Advanced Topics in Numerical Analysis: High Performance Computing

More distributed memory algorithms

Georg Stadler, Dhairya Malhotra
Courant Institute, NYU

Spring 2019, Monday, 5:10–7:00PM, WWH #1302

May 6, 2019

Outline

Organization issues

Summary of previous class

Multigrid

Organization

Scheduling:

- Homework assignment #6 posted last week, due next Monday.
- How are your final projects coming along? Special office hours for final project this week: Wednesday 1-2pm and Thursday noon-1pm in office #1111. Come by!

Topics today:

- Partitioning and balancing: space filling curves
- Multigrid

Outline

Organization issues

Summary of previous class

Multigrid

Partitioning and Load Balancing

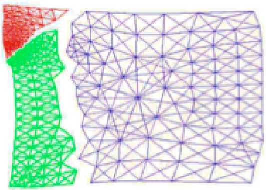
Thanks to Marsha Berger for letting me use many of her slides. Thanks to the Schloegel, Karypis and Kumar survey paper and the Zoltan website for many of these slides and pictures.

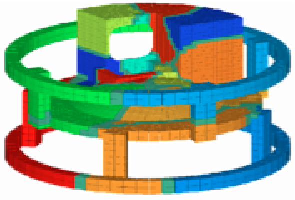
Partitioning

- Decompose computation into tasks to equi-distribute the data and work, minimize processor idle time.

  applies to grid points, elements, matrix rows, particles, ...
- Map to processors to keep interprocessor communication low.

  communication to computation ratio comes from both the partitioning and the algorithm.

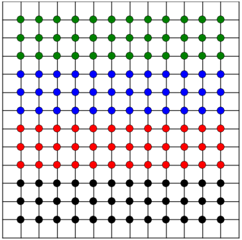


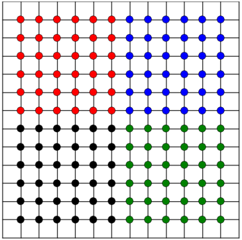


Partitioning

Data decomposition + Owner computes rule:

- Data distributed among the processors
- Data distribution defines work assignment
- Owner performs all computations on its data.
- Data dependencies for data items owned by different processors incur communication





Partitioning

- Static - all information available before computation starts

  use off-line algorithms to prepare before execution time; run as pre-processor, can be serial, can be slow and expensive, starts.

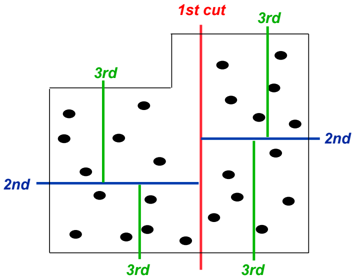
- Dynamic - information not known until runtime, work changes during computation (e.g. adaptive methods), or locality of objects change (e.g. particles move)

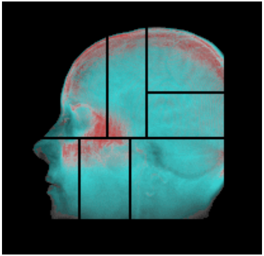
  use on-line algorithms to make decisions mid-execution; must run side-by-side with application, should be parallel, fast, scalable. Incremental algorithm preferred (small changes in input result in small changes in partitions)

will look at some geometric methods, graph-based methods, spectral methods, multilevel methods, diffusion-based balancing,...

Recursive Coordinate Bisection

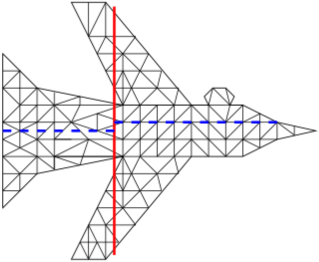
Divide work into two equal parts using cutting plane orthogonal to coordinate axis For good aspect ratios cut in longest dimension.

