Parallel Multigrid Tutorial

19th Copper Mountain Conference on Multigrid Methods

Robert D. Falgout
Center for Applied Scientific Computing

March 24, 2019

LLNL-PRES-770238
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

References and Acknowledgements

- "Introduction to Parallel Computing", Blaise Barney
  – https://computing.llnl.gov/tutorials/parallel_comp/

- "A Parallel Multigrid Tutorial", Jim Jones
  – Copper Mountain tutorial in 1999, 2005, & 2007

- "A Parallel Computing Tutorial", Ulrike Meier Yang
  – IMA Tutorial at Workshop on Fast Solution Techniques, Nov 2010

Outline

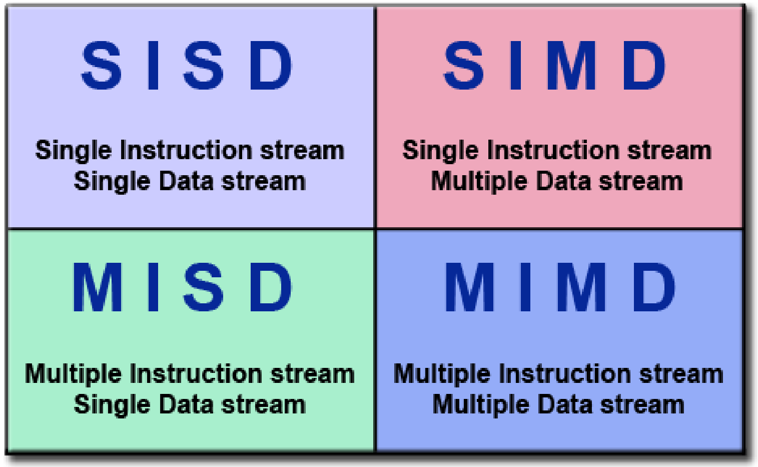
- Parallel Computing
  - Classical computer taxonomy
  - Programming models
  - Parallel performance metrics
  - Parallelizing PDE-based problems

- Parallel Multigrid
  - Parallel Algebraic Multigrid
  - Parallel Multigrid Software Design
  - Some Current Research Topics

Parallel Computing

Classical Taxonomy of Computers due to Michael Flynn in 1972

- Classifies by instruction stream and data stream



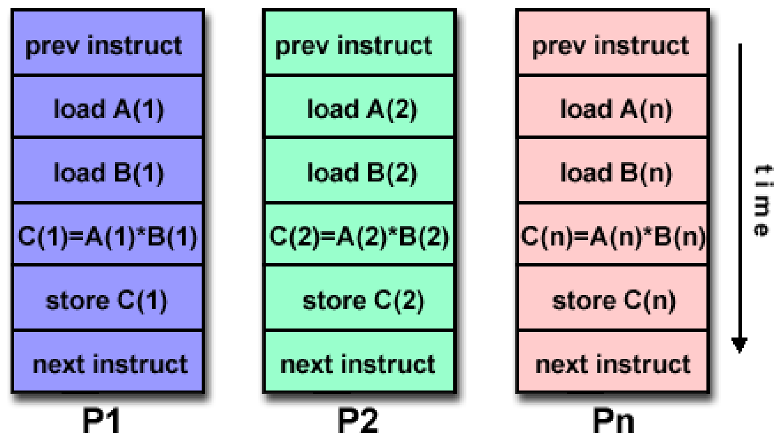
SISD Single Instruction stream Single Data stream	**SIMD** Single Instruction stream Multiple Data stream
MISD Multiple Instruction stream Single Data stream	**MIMD** Multiple Instruction stream Multiple Data stream

- Two main types of parallel systems today
  - SIMD: GPUs, Intel Knights Landing (KNL)
  - MIMD: Most supercomputers, clusters, multi-core PCs

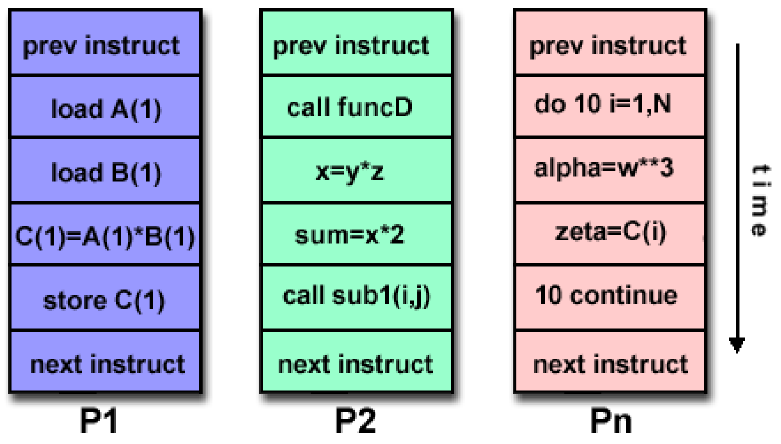
- Recent supercomputers are MIMD with SIMD subcomponents

SIMD vs MIMD in Pictures

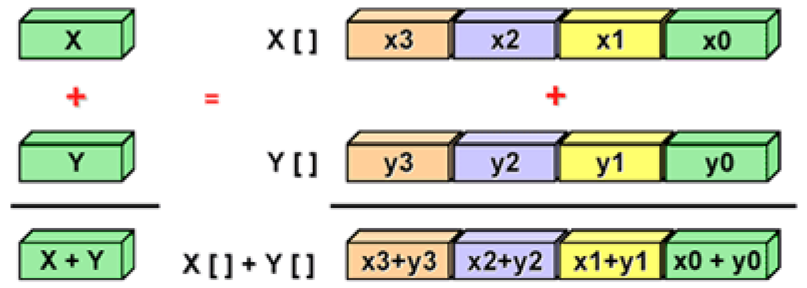
SIMD: instruction stream is the same



MIMD: instruction stream is different

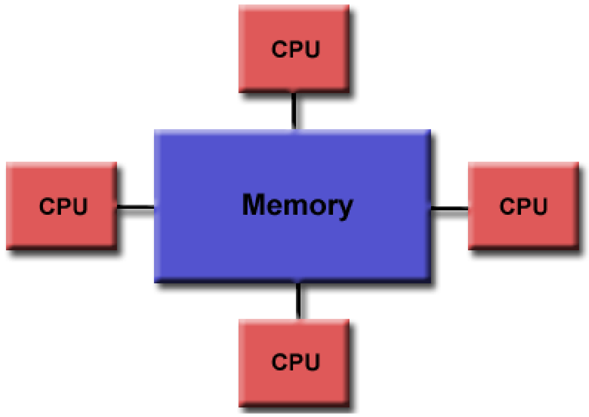


Example SIMD computation: vector addition

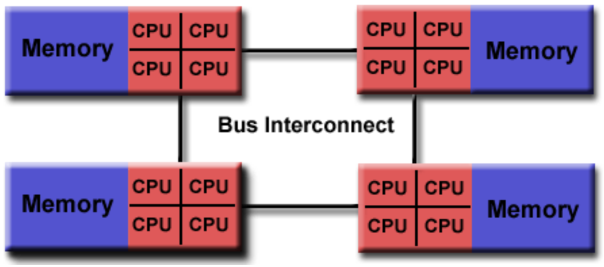


Parallel Computer Memory Architectures – Shared Memory

- All processors can access the same memory
- Uniform memory access (UMA):
  - Same access time to memory
- Non-uniform memory access (NUMA)
  - Different access times to different memories
- Advantages:
  - User-friendly programming perspective to memory (global address space)
  - Data sharing is fast
- Disadvantages:
  - Lack of scalability between memory and CPUs
  - Programmer responsible to ensure "correct" access of global memory



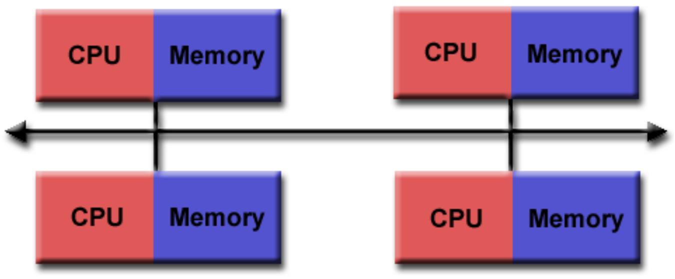
Shared Memory (UMA)



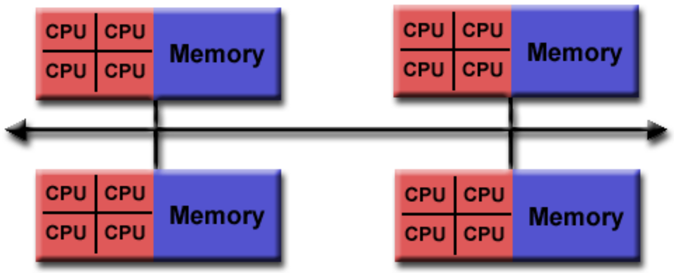
Shared Memory (NUMA)

Parallel Computer Memory Architectures – Distributed Memory

- Require a communication network to connect inter-processor memory
- Advantages:
  - — Memory is scalable with number of processors
  - — No memory interference or overhead for trying to keep cache coherency
- Disadvantages:
  - — Programmer responsible for data communication between processors
  - — Difficult to map data structures to memory
- Hybrid Distributed-Shared Memory
  - — Generally used for today's largest and fastest computers



Distributed Memory



Hybrid Architecture

Parallel Programming Models

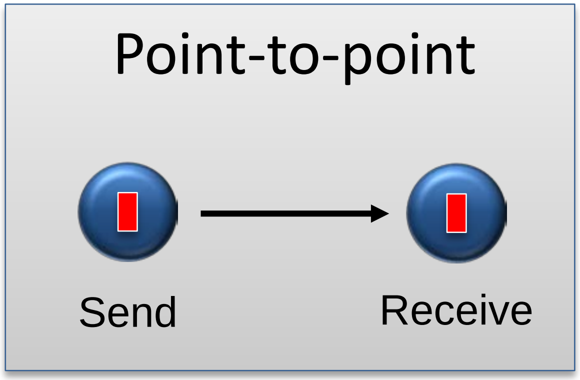
- Shared memory (without threads)
  - API in many operating systems, SHMEM
- Shared memory with threads
  - POSIX Threads (Pthreads), OpenMP (1997)
- Message Passing
  - MPI (Message Passing Interface, 1994)
- Data Parallel
  - Also referred to as Partitioned Global Address Space (PGAS) model
  - Coarray Fortran, Unified Parallel C (UPC), Global Arrays, X10, Chapel
- Hybrid
  - MPI+OpenMP, MPI+CUDA, etc.

- All of these can be implemented on any architecture
  - Most common approach – Single Program Multiple Data (SPMD)

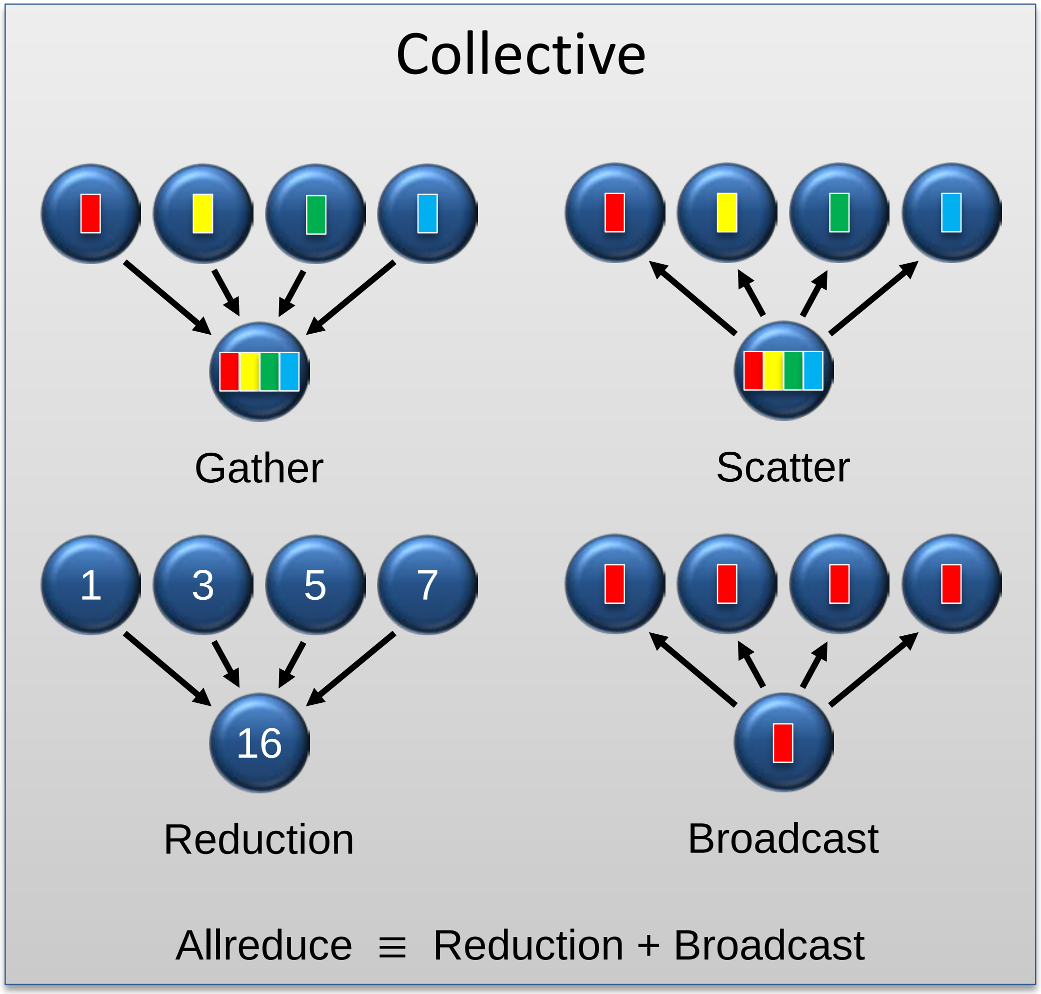
This tutorial will discuss parallel multigrid from a distributed-memory message-passing viewpoint

- Will mostly use conceptual diagrams and descriptions
- Will not focus on implementation details and specific code
- MPI and OpenMP will be the primary languages mentioned

Communication – Two basic types

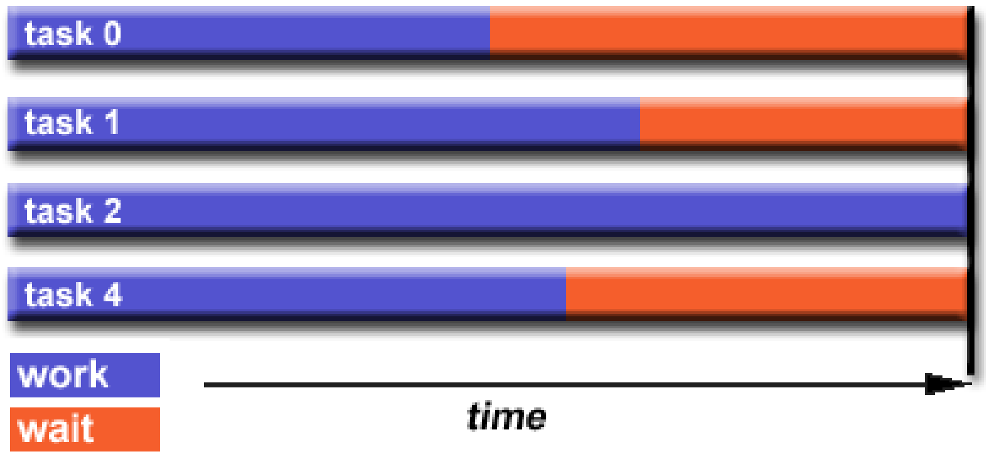


- MG codes mainly use
  - Send, Receive
  - Allreduce (MPI routine)
- Gather and Scatter are not scalable



Load balancing

- Keep all tasks busy all of the time
  - Minimize idle time
- The slowest task will determine the overall performance



Parallel Computing Performance Metrics

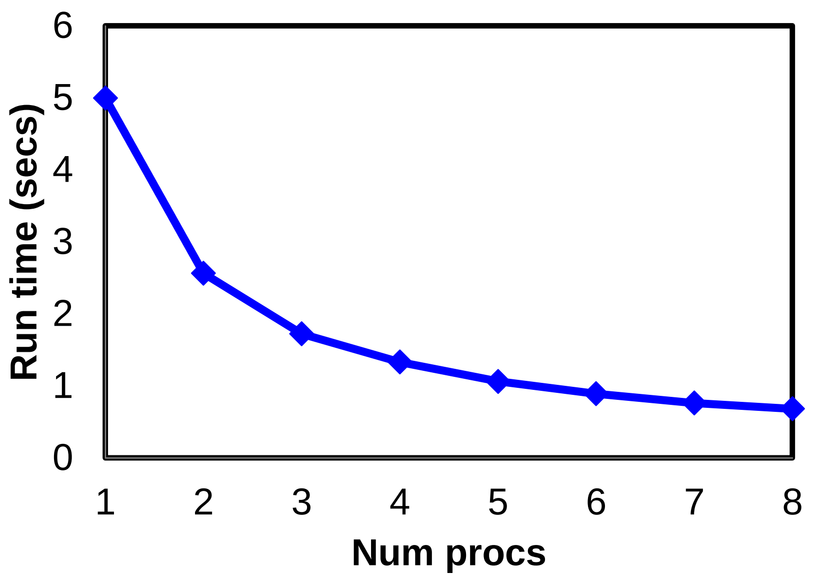
- Let $T(N,P)$ be the time to solve a problem of size N using P processors

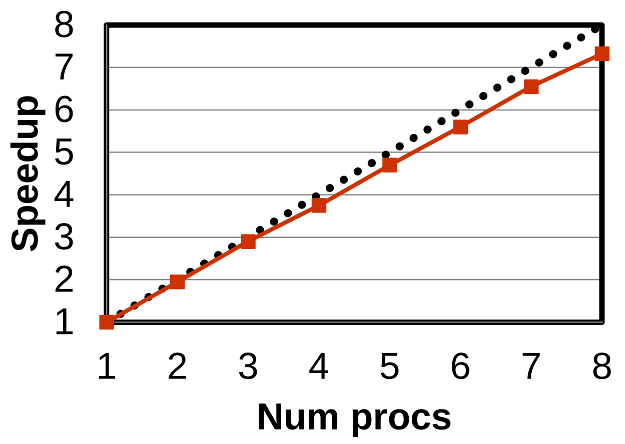
- Strong scaling – problem size is fixed
  - Speedup: $S(N,P) = T(N,1) \,/\, T(N,P)$
  - Efficiency: $E(N,P) = S(N,P) \,/\, P$

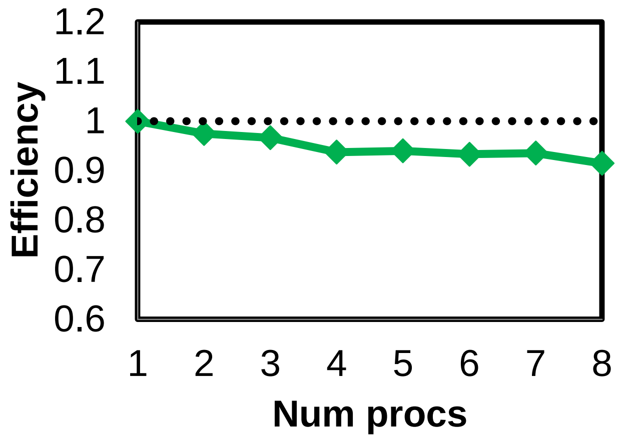
- Weak scaling – problem size is proportional to P
  - Scaled Efficiency: $SE(N,P) = T(N,1) \,/\, T(PN,P)$

- An algorithm is scalable if $T(PN,P) \leq C$ for all P
  - Ideally C is a constant, but this is not always possible
  - For multigrid, C grows as $\log P$ (this is optimal for some problems)

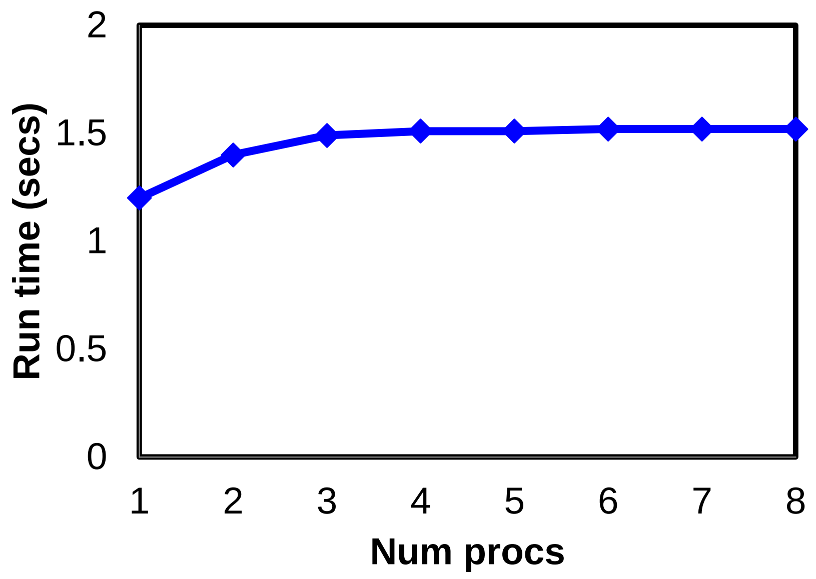
Strong Scaling – fix the problem size and increase the number of processors

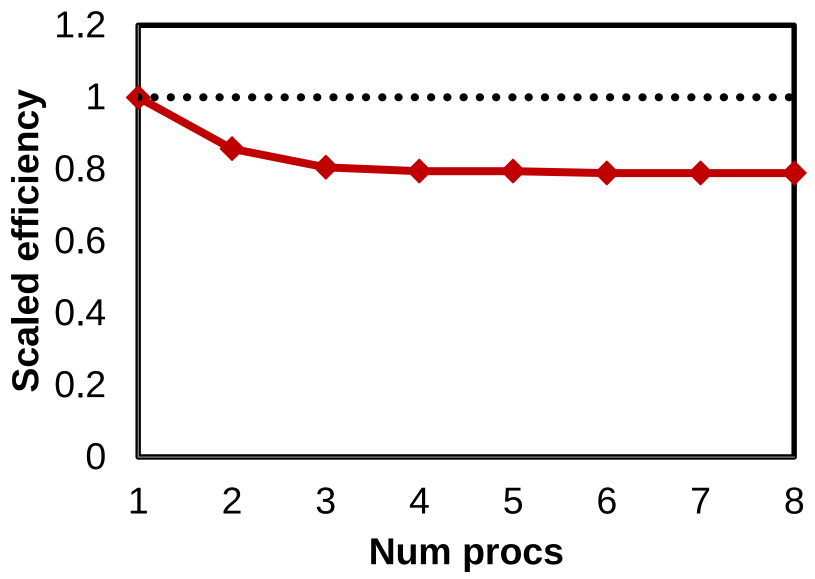






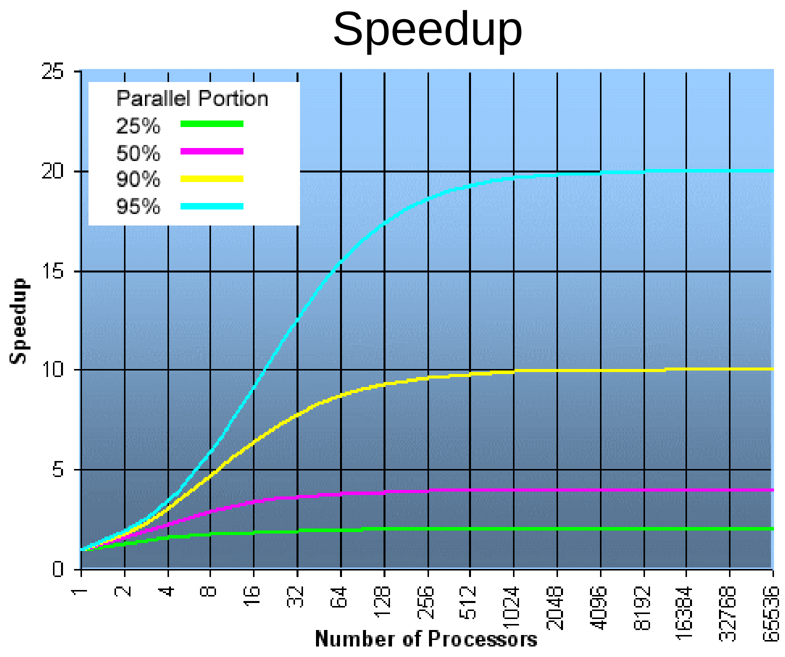
Weak Scaling – increase the problem size and the number of processors proportionately

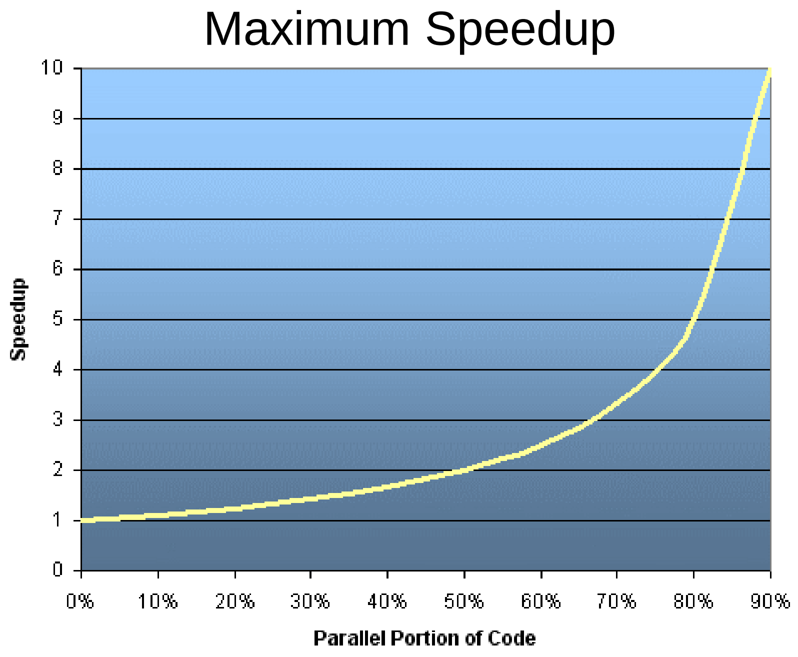




Amdahl's Law models speedup as a function of the serial (non-parallelizable) component of a code

