Skeleton-Based Implementation of Adaptive Multigrid Algorithms

George H. Botorog

Aachen University of Technology

The Goal

Concept for parallel programming ($\Rightarrow$ programming language), which:

1. allows high-level and efficient parallel programming
2. can be used in 'real' applications

Approaches to High-Level Parallel Programming

- HPF (distributed arrays, parallel loops)
- SVM (shared virtual memory)
- automatic parallelization of functional languages
- automatic parallelization of loops in imperative languages (Fortran)
- Linda (tuple space)
- Gamma (chemical reaction model)
- Strand (parallel logic programming)
- BSP (Bulk Synchronous Processing)
- Skeletons
- ...

Algorithmic Skeletons

Skeletons:
- parameterized procedures (polymorphic, higher-order functions)
- common parallelization patterns
- the only parallel constructs in programs

Examples:

map *(forall)*
fold *(collect)*
farm *(master/slave)*
divide&conquer

Skeletons:
- data-parallel
- process-parallel [Freiberg 97]

Efficiency:
- efficiency of the skeletons
- efficiency of the host language

The First Goal

Goal:

- high-level host language
- efficient implementation

Achieved through:

- imperative language, enhanced by a minimal set of functional features:
  - higher-order functions
  - currying
  - polymorphic types
  - distributed data structures
- compile-time elimination of the functional features (minor restrictions on recursively-defined HOFs) [CC 96 (poster)]
- host: *Skil* (*Sk*eleton *i*mperative *l*anguage) [HPDC-5]