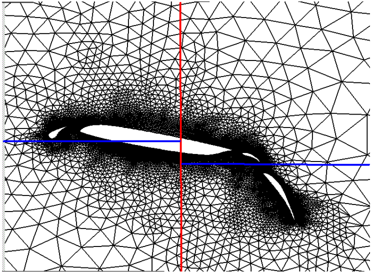




Can generalize to k-way partitions. Finding *optimal* partitions is NP hard. (There are optimality results for a class of graphs as a graph partitioning problem.)

Recursive Coordinate Bisection





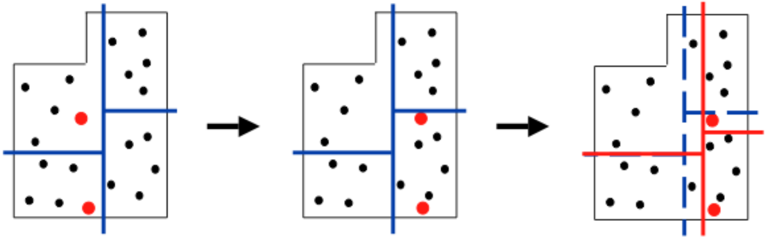
+ Conceptually simple, easy to implement, fast.
+ Regular subdomains, easy to describe

– Need coordinates of mesh points/particles.

– No control of communication costs.

– Can generate disconnected subdomains

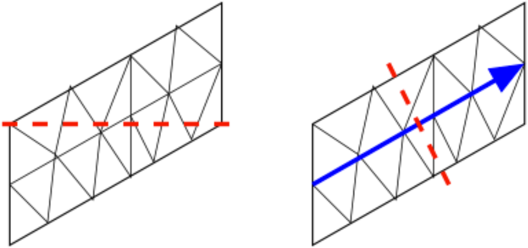
Recursive Coordinate Bisection



Implicitly incremental - small changes in data result in small movement of cuts

Recursive Inertial Bisection

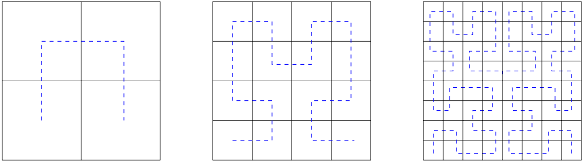
For domains not oriented along coordinate axes can do better if account for the angle of orientation of the mesh.



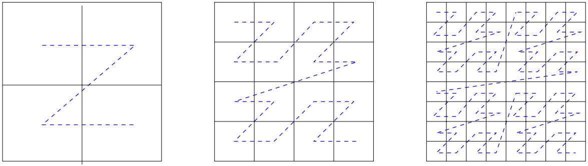
Use bisection line orthogonal to principal inertial axis (treat mesh elements as point masses). Project centers-of-mass onto this axis; bisect this ordered list. Typically gives smaller subdomain boundary.

Space-filling Curves

Linearly order a multidimensional mesh (nested hierarchically, preserves locality)



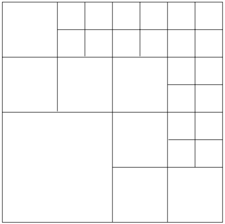
Peano-Hilbert ordering

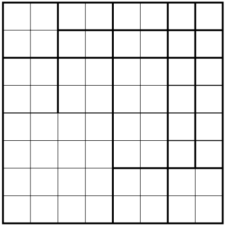


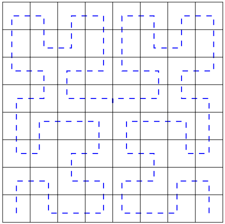
Morton ordering

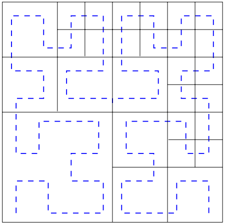
Space-filling Curves

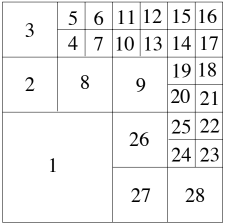
Easily extends to adaptively refined meshes



