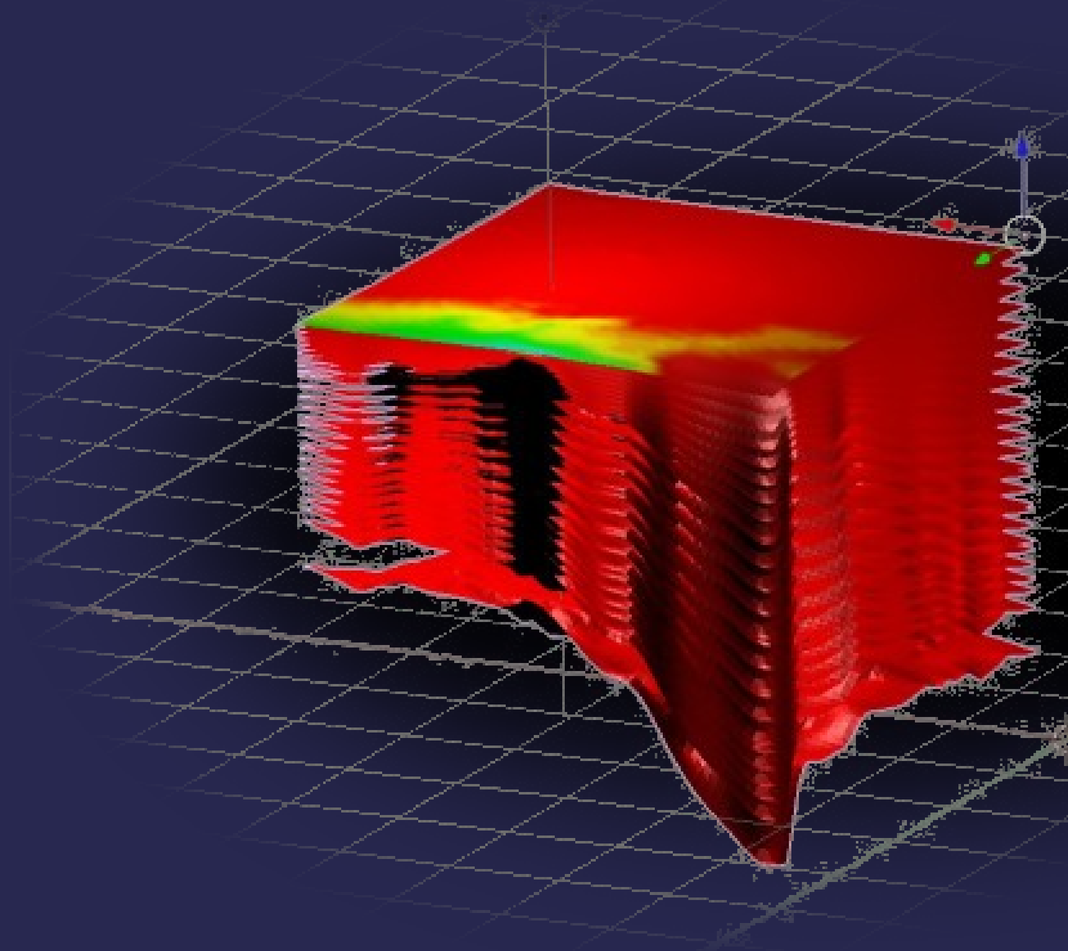
Parallel Computations and the Finite Element Method