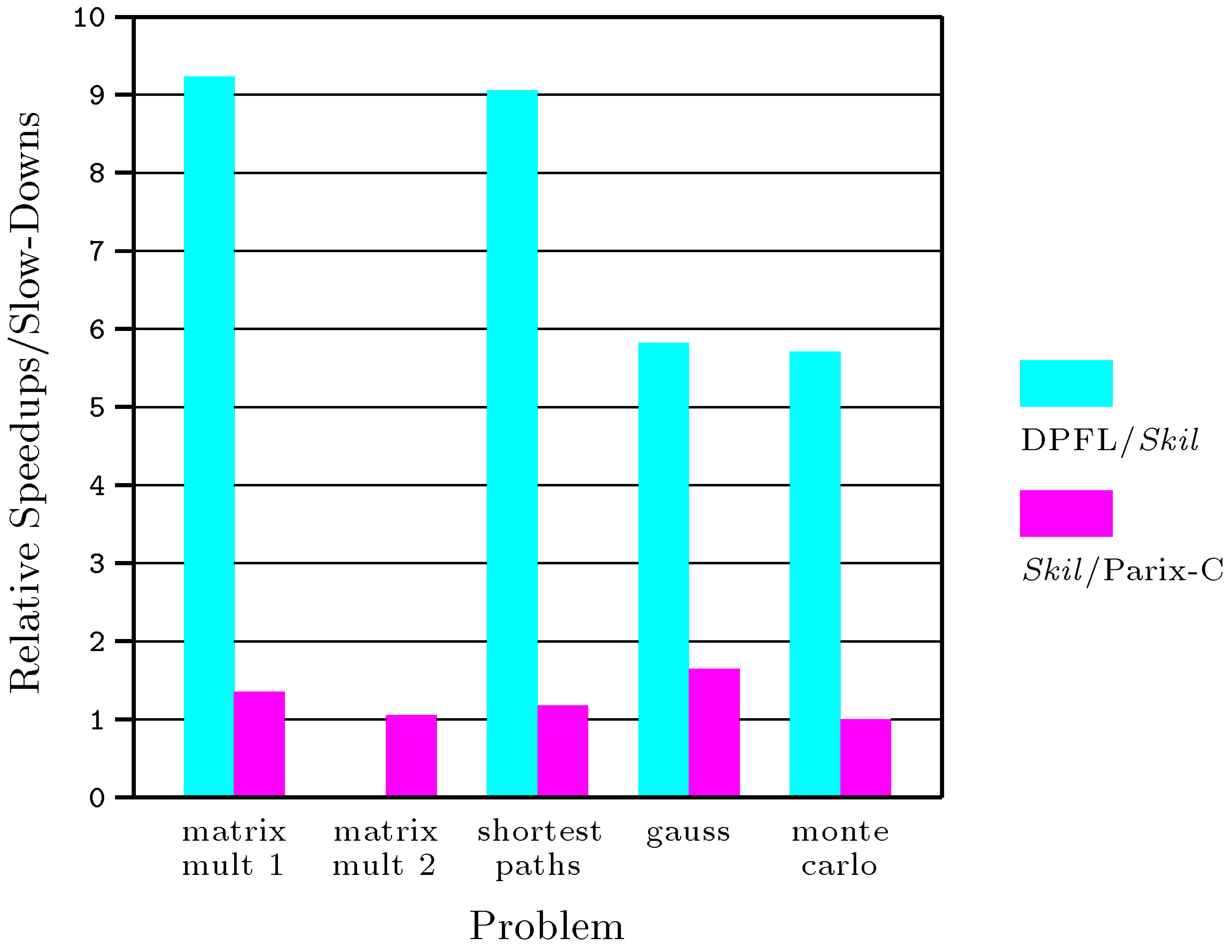
Case Study: Array Skeletons

[EuroPar 96]

```
array <$t> array_create (int dim, Index size, Index blocksize,
                         Index init_lowerbd (int), $t init_f (Index)) ;

void array_map ($t2 map_f ($t1, Index), array <$t1> from,
                array <$t2> to) ;

$t2 array_fold ($t2 conv_f ($t1, Index), $t2 fold_f ($t2, $t2),
                array <$t1> a) ;

void array_permute_blocks (array <$t> a, int perm_f (int)) ;

...
```

Motivation: comparison with the DPFL implementation
(functional language with skeletons [Kuchen 95])

Comparison between ***Skil***, DPFL and C



The Second Goal

Skeletons:

- applicability in solving *"real world"* problems
- same set of skeletons usable in solving *related* problems or *different* problems with *similar* structure

Our application area: Adaptive Multigrid Methods

Why Multigrid ?

Numerics: fastest known methods for solving (elliptic) partial differential equations

Method	Complexity	CPU Time*
Gaussian Elimination	$\mathcal{O}(N^2)$	ca. 1 day
SOR	$\mathcal{O}(N^{\frac{3}{2}} \log N)$	ca. 30 min.
MG	$\mathcal{O}(N \log N)$	18.5 sec.
FMG	$\mathcal{O}(N)$	7.6 sec.

*Mainframe, mid 80's.

Parallelization:
- suited for parallelism
- non-trivial parallelization (especially with adaptive refinement)

Skeletons:
- non-trivial (numerical) application
- *series* of methods, all built from the *same* blocks (w.r.t. computational structure)
- other applications can be implemented with these skeletons (N-body problems)
- adaptive MG $\Rightarrow$ dynamic distributed data structures

Basics

The problem:

$$Lu = f \quad \text{in} \quad \Omega \subset \mathbb{R}^2 \qquad\qquad -\Delta u = f$$
$$u = g \quad \text{in} \quad \partial\Omega$$

$\Downarrow$ discretization (finite differences) $\Downarrow$

$$L_h u_h = f_h \quad \text{in} \quad G_h := \Omega \cap \mathcal{G}_h \qquad\qquad -\Delta_h u_h = f_h$$
$$u_h = g_h \quad \text{in} \quad \partial G_h \qquad\qquad -\Delta_h^* = \frac{1}{h^2}\begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix}$$