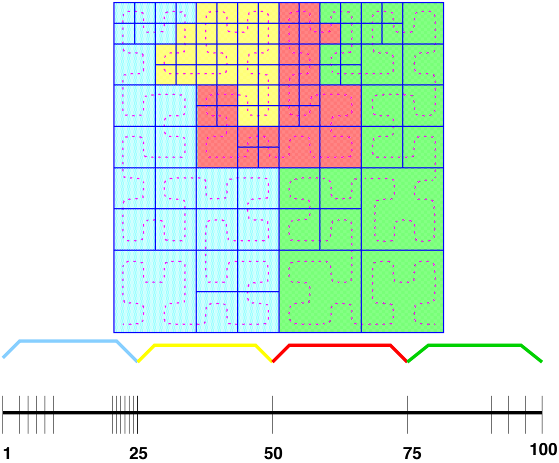






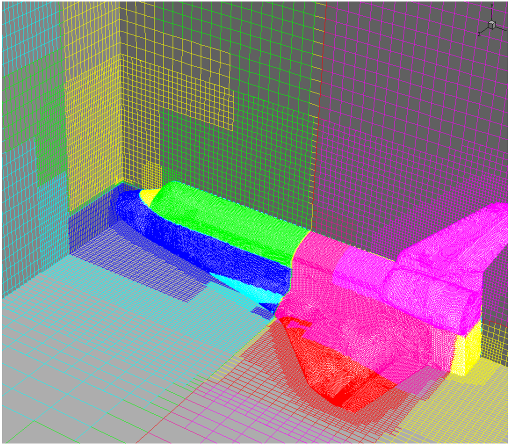


Space-filling Curves



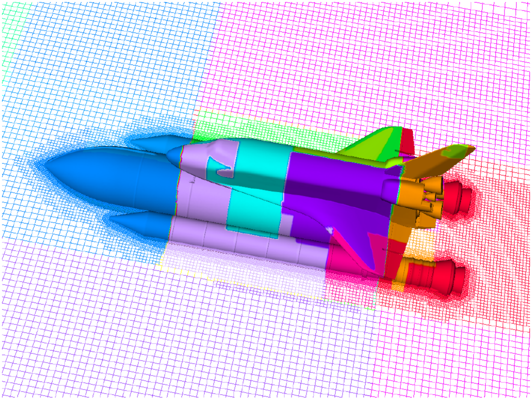
Partition work into equal chunks.

Space-filling Curves



- Generalizes to uneven work loads - incorporate weights.
- Dynamic on-the-fly partitioning for any number of nodes.
- Good for cache performance

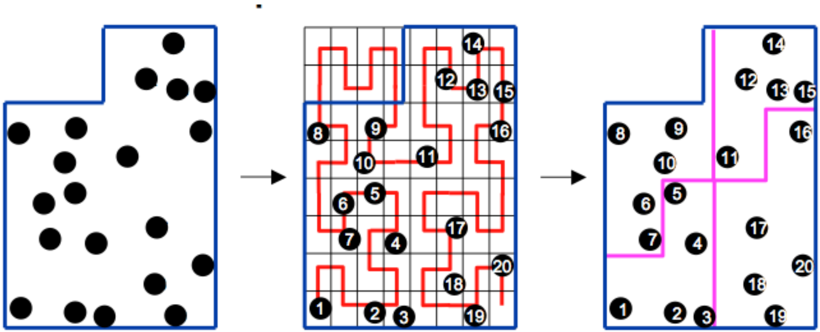
Space-filling Curves



- Red region has more communication - not compact
- Need coordinates

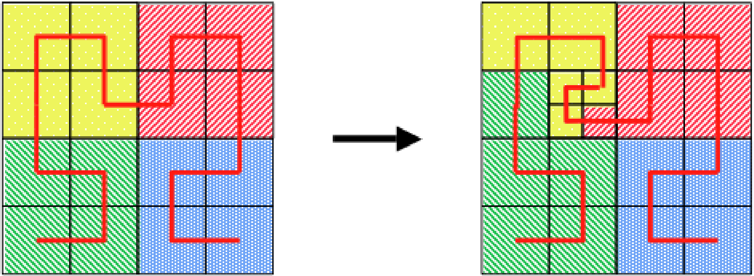
Space-filling Curves

Generalizes to other non-finite difference problems, e.g. particle methods, patch-based adaptive mesh refinement, smooth particle hydro.,