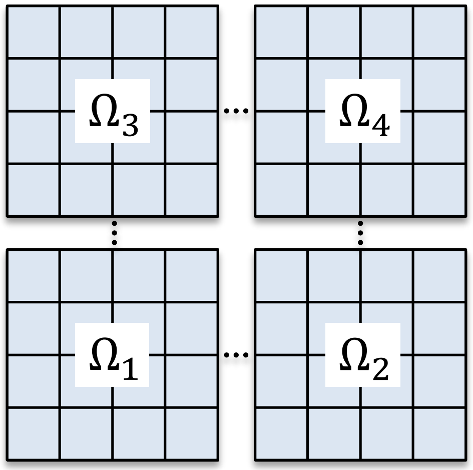
$$\text{Speedup} = \frac{1}{\underbrace{F}_{\text{Parallelizable fraction}}/\underbrace{P}_{\text{Num processors}} + \underbrace{(1-F)}_{\text{Serial fraction}}} < \boxed{\frac{1}{(1-F)}} \leftarrow \text{Maximum Speedup}$$





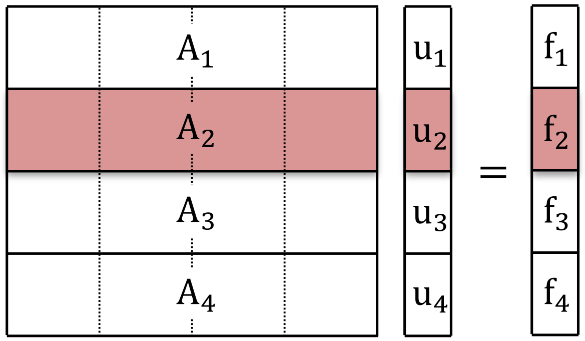
Domain partitioning is the primary approach for parallelizing PDE-based problems

- Example grid Ω partitioned into 4 subdomains Ω_p



- Processor p owns data associated with Ω_p

- Example sparse linear system resulting from a discretized PDE (e.g., 5-pt cell-centered discretization of $\Delta u = f$)

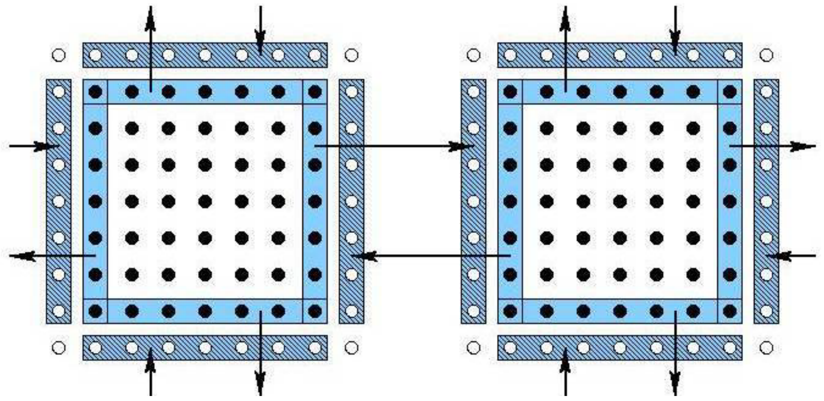


- Each 5-pt stencil corresponds to a row of A

- Processor p owns A_p, u_p, f_p

Basic linear algebra operations are at the core of parallel multigrid algorithms

- Matrix-vector multiply: Ax
- On each processor p:
  - Exchange subdomain boundary data with “nearest neighbors” (MPI Send/Recv)
  - Compute local product $A_p x_p$



- Vector addition: $z = x + y$

z_0	=	x_0	+	y_0	← Proc 0
z_1	=	x_1	+	y_1	← Proc 1
z_2	=	x_2	+	y_2	← Proc 2

- Vector product: $s = x^T y$
  - Compute local product then $s_0 + s_1 + s_2$ (MPI Allreduce)

s_0	=	x_0	*	y_0	← Proc 0
s_1	=	x_1	*	y_1	← Proc 1
s_2	=	x_2	*	y_2	← Proc 2

Parallel programming models provide useful incite into expected performance

- Most common communication/computation model
  - Simple, but effective for providing qualitative understanding
  - Better models account for message contention, network topology, etc.

$$T_{comm} = \alpha + m\beta \quad \text{(communicate m doubles)}$$

$$T_{comp} = m\gamma \quad \text{(compute m flops)}$$

- Values for α, β, γ vary across machines, but communication generally dominates, especially network latency

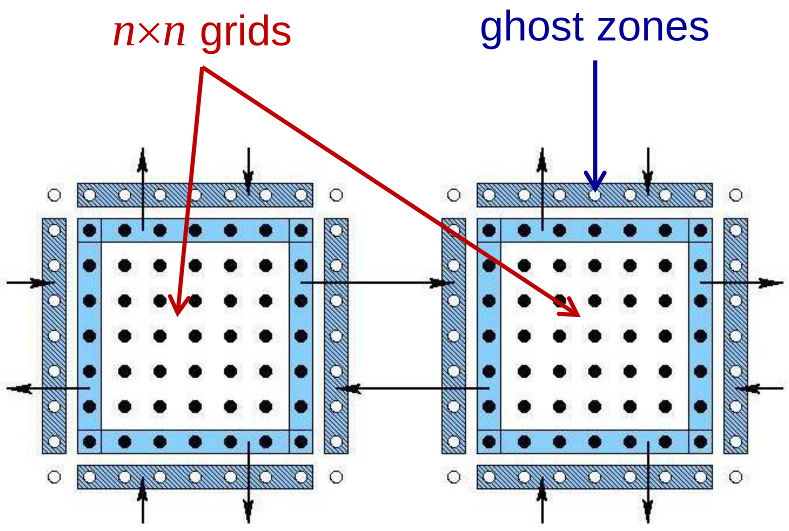
Typical relationship: $\alpha = 10^4\gamma; \quad \beta = 10^1\gamma$

- For sparse linear algebra (especially multigrid), developing approaches to minimize communication is key

Parallel model for matrix-vector multiply – 5-pt discretization of 2D Laplace equation

- On each processor p:
  - Exchange subdomain boundary data with "nearest neighbors"
  - Compute local product $A_p x_p$
- Total time determined by slowest processor
- Time to do a matvec
  - 4 communications of size n data (assuming bi-directional)
  - $5n^2$ computations (multiply-adds)

$$T \approx 4\alpha + 4n\beta + 5n^2\gamma$$

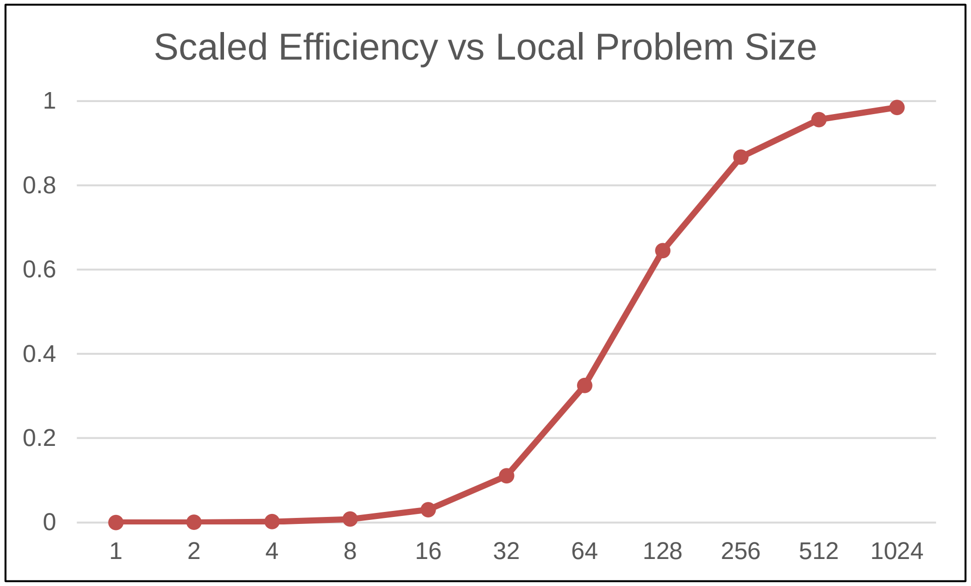


5-pt stencil

$$\begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix}$$

Increase local problem size for efficiency

- Large local problem size → computations dominate

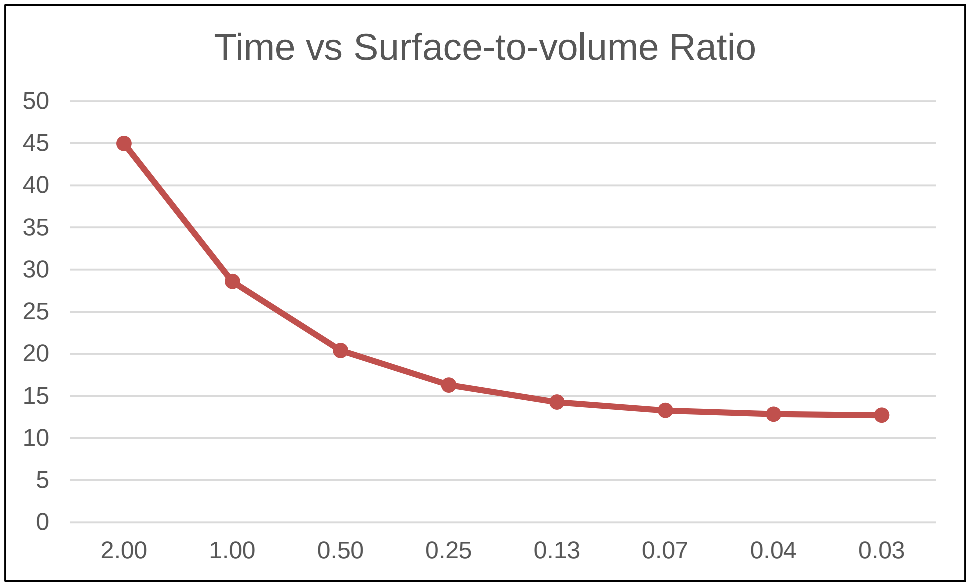


- It's not always possible to increase problem size
  — Competition for memory with the rest of the application

- Easy to make an algorithm look good by choosing large problem
  — Better to show both large and small cases

Minimize surface-to-volume ratio for speed

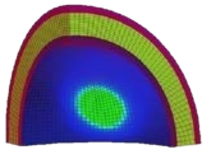
- Consider a local volume of size 16,384 (=128^2), but for rectangles of varying dimensions
  - — Large surface-to-volume ratio = long thin rectangle (1 x 16,384)
  - — Smallest ratio = a cube (128 x 128)



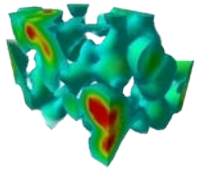
Parallel Multigrid

Multigrid will play an important role for addressing exascale challenges

- For many applications, the fastest and most scalable solvers are multigrid methods

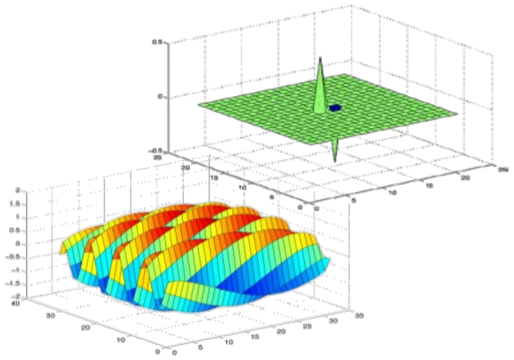

Elasticity / Plasticity

- Exascale solver algorithms will need to:
  - Exhibit extreme levels of parallelism (exascale → 1B cores)
  - Minimize data movement & exploit machine heterogeneity
  - Demonstrate resilience to faults

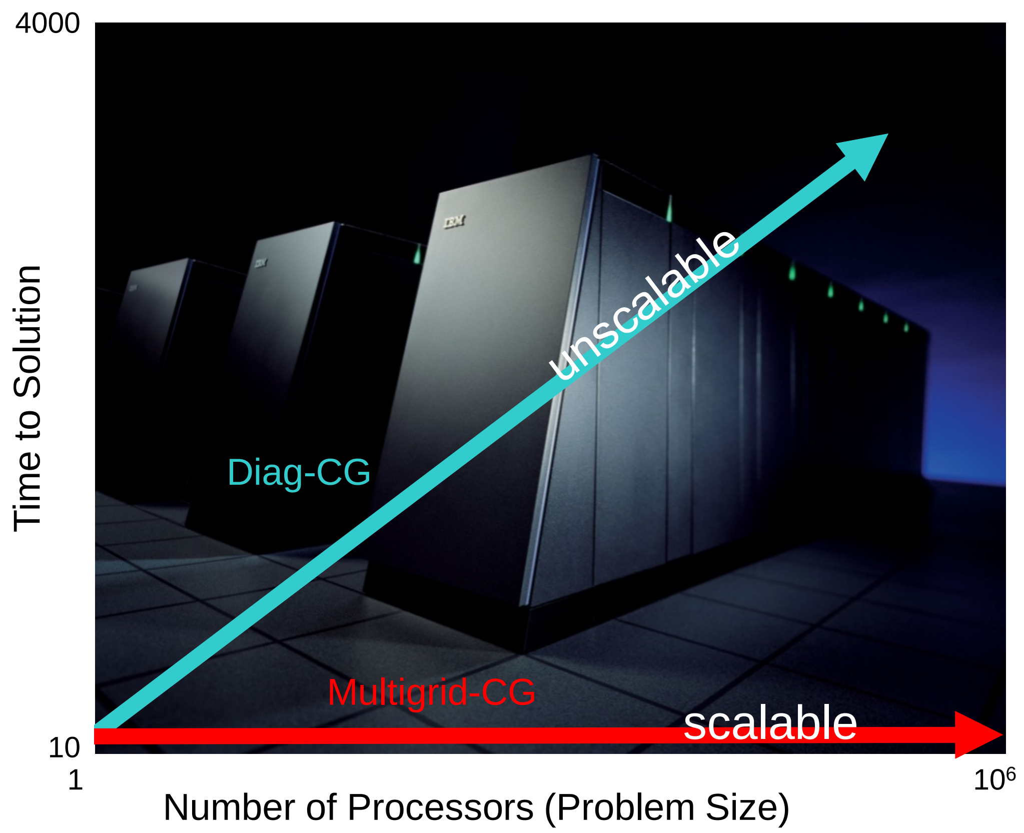

Quantum Chromodynamics

- Multilevel methods are ideal
  - Key feature: Optimal O(N)

- Research challenge:
  - No optimal solvers yet for some applications, even in serial!
  - Parallel computing increases difficulty

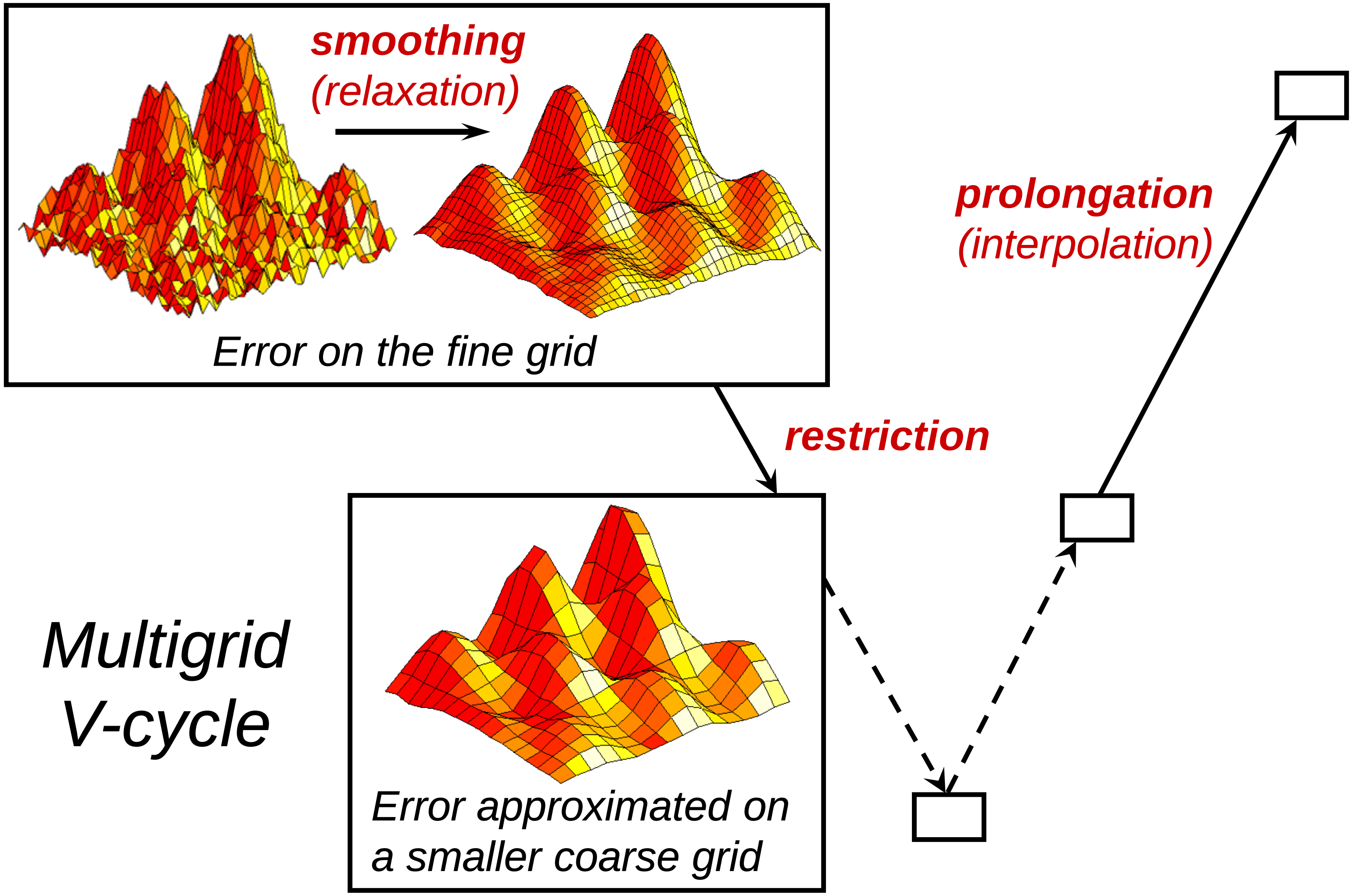

Helmholtz Modes

Multigrid solvers have $O(N)$ complexity, and hence have good scaling potential



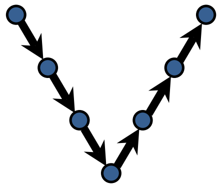
- Weak scaling – want constant solution time as problem size grows in proportion to the number of processors

Multigrid (MG) uses a sequence of coarse grids to accelerate the fine grid solution

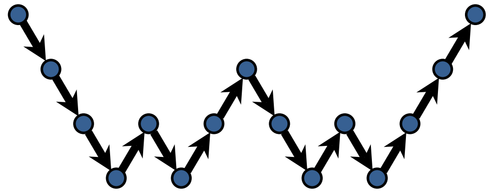


Comments on multigrid cycles

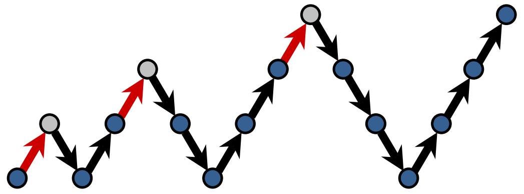
- V-cycle:
  - Most commonly used cycle
  - $O(N)$ work satisfies $||e|| \leq \varepsilon$ for fixed tolerance ε



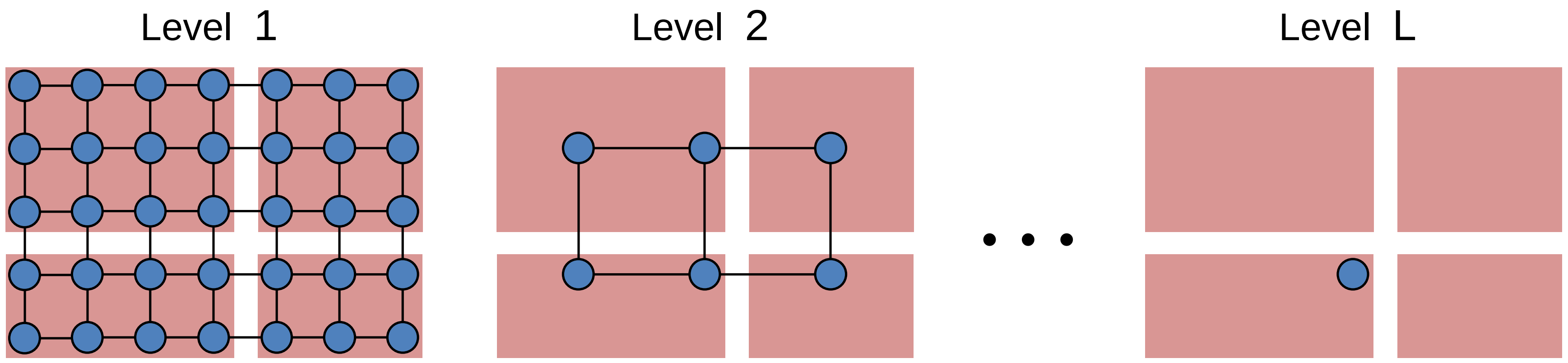
- W-cycle:
  - More robust than V-cycles
  - $O(N)$ work satisfies $||e|| \leq \varepsilon$
  - Not scalable in parallel (discussed later)



- FMG V-cycle:
  - $O(N)$ work satisfies $||e|| \leq kh^p$ where h^p is discretization accuracy



Straightforward MG parallelization yields optimal-order performance for V-cycles



- ~ 1.5 million idle cores on Sequoia!
- Multigrid has a high degree of concurrency
  - Size of the sequential component is only O(log N)!
  - This is often the minimum size achievable
- Parallel performance model has the expected log term

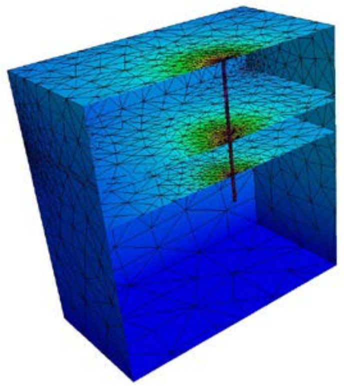
$$T_V = O(\log N)(\text{comm latency}) + O(\Gamma_p)(\text{comm rate}) + O(\Omega_p)(\text{flop rate})$$

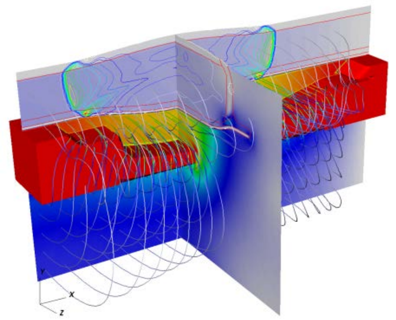
Parallel computing imposes restrictions on multigrid algorithm development

- Avoid sequential techniques
  - Classical AMG coarsening
  - Gauss-Seidel smoother
  - Cycles with large sequential component
    - F-cycle: O($\log^2$ N)
    - W-cycle: O($2^{\log N}$) = O(N)

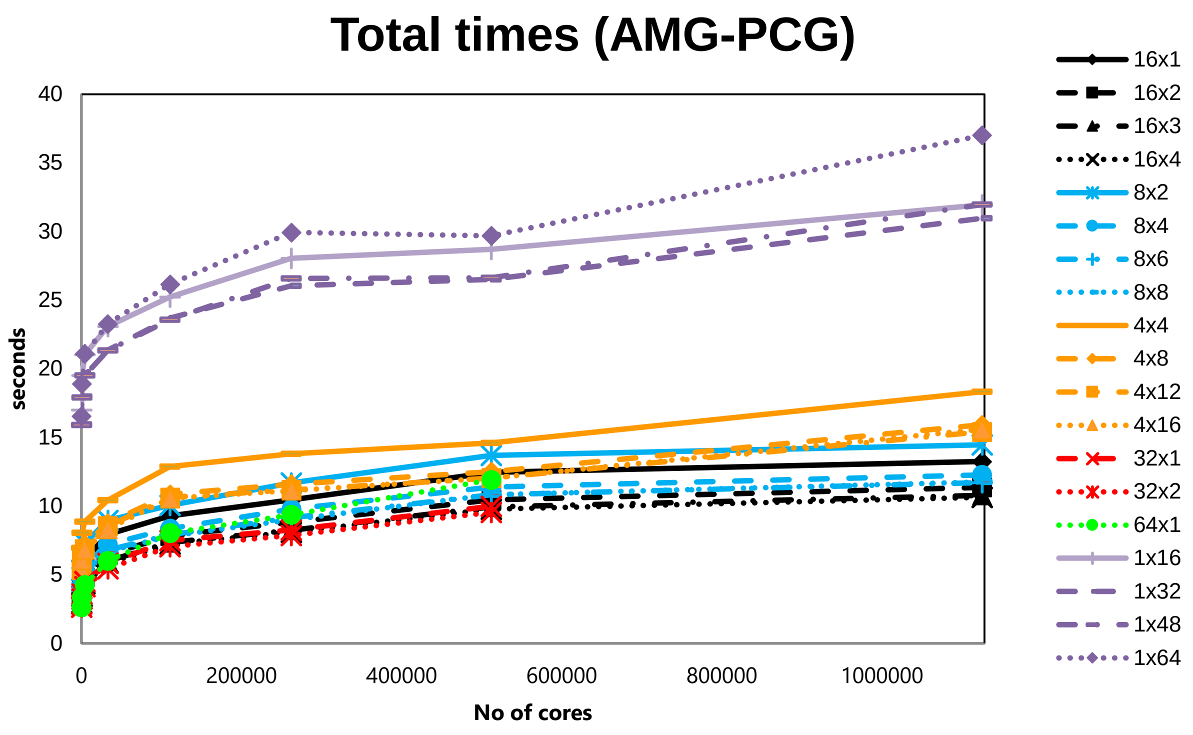
- Control communication
  - Galerkin coarse-grid operators (P^TAP) can lead to high communication costs in AMG

