Parallel Numerics
Remarks for Parallel Algorithms and Implementation

Uwe Küster

kuester@hlrs.de

University of Stuttgart
High-Performance Computing Center Stuttgart (HLRS)
www.hlrs.de

overview

- parallel programming models
- algorithms
- parallelization by domain decomposition
- algorithms for large equation systems
- (implemention)
- (hardware properties)

different aspects of parallelism

- algorithmic view
- software view
- hardware view
- performance view

parallelism: hardware view

- pipelining in a pipeline of concatenated pipelined units
  (load -> add -> store)
- different units execute different instructions
  (load, store, multiply-add, integer add, branch, ...)
- parallel threads (pthread, OpenMP,...) on shared memory systems
- parallel threads on ccNUMA systems
- HPF on shared memory and distributed memory systems
- parallel message passing on distributed memory systems

- future model:
  dynamic generation and destruction of threads

parallelism: software view

- vectorization and software pipelining
  (automatic by Hardware and compiler, F90 array syntax)
- parallel execution of outer or splitted loops (pthread, OpenMP)
- parallel execution of independent program parts (pthread, OpenMP)
- parallel execution in distributed arrays
  (HPF, automatic by compiler)
- parallel execution of different domains
  (OpenMP, Message Passing, MLP, one sided message passing, active messages)

parallelism: performance view

- vectorization and software pipelining
  - factor of 1 - >50 due to programming techniques
  - caching, arrays, no calls
- parallel execution of outer or splitted loops
  - for small number of processors (< 16)
  - penalties by frequent synchronization at loop ends
- parallel execution of independent program parts
  - limited number
- parallel execution of different domains
  - message passing: large domains because of latency overhead
  - OpenMP: avoid cacheline sharing, suppress cache coherency mechanism

parallelism: algorithmic view

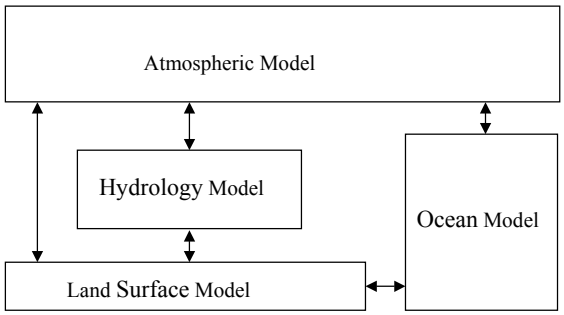
- parallel operations on independent sets of data (SIMD)
- functional decomposition (MPMD)
- recursively generated subtasks, tightly connected threads (??)
- domain decomposition of calculation areas (SPMD)

former Flynn notatation: SISD, SIMD, (MISD), MIMD

parallel operations on independent sets of data:

- parallel jobs on independent data (cluster, internet)
  - most simple and most effective approach
  - very loosly coupling needed
  - Great Internet Mersenne Prime Search (GIMPS)
    - **2^13466917 - 1 39-th prime number**
    - **205000 computers**
    - **http://www.mersenne.org**
- subprograms on independent data
  - spawn processes and collect the data

functional decomposition: example

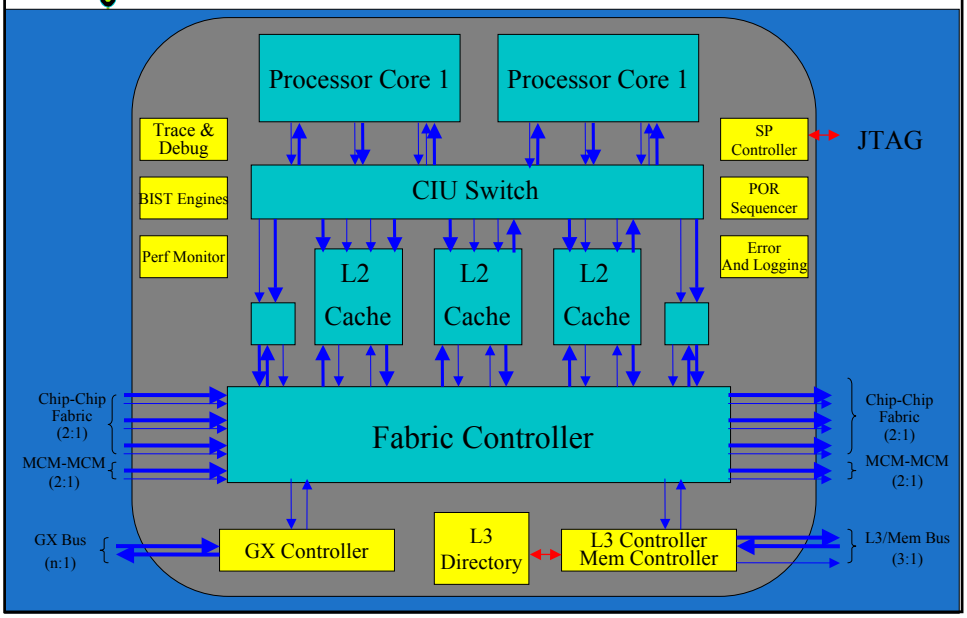


http://www.cs.reading.ac.uk/dbpp/text

recursive task generation, tightly connected threads:

- needs short task creation times
- for shared memory machines
- applicable to CRAY (TERA) MTA
- today not applicable to other machines
- if nested parallelism in OpenMP
- multithreaded processors:
  - Intel Pentium 5+
  - Intel-Compaq-DEC EV8
  - Intel-Madison+
- tightly connected processors:
  - IBM Power 4

IBM Power 4 processor architecture

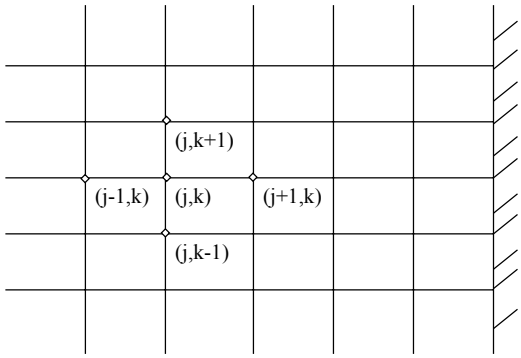


domain decomposition of calculation areas:

- neighbourhoods with a small numbers of neighbours form the base of domain decomposition

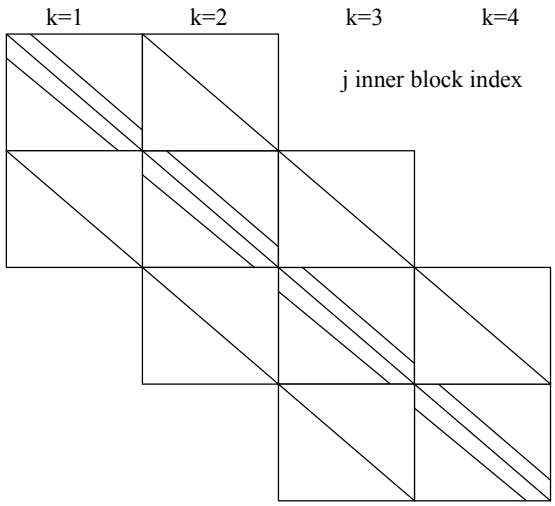
neighbourhoods: Finite Differences

- approximation of differential operators by differences



calculation points on boundaries

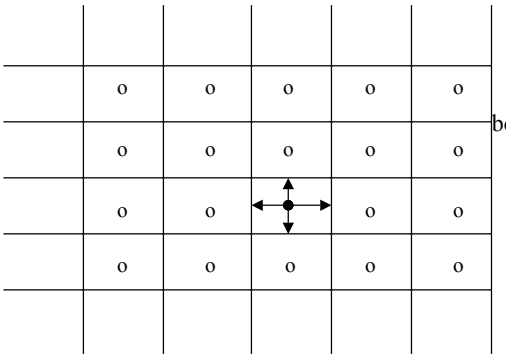
neighbourhoods: Matrix for Regular Grid with 5-Points Stencil



j inner block index

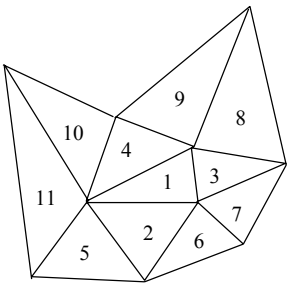
neighbourhoods: Finite Volumes

- flux balance over neighbouring sides



boundary on cell sides

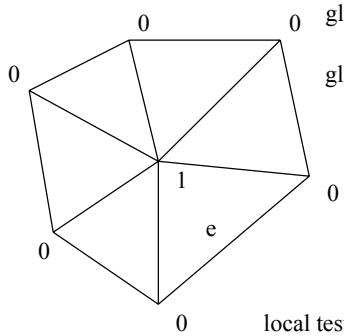
neighbourhoods: Triangular Grid with Relation Matrix



position of matrix elements

	1	2	3	4	5	6	7	8	9	10	11	Σ
1	•	•	•	•								4
2	•	•			•	•						4
3	•		•				•	•				4
4	•			•					•	•		4
5		•			•						•	3
6		•				•	•					3
7			•			•	•					3
8			•					•	•			3
9				•				•	•			3
10				•						•	•	3

neighbourhoods: Finite Elements



global matrix defined by local matrices

global matrices are sums of the element matrices:

$$M^e_{jk} = \int_e \varphi_j \varphi_k \, d\lambda$$

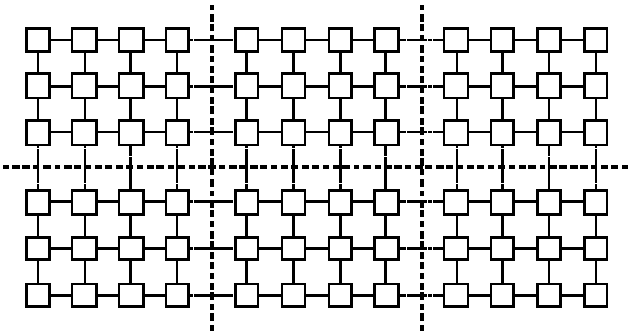
$$G^e_{jk} = \int_e \langle grad\,\varphi_j, grad\,\varphi_k \rangle \, d\lambda$$

local test functions result in sparse global matrices

domain decomposition

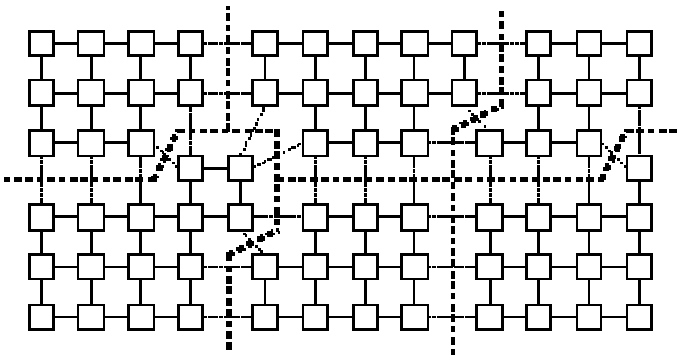
- the natural approach for distribution of work
- the distribution of connected subgrids (subgraphs) to different processors

domain decomposition: regular partitioning of regular grid



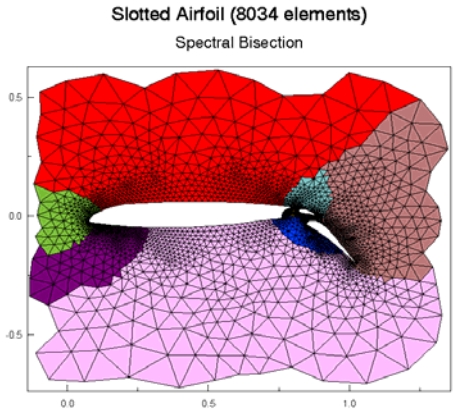
http://www.cs.reading.ac.uk/dbpp/text/node19.html

domain decomposition: irregular partitioning of regular grid



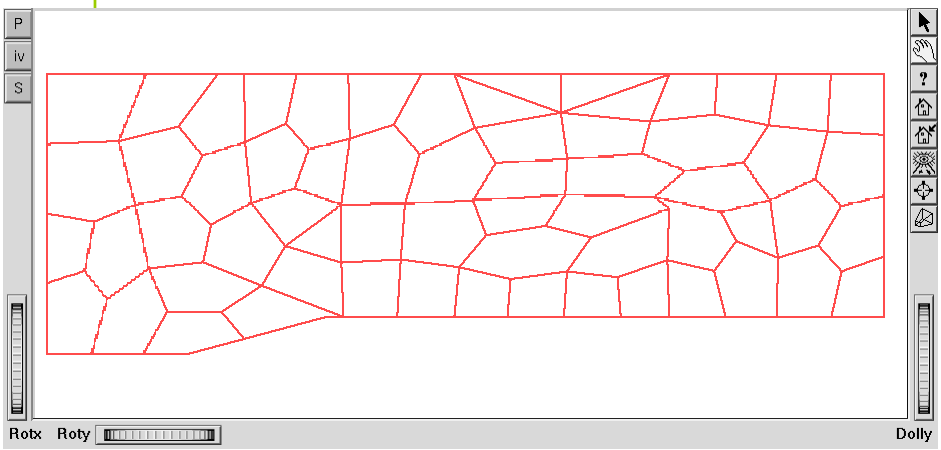
http://www.cs.reading.ac.uk/dbpp/text/node19.html