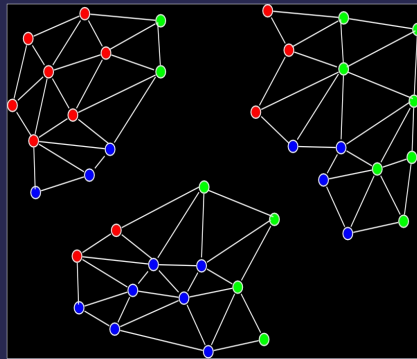
Joachim B Haga

Sintef

March 13, 2008



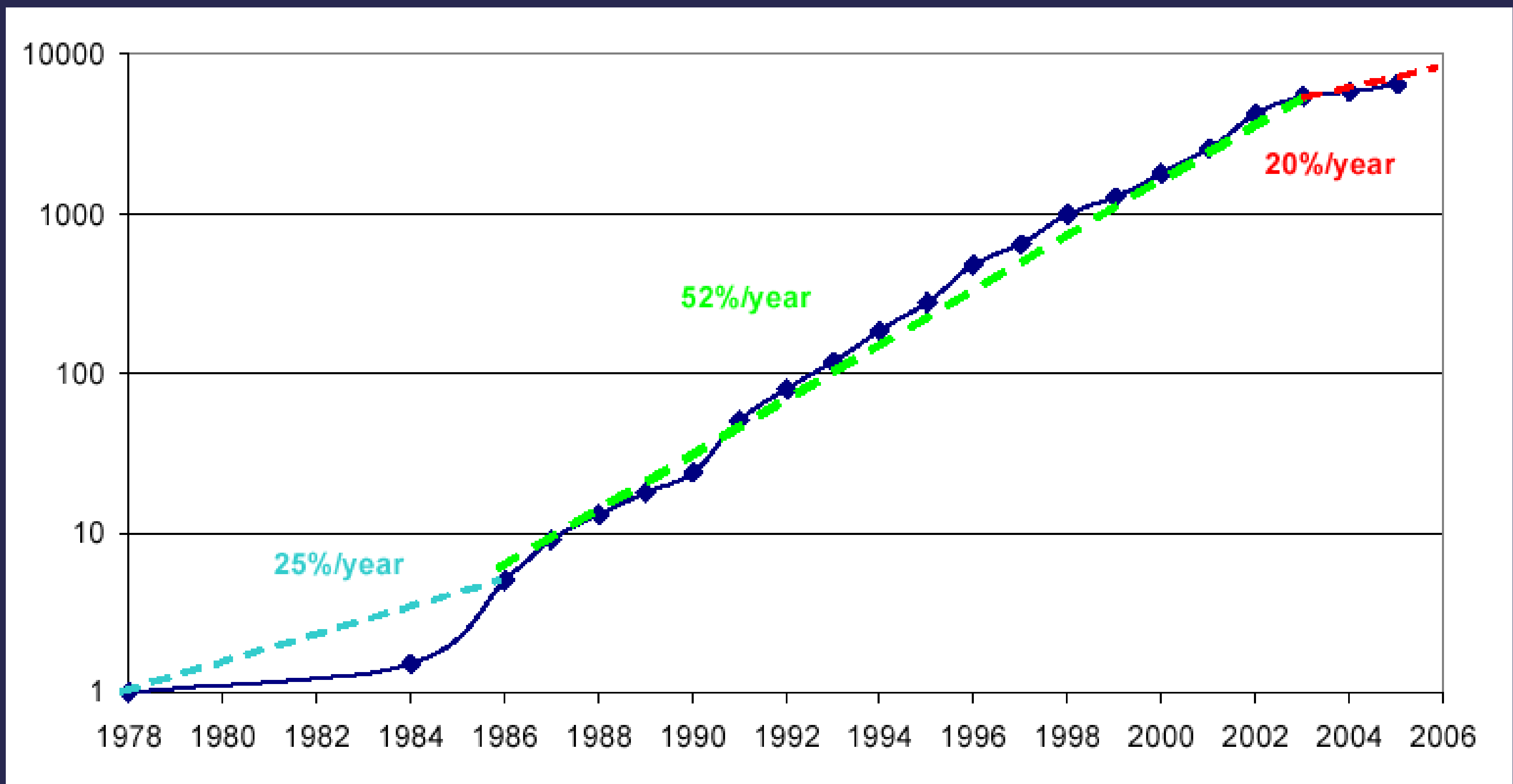
Overview of parallel computing, and why we should care