- Need both CS and Math advances!
  - New methods have new convergence and robustness characteristics
  - Successful addressing issues so far


10x speedup for subsurface problems with new coarsening and interpolation approach

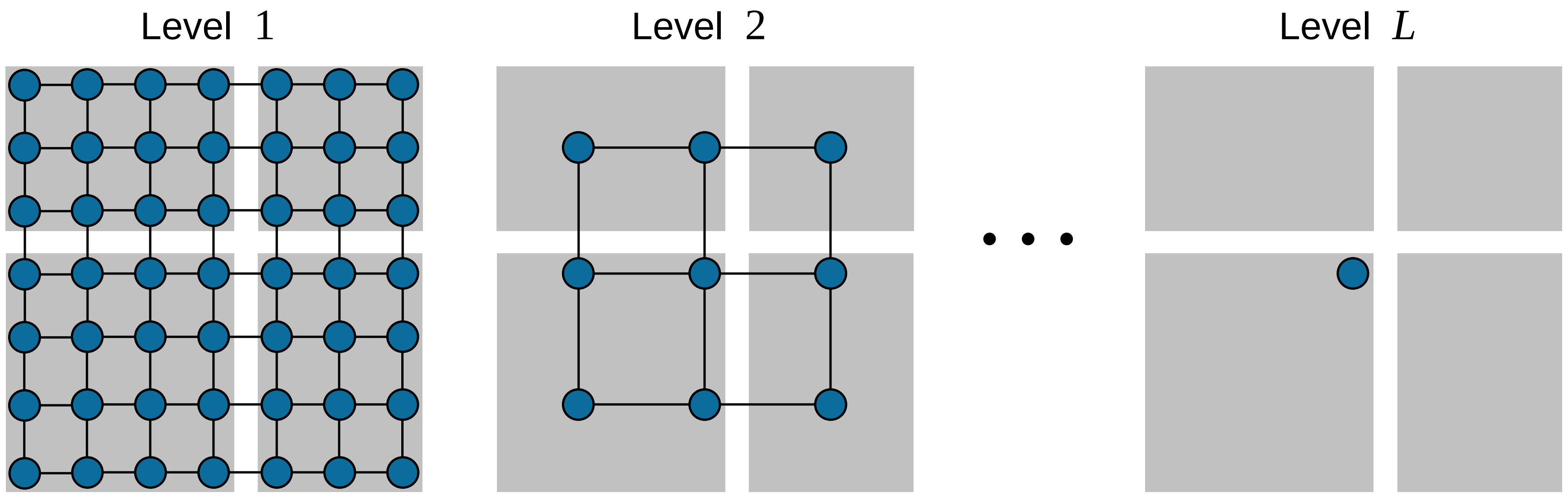

Magnetic flux compression generator simulation enabled by MG smoother research

Parallel AMG in *hypre* now scales to 1.1M cores on Sequoia (IBM BG/Q)



- *m* x *n* denotes *m* MPI tasks and *n* OpenMP threads per node
- Largest problem above: 72B unknowns on 1.1M cores

Approach for parallelizing multigrid is straightforward data decomposition



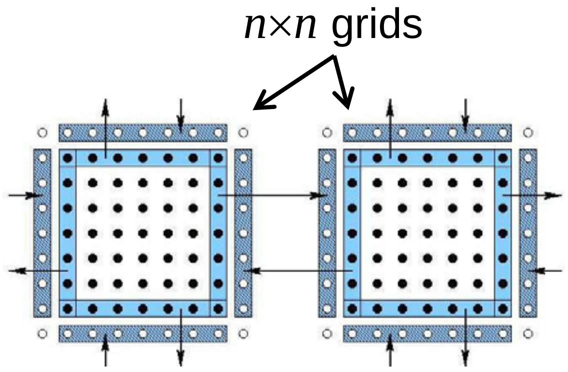
- Basic communication pattern is "nearest neighbor"
  - Relaxation, interpolation, & Galerkin not hard to implement
- Different neighbor processors on coarse grids
- Many idle processors on coarse grids (100K+ on BG/L)
  - Algorithms to take advantage have had limited success

Straightforward parallelization approach is optimal for V-cycles on structured grids (5-pt Laplacian example)

- Standard communication / computation models

$$T_{comm} = \alpha + m\beta \quad \textit{(communicate m doubles)}$$

$$T_{comp} = m\gamma \quad \textit{(compute m flops)}$$



- Time to do relaxation

$$T \approx 4\alpha + 4n\beta + 5n^2\gamma$$

- Time to do relaxation in a V(1,0) multigrid cycle

$$\begin{aligned} T_V &\approx (1+1+\cdots)4\alpha + (1+1/2+\ldots)4n\beta + (1+1/4+\ldots)5n^2\gamma \\ &\approx (\log N)4\alpha + (2)4n\beta + (4/3)5n^2\gamma \end{aligned}$$

- For achieving optimality in general, the *log* term is unavoidable!

- More precise:

$$T_{V,better} \approx T_V + (\log P)(4\beta + 5\gamma)$$

A closer look at the idle processor problem

- The idle processor problem seems severe, but standard parallel V-cycle multigrid performance has optimal order

$$T_V \approx (\log N)4\alpha + (2)4n\beta + (4/3)5n^2\gamma$$

- What are the limits of what we can achieve by trying to use idle processors to accelerate convergence?

- Consider an ideal setting:
  - Fine grid of size N distributed across P procs with n rows per proc (hence $N=nP$)
  - Constant coarsening factor f and grid complexity bound $F=1/(1-1/f)$ (example: 3D cube with full coarsening → $f=8$, $F=8/7$)

- Assume multigrid takes CN work to converge and we don't know how to do less work than this

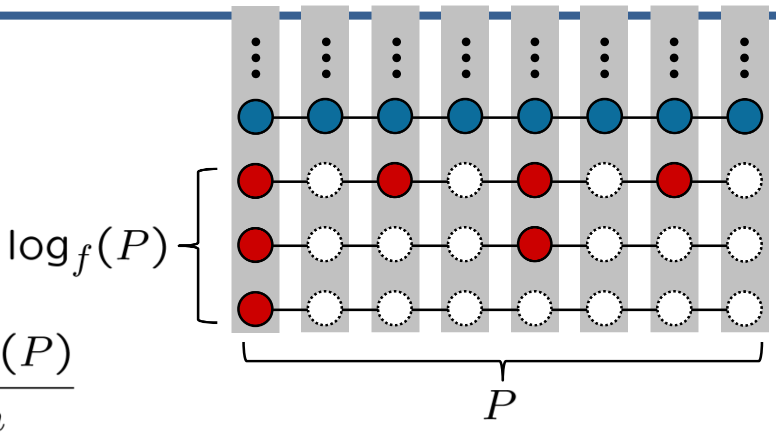
A closer look at the idle processor problem (2)

- Work potential

$$\log_f(P)P - (FP - P)$$

- Best speedup per V-cycle

$$\frac{FN + \log_f(P)P - FP + P}{FN} \leq 1 + \frac{\log_f(P)}{n}$$



- Overall speedup assuming at least one V-cycle is required

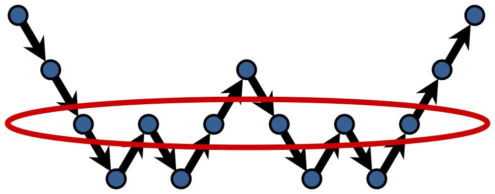
$$\max\left\{1 + \frac{\log_f(P)}{n}, \frac{C}{F}\right\}$$

- Only makes sense if n is very small
  - Example: 3D Laplace on 1M procs: $\log_8(P) < 7$
- And this analysis is extremely optimistic!
  - Assumes computations in cycle 2 can be done before cycle 1

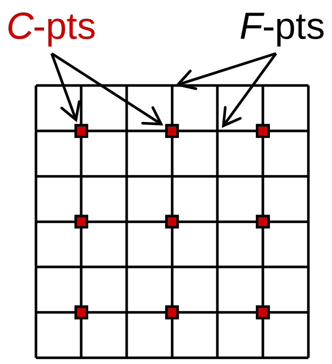
Additional comments on parallel multigrid

- W-cycles scale poorly:

$$T_W \approx (2^{\log N})4\alpha + (\log N)4n\beta + (2)5n^2\gamma$$



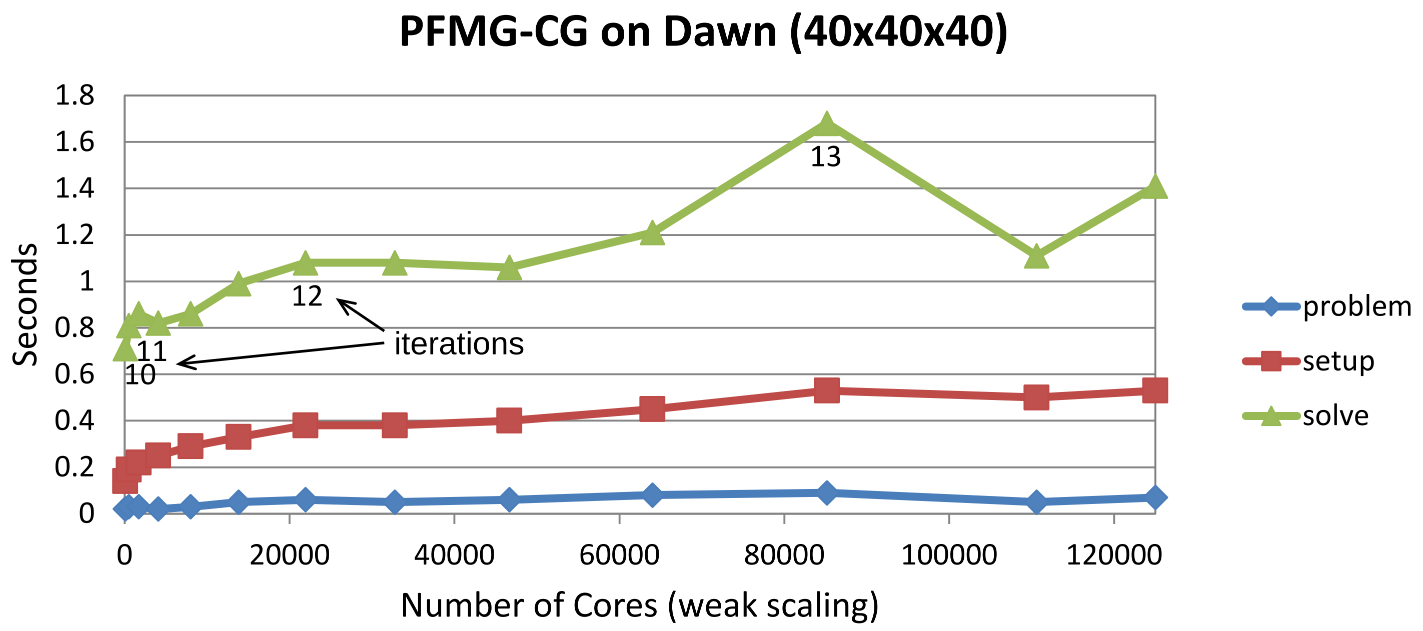
- Lexicographical Gauss-Seidel is too sequential
  - Use red/black or multi-color GS
  - Use weighted Jacobi, hybrid Jacobi/GS, L1
  - Use *C*-*F* relaxation (Jacobi on *C*-pts then *F*-pts)
  - Use Polynomial smoothers



- Parallel smoothers are often less effective

- Survey on parallel multigrid, paper on parallel smoothers:
  - "A Survey of Parallelization Techniques for Multigrid Solvers," Chow, Falgout, Hu, Tuminaro, and Yang, *Parallel Processing For Scientific Computing*, Heroux, Raghavan, and Simon, editors, SIAM, series on Software, Environments, and Tools (2006)
  - "Multigrid Smoothers for Ultra-Parallel Computing," Baker, Falgout, Kolev, and Yang, *SIAM J. Sci. Comput.*, 33 (2011), pp. 2864-2887.

Example weak scaling results on Dawn (an IBM BG/P system at LLNL) in 2011



- Laplacian on a cube; 40^3 = 64K grid per processor; largest had 8 billion unknowns
- PFMG is a semicoarsening multigrid solver in *hypre*
- Constant-coefficient version - 1 trillion unknowns on 131K cores in 83 seconds
- Can improve setup (these results already use the assumed partition algorithm described later)

Parallel Algebraic Multigrid (AMG)

Preliminaries… the Galerkin coarse-grid operator

- As before, consider solving the $N \times N$ linear system

$$Au = f$$

- Let P be prolongation (interpolation) and P^T restriction

- The coarse-grid operator is defined by the Galerkin procedure, $A_c = P^T A P$

- This gives the "best" coarse-grid correction in the sense that the solution e_c of the coarse system

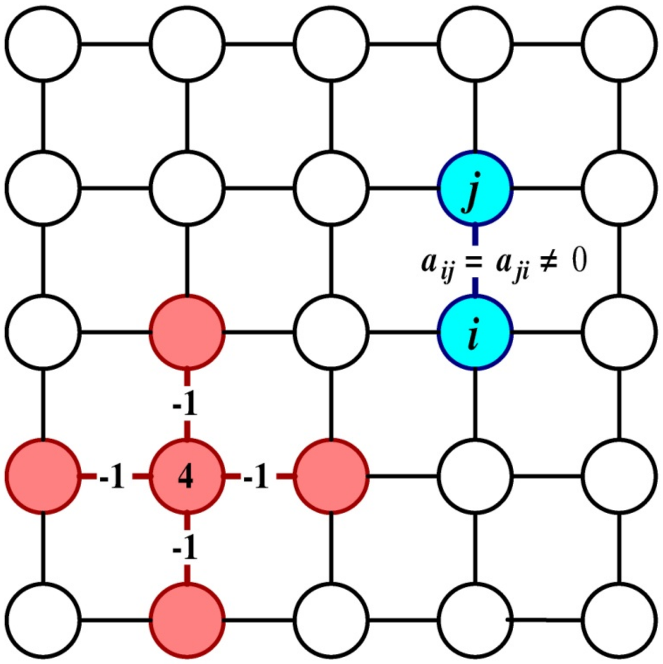
$$A_c e_c = P^T r$$

satisfies

$$e_c = \text{arg min} \; \|e - P e_c\|_A$$

Preliminaries… AMG "grids"

- Matrix adjacency graphs play an important role in AMG:
  - grid = set of graph vertices
  - grid point i = vertex i
- As a visual aid, it is highly instructive to relate the matrix equations to an underlying PDE and discretization
- We will often draw the grid points in their geometric locations
- Remember that AMG doesn't actually use this geometric information!



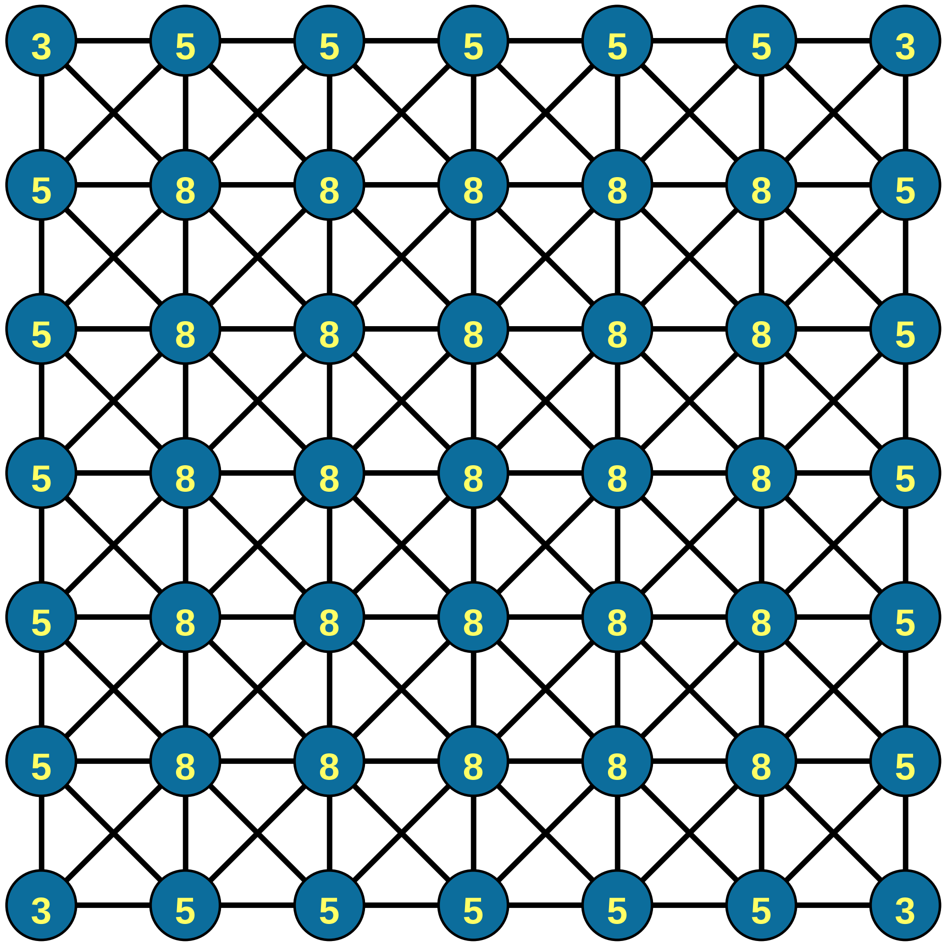
Choosing the coarse grid

- Classical AMG (C-AMG) – coarse grid is a subset of the fine grid

- The basic coarsening procedure is as follows:
  - Define a strength matrix A_s by deleting weak connections in A
  - First pass: Choose an independent set of fine-grid points based on the graph of A_s
  - Second pass: Choose additional points if needed to satisfy interpolation requirements

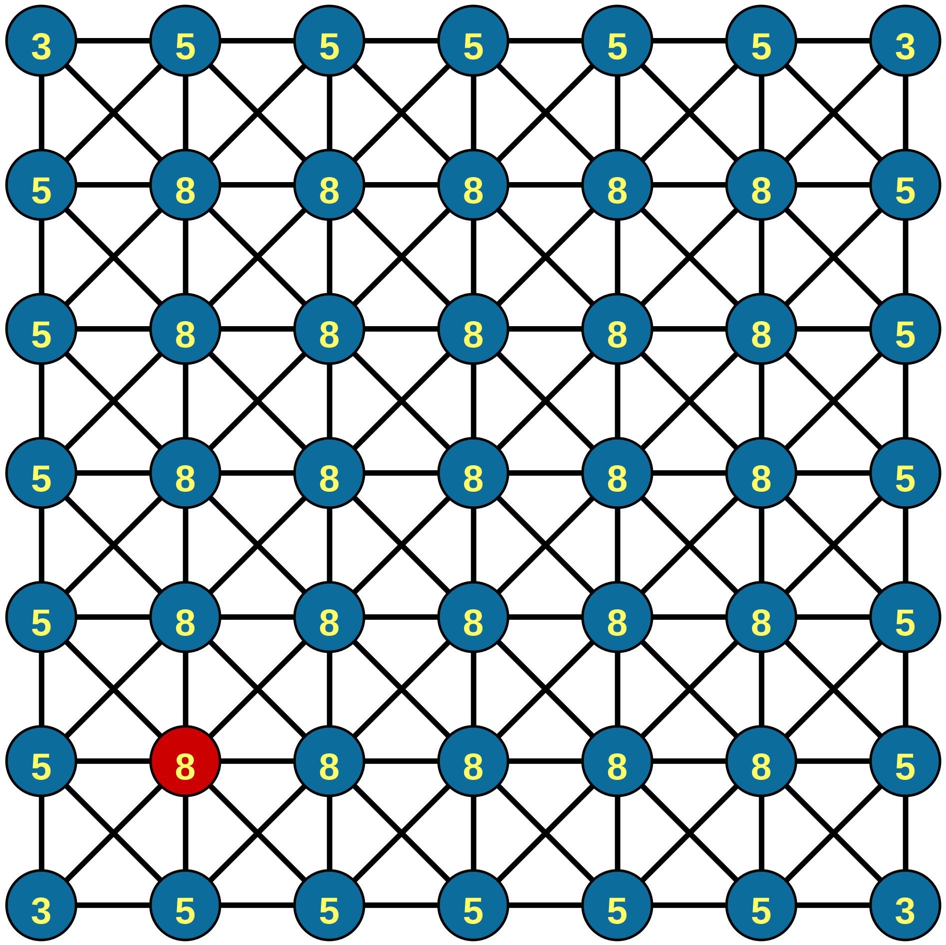
- Coarsening partitions the grid into *C*- and *F*-points

C-AMG coarsening



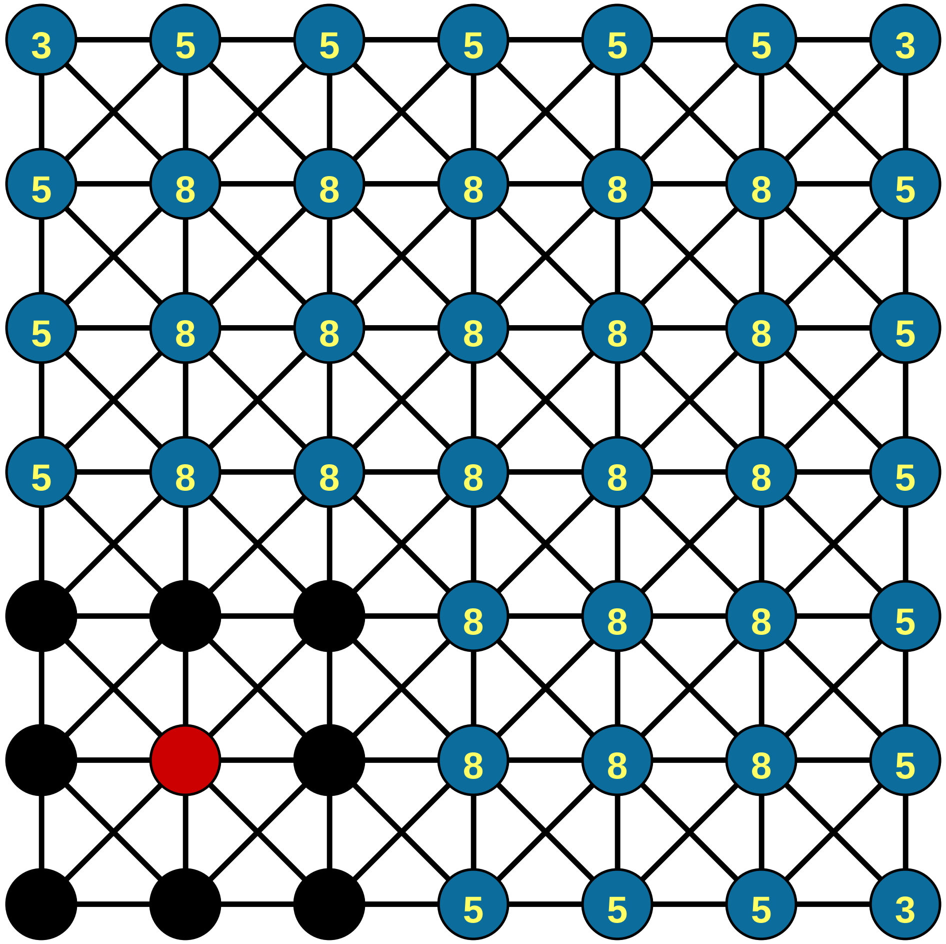
- **select C-pt with maximal measure**
- **select neighbors as F-pts**
- **update measures of F-pt neighbors**

C-AMG coarsening



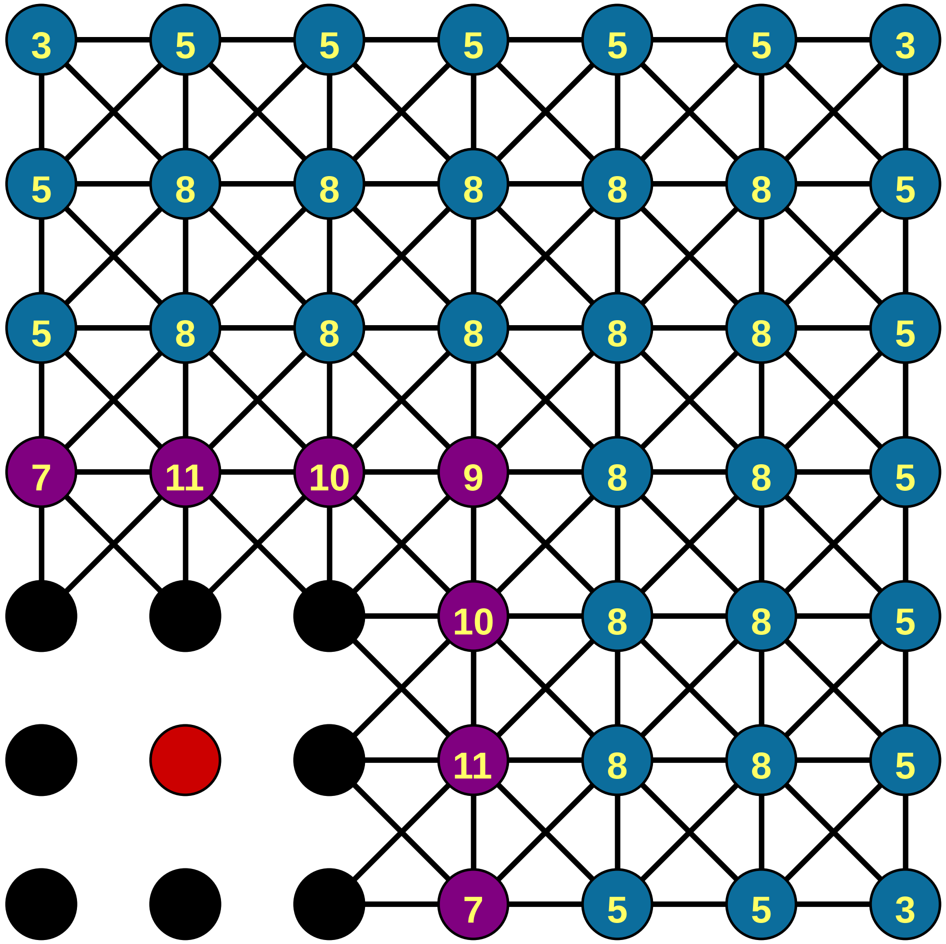
- select C-pt with maximal measure
- select neighbors as F-pts
- update measures of F-pt neighbors