Space-filling Curves

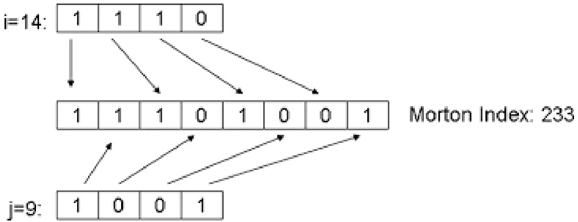
Implicitly incremental - small changes in data results in small movement of cuts in linear ordering



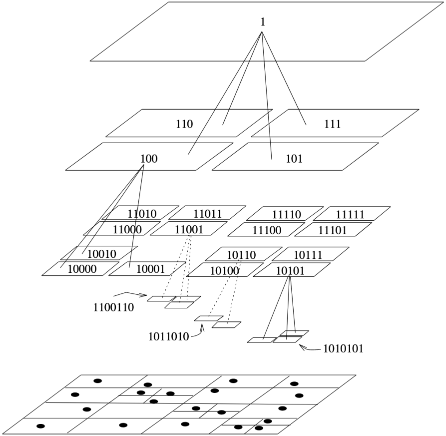
Morton Ordering

Computing Morton Indedx:

- convert coordinate to integers
- interleave the bits to generate a new integer
- works for arbitrary dimension (1D, 2D, 3D, 4D, ...)



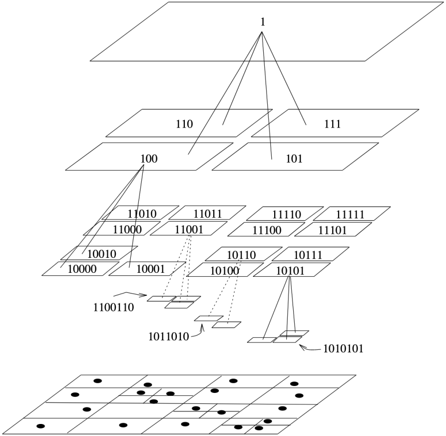
Application to N-body codes



References

- H. Sundar, R.S. Sampath, G. Biros - *Bottom Up Construction and 2:1 Balance Refinement of Linear Octrees in Parallel*
- M.S.Warren and J.K.Salmon - *A Parallel HashedOct-Tree N-Body Algorithm*

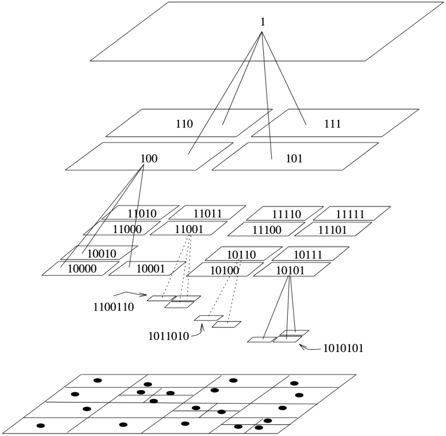
Application to N-body codes



Parallel Tree Construction

- compute Morton Ids of particles;
- parallel sort and partition across processes
- construct local tree on each process
- adjust for overlapping tree nodes at process boundaries.

Application to N-body codes



Communication

- all processes store the starting Morton IDs of each partition
- to fetch a data element with given coordinates, we can determine the process ID to communicate with using a binary search in the list of starting Morton IDs ($O(\log p)$ cost).

Visualization

- Useful for interpreting results.
- Can also be helpful for debugging!