domain decomposition: recursive Spectral Bisection

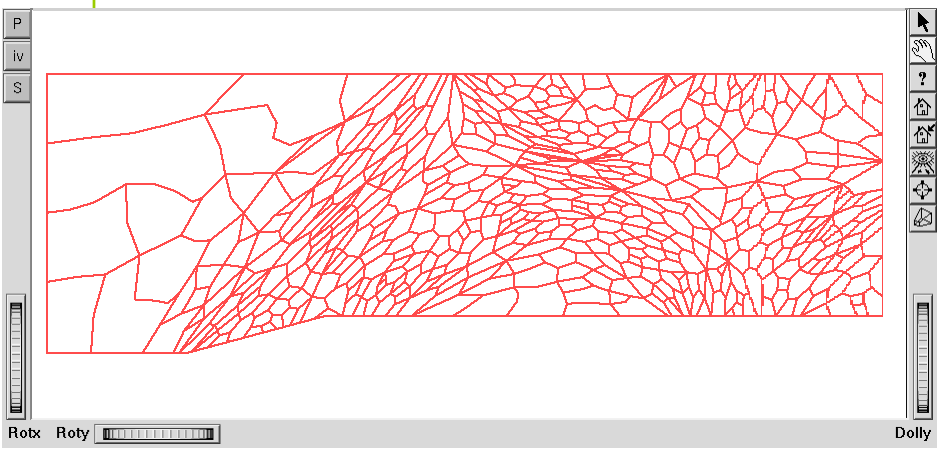


http://www.cs.sandia.gov/CRF/gif/domain_mapping_fig2.gif

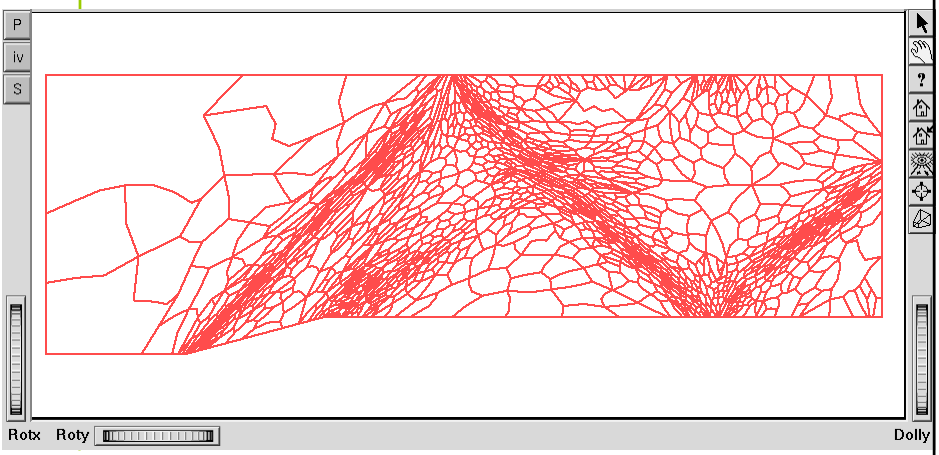
unstructured self adaptive grids: CEQ 1



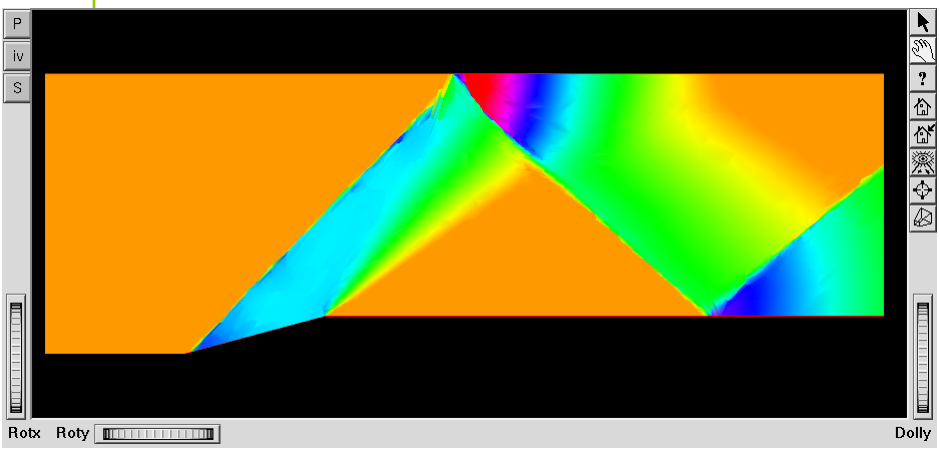
unstructured self adaptive grids: CEQ 2



unstructured for self adaptive grids: CEQ 3



unstructured self adaptive grids: CEQ 4



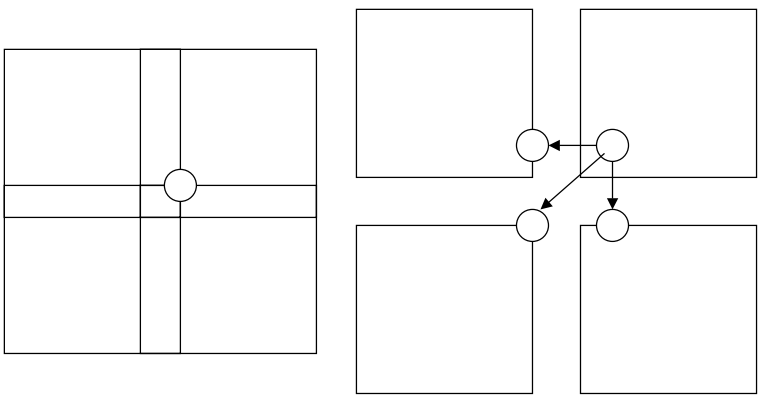
domain decomposition: unstructured self adaptive grids CEQ