Parallelisation of the finite element method

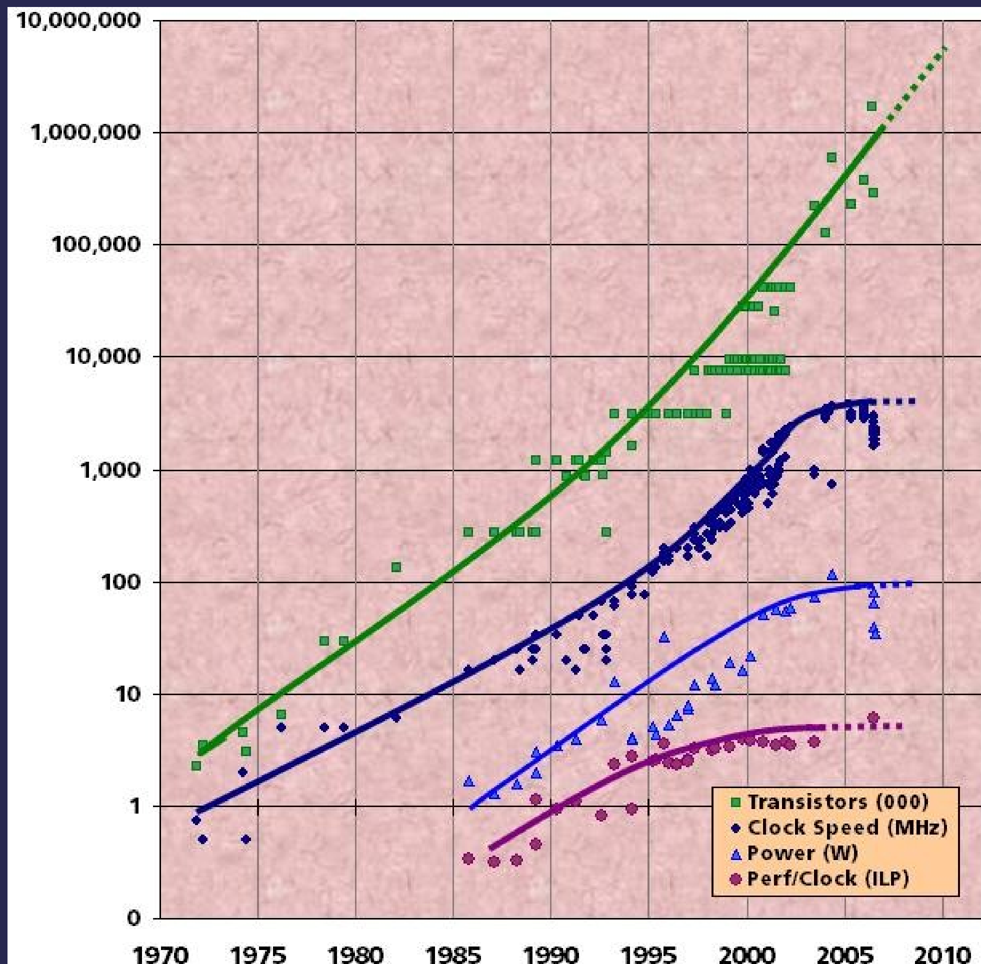
Algebraic multigrid as a parallel preconditioner

Single-processor performance has stalled since ~2002 ...



(Hennessy and Patterson, 2006)

... so the future is parallel



(Olokotun and Sutter)

- Power wall
- ILP wall
- Memory wall
- VLSI wall

Lower clock speeds
Simpler modules
Multiple cores

A number of alternatives exist for parallel computations

Distributed memory (MPI)

Shared memory (OpenMP)

GPU programming (CUDA, CTM)

Cell (PS3)

SIMD vector engines (Cray etc.)

- There is no consensus on the paradigm for the parallel future
  - so stick with established standards

OpenMP vs MPI example: inner product

```
double inner(vec a, vec b)
{
    double sum = 0;

    for (int i=0; i<a.size(); i++)
        sum += a[i]*b[i];

    return sum;
}
```

OpenMP vs MPI example: OpenMP version (transparent to caller)

```
double inner(vec a, vec b)
{
    double sum = 0;

#   pragma omp parallel for \
            private(i) \
            reduction(+:sum)
    for (int i=0; i<a.size(); i++)
        sum += a[i]*b[i];

    return sum;
}
```