The residual equation:

$$L_h u_h = f_h \tag{1}$$

Iterative solution $\Rightarrow$ approximation $w_h^{(n)} \approx u_h$

Algebraic error: $\tilde{e}_h^{(n)} \overset{\text{def}}{=} u_h - w_h^{(n)}$

Residual: $r_h^{(n)} \overset{\text{def}}{=} f_h - L_h w_h^{(n)}$

Then: $r_h^{(n)} = f_h - L_h(u_h - \tilde{e}_h^{(n)}) \overset{\text{lin}}{=} f_h - L_h u_h + L_h \tilde{e}_h^{(n)} \Rightarrow$

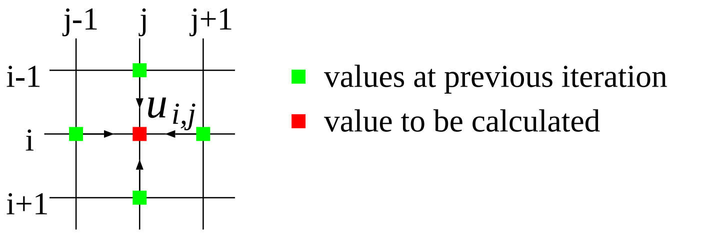
$$L_h \tilde{e}_h^{(n)} = r_h^{(n)} \tag{2}$$

Idea: (iterative) method for solving (1) $\Leftrightarrow$
(iterative) method for solving (2)

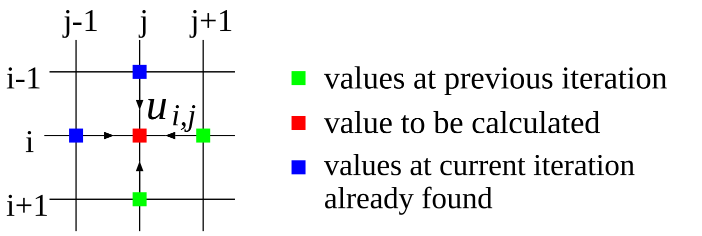
Relaxation

Why:
- usually huge equation systems
- iterative methods faster than exact ones
- solution approximative, due to discretization