C-AMG coarsening



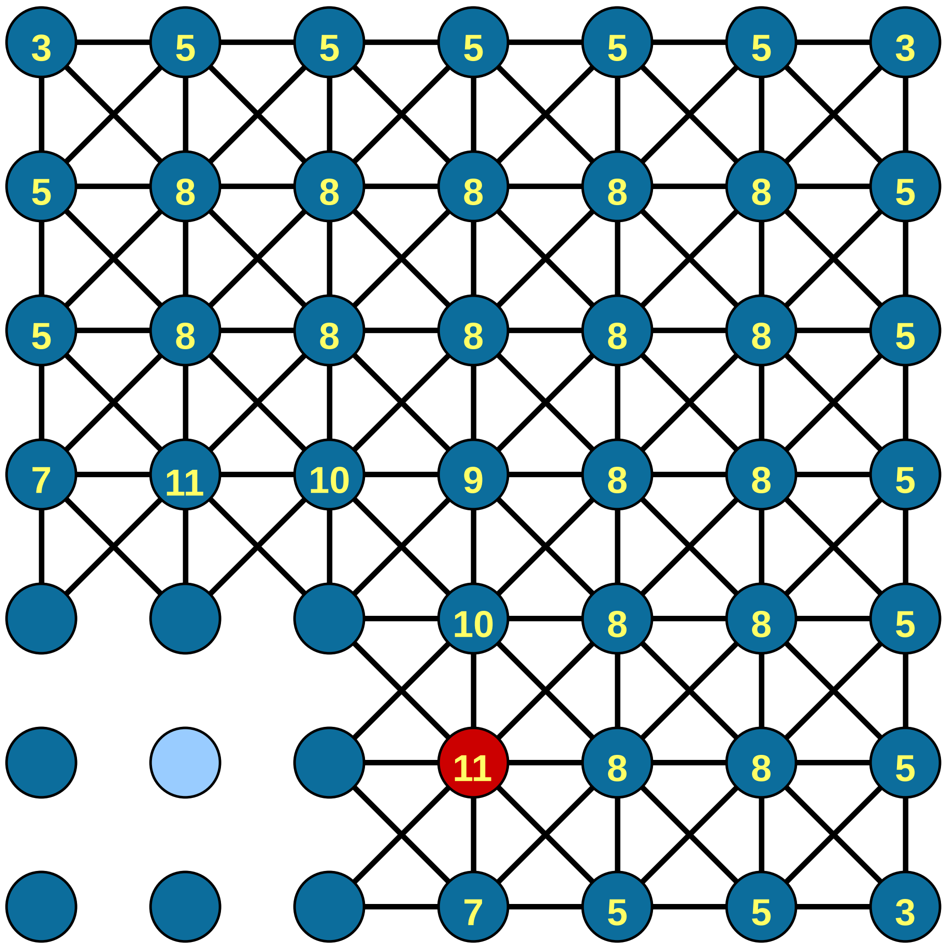
- select C-pt with maximal measure
- select neighbors as F-pts
- update measures of F-pt neighbors

C-AMG coarsening



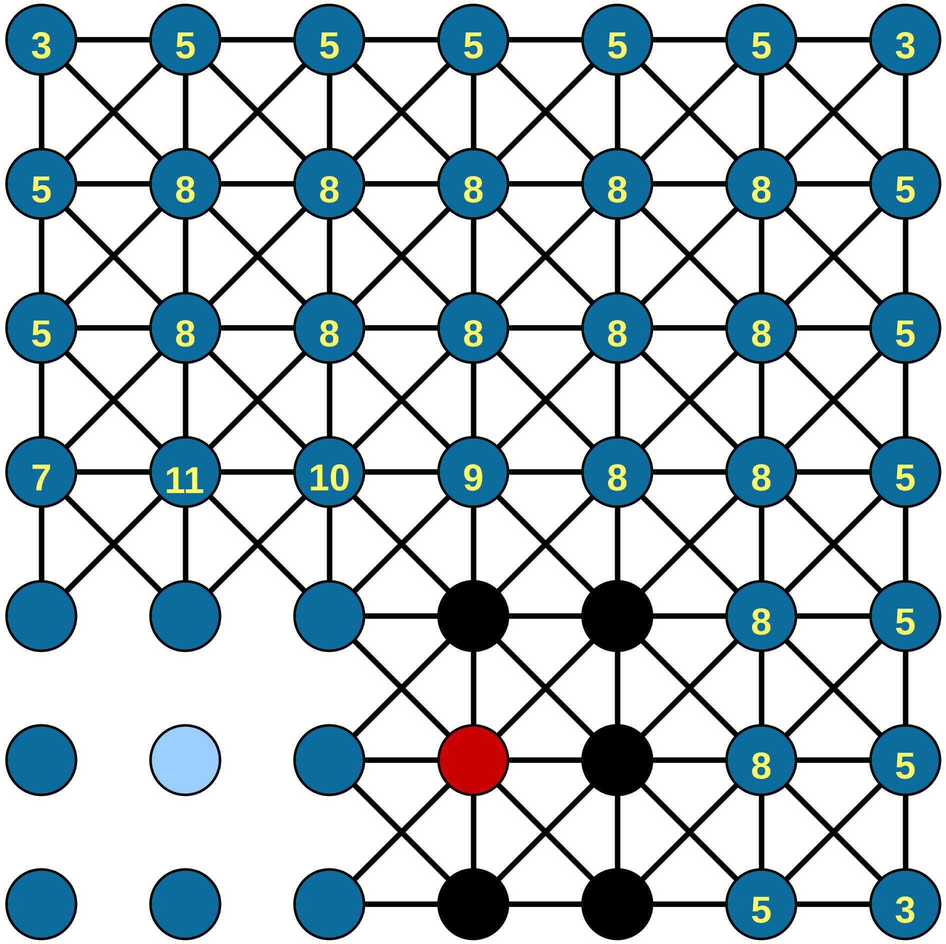
- select C-pt with maximal measure
- select neighbors as F-pts
- update measures of F-pt neighbors

C-AMG coarsening



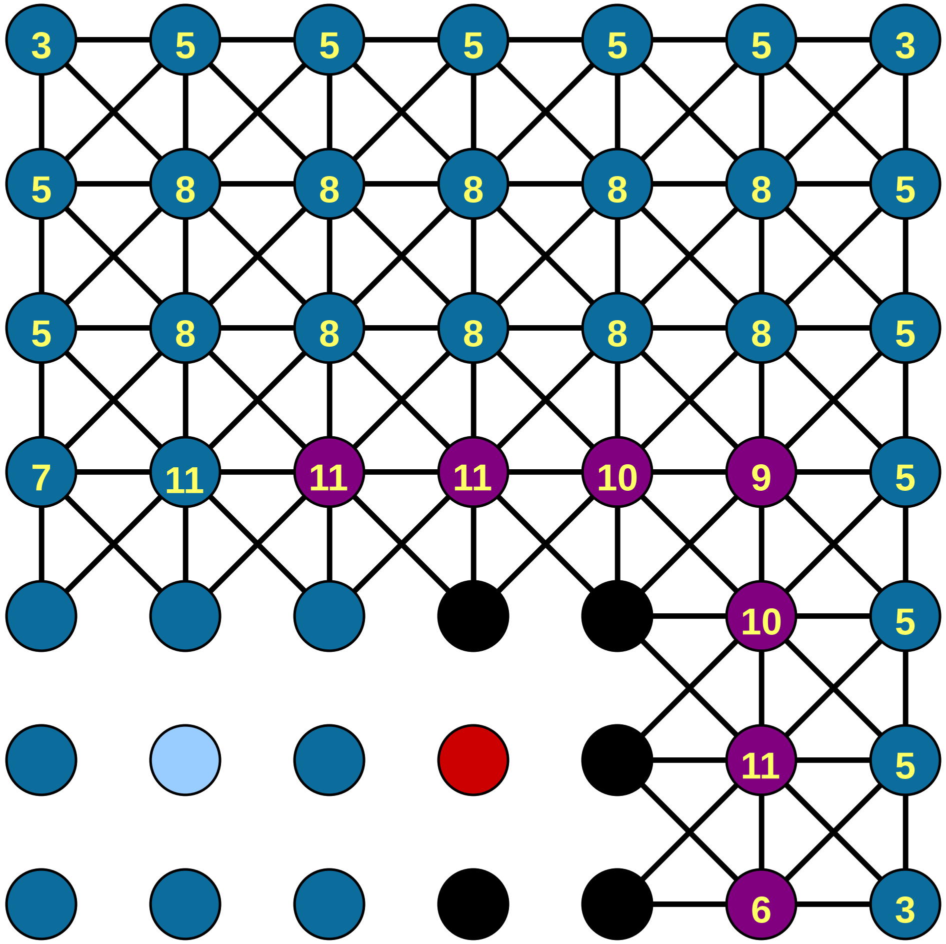
- select C-pt with maximal measure
- select neighbors as F-pts
- update measures of F-pt neighbors

C-AMG coarsening



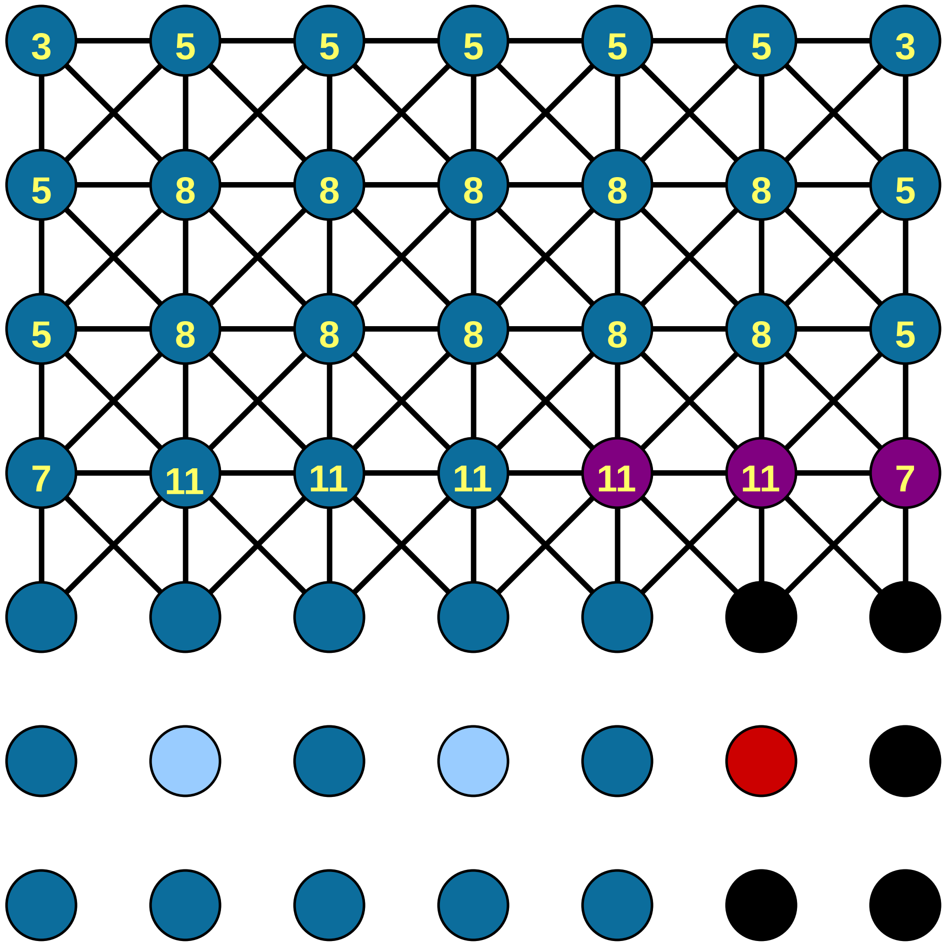
- select C-pt with maximal measure
- select neighbors as F-pts
- update measures of F-pt neighbors

C-AMG coarsening



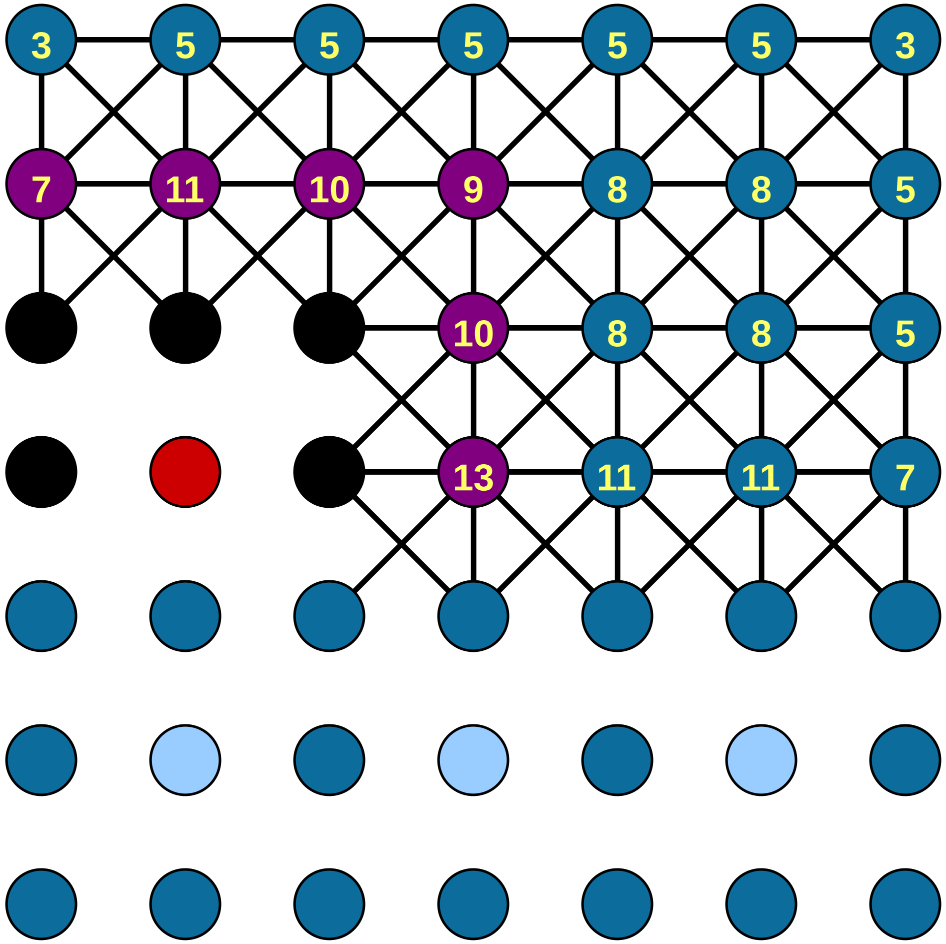
- select C-pt with maximal measure
- select neighbors as F-pts
- update measures of F-pt neighbors

C-AMG coarsening



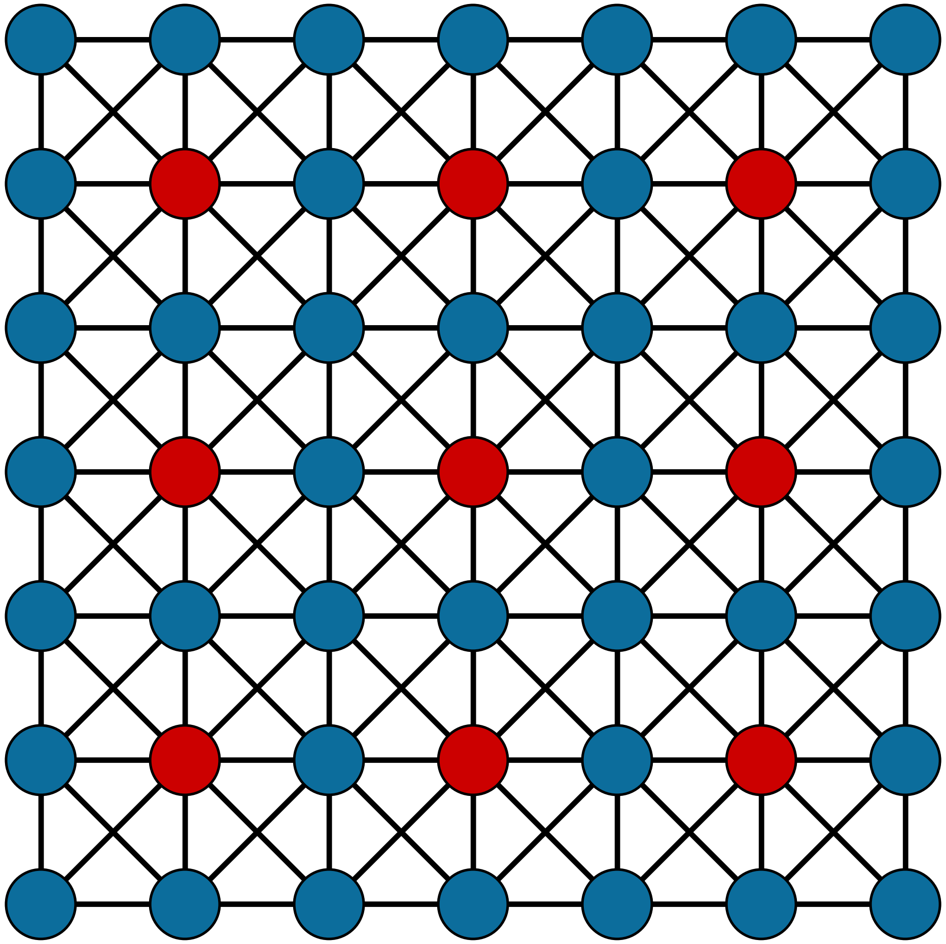
- select C-pt with maximal measure
- select neighbors as F-pts
- update measures of F-pt neighbors

C-AMG coarsening



- select C-pt with maximal measure
- select neighbors as F-pts
- update measures of F-pt neighbors

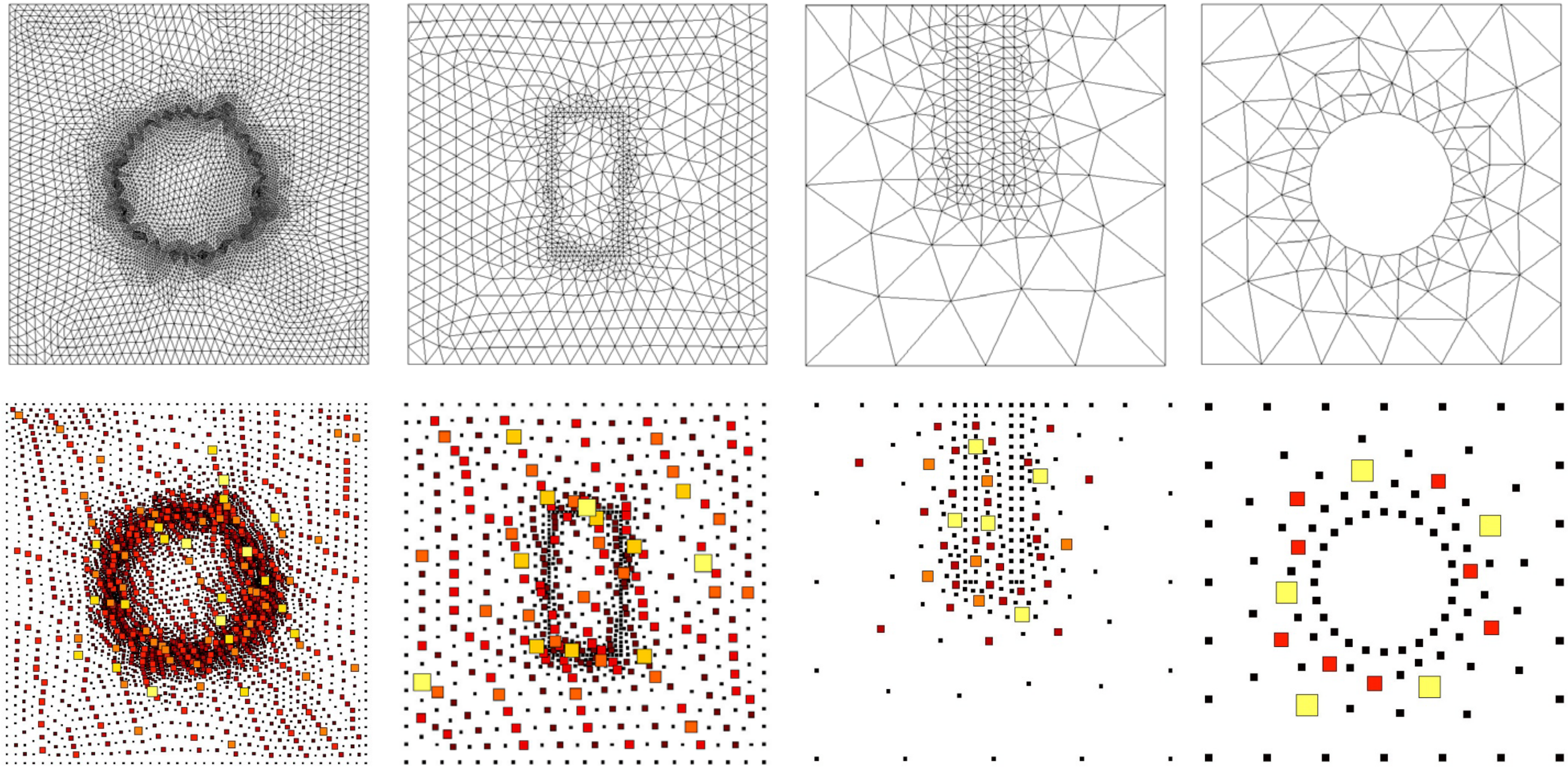
C-AMG coarsening is inherently sequential



- select C-pt with maximal measure
- select neighbors as F-pts
- update measures of F-pt neighbors

AMG grid hierarchies for several 2D problems

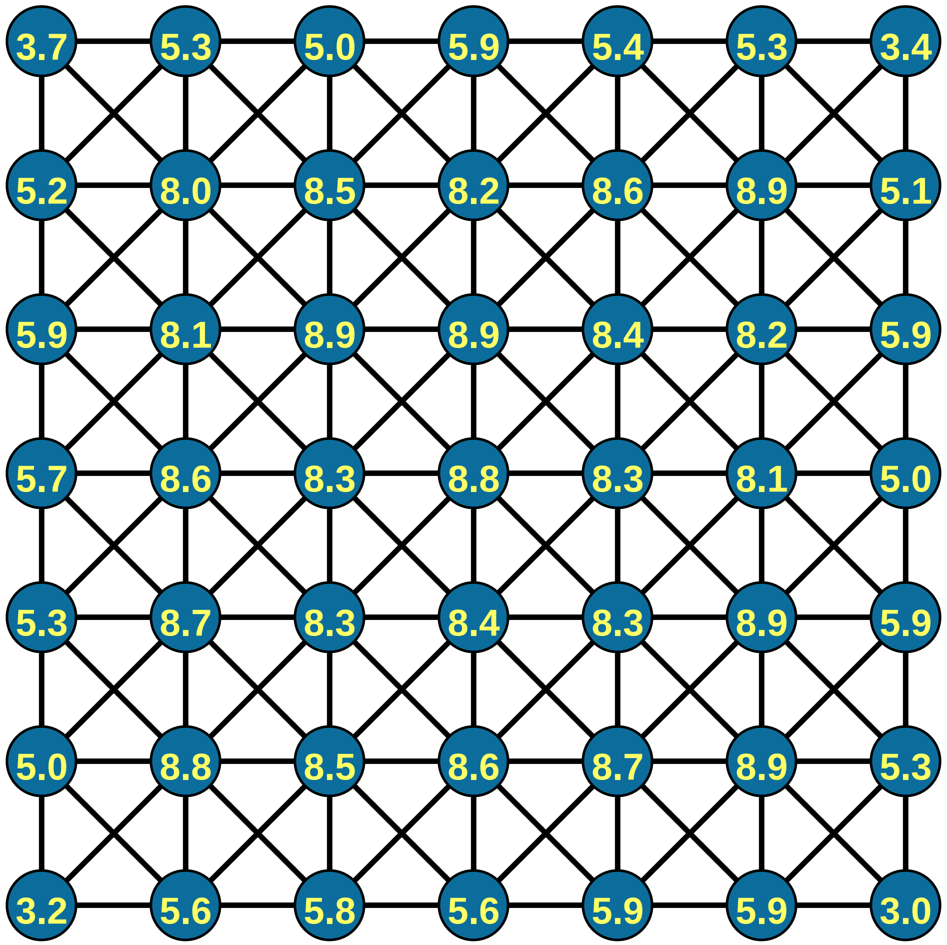
domain1 - 30^o	domain2 - 30^o	pile	square-hole