OpenMP vs MPI example: MPI version (requires caller knowledge)

```
double inner(vec a, vec b)
{
    double sum = 0, glob_sum;

    for (int i=0; i<n_local_rows; i++)
        sum += a[i]*b[i];

    MPI_Allreduce(&sum, &glob_sum, 1, MPI_DOUBLE,
                  MPI_SUM, MPI_COMM_WORLD);
    return glob_sum;
}
```

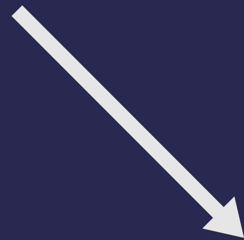
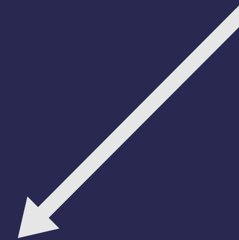
OpenMP, MPI, hybrid: pros and cons

OpenMP

- shared memory
- allows gradual parallelisation
- does little for the “memory wall”

MPI

- distributed memory
- allows clusters
- not transparent

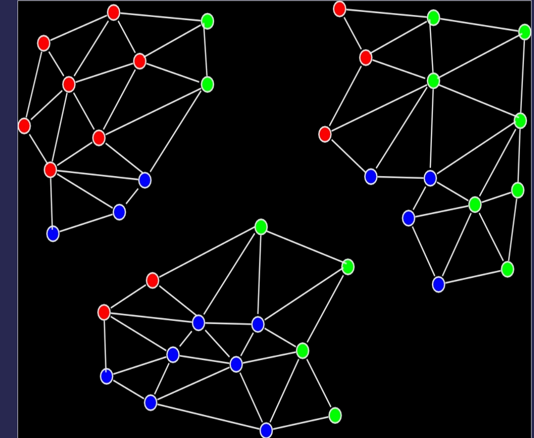



Hybrid

- OpenMP + MPI
- scales better than either alone

Parallelising the Finite Element Method

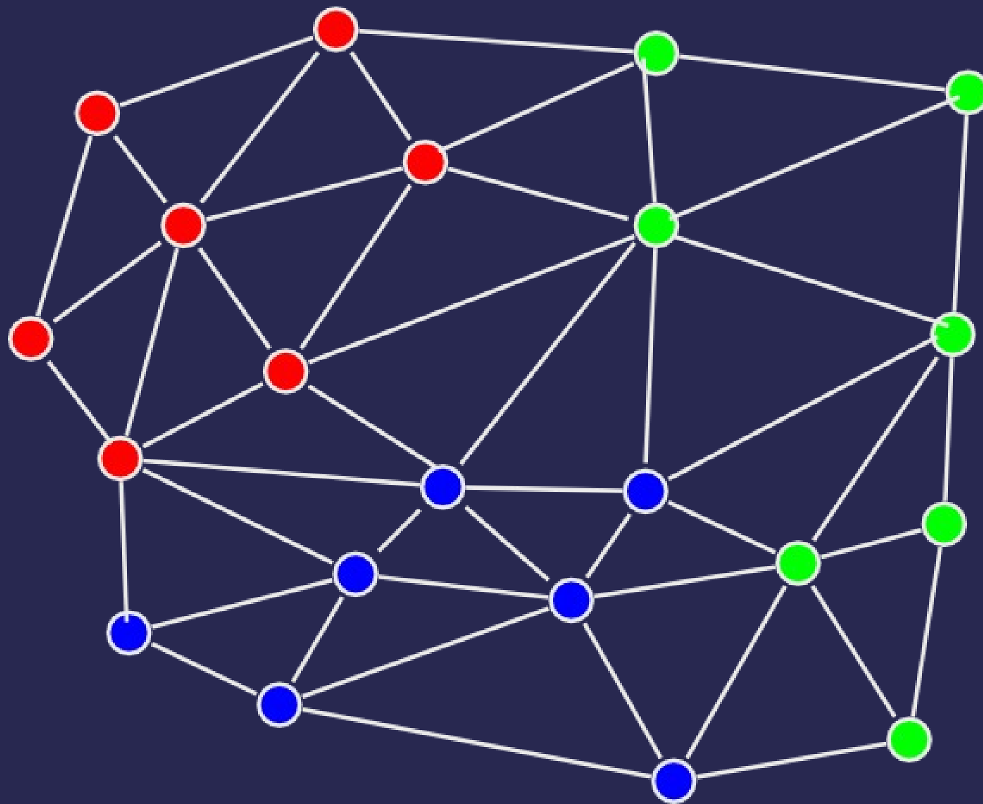
- Grid partitioning
- Parallel assembly
- Parallel linear algebra
- Parallel I/O