- Finite volume mechanism:
  - each cell is defined by cell_to_face relation and momentums
  - each face is defined by face_to_node relation and momentums
  - each cell contains additionally states in a separate data structure
  - each edge contains additionally states in a separate data structure
- states have to be communicated at each time step
- relations and momentums have to be communicated at each refinement and recoarsening step

domain decomposition: the difficult part of implementation

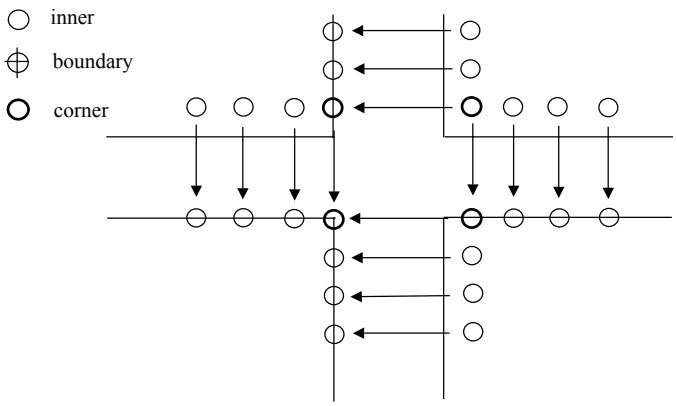
- the grids/graphs are not independent
- communication of overlapping boundaries has to ensure consistent behaviour of algorithmic steps
- the implementation has to be correct and efficient
- possible on all kinds of interconnected computers
- simpler on shared memory machines

domain decomposition: overlapping grids and communication



communications may have complicated pattern for special nodes

domain decomposition: data exchange including the corner

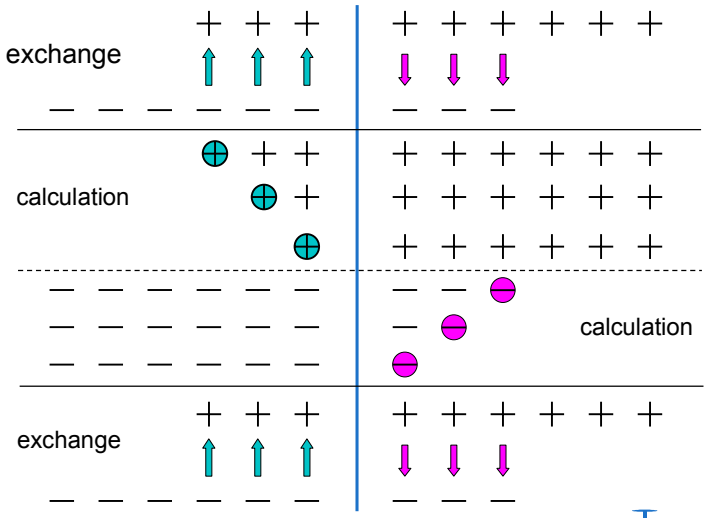


communication in 2 serial steps (2D)

domain decomposition: data exchange including the corners / edges

- usefull if corner points are part of the discretization stencil and if latencies are large
- reduces number of messages
  - for 2D regular patches 4 messages per patch instead of 4+4 messages per patch
  - but 2 serial message groups
- important for 3D problems
  - 6 messages per patch instead of 6+12+8 messages
  - but 3 serial message groups

domain decomposition: working with extended halo



domain decomposition: general properties

- additional operations by overlapping grids
- small patches may fit into caches
- numerical and communication problems for (semi-) implicit procedures
- enumeration problem:
  - mapping numbers/pointers of the neighbouring grids
  - global numbering would be simpler, but no support on pure distributed memory machines
- essential approach for parallelization
- may help in cache reuse for larger caches

domain decomposition: communicated surface for cubus

2D case

grid size: $m * m = n$

surface size: $4m = 4n^{1/2}$

relation surface/volume: $4 / n^{1/2}$

3D case

grid size: $m * m * m = n$

surface size: $6m * m = 6n^{2/3}$

relation surface/volume: $6 / n^{1/3}$

communication/calculation ratio worse for 3D!

domain decomposition: load balancing

- load for all processes should be the same
- time defect of only one processor affects hundreds of other processors
- different techniques of load balancing
- problem of load shifting for dynamically generated load

Recursive Spectral Bisection as example

nodes even number of nodes

E edge relation of a graph

graph Laplacian is given by

$$Q = (q_{vw})_{v,w \in nodes}$$

$$q_{vw} = \begin{cases} -1 & (v,w) \in E \\ \deg ree(v) & v = w \\ 0 & otherwise \end{cases}$$

determine Eigenvector (Fiedlervector) x for the first Eigenvalue $\lambda > 0$

calculate

$$median = \frac{1}{|nodes|} \sum_{v \in nodes} x_v$$

define sets by

$$p_v^+ = \begin{cases} 1 & x_v > median \\ 0 & otherwise \end{cases}$$

$$p_v^- = \begin{cases} 1 & x_v < median \\ 0 & otherwise \end{cases}$$

if $p^+ + p^- = 1$

then $p^+ and\ p^-$ are connected

and best in some sense

large systems

Solution of Large Systems

large systems

- the discrete solution of partial differential equations results in the problem of solving a large (non)linear system (dimension of the system: 10^4 - 10^9)
- nonlinear systems are linearized: replace

$$L(q) = f$$

by

$$\partial L / \partial q \, \Delta q = f - L(q)$$

$$q_{new} = q + \Delta q$$

large systems: two different approaches

- explicit procedures like

$$q^{n+1} - q^n = \Delta t \, A q^n$$

only need a single operator evaluation (or a few for Runge Kutta)