Software

- Paraview
- Visit
- TecPlot

Visualization ToolKit (VTK) File Format

Reference: The VTK User's Guide

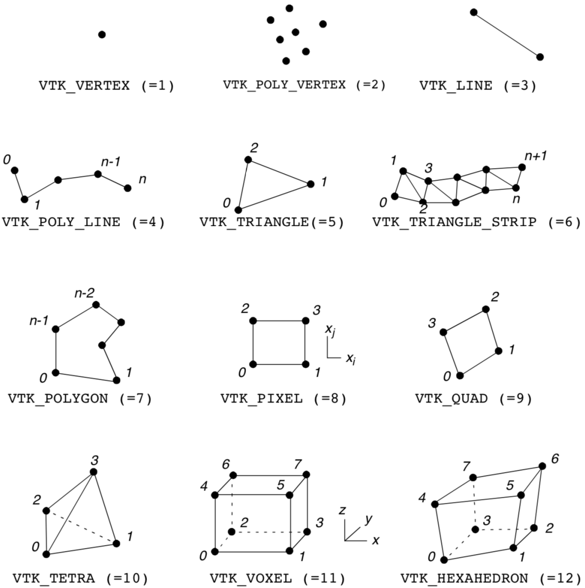
Simple Text Format

```
# vtk DataFile Version 2.0
Volume example
ASCII
DATASET STRUCTURED_POINTS
DIMENSIONS 3 4 6
ASPECT_RATIO 1 1 1
ORIGIN 0 0 0
POINT_DATA 72
SCALARS volume_scalars char 1
LOOKUP_TABLE default
0 0 0 0 0 0 0 0 0 0 0 0
0 5 10 15 20 25 25 20 15 10 5 0
0 10 20 30 40 50 50 40 30 20 10 0
0 10 20 30 40 50 50 40 30 20 10 0
0 5 10 15 20 25 25 20 15 10 5 0
0 0 0 0 0 0 0 0 0 0 0 0
```

XML Format

```
<VTKFile type="UnstructuredGrid" ...>
  <UnstructuredGrid>
    <Piece NumberOfPoints="#" NumberOfCells="#">
      <PointData>...</PointData>
      <CellData>...</CellData>
      <Points>...</Points>
      <Cells>...</Cells>
    </Piece>
  </UnstructuredGrid>
</VTKFile>
```

VTK Cell Types



Examples: https://github.com/NYU-HPC19/lecture13

Outline

How solve large linear systems?

Many linear solvers are available: factorization-based solvers (LU, Choleski), fast direct solvers (for specific problems), Krylov solvers (CG, MINRES, GMRES,...), optimal complexity ($\mathcal{O}(n)$) solvers for certain problems (multigrid, FMM)

Solver choice depends on:

- is the system sparse or dense? how sparse?
- symmetric? positive definite? explicitly available?
- properties of the matrix? what do I know about the eigenvalues?
- do I have a good preconditioner?
- do I need the exact solution or can I allow for ε-errors?
- what computing resources do I have? can I store the matrix?
- how fast/often do I need to solve systems?

Reading/Sources

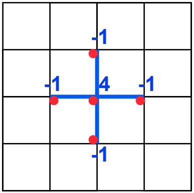
Why Multigrid Methods are so efficient:
http://www.cs.technion.ac.il/people/irad/online-publications/Yav06.pdf

2D Poisson's equation

- Similar to the 1D case, but the matrix T is now

$$
T = \left(\begin{array}{ccc|ccc|ccc}
4 & -1 & & -1 & & & & & \\
-1 & 4 & -1 & & -1 & & & & \\
 & -1 & 4 & & & -1 & & & \\
\hline
-1 & & & 4 & -1 & & -1 & & \\
 & -1 & & -1 & 4 & -1 & & -1 & \\
 & & -1 & & -1 & 4 & & & -1 \\
\hline
 & & & -1 & & & 4 & -1 & \\
 & & & & -1 & & -1 & 4 & -1 \\
 & & & & & -1 & & -1 & 4
\end{array}\right)
$$

Graph and "stencil"



- 3D is analogous

Algorithms for 2D/3D Poisson Equation with n unknowns

Algorithm	2D ($n= N^2$)	3D ($n=N^3$)
Dense LU	n^3	n^3
Band LU	n^2	$n^{7/3}$
Explicit Inv.	n^2	n^2
Jacobi/GS	n^2	n^2
Sparse LU	$n^{3/2}$	n^2
Conj.Grad.	$n^{3/2}$	$n^{3/2}$
RB SOR	$n^{3/2}$	$n^{3/2}$
FFT	$n*\log n$	$n*\log n$
Multigrid	n	n
Lower bound	n	n