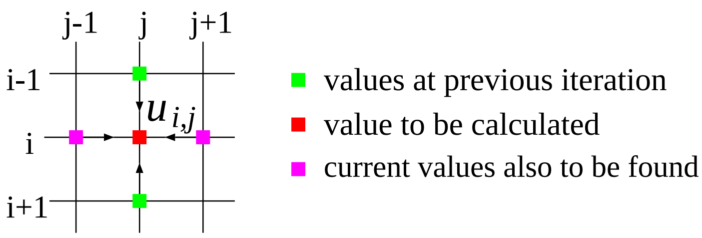
How: Jacobi



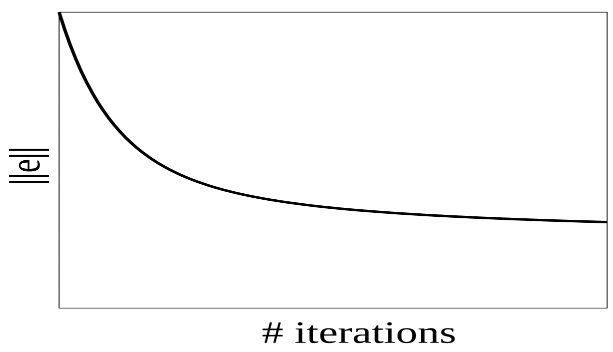
Gauss-Seidel

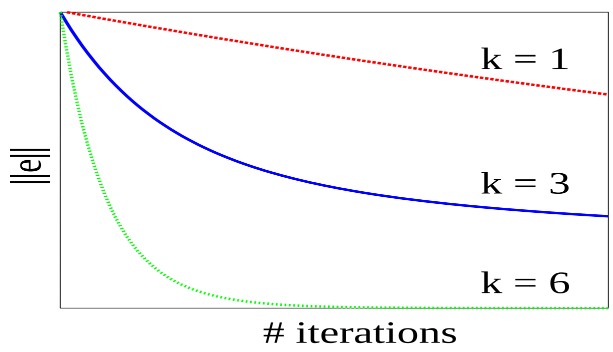


LSOR



Problem: slow convergence





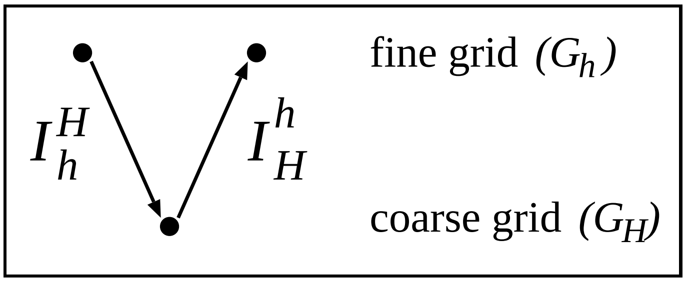
High frequencies are damped fast $\Rightarrow$ smoothers.

Coarse Grid Correction

Idea: $L_h \tilde{e}_h = r_h \quad \Rightarrow \quad L_H \tilde{e}_H = r_H \quad (H > h)$

Coarse grid correction:

residual computation	$r_h^{(n)} := f_h - L_h w_h^{(n-1)}$
residual restriction	$r_H^{(n)} := I_h^H r_h^{(n)}$
exact solution on coarse grid	$L_H \tilde{e}_H = r_H^{(n)}$
correction transfer	$\tilde{e}_h^{(n)} := I_H^h \tilde{e}_H^{(n)}$
correction	$w_h^{(n)} := w_h^{(n-1)} + \tilde{e}_h^{(n)}$



Correction scheme (CGC + relaxation):

pre-smoothing	$\overline{w}_h^{(n)} := RELAX^{\nu_1}(w_h^{(n-1)}, L_h, f_h)$
residual computation	$r_h^{(n)} := f_h - L_h \overline{w}_h^{(n)}$
$\vdots$	$\vdots$
correction	$w_h^{(n)} := \overline{w}_h^{(n)} + \tilde{e}_h^{(n)}$
post-smoothing	$w_h^{(n)} := RELAX^{\nu_2}(\overline{w}_h^{(n)}, L_h, f_h)$

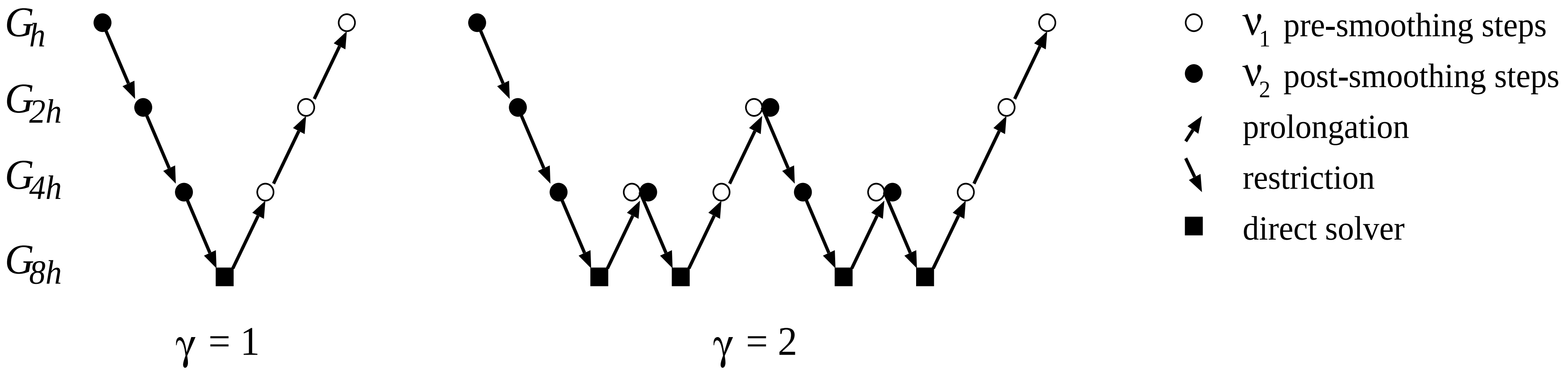
The Multigrid Cycle

Idea: instead of *exact* solution on coarse grid, *recursive* application of CS (γ times), until direct coarse grid solution is possible

$$\Downarrow$$

multigrid cycle

V and W cycles:



The Full Approximation Scheme (FAS)

Idea: • *classic multigrid:* – approximate solution on finest grid
– corrections on all coarser grids

• *FAS:* approximations of the solution on all grids

Coarse grid correction (2-level):

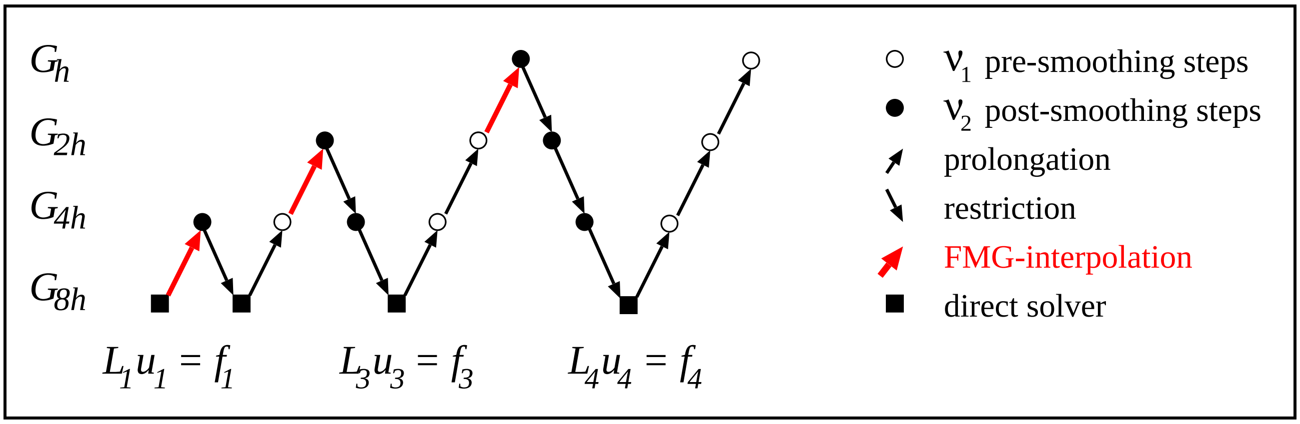
pre-smoothing	$\overline{w}_h^{(n)} := RELAX^{\nu_1}(w_h^{(n-1)}, L_h, f_h)$
residual computation	$r_h^{(n)} := f_h - L_h \overline{w}_h^{(n)}$
residual restriction	$r_H^{(n)} := I_h^H r_h^{(n)}$
exact solution on coarse grid	$L_H u_H^{(n)} = L_H \hat{I}_h^H \overline{w}_h^{(n)} + r_H^{(n)}$
correction computation	$\tilde{e}_H^{(n)} := u_H^{(n)} - \hat{I}_H^h \tilde{e}_H^{(n)}$
correction transfer	$\tilde{e}_h^{(n)} := I_H^h \tilde{e}_H^{(n)}$
correction	$w_h^{(n)} := \overline{w}_h^{(n)} + \tilde{e}_h^{(n)}$
post-smoothing	$w_h^{(n)} := RELAX^{\nu_2}(\overline{w}_h^{(n)}, L_h, f_h)$

Full Multigrid

Problem: initial approximation of solution on the finest grid

Idea: "nested iteration", i.e. coarser grids are used to provide good initial approximations on finer grids