Nodal grid partitioning

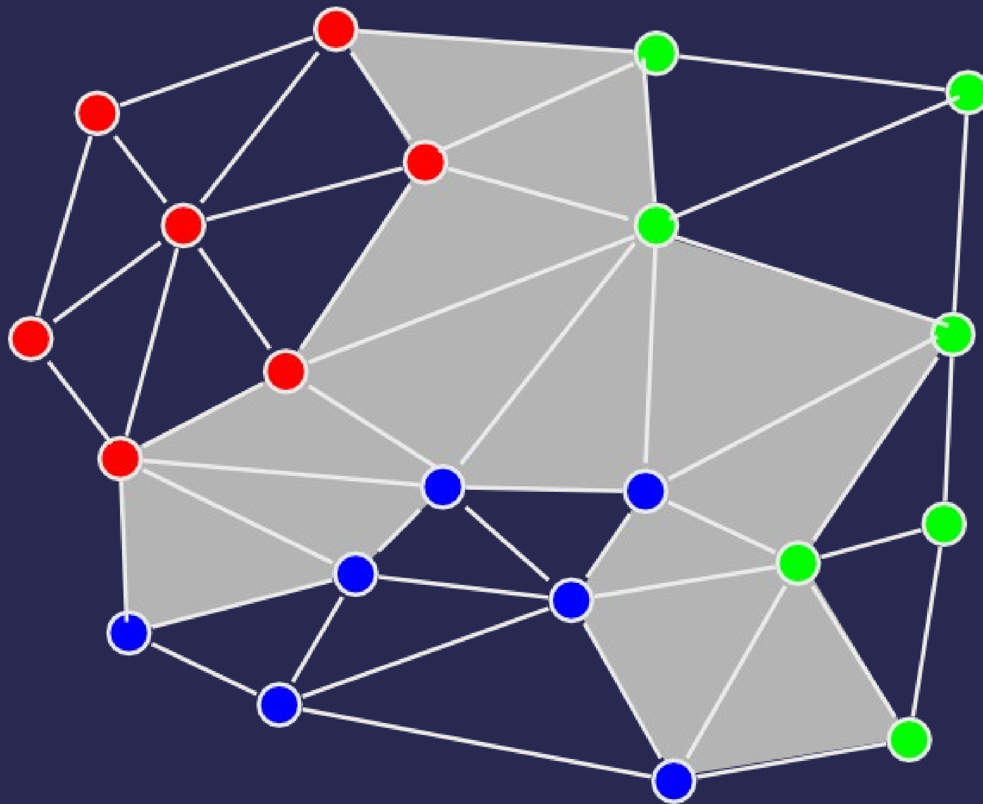
- Construct a graph from the mesh

Nodal grid partitioning



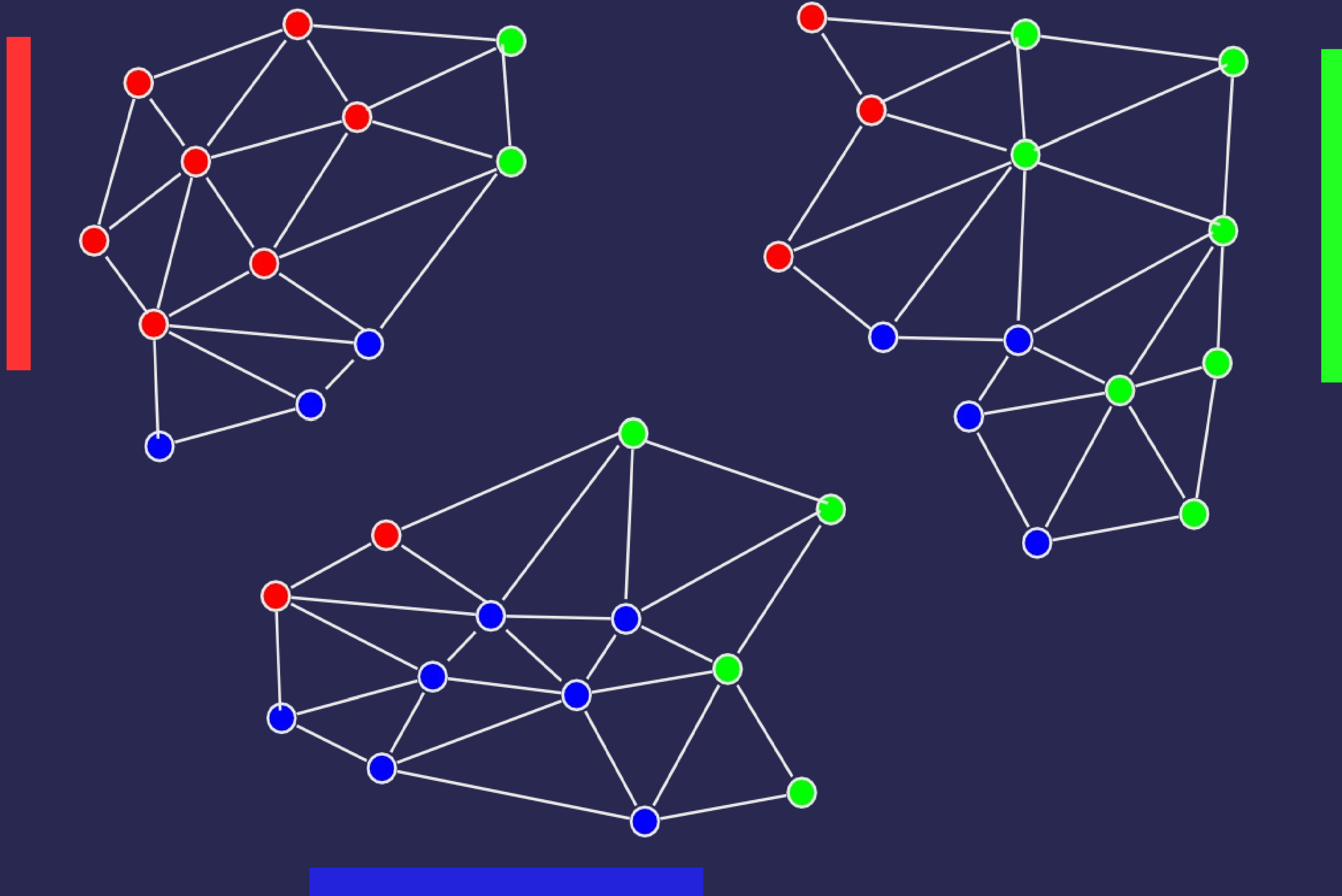
- Construct a graph from the mesh
- Partition the graph
  - METIS, ParMETIS
  - Scotch, PScotch

Nodal grid partitioning



- Construct a graph from the mesh
- Partition the graph
  - METIS, ParMETIS
  - Scotch, PScotch
- An element is on a cpu if any of its nodes are
- Result: A shared band of border elements

Nodal grid partitioning



The advantage of nodal grid partitioning is that each matrix row is complete on one processor

Dual grid partitioning (of the elements) creates a shared band of *nodes* instead of *elements*

- fewer nodes overall
- slightly smaller communication cost
- the shared nodes have no canonical placement

But with nodal partitioning, every node is the responsibility of exactly one processor

- the matrix row associated with a node is in one place
- makes e.g. algebraic multigrid much easier