Multigrid is much more general than FFT approach (many elliptic PDE)

Different approaches

Jacobi: M = Diagonal(A)

Gauss Seidel: M = LowerTriangular(A)

SOR & SSOR: Combination

Jacobi	$x = D^{-1}(b + Ox)$
Gauss Siedel	$x = L^{-1}(B + Ux)$
SOR	$x = (D+wL)^{-1}(wb-[(w-1)D-wU)x)$

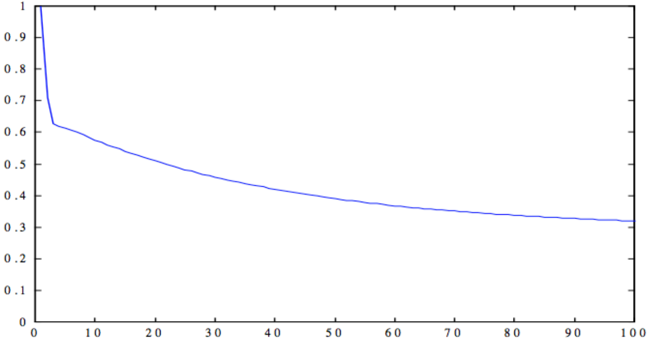
Iterations

Jacobi: $O(1/h)$

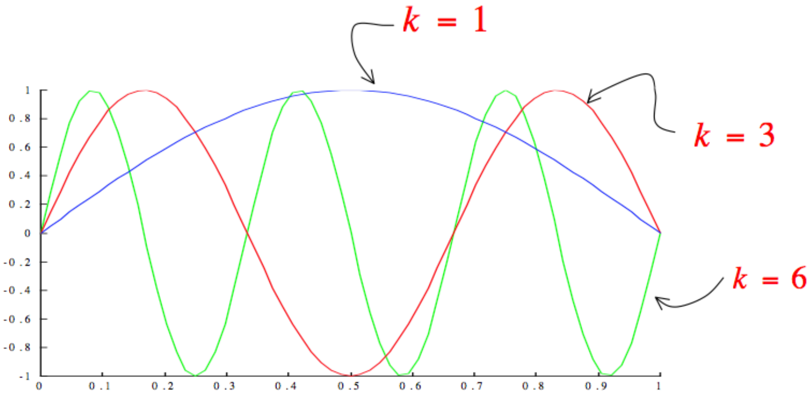
SOR: $O(1/h^{-1/2})$

1D problem

- Error $\|e\|_\infty$ plotted against iteration number:

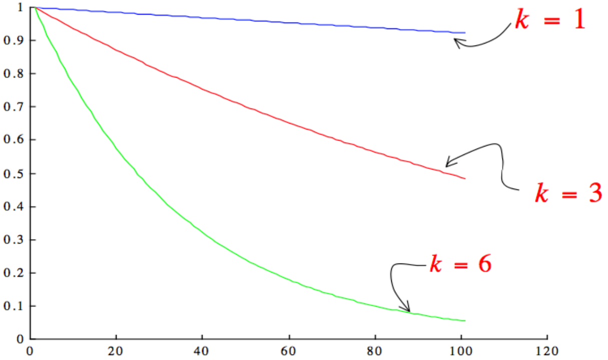


Eigenvectors



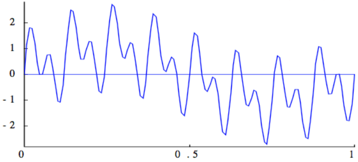
Error /eigenvectors

- Error, $||e||_\infty$, in weighted Jacobi on Au = 0 for 100 iterations using initial guesses of v_1, v_3, and v_6