FMG: MG + nested iteration

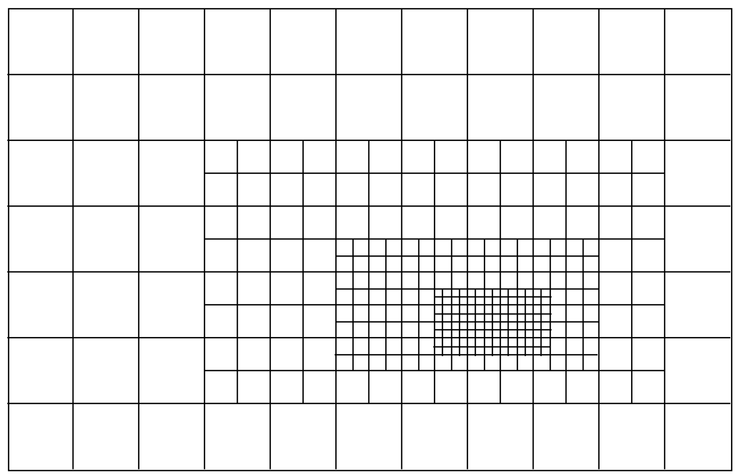


FMG with V correction cycles

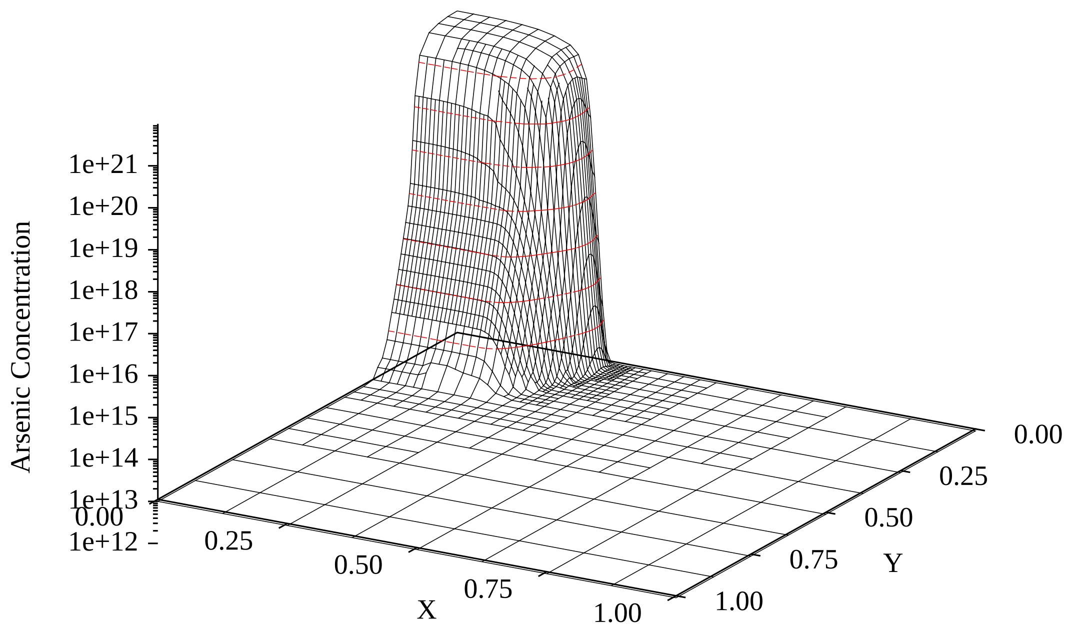
Complexity: $\mathcal{O}(N)$, where N = number of grid points on finest level

The Multi-Level Adaptive Technique

Dual view: finer grids as correction grids $\Rightarrow$
adaptively refined grids