- implicit problems

$$q^{n+1} - \Delta t \, A q^{n+1} = q^n$$

require the solution of a large linear system

large systems: main directions of linear solvers

- direct solvers for ‚small' matrices
- simple iterative procedures
- Krylov space procedures
- Multi level procedures O(n), O(n log n), O(n (log n)^2)

direct solvers

- good public domain software for dense matrices:
  direct solvers for dense matrices and
  eigenvalue solvers for dense matrices in
  - Lapack-3,
  - Scalapack
- direct solvers for some types of sparse matrices (SPD)
  - MUMPS

simple iterative procedures: Gauss-Seidel

$$M\Delta q = -(Aq - f) \quad \leftrightarrow$$
$$q_{new} = q + \omega\Delta q$$

Gauss-Seidel Iteration for M=L+D

```
do element in all_elements
   delta = operation_of( element%value , neighbourhood_elements%value )
   element%value= element%value + omega*delta
enddo
```

in general non parallelizable recursion

parallelizable by coloring

simple iterative procedures: Jakobi

$$M\Delta q = -(Aq - f) \quad \leftrightarrow$$
$$q_{new} = q + \omega\Delta q$$

Jakobi Iteration for M=D

```
do element in all_elements
   delta(element)=operation_of(element%value,neighbourhood_elements%value)
enddo
do element in all_elements
   element%value= element%value+omega*delta(element)
enddo
```

all operations can be done in parallel

Krylov space algorithms

- CG
- Lanczos
- BiCO
- CGS
- BiCGSTAB(l)
- TFQMR
- ORTHOMIN
- GMRES
- GMRESR

all have the same building blocks

Conjugate Gradient Squared (CGS)

start values

$r_0 = L^{-1} A x_0 - b \quad \leftrightarrow$

$\bar{r}_0 = ? \quad \leftrightarrow$

$q_0 = 0$

$p_{-1} = 0$

$\rho_{-1} = 1$

Iteration

$do\ k = 1, k\max$

$\rho_k = \langle \bar{r}_0, r_k \rangle \quad \leftrightarrow$

$\beta_k = \rho_k / \rho_{k-1}$

$u_k = r_k + \beta_k q_k$

$p_k = u_k + \beta_k (q_k + \beta_k p_{k-1})$

$v_k = L^{-1} A U^{-1} p_k \quad \leftrightarrow$

$\sigma_k = \langle \bar{r}_0, v_k \rangle \quad \leftrightarrow$

$\alpha_k = -\rho_k / \sigma_k$

$q_{k+1} = u_k + \alpha_k v_k$

$w_k = \alpha_k U^{-1} (u_k + q_{k+1}) \quad \leftrightarrow$

$r_{k+1} = r_k + L^{-1} A w_k \quad \leftrightarrow$

$x_{k+1} = x_k + w_k$

$enddo$

$\leftrightarrow$ data exchange between processors

BiCGSTAB(2)

start values

x_0 *is an initial guess*; $r_0 = b - Ax_0$;
$\bar{r}_0$ *is an arbitrary vector, such that* $(r, \bar{r}_0) \neq 0$,
e.g. $\bar{r}_0 = r$;
$\rho_0 = 1; u = 0; \alpha = 0; \omega_2 = 1;$

even Bi-CG step

$for\ i = 0,2,4,6,\cdots$
$\rho_0 = -\omega_2 \rho_0$
$\rho_1 = (\bar{r}_0, r_i)$ $\leftrightarrow$
$\beta = \alpha\, \rho_1 / \rho_0; \rho_0 = \rho_1$
$u = r_i - \beta u;$
$v = Au$ $\leftrightarrow$
$\gamma = (v, \bar{r}_0)$ $\leftrightarrow$
$\alpha = \rho_0 / \gamma;$
$r = r_i - \alpha v;$
$s = Ar$ $\leftrightarrow$
$x = x_i + \alpha\, u;$

odd Bi-CG step

$\rho_1 = (\bar{r}_0, s)$ $\leftrightarrow$
$\beta = \alpha\, \rho_1 / \rho_0; \rho_0 = \rho_1$
$v = s - \beta v;$
$w = Av$ $\leftrightarrow$
$\gamma = (w, \bar{r}_0)$ $\leftrightarrow$
$\alpha = \rho_0 / \gamma;$
$u = r - \beta u$
$r = r - \alpha v$
$s = s - \alpha w$
$t = As$ $\leftrightarrow$