The major components of a finite element solver

Parallel assembly

- trivial, just assemble locally

Parallel linear algebra

- requires care
- preconditioning remains a problem

Parallel I/O

- commonly to local disk
- gather result post-process

Distributed memory parallel linear algebra

Vector addition

- no communication

Vector inner products, norms

- exclude ghost-nodes from local norm
- reduction (sum) operation after local norm

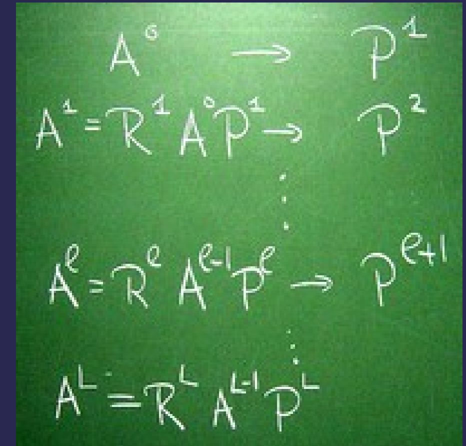
Matrix-vector product

- update ghost-node values before multiplication
- (optionally) update again after multiplication

Preconditioning

- hard...

Parallel preconditioning



- A few preconditioners are easy to parallelise (Jacobi, for example, is trivial)
- But many popular ones are not (e.g. ILU)
- Geometric multigrid is possible, but that is not an option on unstructured grids
- So what about algebraic multigrid?

Algebraic multigrid phases

Coarsening

- problem with coarsening across processor boundaries
- can operate decoupled (sub-optimal)
- various coupling strategies
- limits coarse grid size (to #cpus)

Projection / interpolation

- requires no communication

Smoothing

- Jacobi smoothing is decoupled
- Gauss-Seidel most popular?

Changes needed to support parallel operations for Trilinos/ML are minor (with nodal partitioning)

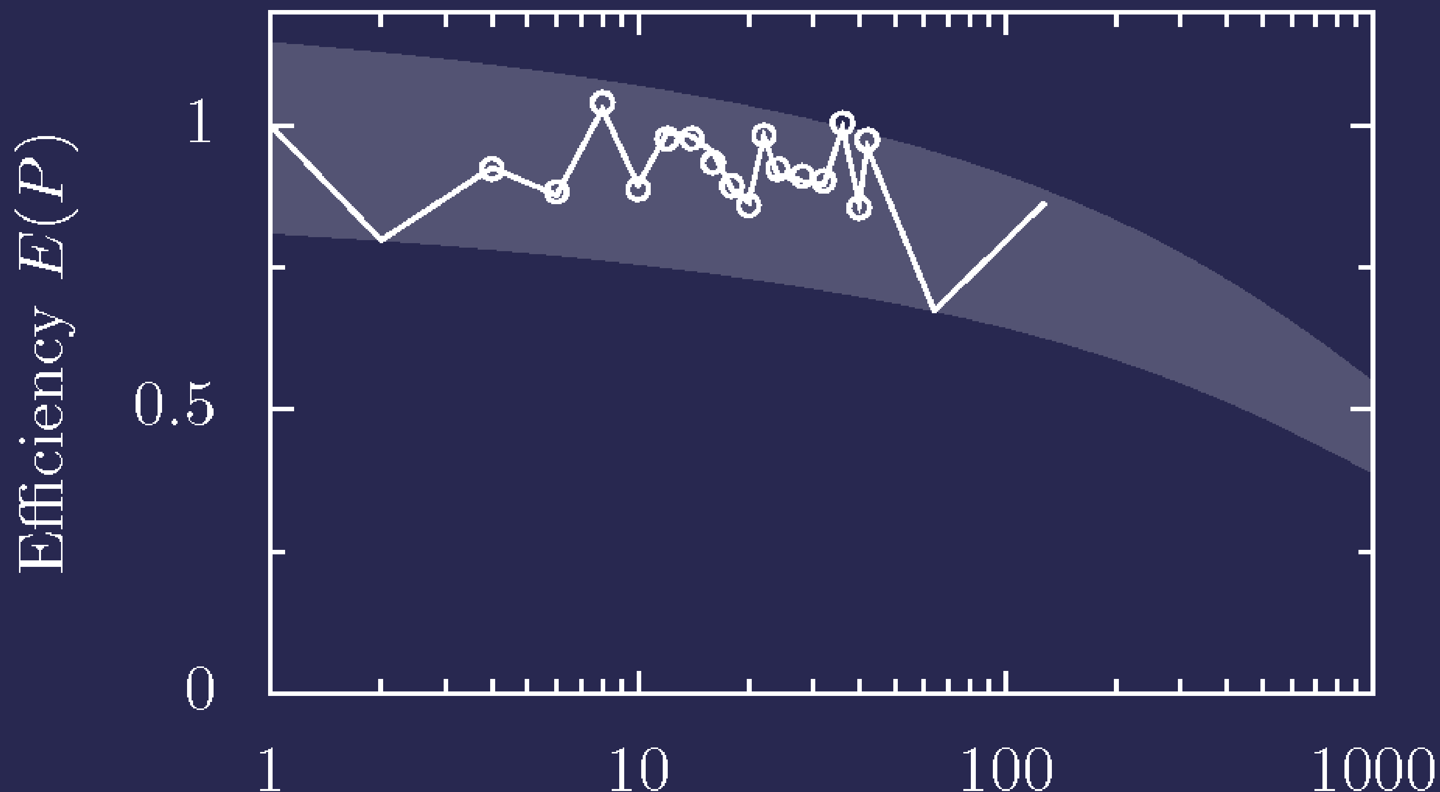
In addition to the sequential interface code, we need to pass in