Parallel Coarsening Algorithms

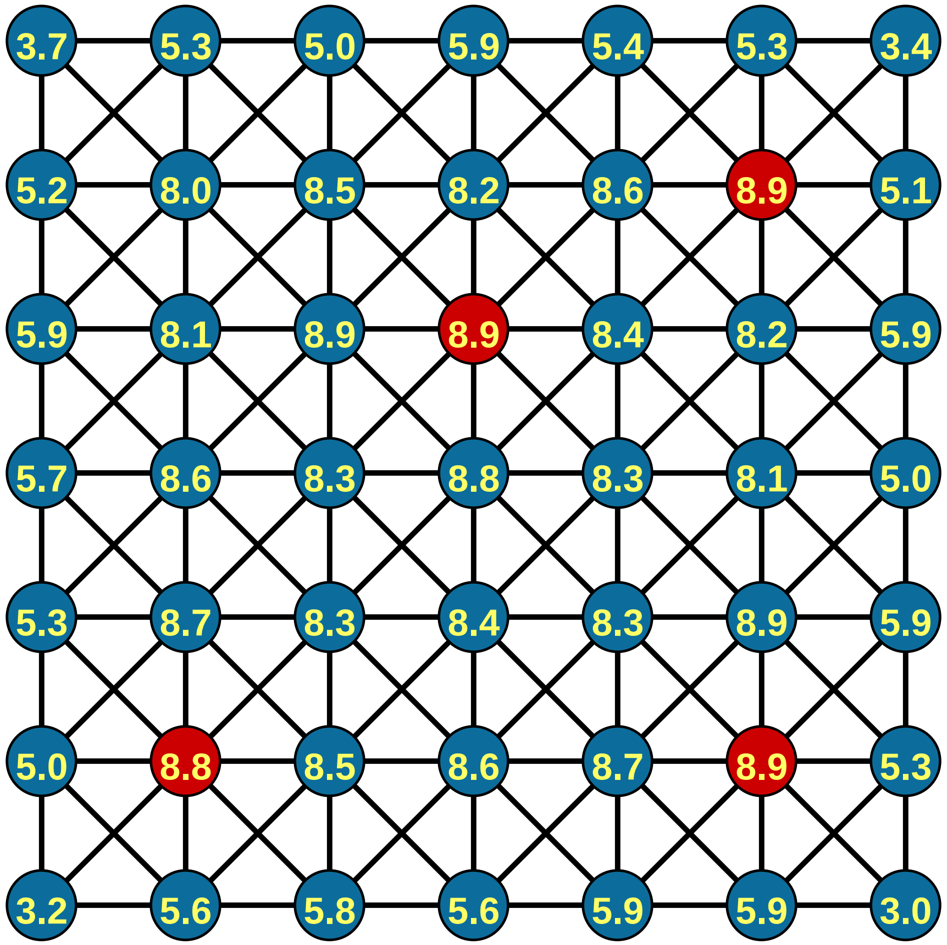
- C-AMG coarsening algorithm is inherently sequential
- Several parallel algorithms (in *hypre*):
  - CLJP (Cleary-Luby-Jones-Plassmann) – one-pass approach with random numbers to get concurrency (illustrated next)
  - Falgout – C-AMG on processor interior, then CLJP to finish
  - PMIS – CLJP without the 'C'; parallel version of C-AMG first pass
  - HMIS – C-AMG on processor interior, then PMIS to finish
  - CGC (Griebel, Metsch, Schweitzer) – compute several coarse grids on each processor, then solve a global graph problem to select the grids with the best "fit"
  - …
- Other parallel AMG codes use similar approaches

CLJP coarsening is fully parallel



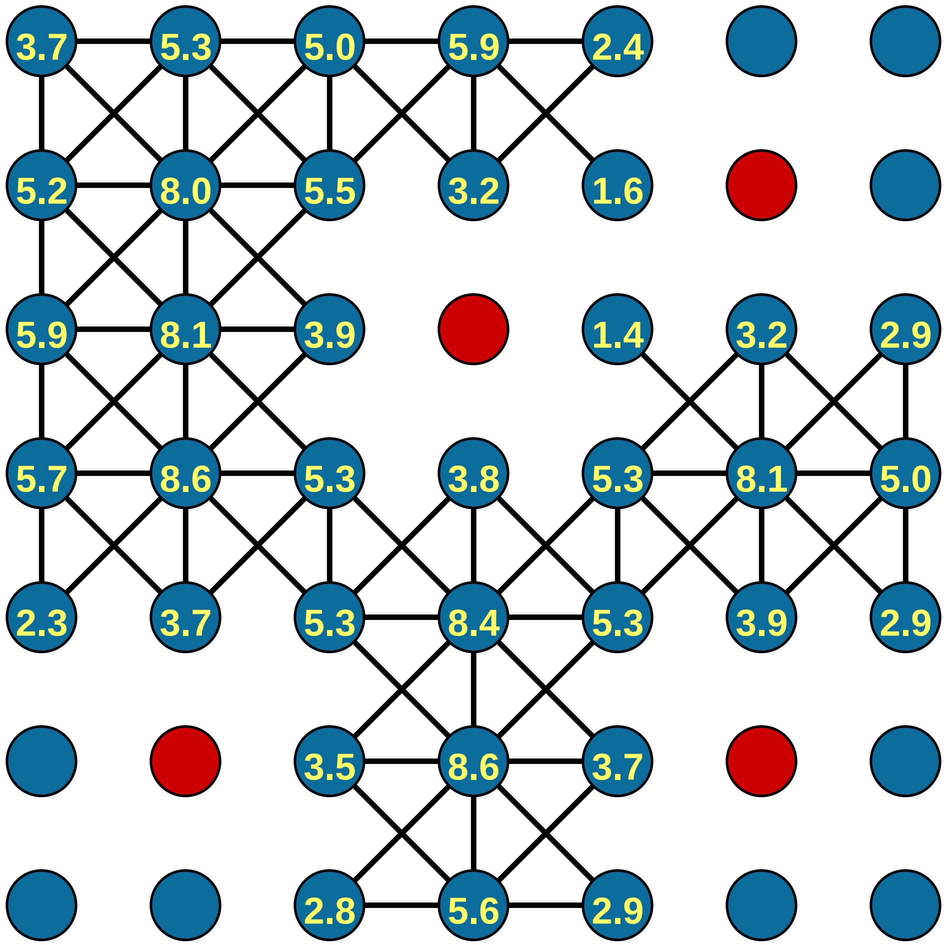
- **select C-pts with maximal measure locally**
- **remove neighbor edges**
- **update neighbor measures**

CLJP coarsening is fully parallel



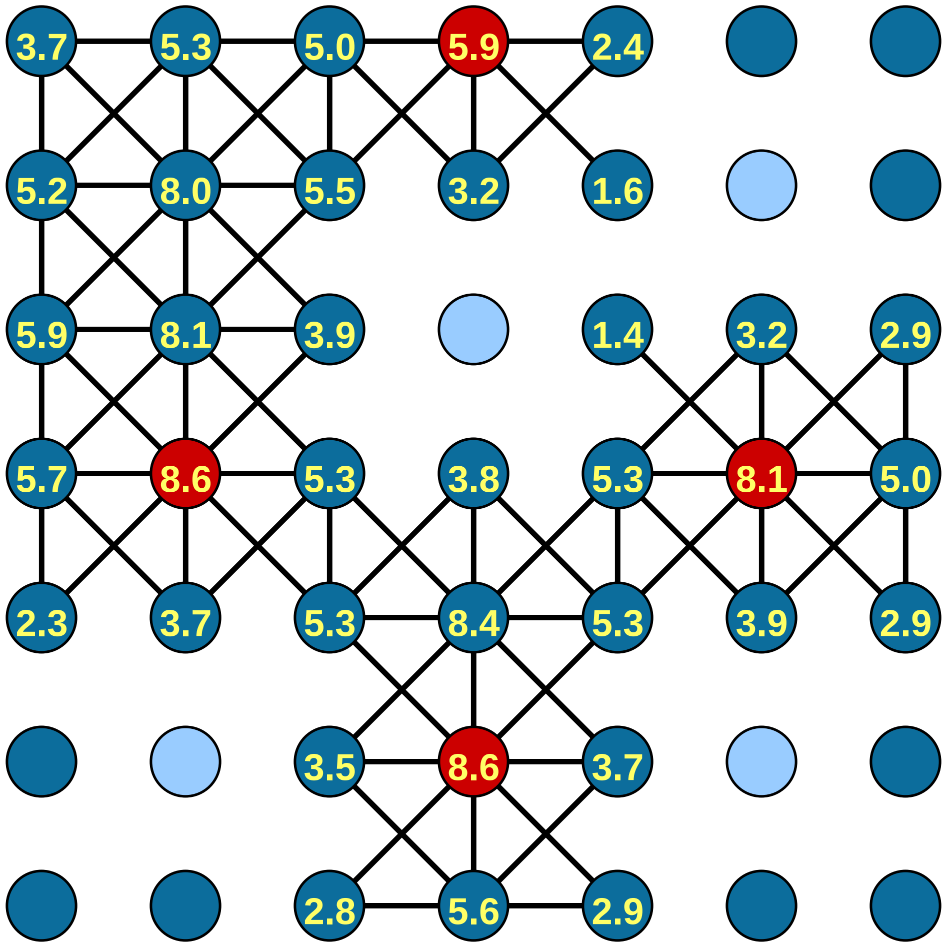
- **select C-pts with maximal measure locally**
- **remove neighbor edges**
- **update neighbor measures**

CLJP coarsening is fully parallel



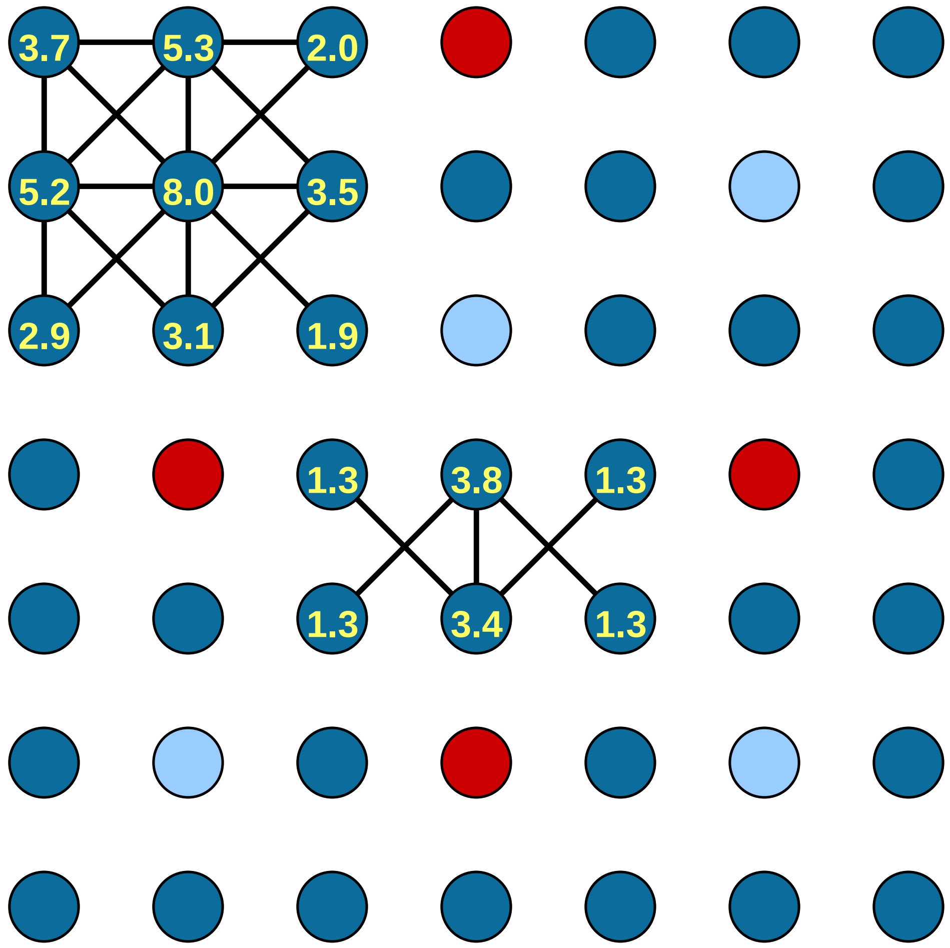
- select C-pts with maximal measure locally
- remove neighbor edges
- update neighbor measures

CLJP coarsening is fully parallel



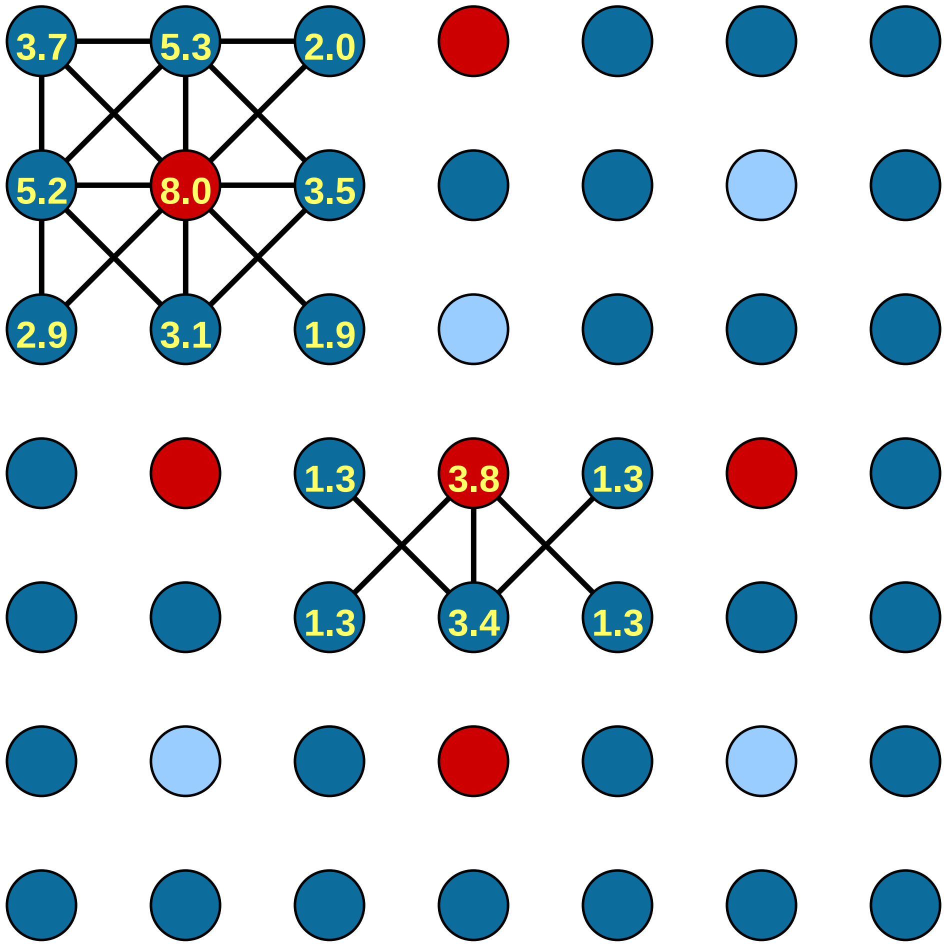
- select C-pts with maximal measure locally
- remove neighbor edges
- update neighbor measures

CLJP coarsening is fully parallel



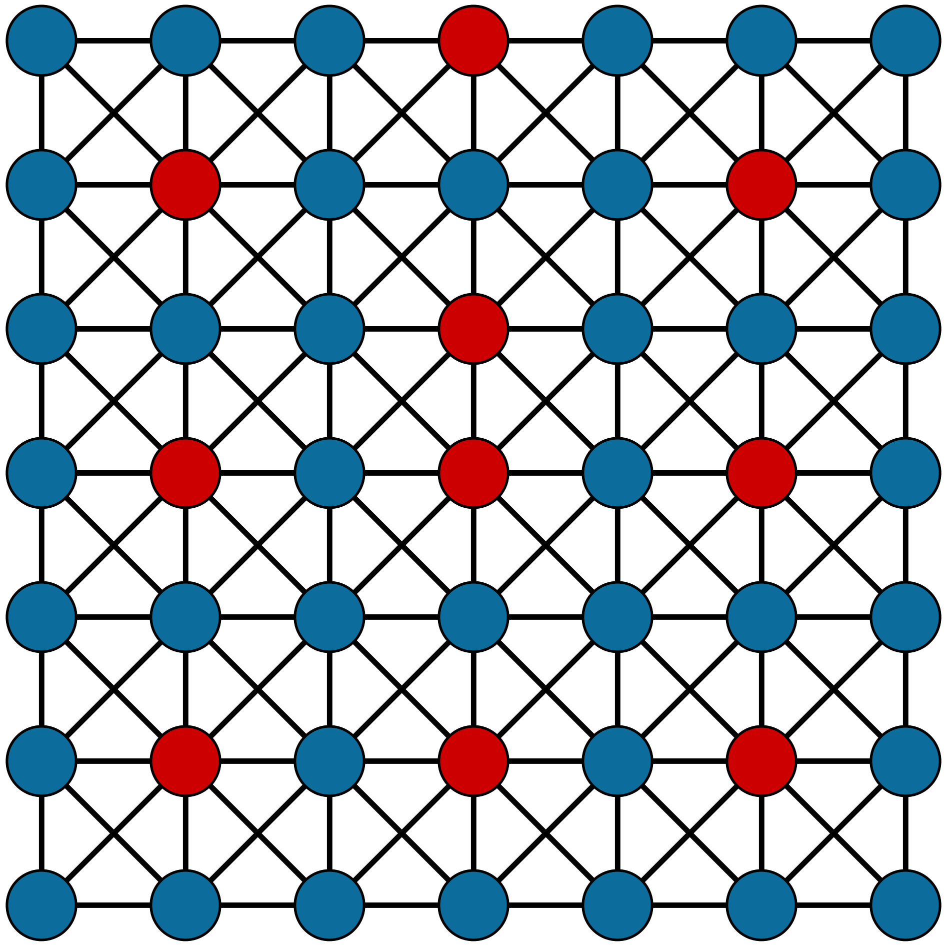
- select C-pts with maximal measure locally
- remove neighbor edges
- update neighbor measures

CLJP coarsening is fully parallel



- select C-pts with maximal measure locally
- remove neighbor edges
- update neighbor measures

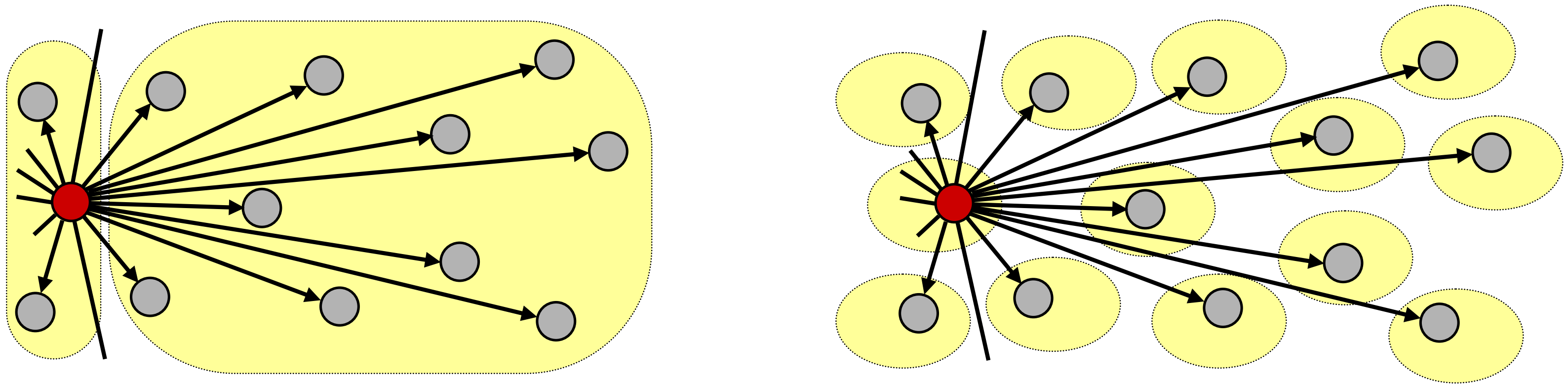
CLJP coarsening is fully parallel



- **10 C-points selected**
- **Standard AMG selects 9 C-points**

Parallel coarse-grid selection in AMG can produce unwanted side effects

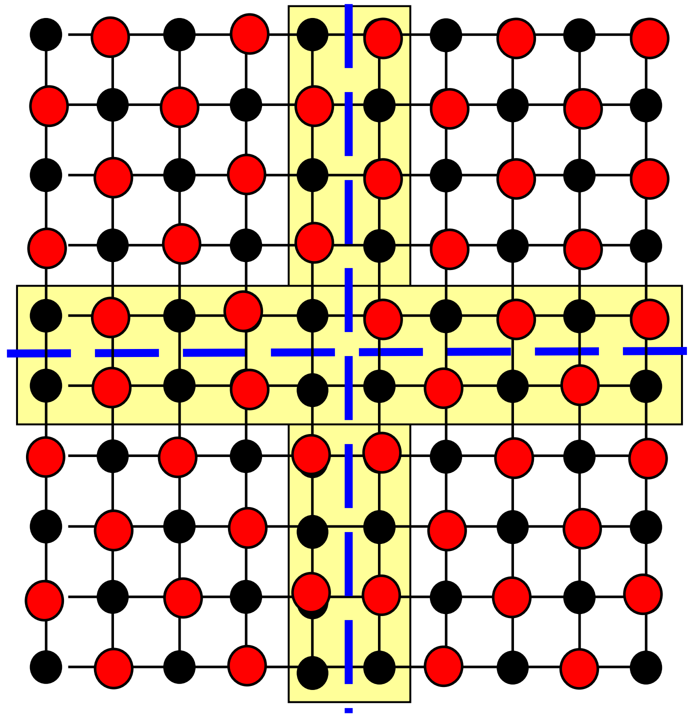
- Non-uniform grids can lead to increased operator complexity and poor convergence
- Operator "stencil growth" reduces parallel efficiency



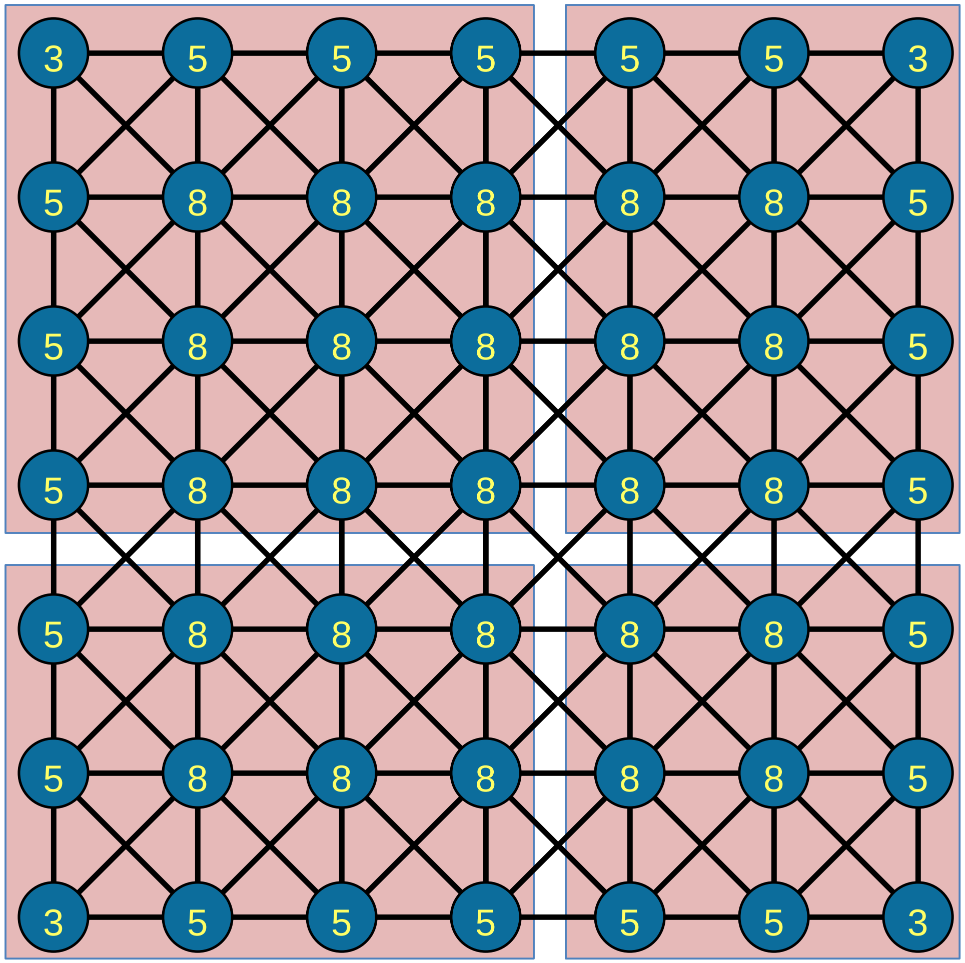
- Currently no guaranteed ways to control complexity
- Can ameliorate with more aggressive coarsening
- Requires long-range interpolation approaches

More aggressive coarsening in parallel – PMIS eliminates the second pass

- Parallel coarsening algorithms – perform sequential algorithm on each processor, then deal with processor boundaries