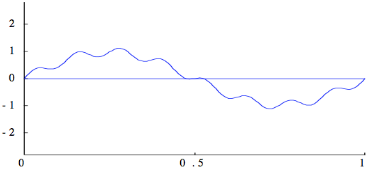


Error for high frequencies

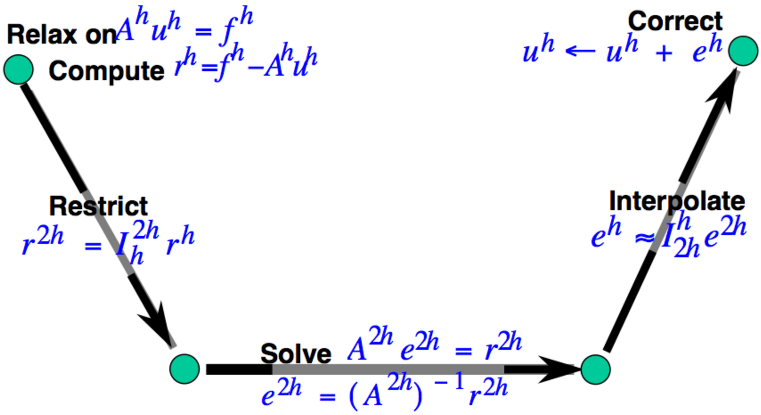
- Initial error: $v_{kj} = \sin\left(\frac{2j\pi}{N}\right) + \frac{1}{2}\sin\left(\frac{1}{}\right.$



- Error after 35 iteration sweeps:

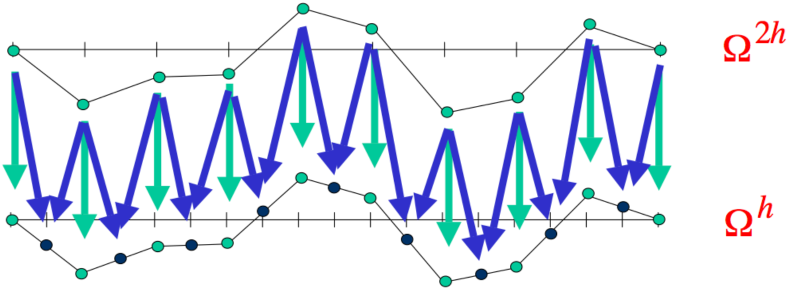


Two-level scheme

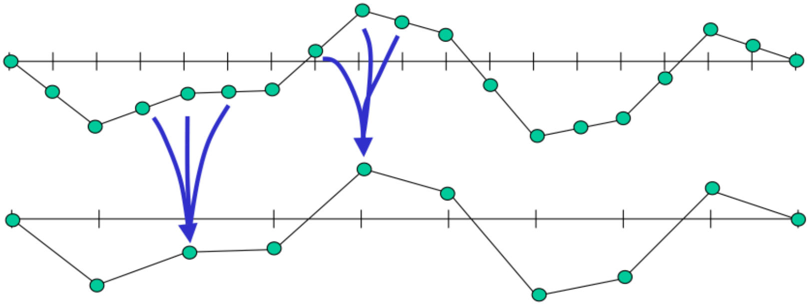


Prolongation

- Values at points on the coarse grid map unchanged to the fine grid
- Values at fine-grid points NOT on the coarse grid are the averages of their coarse-grid neighbors



Restriction



V-cycle

$$\underline{v^h \leftarrow MV^h(v^h, f^h)}$$

1) Relax α_1 times on $A^h u^h = f^h$, initial v^h arbitrary

2) If Ω^h is the coarsest grid, go to 4)

Else:

$$f^{2h} \leftarrow I_{2h}^h (f^h - A^h v^h)$$

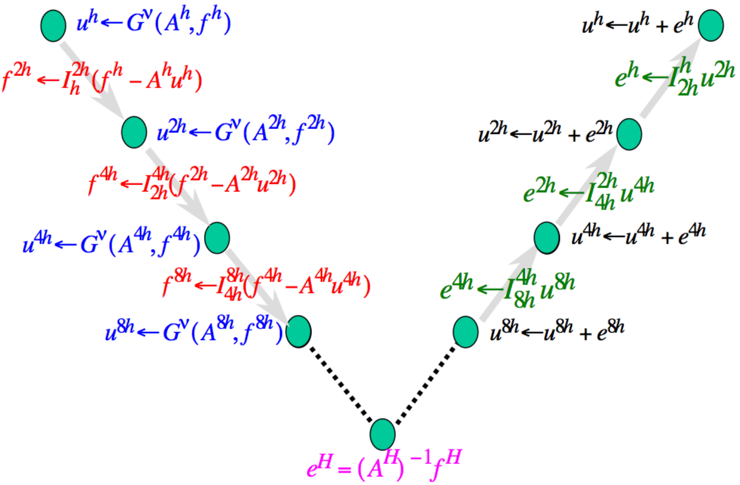
$$v^{2h} \leftarrow 0$$

$$v^{2h} \leftarrow MV^{2h}(v^{2h}, f^{2h})$$

3) Correct $v^h \leftarrow v^h + I_{2h}^h v^{2h}$

4) Relax α_2 times on $A^h u^h = f^h$, initial guess v^h

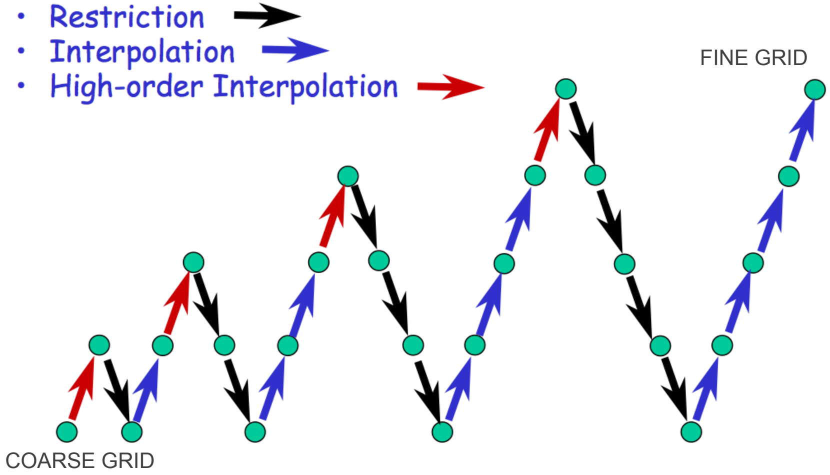
Multigrid



$u^h \leftarrow G^\nu(A^h, f^h)$
$f^{2h} \leftarrow I_h^{2h}(f^h - A^h u^h)$
$u^{2h} \leftarrow G^\nu(A^{2h}, f^{2h})$
$f^{4h} \leftarrow I_{2h}^{4h}(f^{2h} - A^{2h} u^{2h})$
$u^{4h} \leftarrow G^\nu(A^{4h}, f^{4h})$
$f^{8h} \leftarrow I_{4h}^{8h}(f^{4h} - A^{4h} u^{4h})$
$u^{8h} \leftarrow G^\nu(A^{8h}, f^{8h})$
$e^H = (A^H)^{-1} f^H$
$u^{8h} \leftarrow u^{8h} + e^{8h}$
$e^{4h} \leftarrow I_{8h}^{4h} u^{8h}$
$u^{4h} \leftarrow u^{4h} + e^{4h}$
$e^{2h} \leftarrow I_{4h}^{2h} u^{4h}$
$u^{2h} \leftarrow u^{2h} + e^{2h}$
$e^h \leftarrow I_{2h}^h u^{2h}$
$u^h \leftarrow u^h + e^h$

Full multigrid (O(N))

- Restriction
- Interpolation
- High-order Interpolation



Parallelizing multigrid

Regular grids

- grid coarsening is trivial
- grid partitioning is trivial
- coloring can be constructed analytically
- smoother

Unstructured grids/graphs

- graph coarsening

Multigrid for generic matrices

Algebraic multigrid

- Define coarse grid in terms of strengths of connections
- Use MIS to define nodes
- No need to create new edges
- Interpolation and restriction operations based on the smooth error idea
- Only requires matrix entries
- Works for graph Laplacians