- the number of local rows
- a function which updates ghost-nodes in a vector

Also, the “matvec” and “apply” functions must of course be parallel-aware

- “matvec” updates ghost-nodes before multiplication
- “apply” updates ghost-nodes after V cycle

Algebraic multigrid shows great promise as parallell preconditioner for the coupled system



Parallel efficiency of the AMG-preconditioned BiCGStab solver (ML/Trilinos+Diffpack)

Some key questions

What is the target?

Multicore, < 8 cpus	➔ OpenMP
Cluster, or > 16 cpus	➔ MPI
> 100 cpus	➔ Hybrid
> 1000 cpus	➔ ???

Can existing libraries be used?

- Trilinos (MPI)
- Hypre (hybrid)

The geomechanical model: Equation for the fluid pressure

$$S\frac{\partial p}{\partial t} = \nabla \cdot (\Lambda \nabla p) - \nabla \cdot (\Lambda \rho_{\mathtt{f}} (1 - \beta_{\mathtt{f}} (T - T_{\mathtt{0}})) \mathtt{g})$$

S	storage coefficient
Λ	mobility of flow
ρ	density
β	thermal expansion coefficient

The geomechanical model: Equation for the temperature

$$C\frac{\partial T}{\partial t} + \rho_{\mathsf{f}} C_{\mathsf{f}} \mathsf{v}_{\mathsf{D}} \cdot \nabla T = \nabla \cdot (\kappa \nabla T)$$

v_{D}	Darcy velocity (in porous media) $= \phi \mathsf{v}_{\mathsf{f}} = -\Lambda(\nabla p - \rho_{\mathsf{f}}(1 - \beta_{\mathsf{f}}(T - T_{\mathsf{0}})\mathsf{g})$
ϕ	porosity
C	bulk heat capacity $= \phi \rho_{\mathsf{f}} C_{\mathsf{f}} + (1-\phi)\rho_{\mathsf{s}} C_{\mathsf{s}}$
C_{f}	specific heat
κ	thermal conductivity

The geomechanical model: Equation for the deformation (elastic)

$$0 = \nabla \cdot \sigma + \rho \mathsf{g}$$

$$\sigma = (\lambda \nabla \cdot \mathsf{u} - \alpha p - \beta_{\mathsf{s}}(3\lambda + 2\mu)(T - T_{\mathtt{0}}))\mathtt{I} + 2\mu\epsilon$$

μ, λ Lamé material constants

α Biot factor ≈ 1

u displacement field

ϵ deformation tensor $= (\nabla \mathsf{u} + (\nabla \mathsf{u})^{\mathsf{T}})/2$