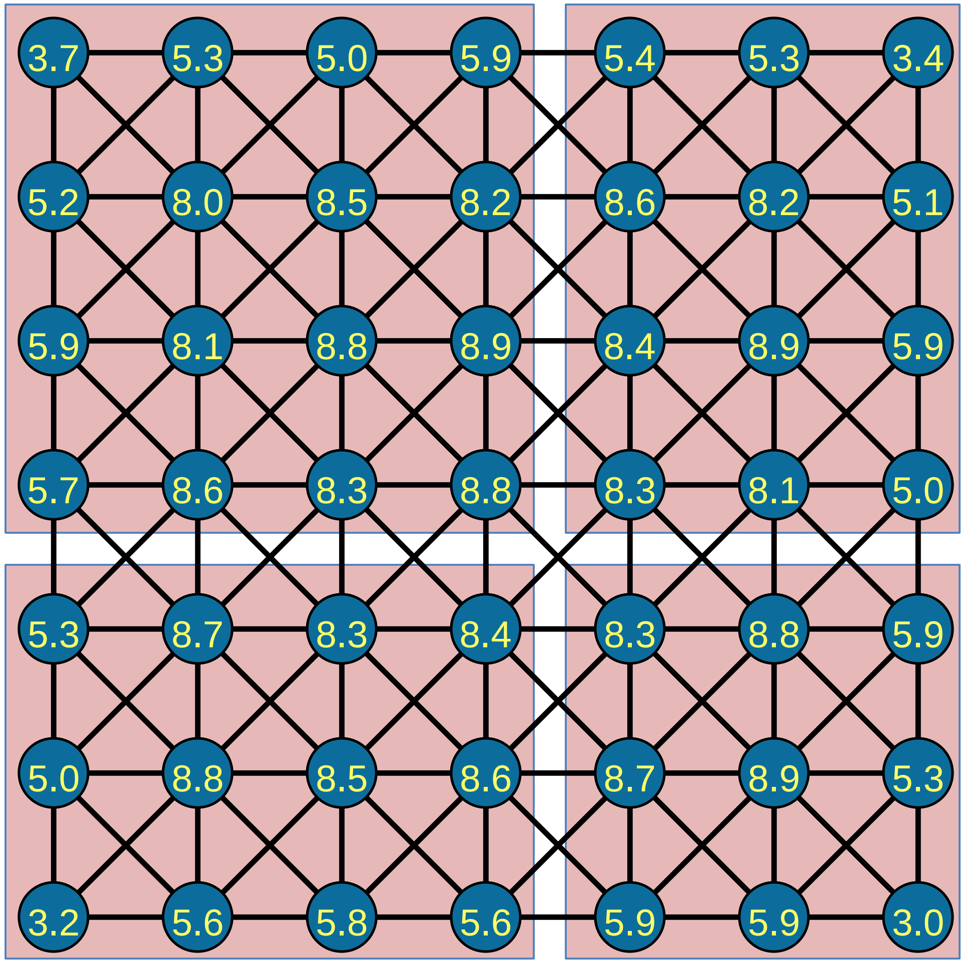


PMIS: start



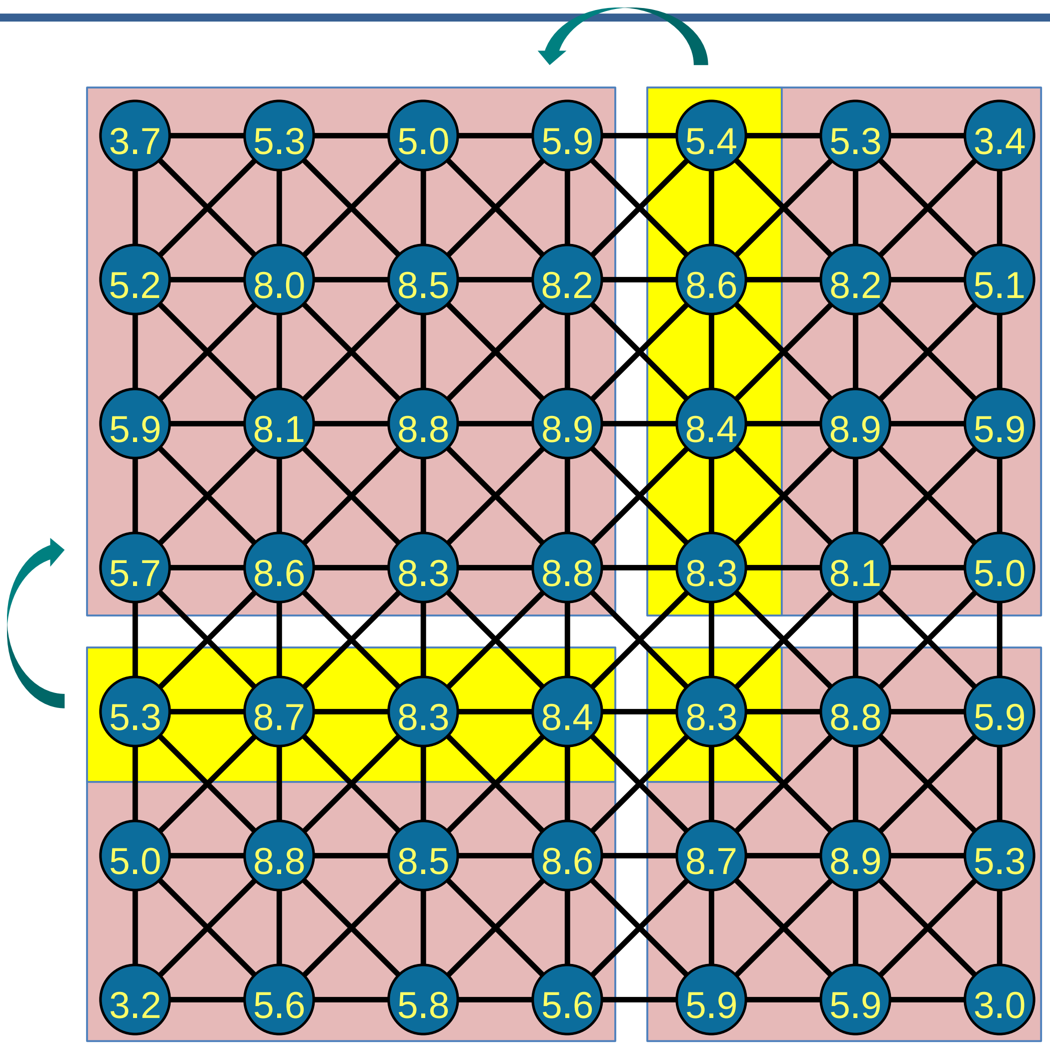
- select C-pt with maximal measure
- select neighbors as F-pts

PMIS: add random numbers



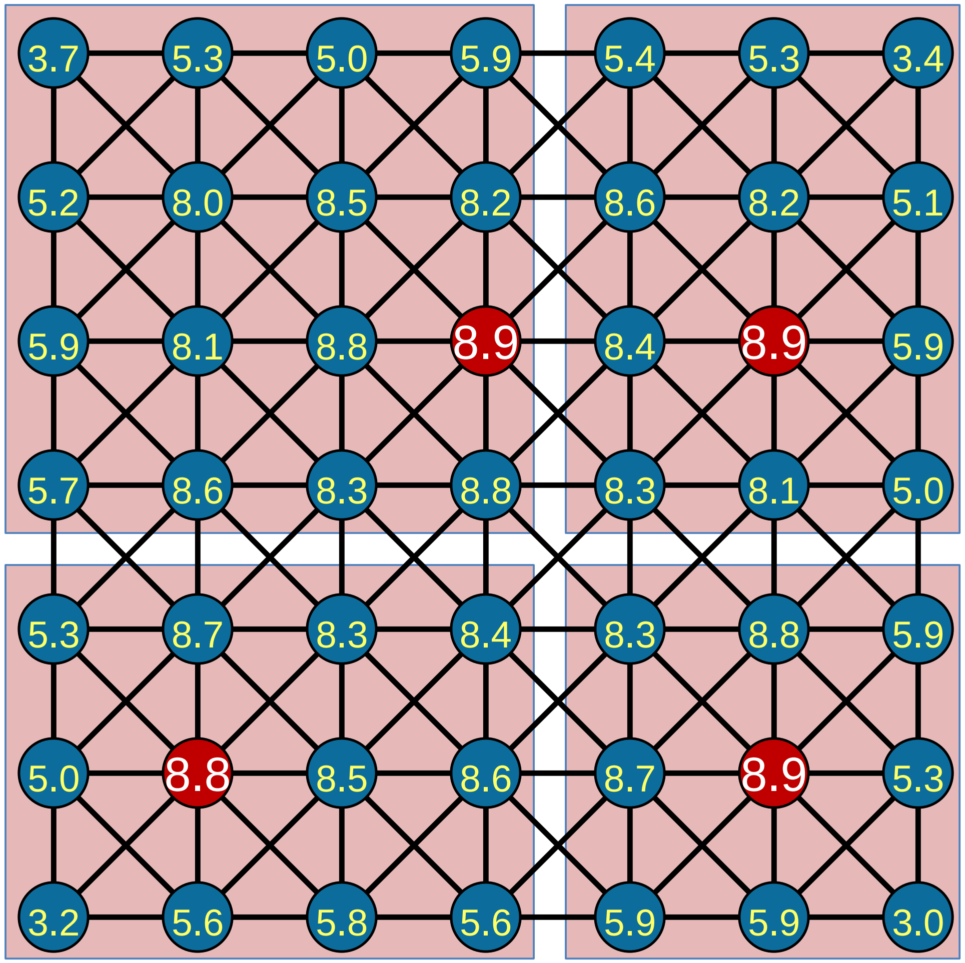
- select C-pt with maximal measure
- select neighbors as F-pts

PMIS: exchange neighbor information



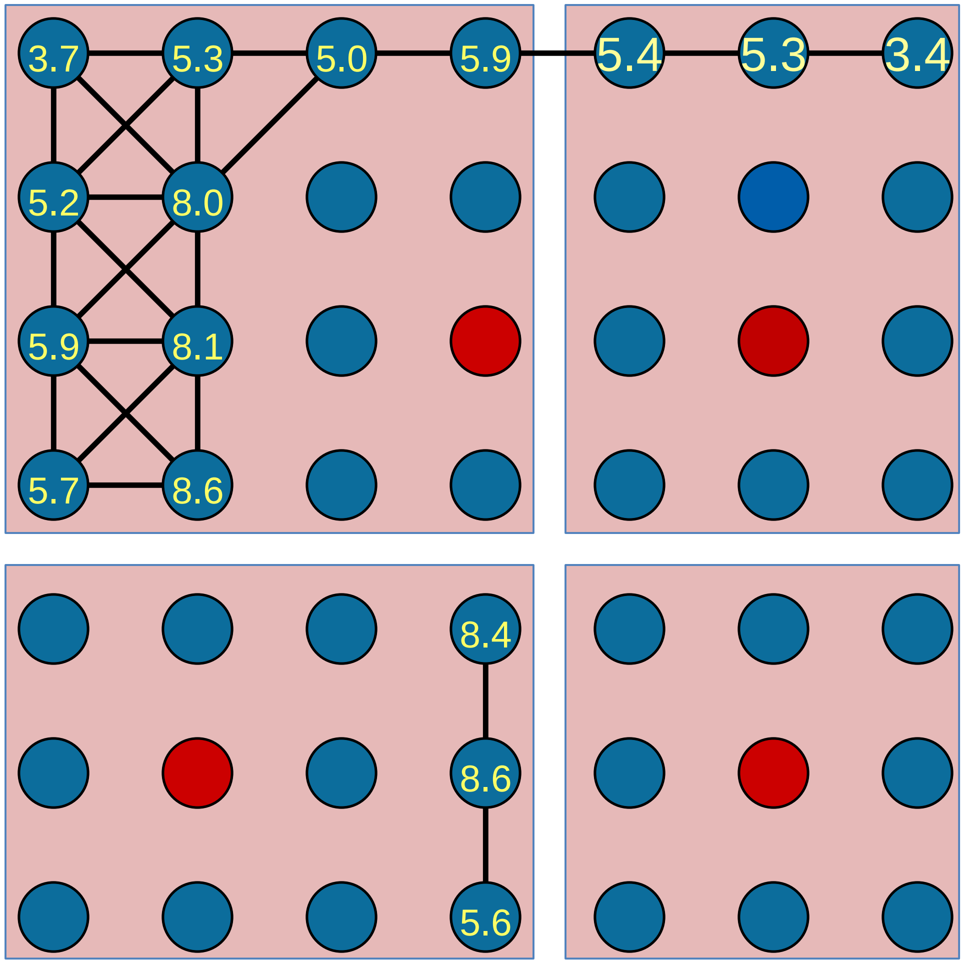
- select C-pt with maximal measure
- select neighbors as F-pts

PMIS: select



- select C-pts with maximal measure locally
- make neighbors F-pts

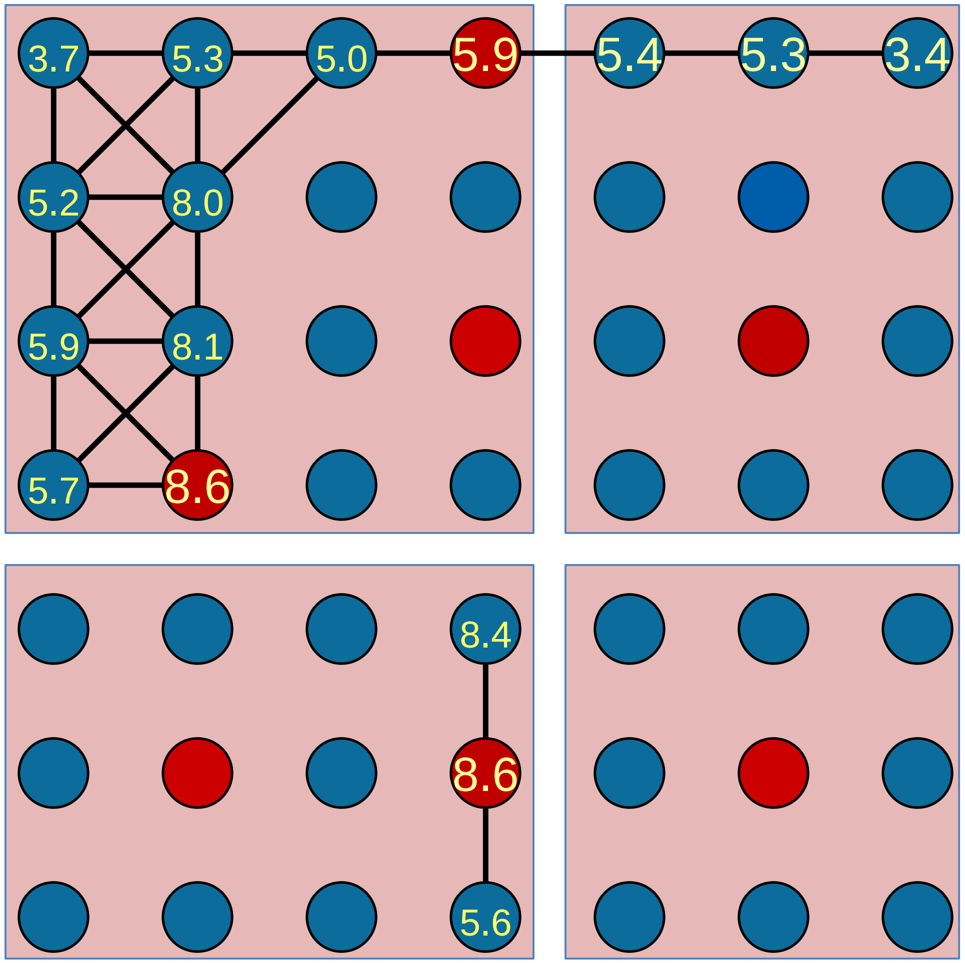
PMIS: update 1



- select C-pts with maximal measure locally

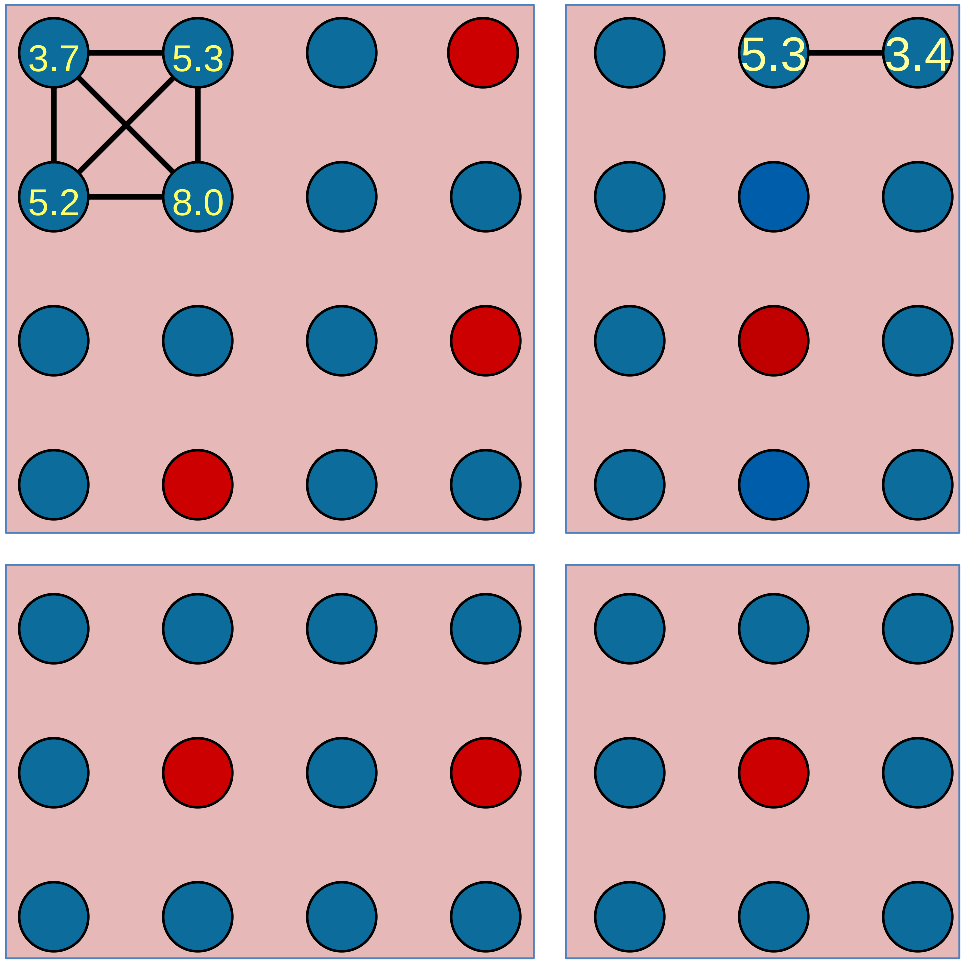
- make neighbors F-pts
(requires neighbor
 info)

PMIS: select



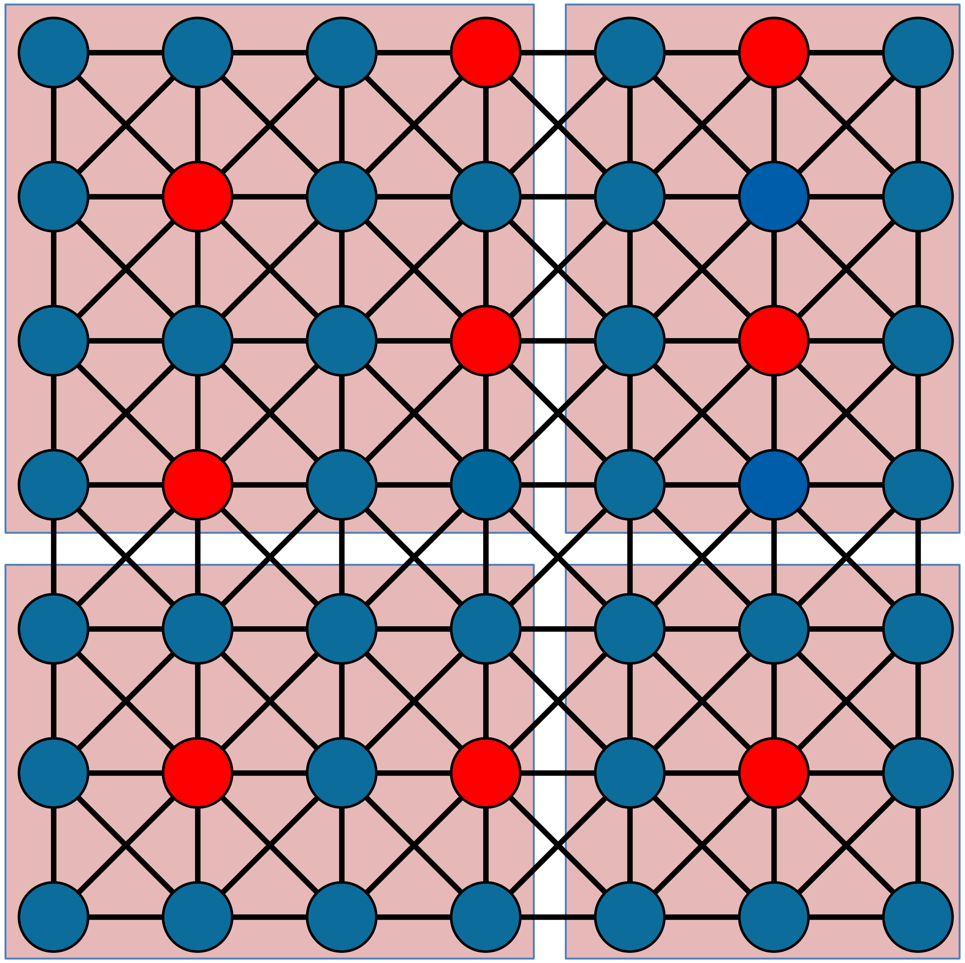
- select C-pts with maximal measure locally
- make neighbors F-pts

PMIS: update 2



- select C-pts with maximal measure locally
- make neighbors F-pts

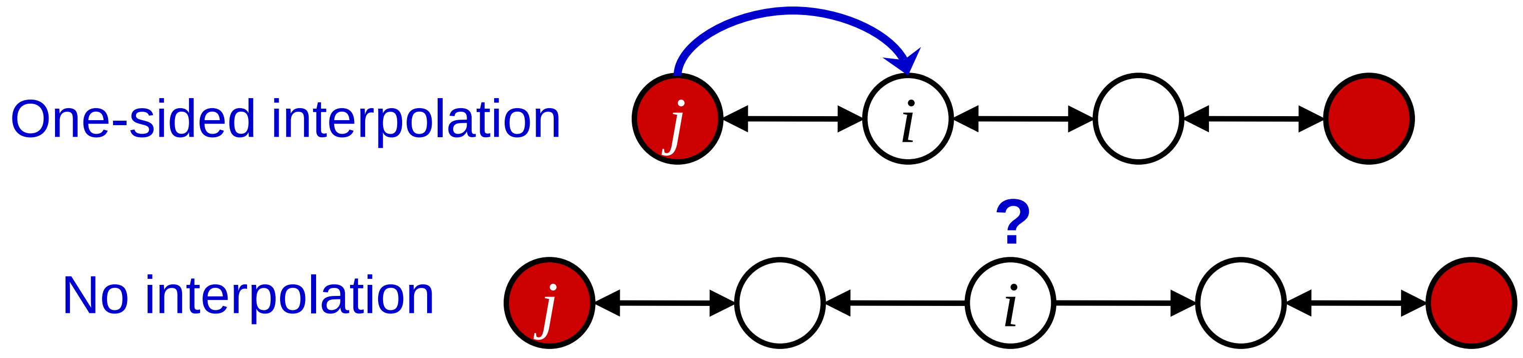
PMIS: final grid



- select C-pts with maximal measure locally
- make neighbor F-pts
- remove neighbor edges

C-AMG interpolation is not suitable for more aggressive coarsening

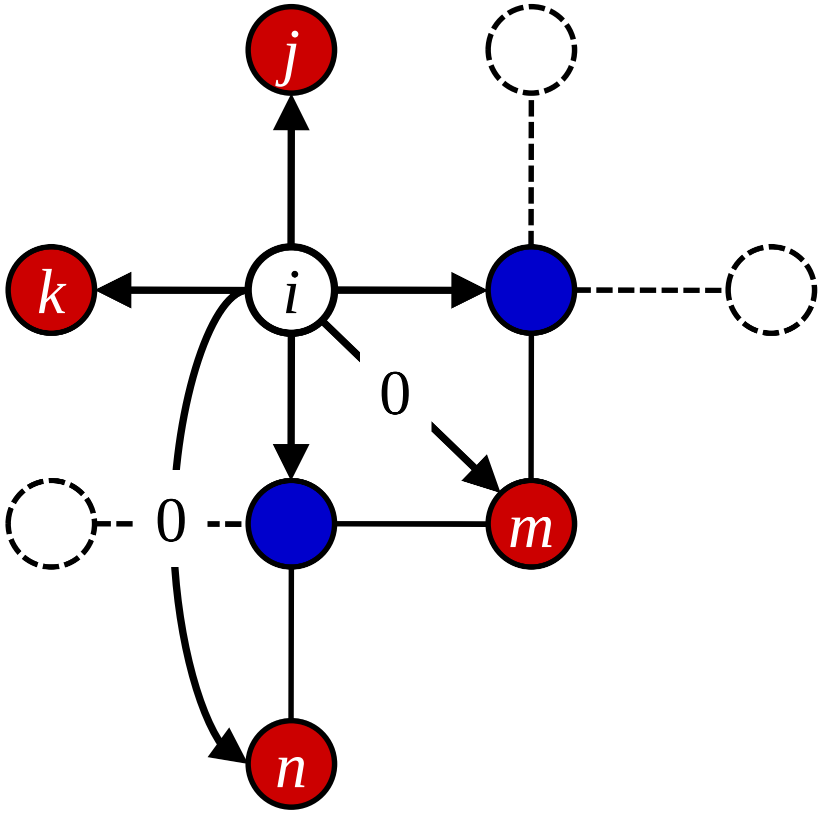
- PMIS is parallel and eliminates the second pass, which can lead to the following scenarios:



- Want above *i*-points to interpolate from both *C*-points
- Long-range (distance two) interpolation!