GCR(2) - part

$\omega_1 = (r, s); \mu = (s, s); v = (s, t); r = (t, t)$
$\omega_2 = (r, t)$ $\leftrightarrow$
$r = r - v^2 / \mu;$
$\omega_2 = (\omega_2 - v\omega_1 / \mu) / r;$
$\omega_1 = (\omega_1 - v\omega_2) / \mu$
$x_{i+2} = x + \omega_1 r + \omega_2 s + \alpha u$
$r_{i+2} = r - \omega_1 s - \omega_2 t$
$if\ x_{i+2}\ accurate\ enough\ then\ quit$
$u = u - \omega_1 v - \omega_2 w$
end

$\leftrightarrow$ data exchange between processors

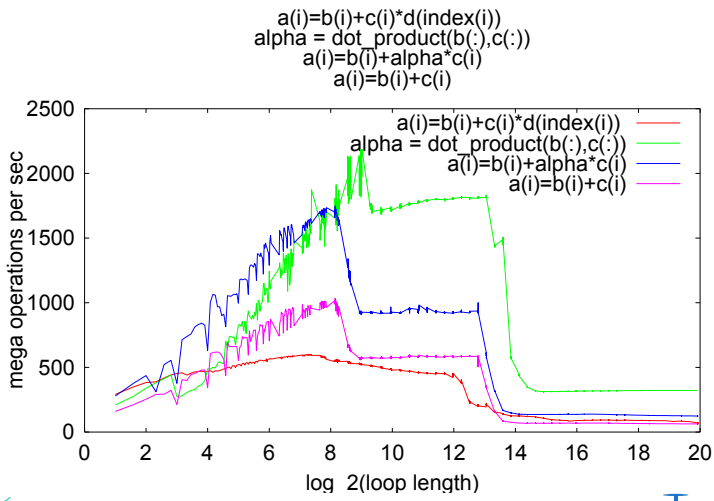
operations of Krylov space algorithms

- some scalar operations
- scalarproduct of vectors
- a = b + alpha*c (daxpy)
- vector = matrix * vector (time critical)
- preconditioning

operations of Krylov space algorithms

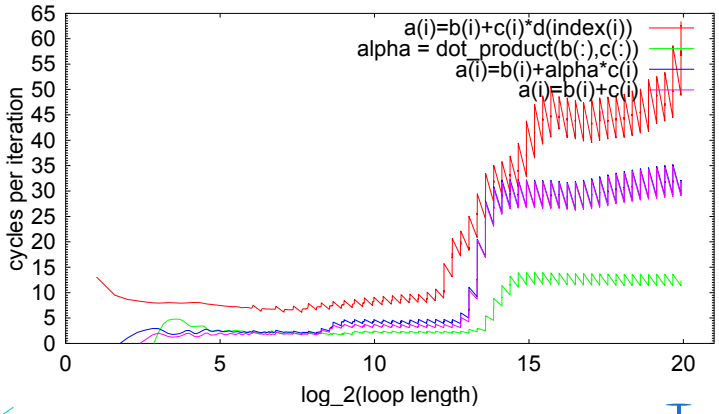
- all these operations are vectorizable and parallelizable
- all operations allow domain decomposition
- arrays as datastructure
- work can be done in loops
- all essential loops are large
- limited cache reuse
- matrix X vector multiply may be formulated in general way
  - if possible, use your specific formulation

Intel Xeon 2GHz, performance of CG specific operations



Intel Xeon 2GHz, cycles of CG specific operations

a(i)=b(i)+c(i)*d(index(i)) ,_start_up=7.93_cycles,_cycles_per_it=7.99
alpha = dot_product(b(:),c(:)),_start_up=84.69_cycles,_cycles_per_it=3.65
a(i)=b(i)+alpha*c(i),_start_up=37.86_cycles,_cycles_per_it=2.28
a(i)=b(i)+c(i),_start_up=37.53_cycles,_cycles_per_it=1.60



preconditioning

- Krylov space are fast for matrices with small condition
- preconditioning decreases condition number

$$Ax = b \quad \rightarrow \quad (L^{-1}AU^{-1})\,Ux = L^{-1}b$$

- preconditioning is a problem for vectorization and parallelization
- Jakobi (diagonal) preconditioning simple
- Block Jakobi (diagonal blocks) may be efficient

ILU preconditioning

- ILU is a LU decomposition with no or limited fill in with respect to the original sparse matrix. Neglected elements may be added to the diagonal.
- ILU preconditioning is highly recursive as well as for the decomposition step as well as for the calculation step
  - applicable to microprocessors
  - not applicable to vector computers
- ILU shows the same parallelization problems as the sparse matrix vector multiplication. Additional synchronization points are necessary as well for the decomposing step as for the preconditioning step
- ILU for the blocks of a domain decomposition

