream for $x = y + \alpha z$ is 1084 MB/s
- Compiler: GNU C++ compiler version 4.0
- Scalar product of two vectors (block size 1)

N	500	5000	50000	500000	5000000
MFLOPS	896	775	167	160	164

- daxpy operation $y = y + \alpha x$, 1200 MB/s transfer rate for large N

N	500	5000	50000	500000	5000000
MFLOPS	936	910	108	103	107

- Matrix-vector product, BCRSMatrix, 5-point stencil, b: block size

N, b	100,1	10000,1	1000000,1	1000000,2	1000000,3
MFLOPS	388	140	136	230	260

Example: Generic Gauss-Seidel

```
template<class M, class X, class Y, class K>
static void dbgs (const M& A, X& x, const Y& b, const K& w) {
  typedef typename M::ConstRowIterator rowiterator;
  typedef typename M::ConstColIterator coliterator;
  typedef typename Y::block_type bblock;
  typedef typename X::block_type xblock;
  bblock rhs; X xold(x); rowiterator endi=A.end();
  for (rowiterator i=A.begin(); i!=endi; ++i) {  // loop over rows
    rhs = b[i.index()];                          // initialize rhs
    coliterator endj=(*i).end();                 // end of row i
    coliterator j=(*i).begin();                  // start of row i
    for (; j.index()<i.index(); ++j)             // lower triangle
      (*j).mmv(x[j.index()],rhs);                // minus matrix vecto
    coliterator diag=j;                          // remember diagonal
    for (; j!=endj; ++j)                         // upper triangle
      (*j).mmv(x[j.index()],rhs);                // minus matrix vecto
    algmeta_itsteps<I-1>::dbgs(*diag,x[i.index()],rhs,w);//''solve''
  }
  x *= w; x.axpy(1-w,xold);                      // update with dampi
}
```

Performance II

- Damped Gauss-Seidel solver
- 5-point stencil on 1000 by 1000 grid
- Comparison of generic implementation in ISTL with specialized C implementation in AMGLIB

	AMGLIB	ISTL
Time per iteration [s]	0.17	0.18

- Corresponds to about 150 MFLOPS

Operator and Solver Concept

Operator Concept

- Let $A : X \longmapsto Y$, $x \longmapsto A(x)$ be a linear Operator with X, Y vector spaces.
- Class LinearOperator
  - apply(const X& x, Y& y) : $y = A(x)$
  - applyscaleadd(field_type alpha, const X& x, Y& y): $y = y + \alpha A(x)$
- *Problem:* Find $x \in X$ such that $A(x) = b$ for $b \in Y$.

Solver Concept

- Preconditioned iterative solvers, e. g. LoopSolver, CGSolver, BiCGStabSolver
- Inherit from abstract base class InverseOperator
- Use only abstract Operator interface functions, provided scalarproduct and preconditioner

Operator and Solver Concept

Operator Concept

- Let $A : X \longmapsto Y$, $x \longmapsto A(x)$ be a linear Operator with X, Y vector spaces.
- Class LinearOperator
  - apply(const X& x, Y& y) : $y = A(x)$
  - applyscaleadd(field_type alpha, const X& x, Y& y): $y = y + \alpha A(x)$
- *Problem:* Find $x \in X$ such that $A(x) = b$ for $b \in Y$.

Solver Concept

- Preconditioned iterative solvers, e. g. LoopSolver, CGSolver, BiCGStabSolver
- Inherit from abstract base class InverseOperator
- Use only abstract Operator interface functions, provided scalarproduct and preconditioner

Parallelism

- Solvers programmed to the interface of precondtioner, scalarproduct and operator.
- Parallelism is hidden in Operator, Preconditioner and Scalarproduct.
- E. g. OverlappingSchwarzScalarProduct, BlockPreconditioner, parallel agglomeration algebraic multigrid.

```
typedef Dune::OverlappingSchwarzScalarProduct<Vector,
                                         Communication> ScalarProduct;
typedef Dune::SeqJac<BCRSMat,Vector,Vector> SeqPrec;
typedef Dune::BlockPreconditioner<Vector,Vector,
                             Communication,SeqPrec> ParPrec;
ScalarProduct sp(comm);
SeqPrec sprec(fop.getmat(),1,1);
ParSmoother pprec(sprec,comm);
Dune::CGSolver<Vector> cg(fop, sp, pprec, 10e-08, 10, 0);
cg.apply(x,b,r);
```