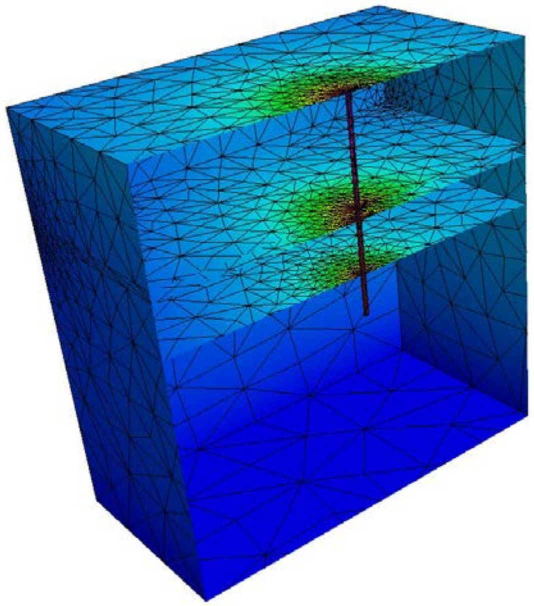
One possibility for long-range interpolation is extended interpolation

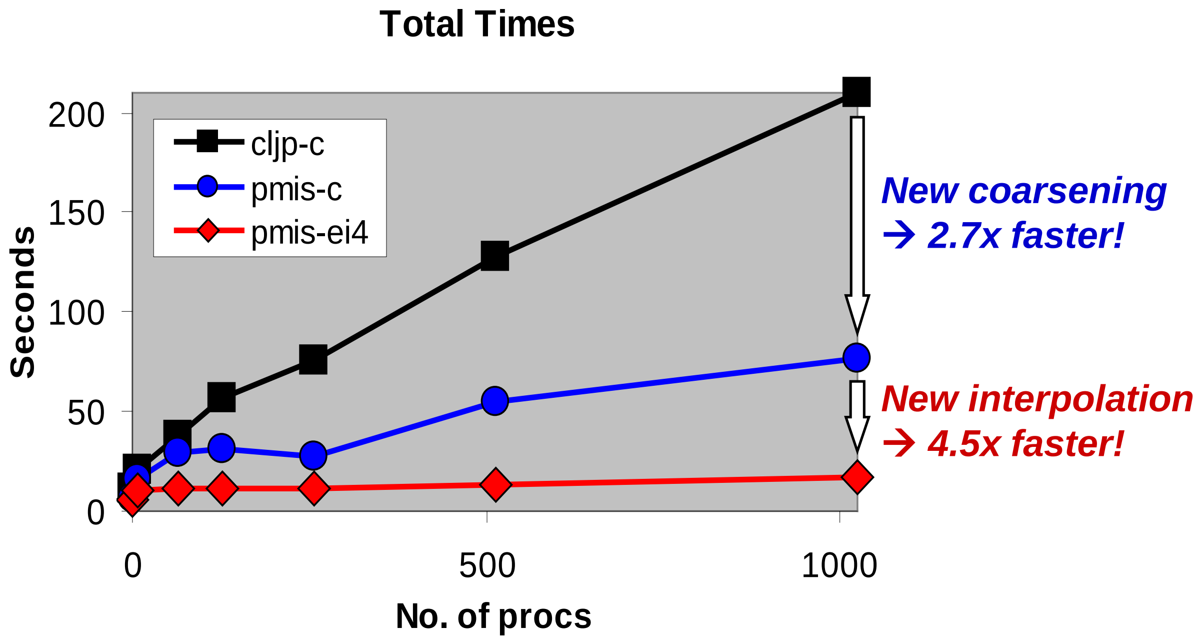
- C-AMG: $C_i = \{j,k\}$
- Long-range: $C_i = \{j,k,m,n\}$
- Extended interpolation – apply C-AMG interpolation to an extended stencil
- Extended+i interpolation is the same as extended, but also collapses to point *i*
- Improves overall quality



New parallel coarsening and long-range interpolation methods improve scalability

- Unstructured 3D problem with material discontinuities
- About 90K unknowns per processor on MCR (Linux cluster)
- AMG - GMRES(10)





Agglomeration of coarsest grid

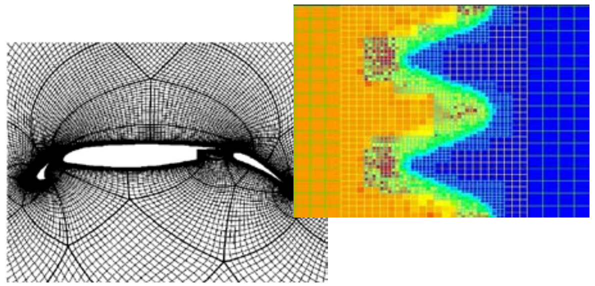
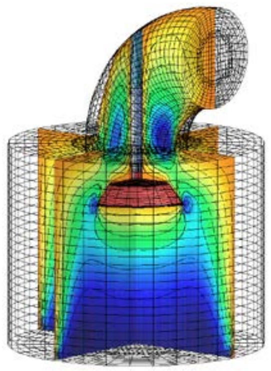
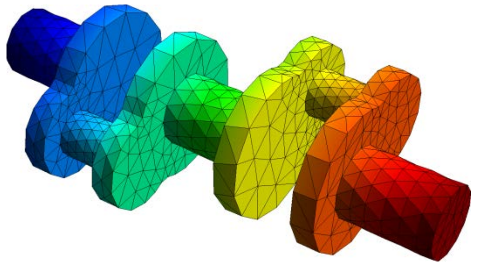
- One technique that can be useful is to gather (agglomerate) the system matrix at some coarse level to a single processor
- This costs O(log P) communications, but so does simply continuing the V-cycle
- This helps when the matrix rows have grown in complexity
  - Avoids cost of communicating with many neighbors

Multigrid Software Design

hypre
high performance preconditioners

Simulation codes present a wide array of challenges for scalable linear solver libraries

- Different applications
  - Diffusion, elasticity, magnetohydrodynamics (MHD)
- Different discretizations and meshes
  - Structured, block-structured, structured AMR, overset, unstructured

- Different languages – C, C++, Fortran
- Different programming models – MPI, OpenMP
- Scalability beyond 100,000 processors!

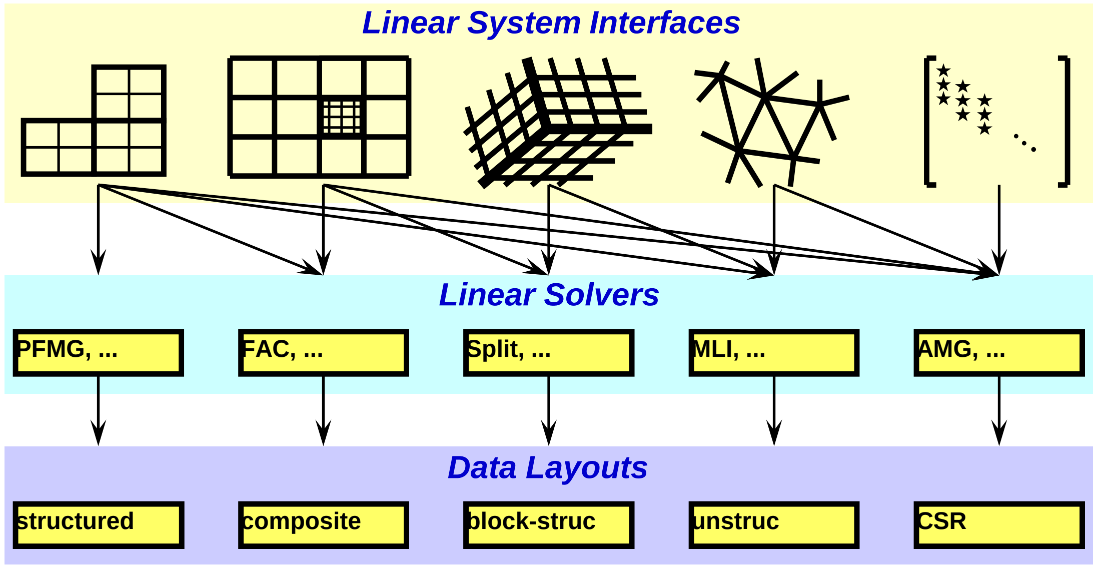
Challenge: Software design

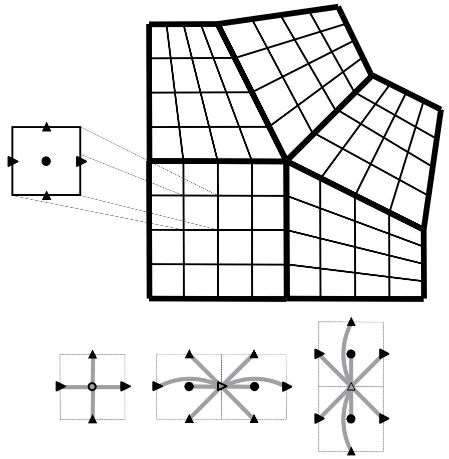
- The “best” solver for a given application usually takes advantage of the setting
  - structured grids, constant coefficients, FE discretization, etc.

- Traditional linear solver libraries take in only generic matrix-vector information

- How do we supply these “best solvers” in library form?

Unique software interfaces in *hypre* provide efficient solvers not available elsewhere





Block-structured grid with 3 variable types and 3 discretization stencils

- Example: *hypre*'s interface for semi-structured grids
  — Based on "grids" and either "stencils" or "finite elements" (new)
  — Allows for specialized solvers for structured AMR
  — Also provides for more general solvers like *AMG*

Challenge: Parallel implementation

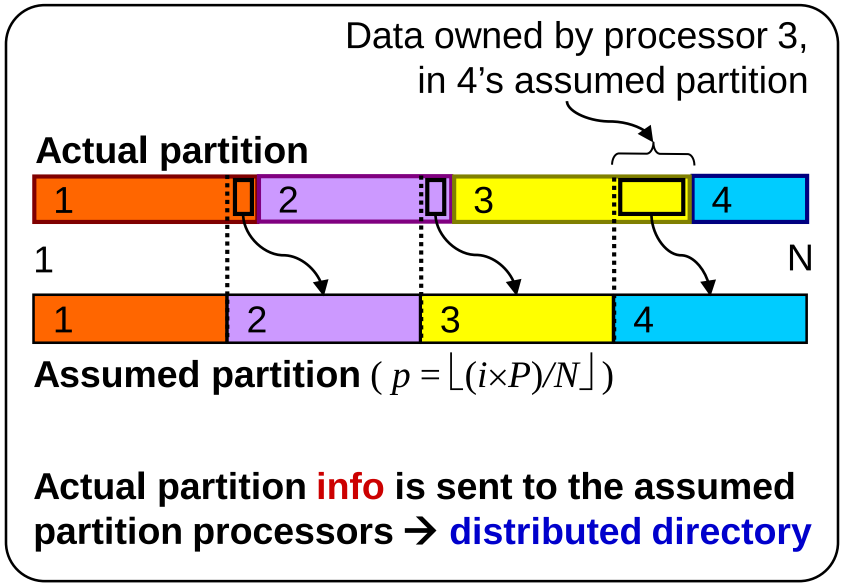
- Simple algorithms can be used for modest numbers of processors (< 100)
  - e.g., store O(P) data and do $O(P^2)$ computations to determine send/receive patterns

- On large numbers of processors (1K – 10K), algorithms get more complex
  - e.g., store O(P) data and do O(P) computations to determine send/receive patterns

- What about 100K – 1B processors?

Assumed partition (AP) algorithm enables scaling to 100K+ processors

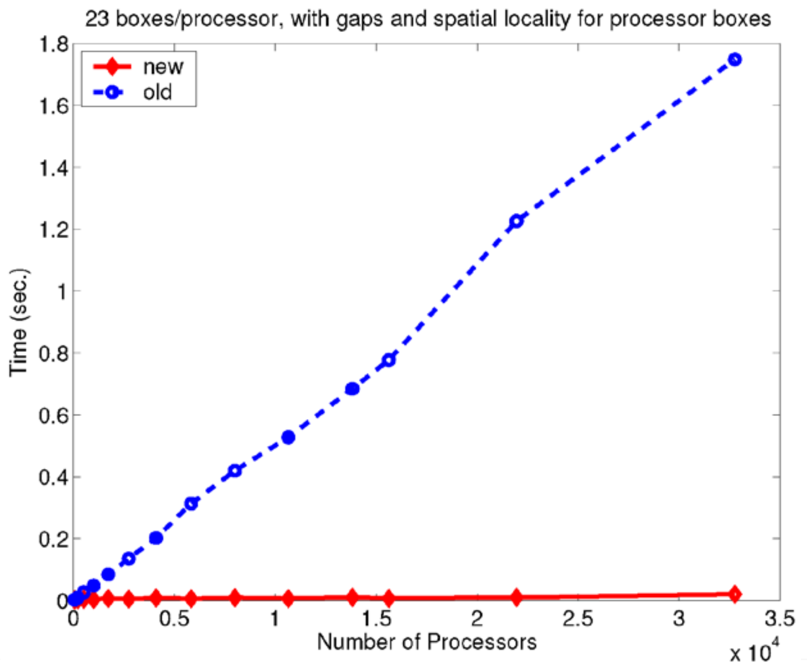
- Answering global distribution questions previously required $O(P)$ storage & computations
- On BG/L, $O(P)$ storage may not be possible
- AP algorithm requires
  - $O(1)$ storage
  - $O(\log P)$ computations
- Default approach in *hypre*
- AP has general applicability beyond *hypre*

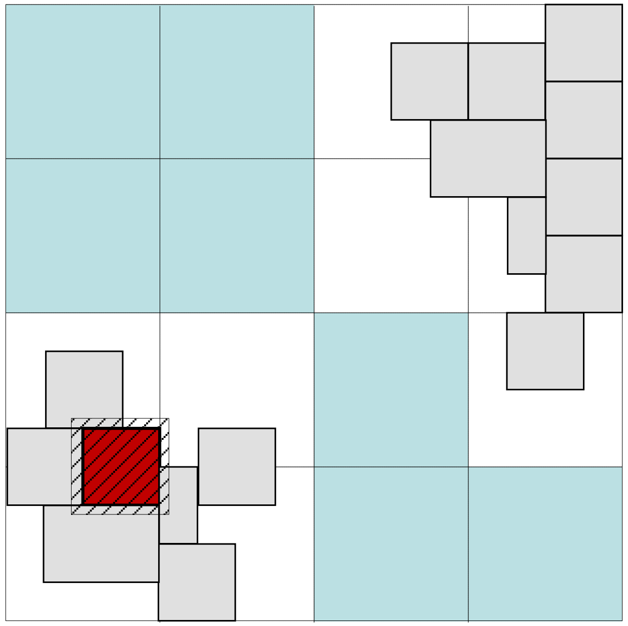

Data owned by processor 3, in 4's assumed partition
Actual partition
1
2
3
4
1
N
1
2
3
4
Assumed partition ($p = \lfloor (i \times P)/N \rfloor$)

Actual partition info is sent to the assumed partition processors → distributed directory

Assumed partition (AP) algorithm is more challenging for structured AMR grids

- AMR can produce grids with “gaps”
- Our AP function accounts for these gaps for scalability
- Demonstrated on 32K procs of BG/L



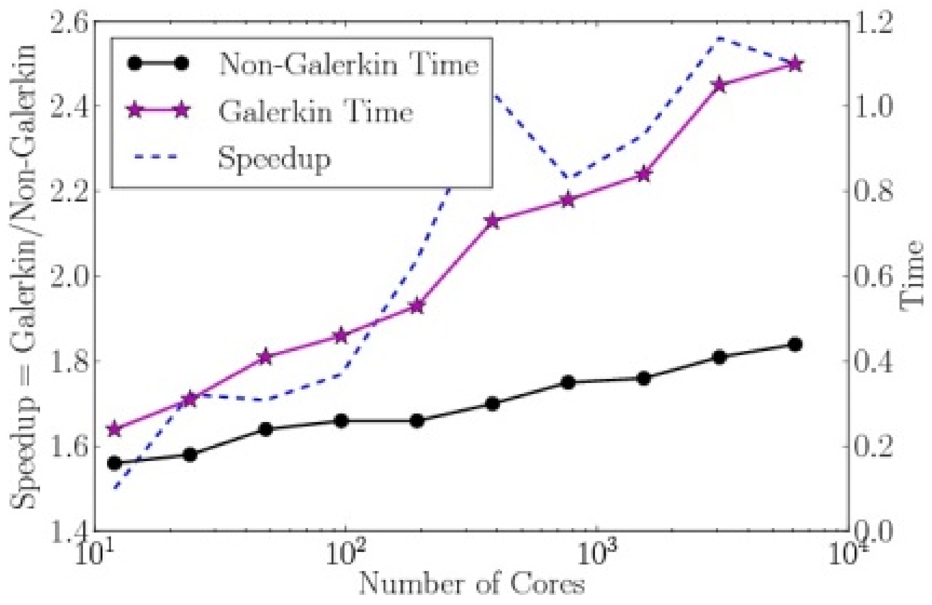


Simple, naïve AP function leaves processors with empty partitions

Some Recent Research Topics

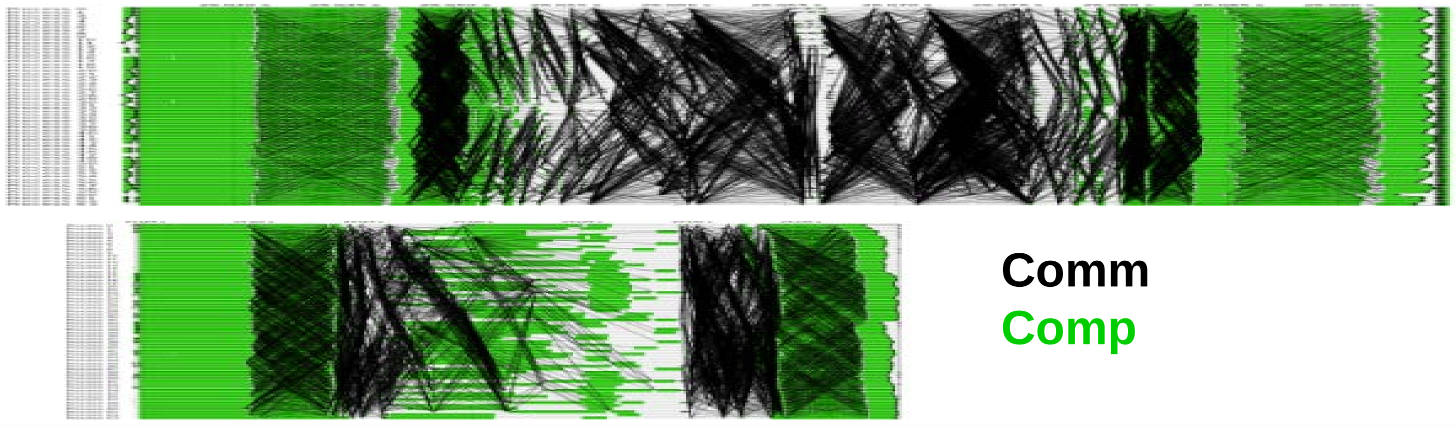
Recent spatial-MG research focuses on reducing parallel communication costs (1)

- Non-Galerkin AMG replaces the usual coarse-grid operators with sparser ones
  - Speedups from 1.2x - 2.4x over existing AMG
  - In *hypre* 2.10.0b



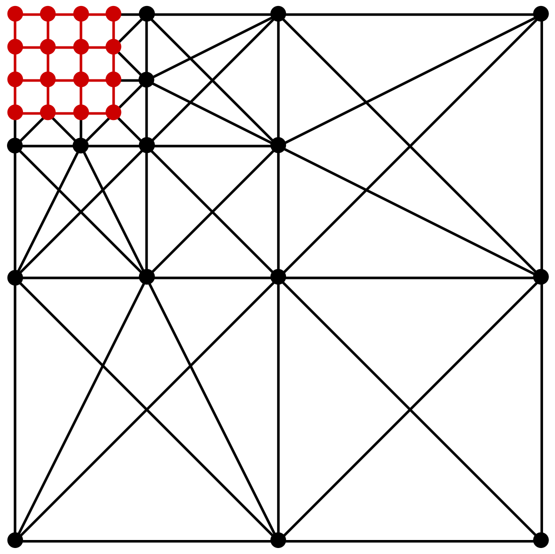
Recent spatial-MG research focuses on reducing parallel communication costs (2)

- Mult-additive AMG exploits a theoretical identity to inherit the parallelization benefits of additive methods and the convergence properties of multiplicative
  - Additive MG is good at overlapping communication and computation, but converges slower than multiplicative MG
  - Speedups of 2x over existing AMG
  - In *hypre* 2.10.0b



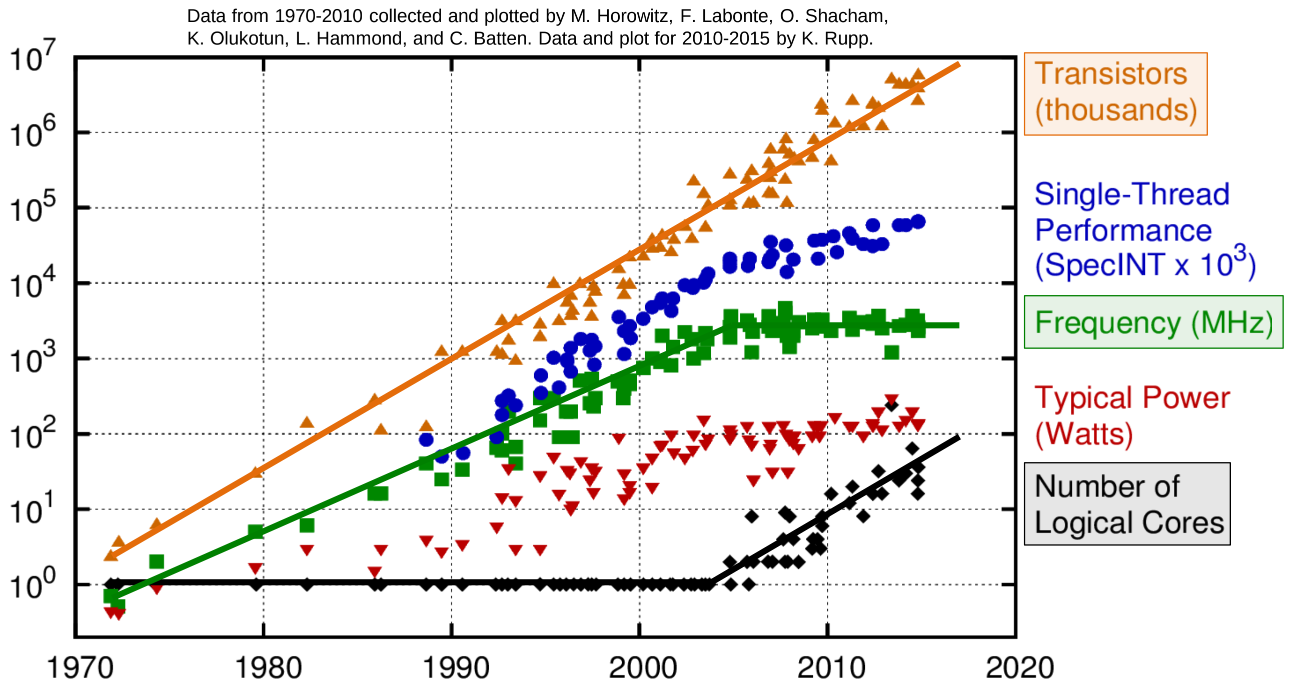
Recent spatial-MG research focuses on reducing parallel communication costs (3)

- AMG domain decomposition (AMG-DD) employs cheap global problems to speed up convergence
  - Constructs problems algebraically from an existing method
  - Potential for FMG convergence with only log N latency (vs $\log^2$ N)!
  - Implementing parallel code



Parallel Time Integration (Parallel Multigrid in Time)

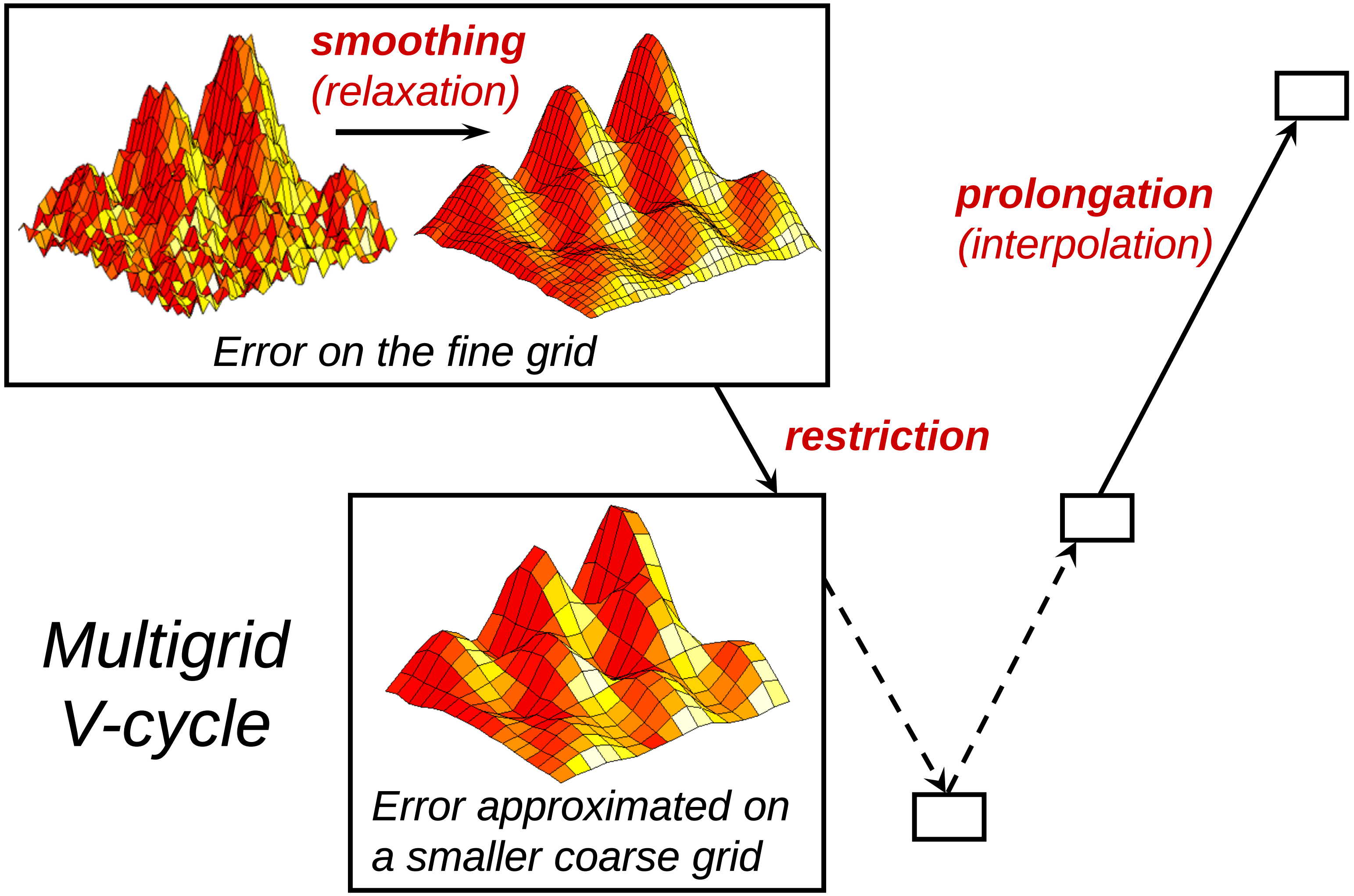
Parallel time integration is a major paradigm shift driven by hardware design realities



- Architecture trend: flat clock rates, more concurrency
  - Traditional time stepping is becoming a sequential bottleneck

- Continued advancement in scientific simulation will require algorithms that are parallel in time