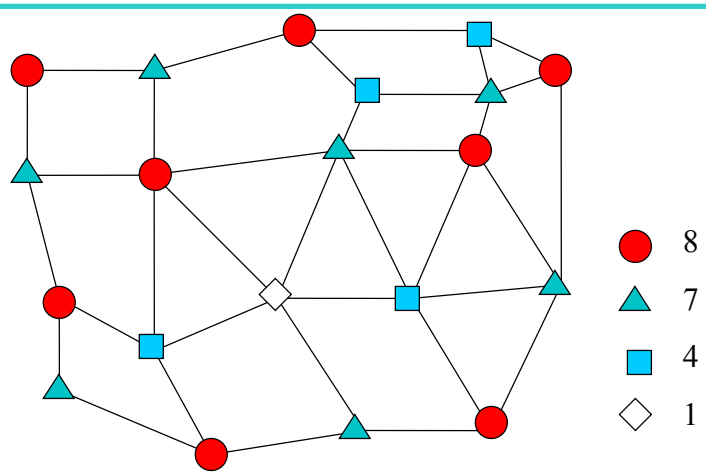
MILU for general matrix

```
do   l = 1, n-1
        do  j = l+1, n
         if ((j, l) ∈  S ) then
            aa = a_{jl}/a_{ll}
            aa_{jl} = aa
            do k = l+1, n
               if ((l, k) ∈  S ) then
                  if ((j, k) ∈   S ) then
                     a_{jk} = a_{jk} - aa*a_{lk}
                  else
                     a_{jj}  = a_{jj}  - aa*a_{lk}
                  endif
               endif
            enddo
         endif
      enddo
   enddo
```

graph coloring: defining independent sets for parallelization

- helps preconditioning
- histogramm loop
- assembling stiffness matrices
- assembling forces on nodes
- distribution of states on the neighbourhood
- (collection is no problem)

graph coloring: my neighbour has not my color



graph coloring:

- vectorization or parallelization for all elements in a single color
- a color may be a complete patch in a decomposed domain
- only a small number of colors for large independent sets
- finding the smallest number of colors is np-complete !
  some approximation sufficient
- calculation of colors is expensive
- pays out only if used several times

Multigrid 1

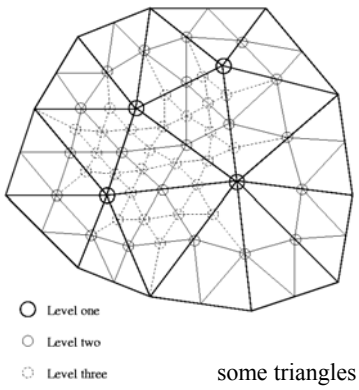
to be solved	$L_f(x_f) = f_f$
presmoothing	$x_f := S_f^{\nu_1}(x_f)$
defect	$d_f := -(L_f(x_f) - f_f)$
restriction	$r_g := I_{f\to g}(d_f)$
coarse grid correction	$L_g(x_g + \varepsilon\,\Delta x_g) = L_g(x_g) + \varepsilon\, r_g$
prolongation	$x_f := x_f + I_{g\to f}(\Delta x_g)$
postsmoothing	$x_f := S_f^{\nu_2}(x_f)$

solve coarse grid correction with the next coarser grid

Multigrid 2

- smoothing and postsmoothing by a usual relaxation technique
- restriction and prolongation should be a problem dependent interpolation
- coarse grid correction do be done by the same mechanism
- coarse grids should reside on the same processor as the fine grids
- communication necessary in all steps
- very coarse grids should be calculate on only one processor
- distribution of load difficult
- poor cache reuse but fast mxing of information

Hierarchy of triangles



some triangles have only 2 children!

Concluding remarks

- programs with high performance today are a combination of pipelined (vectorized) code and parallel execution with a domain decomposition technique
- decrease the bandwidth needs of the code!
- try to increase the size of the patches and to decrease their surface
- load balancing and load distributing can be difficult
- use arrays and loops for the computing intensive parts
- there is a trade off between ‚good' programming style and processor performance
- flexibility and dynamic features decrease performance in the time dominant parts

Books and URLs 1

Andreas Meister:
Numerik linearer Gleichungssysteme - Eine Einführung in moderne Verfahren.
Vieweg 1999.

Ian Foster
Designing and Building Parallel Programs
Addison-Wesley, 1995
ISBN 0-201-57594-9
http://www.cs.reading.ac.uk/dbpp/text

Books and URLs 2

JOSTLE
http://www.gre.ac.uk/~jjg01/

DRAMA
http://www.cs.kuleuven.ac.be/cwis/research/natw/DRAMA/index.html

Patrick Amestoy, Iain Duff, Jean Yves L'Excellent, and Petr Plechác.
PARASOL An integrated programming environment for parallel sparse matrix solvers.

Technical report, Department of Computation and Information, 1998.
http://www.genias.de/projects/parasol/

end