Characteristics of Agglomeration AMG

Simple multigrid algorithm

- P_l: piecewise constant
- $R_l = P_l^T$
- $A_{l-1} = R_l A_l P_l$
- Proposed by Raw, Vanek et al., Braess

Clustering controlled by

- Strong coupling
- desired size (4, 8)
- minimize fill-in

Observations

- Preserves FV discretization
- Preserves sign of M-matrix
- $O(J)$ iterations for model problem in $d = 2, 3$
- Quite robust for variable coefficient elliptic problems
- $O(J)$ optimal anyway for 2d variable coefficient problems
- Reasonable coarse grid operator for systems
- Allows efficient data-parallel implementation

Characteristics of Agglomeration AMG

Simple multigrid algorithm

- P_l: piecewise constant
- $R_l = P_l^T$
- $A_{l-1} = R_l A_l P_l$
- Proposed by Raw, Vanek et al., Braess

Clustering controlled by

- Strong coupling
- desired size (4, 8)
- minimize fill-in

Observations

- Preserves FV discretization
- Preserves sign of M-matrix
- $O(J)$ iterations for model problem in $d = 2, 3$
- Quite robust for variable coefficient elliptic problems
- $O(J)$ optimal anyway for 2d variable coefficient problems
- Reasonable coarse grid operator for systems
- Allows efficient data-parallel implementation

Characteristics of Agglomeration AMG

Simple multigrid algorithm

- P_l: piecewise constant
- $R_l = P_l^T$
- $A_{l-1} = R_l A_l P_l$
- Proposed by Raw, Vanek et al., Braess

Clustering controlled by

- Strong coupling
- desired size (4, 8)
- minimize fill-in

Observations

- Preserves FV discretization
- Preserves sign of M-matrix
- $O(J)$ iterations for model problem in $d = 2, 3$
- Quite robust for variable coefficient elliptic problems
- $O(J)$ optimal anyway for 2d variable coefficient problems
- Reasonable coarse grid operator for systems
- Allows efficient data-parallel implementation

Scalar Elliptic Problem

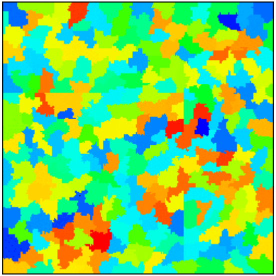
- Solve $\nabla \cdot \{k(x,y)\nabla u\} = f \quad \text{in } (0,1)^2; \quad u = g \quad \text{on } \partial\Omega$
- AMG(2,2,1) SSOR used as preconditioner in CG
- Iteration numbers for 10^{-8} reduction

N	$k(x,y) = 1$	$k(x,y) = \begin{matrix} & \vdots & & \\ & 10^{-1} & 10^{-3} & \\ & 10^{3} & 10^{1} & \ldots \end{matrix}$
64^2	7	12
128^2	9	26
256^2	10	39
512^2	12	44
1024^2	14	60
2048^2	16	52

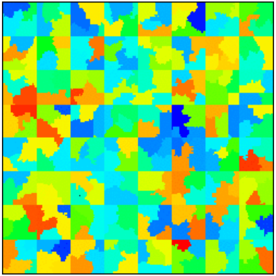
- Coarsening costs about 3 iterations

Illustration of Agglomeration

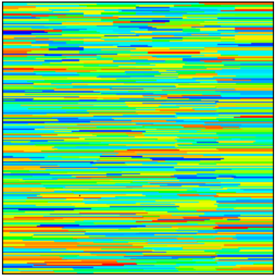
Agglomeration is matrix dependent
Follows "strong" connections