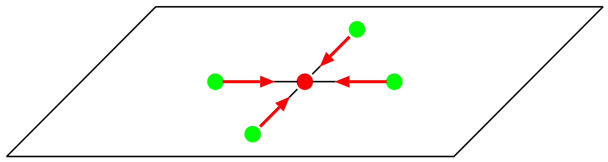


MG + FAS + adaptive refinement $\Rightarrow$ MLAT

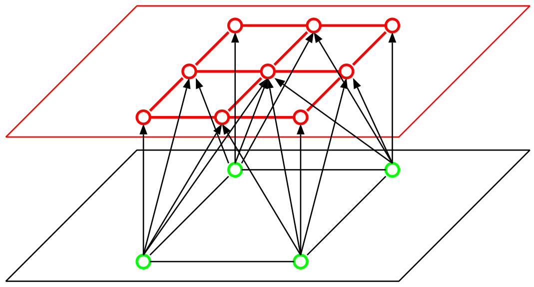


Data Dependencies

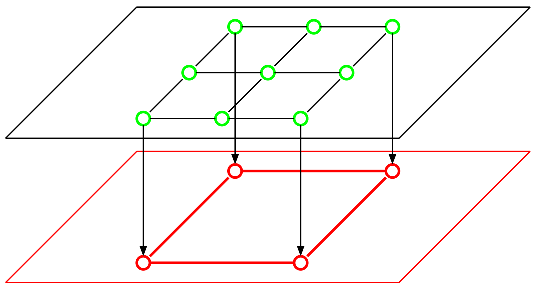
Relaxation:



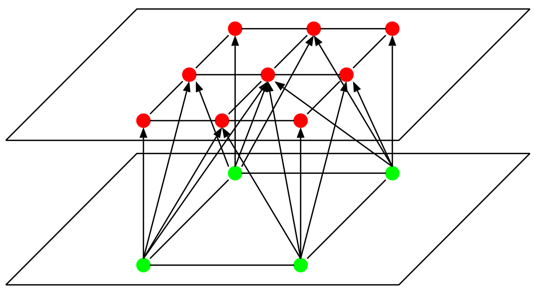
Refinement:



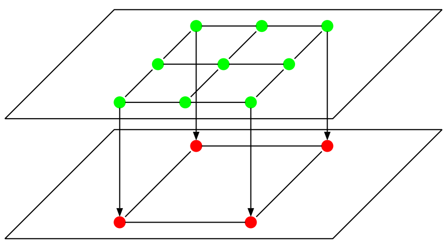
Coarsening:

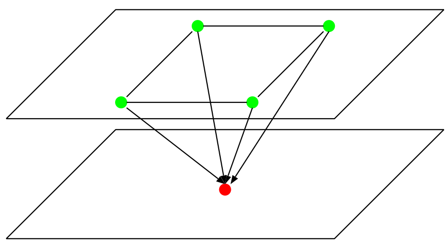


Prolongation:



Restriction:





Multigrid Skeletons

[Irregular 95]

```
pardata MGrid <$t> struct { ... $t data; Coords coords; ... } ;

MGrid <$t> mg_create (Domain dom, int ovl_depth, float h0) ;
void mg_destroy (MGrid <$t> mg) ;
void mg_refine (MGrid <$t> mg, Coords *refine_f (Coords coords)) ;
void mg_coarsen (MGrid <$t> mg, Coords *coarsen_f (Coords coords)) ;
void mg_remove_finest (MGrid <$t> mg) ;
void mg_remove_coarsest (MGrid <$t> mg) ;
void mg_balance (MGrid <$t> mg) ;
void mg_map (MGrid <$t> mg, $t map_f ($t data, Coords coords)) ;
void mg_mapenv (MGrid <$t> mg, int env_depth, int depdcy_f ($t *data,
                Coords *coords), $t mapenv_f ($t *data, Coords *coords),
                Bool use_overlap, Bool upd_overlap) ;
void mg_mapblock (MGrid <$t> mg, int xsize, int ysize, $t mapenv_f (
                  $t *data, Coords *coords)) ;
void mg_get_parents (Coords coords, $t **p_data, Coords **p_coords) ;
void mg_get_children (Coords coords, $t **ch_data, Coords **ch_coords) ;
```