homogeneous



checker board



anisotropic

Illustration of Algorithm

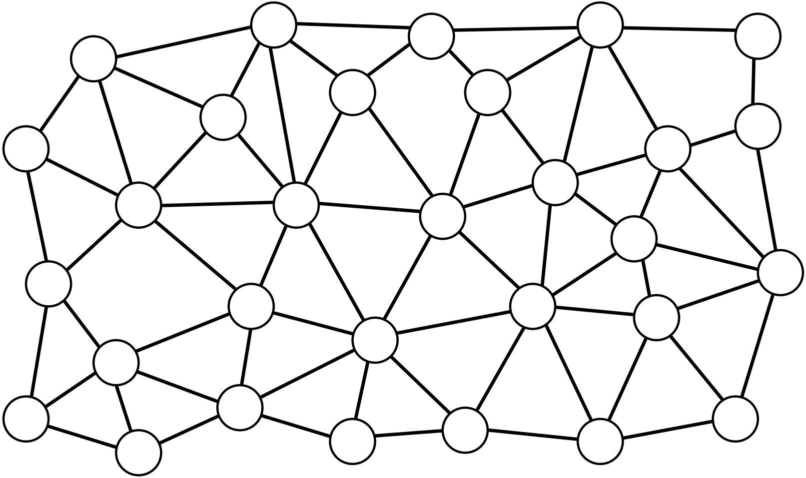


Illustration of Algorithm

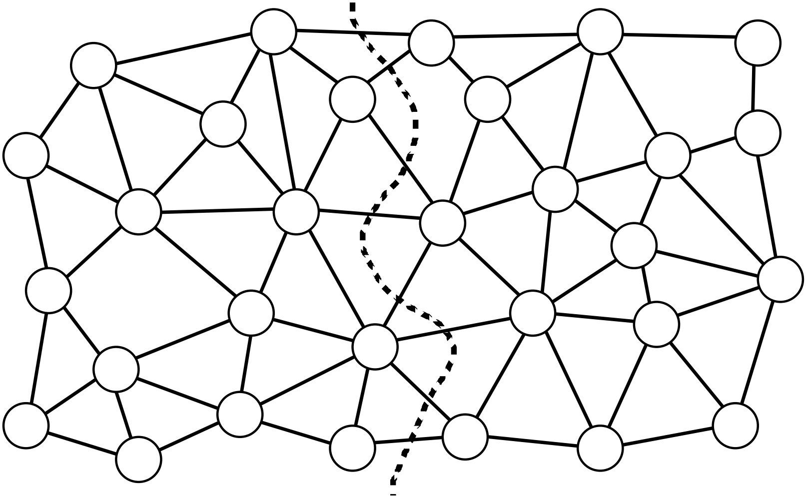


Illustration of Algorithm

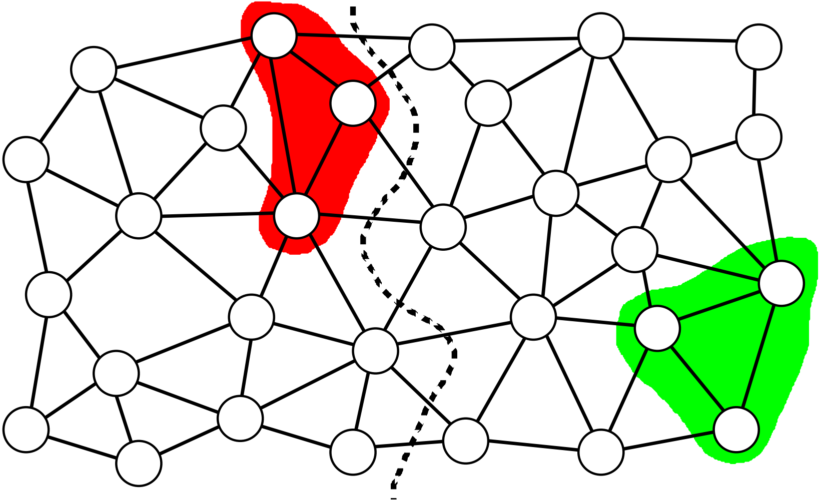


Illustration of Algorithm

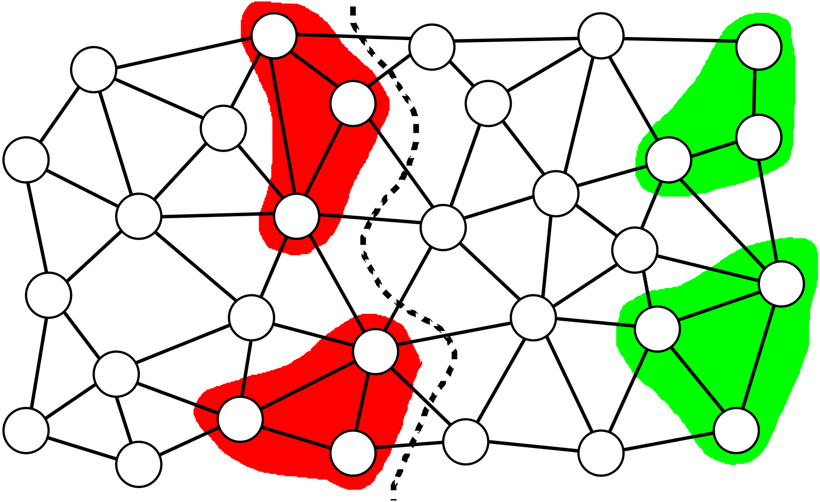


Illustration of Algorithm

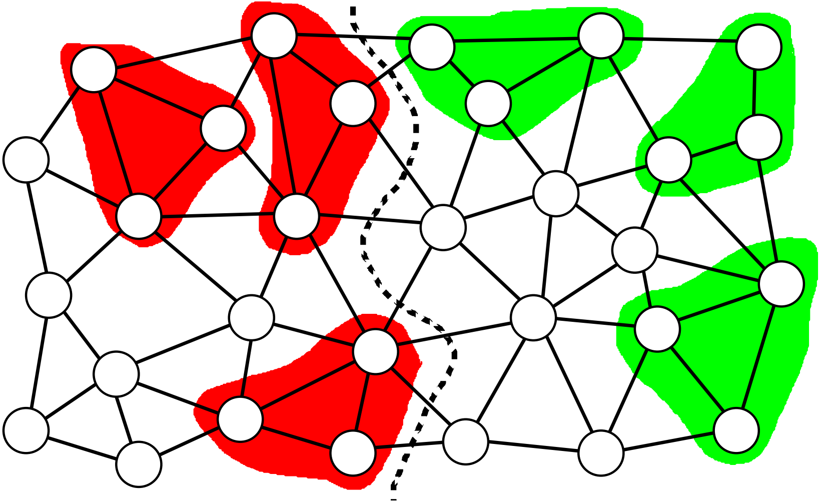


Illustration of Algorithm

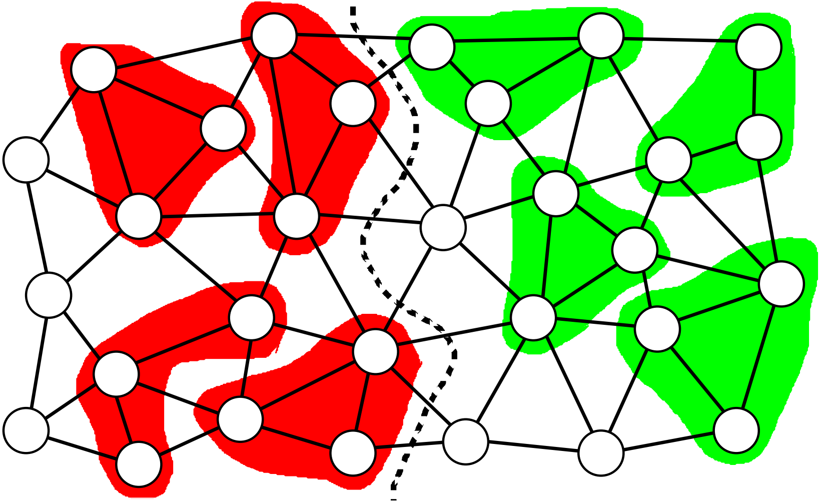


Illustration of Algorithm

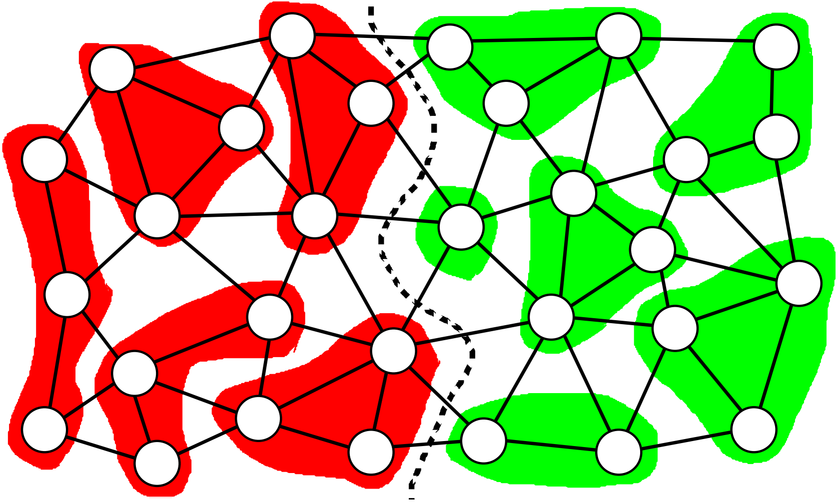


Illustration of Algorithm

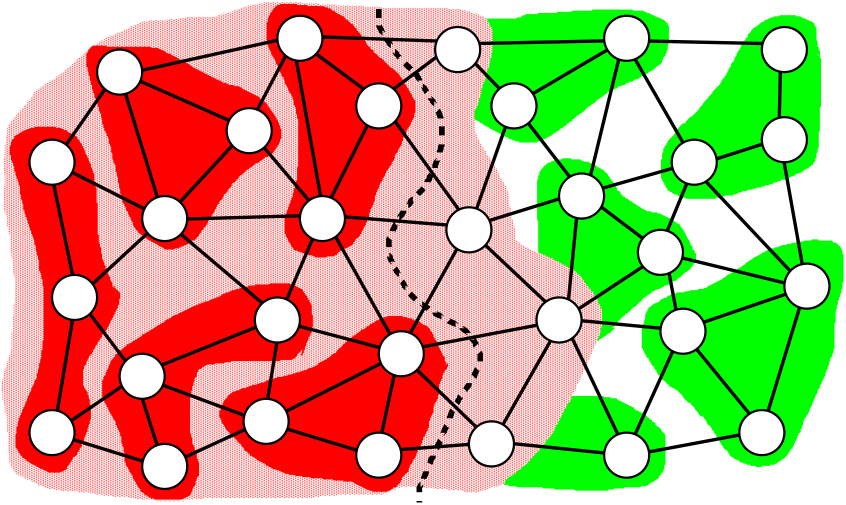
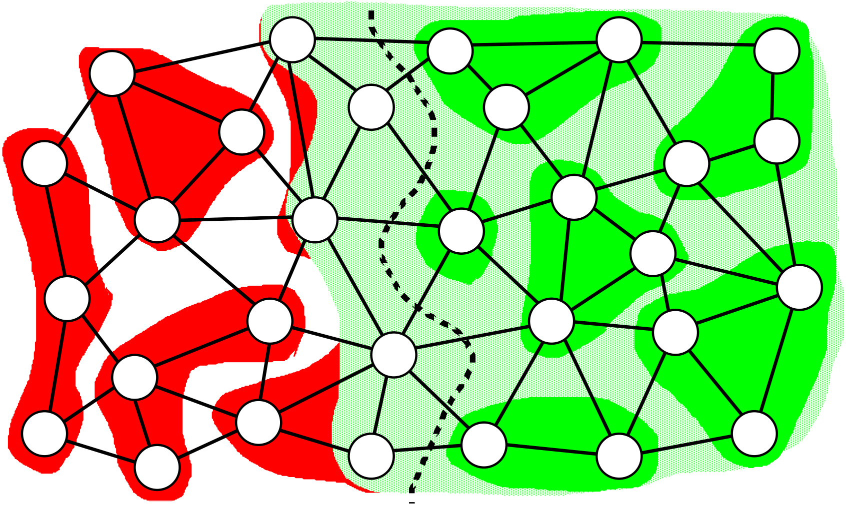


Illustration of Algorithm



Scalability Results

CG + AMG prec. + Jacobi(2) smoother, 10^{-8} residual reduction
2D heterogeneous problem: $P \cdot 2000^2$, max. $1.6 \cdot 10^9$ unknowns.

P	1	4	16	64	256	400
T_{build}[s]	96	103	110	118	123	128
T_{solve}[s]	209	298	331	366	410	407
T_{it}[s]	8.0	9.9	10.0	10.2	10.3	10.4
#IT	26	30	33	36	40	39

3D heterogeneous problem $P \cdot 150^3$, max. $7.3 \cdot 10^8$ unknowns.

P	1	8	27	64	125	216
T_{build}[s]	216	228	242	245	251	276
T_{solve}[s]	213	352	294	467	519	443
T_{it}[s]	7.6	9.8	7.7	10.2	9.1	10.3
#IT(10^{-8})	28	36	38	46	57	43

Conclusions

- ISTL is based on the following principles
  - Matrix and vector interface recursive block struture.
  - Algorithms use structure of the finite element methods.
  - No performance lack.
  - Same solver algorithms and code for all implementations due to generic programming.
  - Solver algorithms support sequential and parallel usage.
  - Robust preconditioners for heterogeneous problems
- Current plans
  - Release 1.0 of Dune http://dune.uni-hd.de
  - Apply AMG to DG discretizations (next talk!)