Skeleton-Based Implementation of FMG (1)

```
typedef struct _Data { double u, f, e, r; } *Data ;

void FMG () {
    MGrid <Data> mg ;  int l = 0 ;  Domain dom = ... ;  float h0 = ... ;
    mg = mg_create (dom, 1, h0) ;
    solve exactly on coarsest grid ;
    while (! convergence) {
        mg_refine (mg, local_ad_ref) ;
        mg_balance (mg) ;
        mg_mapenv (mg, 1, int_depdcy_f, interpolate, True, False) ;
        for (i = 0, l++; i < δ; i++)
            CGC (mg, l) ;
    }
    output ;
    mg_destroy (mg) ;
}
```

Skeleton-Based Implementation of FMG (2)

```
void CGC (MGrid <Data> mg, int l) {
    if (l == 0)
        compute exact solution on coarsest grid ;
    else {
        for (i = 0; i < ν1; i++)
            mg_mapenv (mg, 1, null_f, jacobi (ω), True, True) ;

        mg_map (mg, compute_residual) ;
        mg_map (mg, inject_residual) ;

        for (i = 0; i < γ; i++)
            CGC (mg, l-1) ;

        mg_mapenv (mg, 1, int_depdcy_f, interpolate, True, False) ;
        mg_map (mg, { u = u - e ; }) ;

        for (i = 0; i < ν2; i++)
            mg_mapenv (mg, 1, null_f, jacobi (ω), True, True) ;
}
```

The Relaxation

GS: $\texttt{mg\_mapenv (mg, 1, }\left\{\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}\right\}\texttt{, gauss\_seidel, False, True) ;}$

SOR: $\texttt{mg\_mapenv (mg, 1, }\left\{\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}\right\}\texttt{, sor (}\omega\texttt{), False, True) ;}$

LSOR: `mg_mapblock (mg, 0, 1, lsor) ;`

RB-GS: $\texttt{mg\_mapenv (mg, 1, }\{\texttt{if odd (coords) then }\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
$\texttt{else }\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}\}\texttt{, gauss\_seidel, False, True) ;}$

$\texttt{mg\_mapenv (mg, 1, }\{\texttt{if even (coords) then }\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
$\texttt{else }\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}\}\texttt{, gauss\_seidel, False, True) ;}$

Applications

Goal: achieve decoupling of parallelism and numerics

Application classes:
- relative small MG programs
- ‘real world’ applications

Restrictions:
- structured, adaptive grids
- work with stencils
- discretization by finite differences

GMD:
- MG 00
- MG 01
- [Music]

MG 00 and MG 01

MG 00:

- 'didactic' MG program (*sequential*)
- solves the Poisson equation
$$-\Delta u = f$$
on unit square with Dirichlet boundary conditions
- 'classic' multigrid
- V-cycle correction
- no adaptive grid generation
- small (*sequential*)

MG 01: [Stüben, Trottenberg 84]

- solves the Helmholtz equation
$$-\Delta u + c(x,y)u = f$$
on bounded domains described by two functions and Dirichlet boundary conditions
- FMG
- FAS
- V or W-cycle correction
- no adaptive grid generation
- bigger (*sequential*)

Music

[Joppich, Mijalković 91,93]

- MUSIC = Multigrid Simulator for IC Fabrication Processes
- simulation of: ion implantation, diffusion, oxidation
- equation of the type: $\frac{\partial u}{\partial t} - Au = f \;\; \text{in} \;\; \Omega \times (0, T]$
  $Bu = g \;\; \text{in} \;\; \partial\Omega, \; \forall t \in (0, T]$
  $u = u^0 \;\; \text{in} \;\; \overline{\Omega} \text{ and } t = 0$
- FMG, FAS, V-cycles
- adaptive refinement
- in addition: time discretization
- big (*sequential*)

Conclusions

Skeletons:

- high-level parallel programming
- efficiency:
  - skeleton implementation
  - host language
    - imperative
    - enhanced by functional features
    - compile-time elimination
- ‘real-world’ applications

Multigrid:

- skeletons for MG
- MG applications

To do:

- implementation of MG skeletons
- parallel implementation of the MG applications based on the skeletons