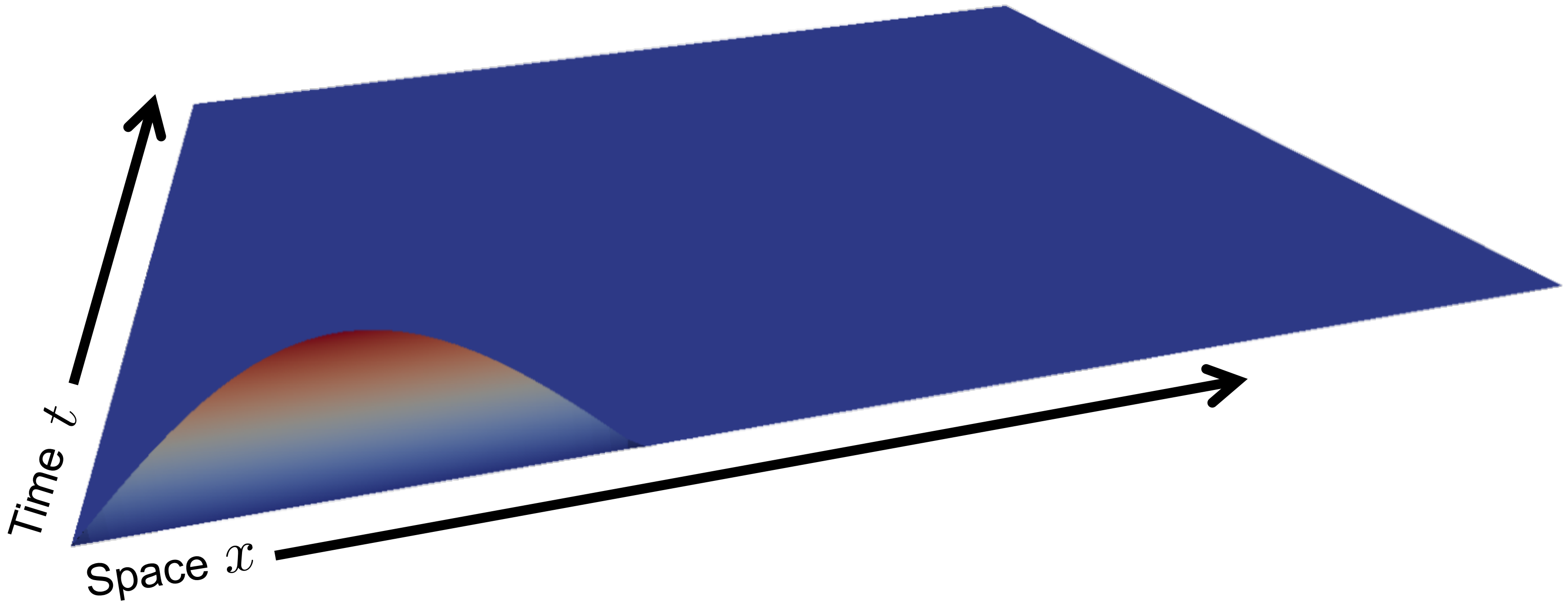
One approach for parallel-in-time: apply multigrid ideas to the (space-)time dimension



Multigrid V-cycle

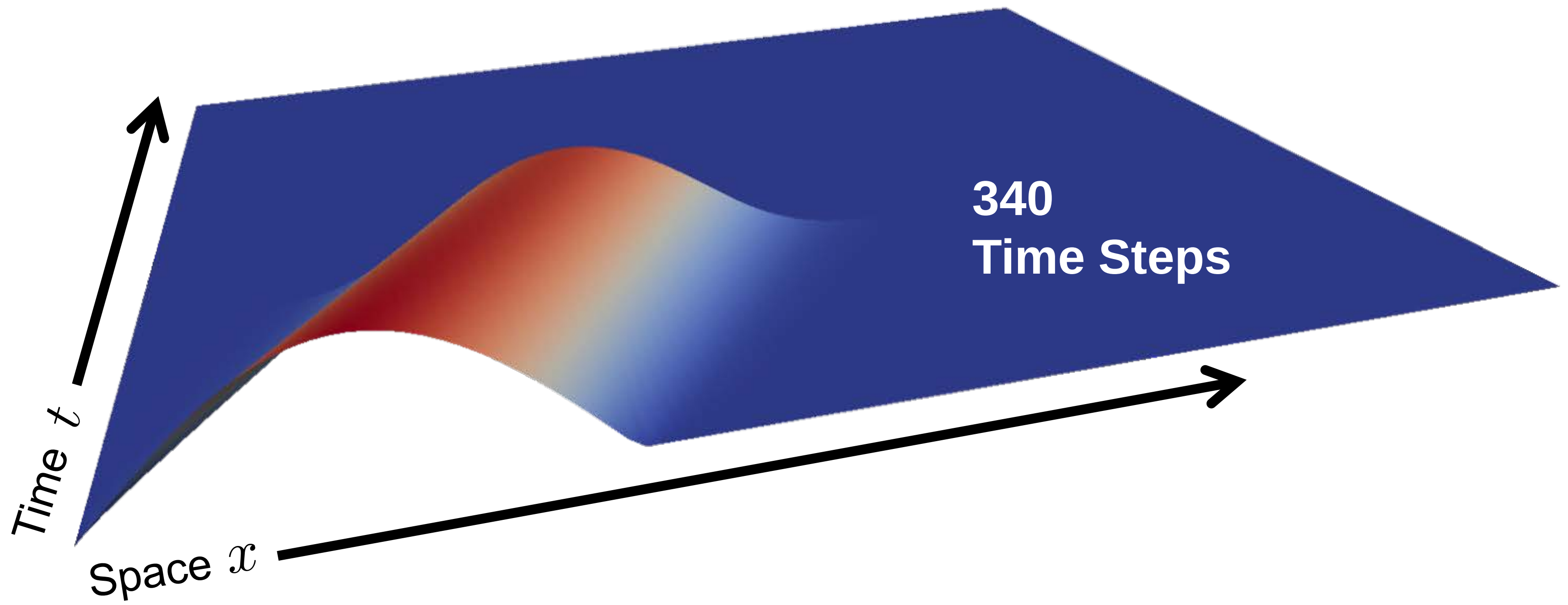
Time stepping is sequential

- Simple advection equation, $u_t = -cu_x$
- Initial condition is a wave



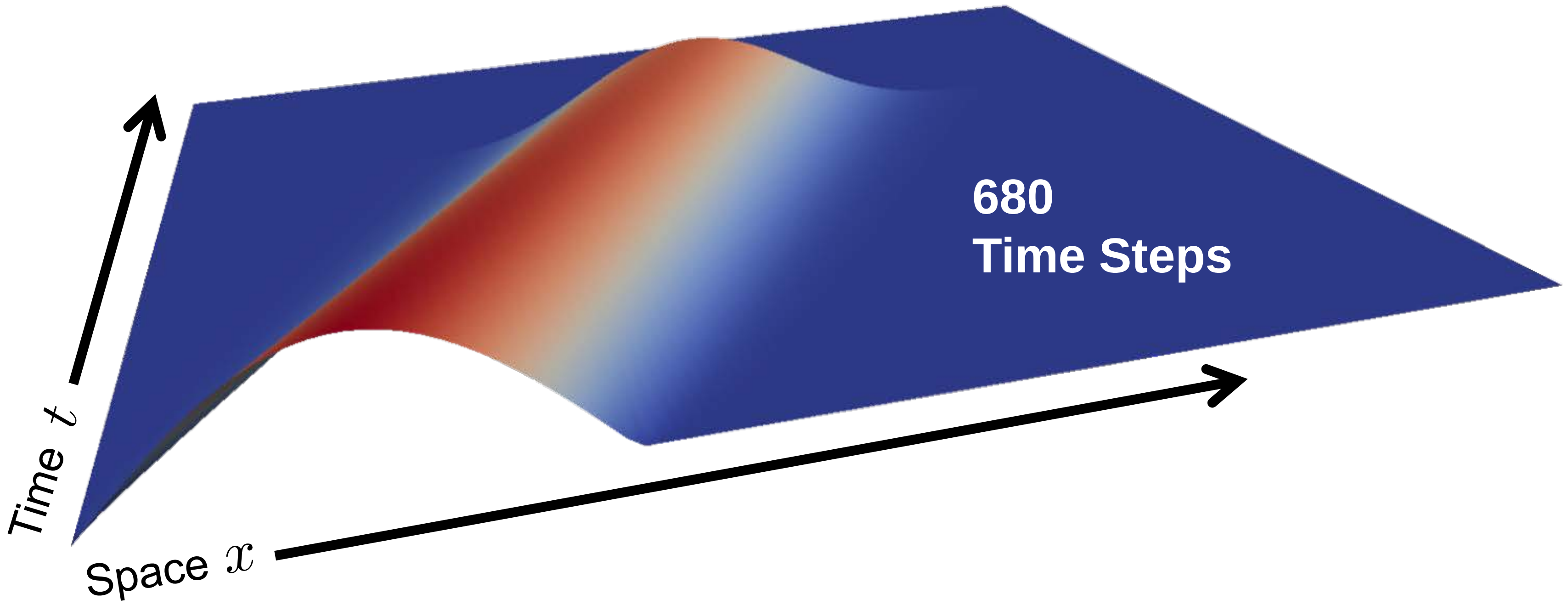
Time stepping is sequential

- Simple advection equation, $u_t = -cu_x$
- Wave propagates serially through space



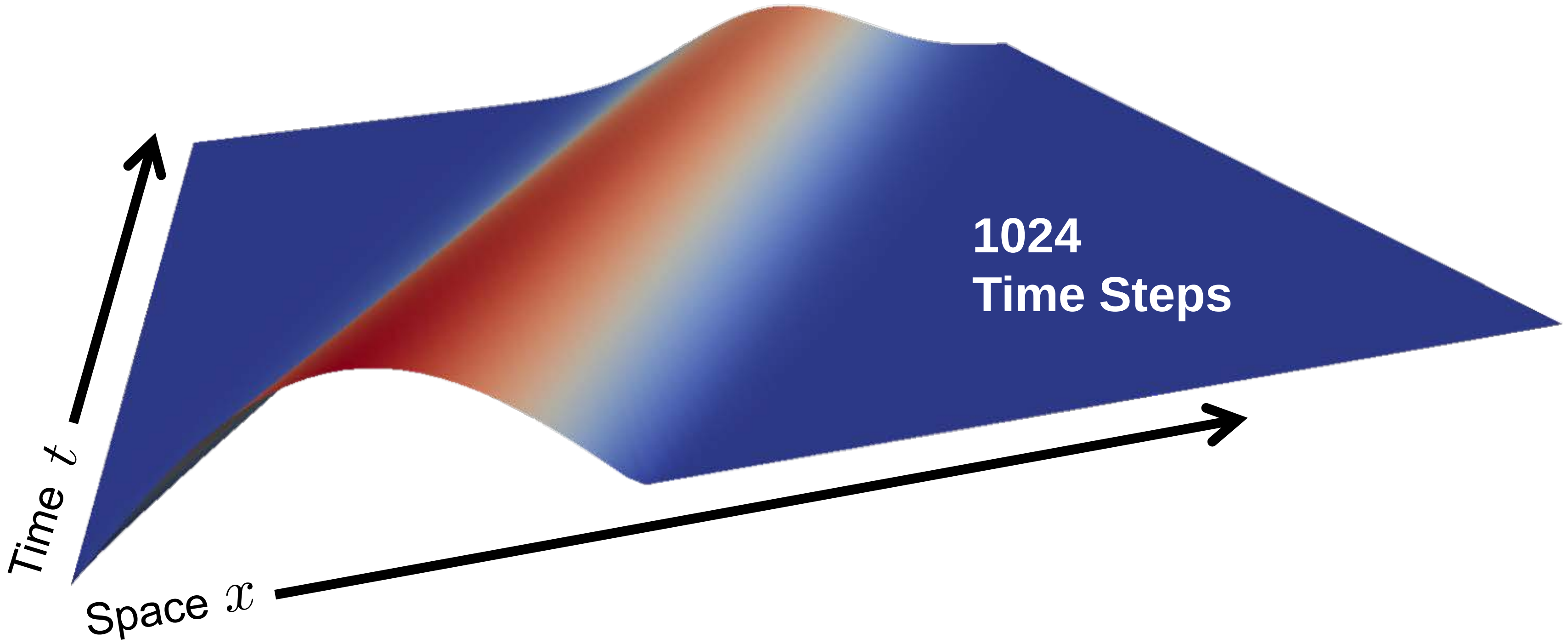
Time stepping is sequential

- Simple advection equation, $u_t = -cu_x$
- Wave propagates serially through space



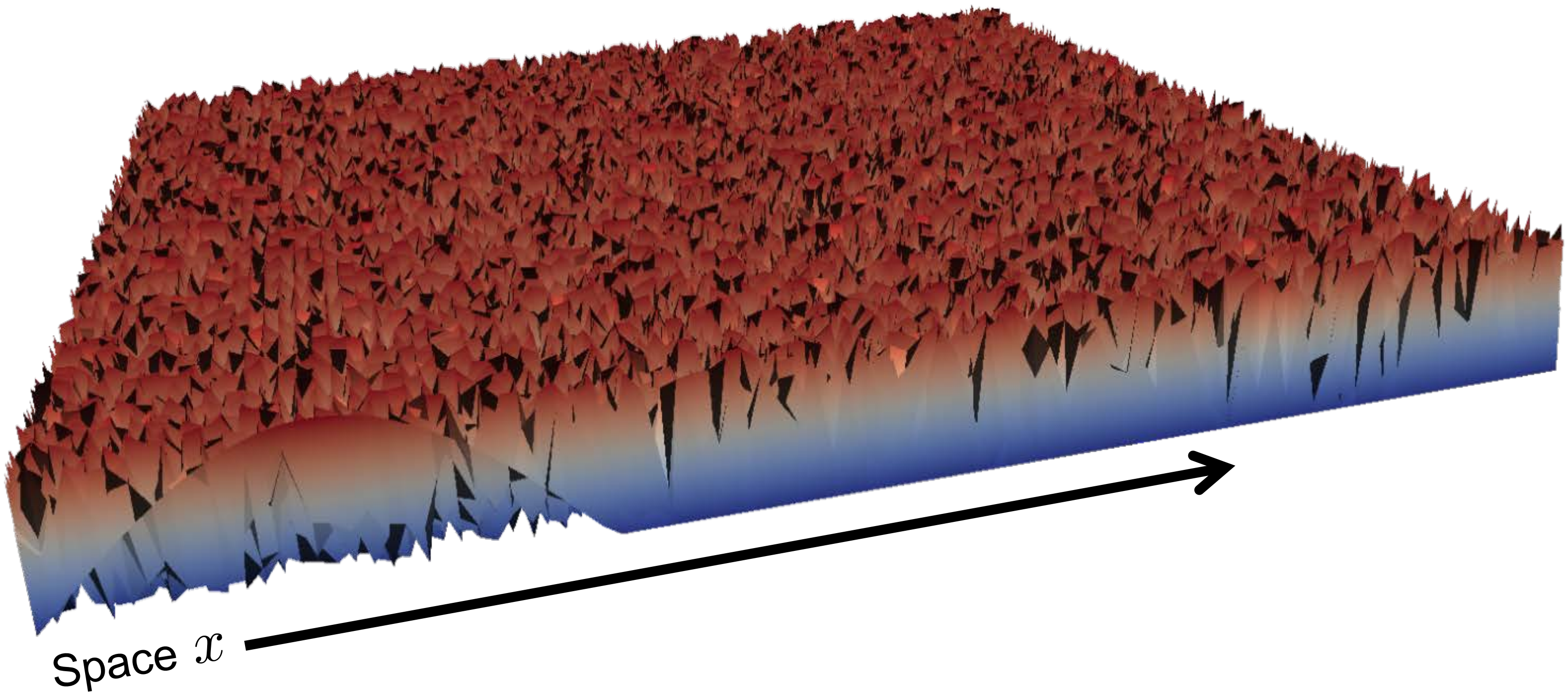
Time stepping is sequential

- Simple advection equation, $u_t = -cu_x$
- Wave propagates serially through space



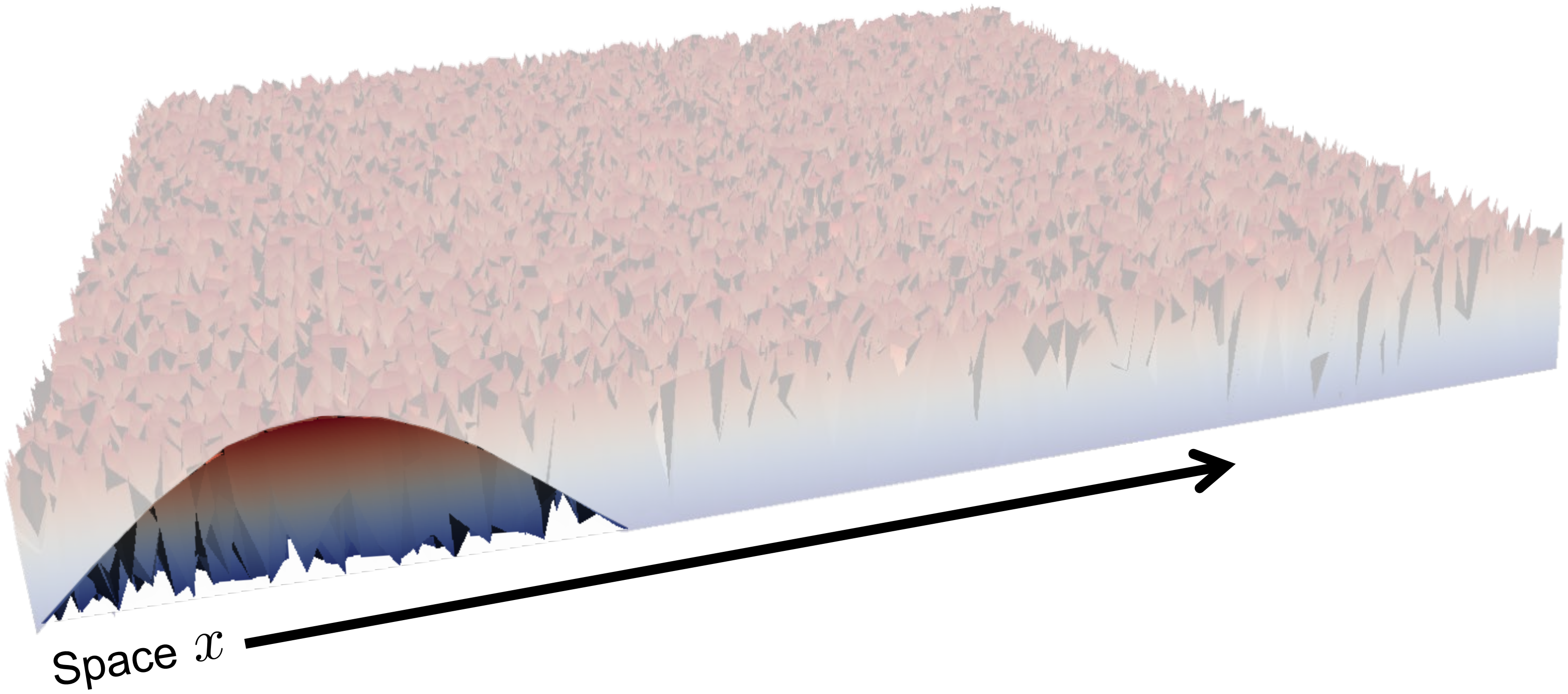
Multigrid-in-time converges to the serial space-time solution in parallel

- Simple advection equation, $u_t = -cu_x$
- Random initial space-time guess (only for illustration)



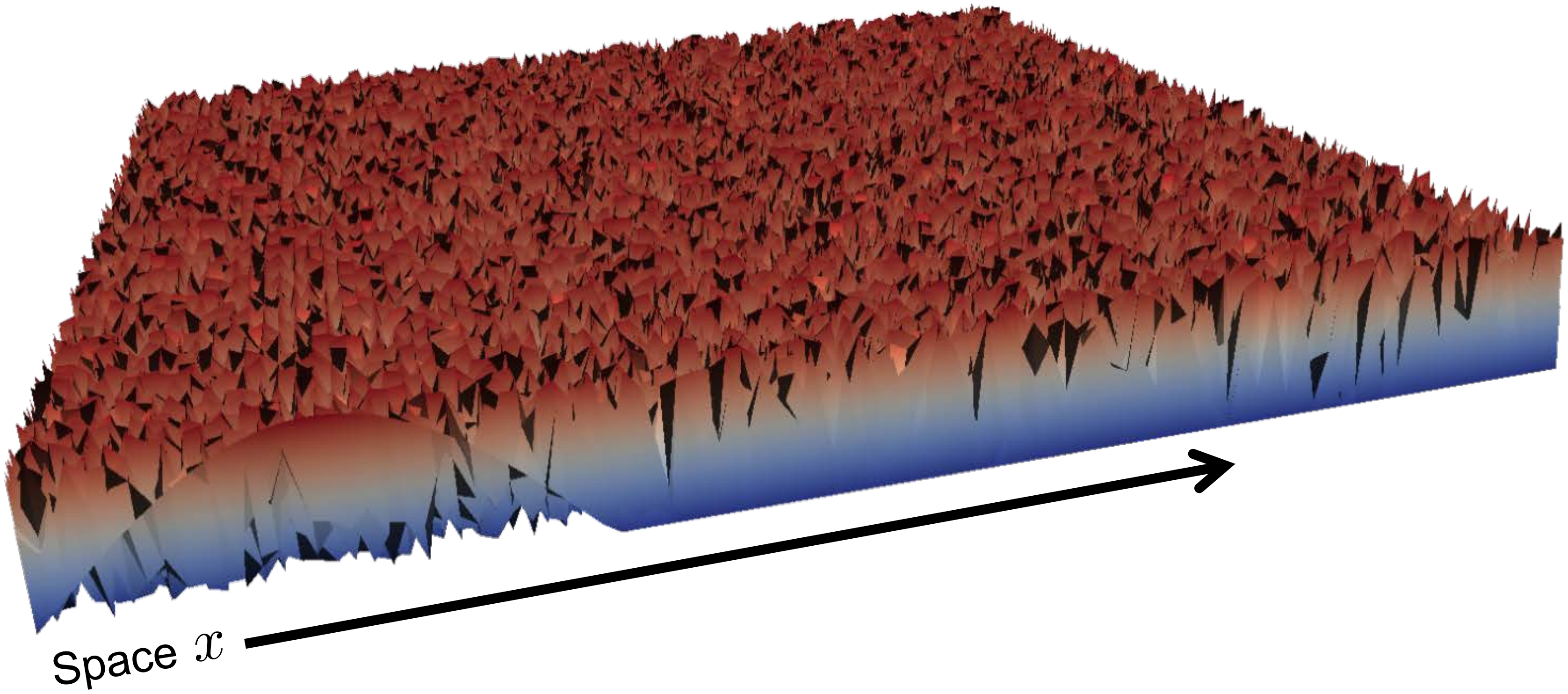
Multigrid-in-time converges to the serial space-time solution in parallel

- Simple advection equation, $u_t = -cu_x$
- Initial condition is a wave



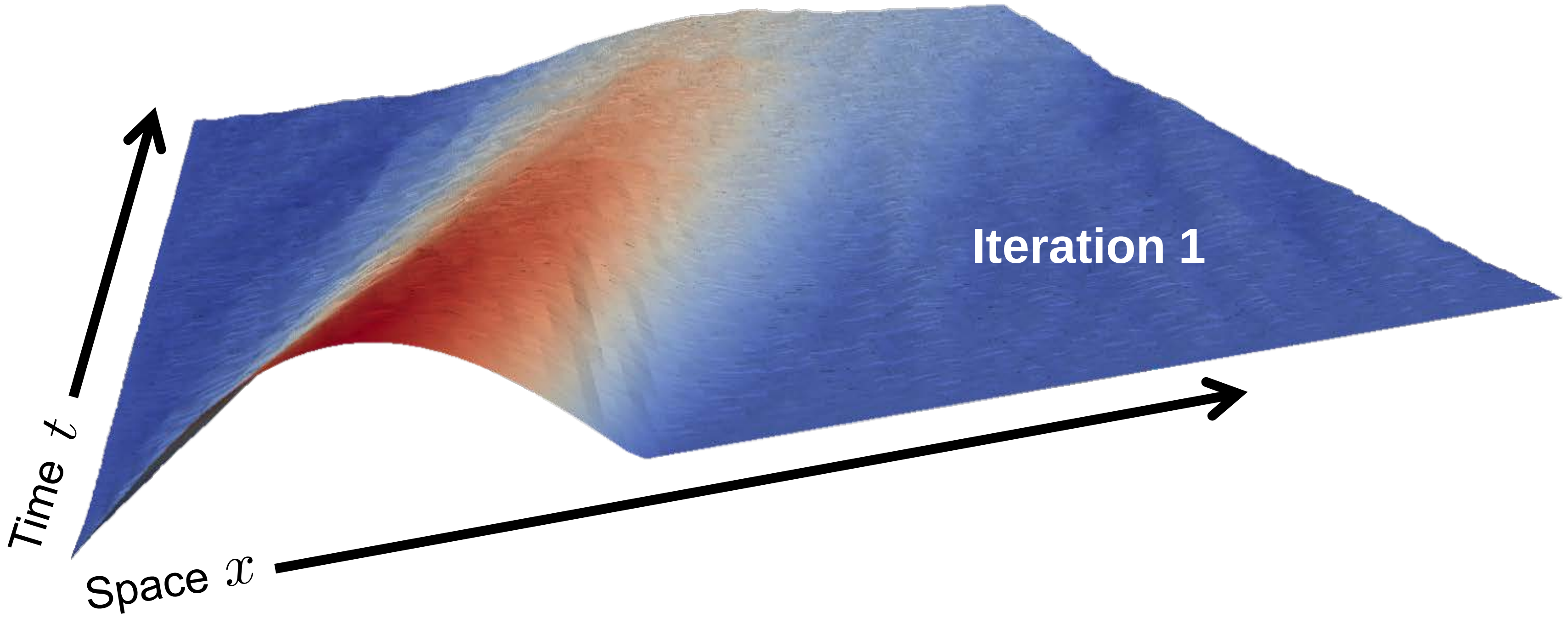
Multigrid-in-time converges to the serial space-time solution in parallel

- Simple advection equation, $u_t = -cu_x$
- Initial condition is a wave



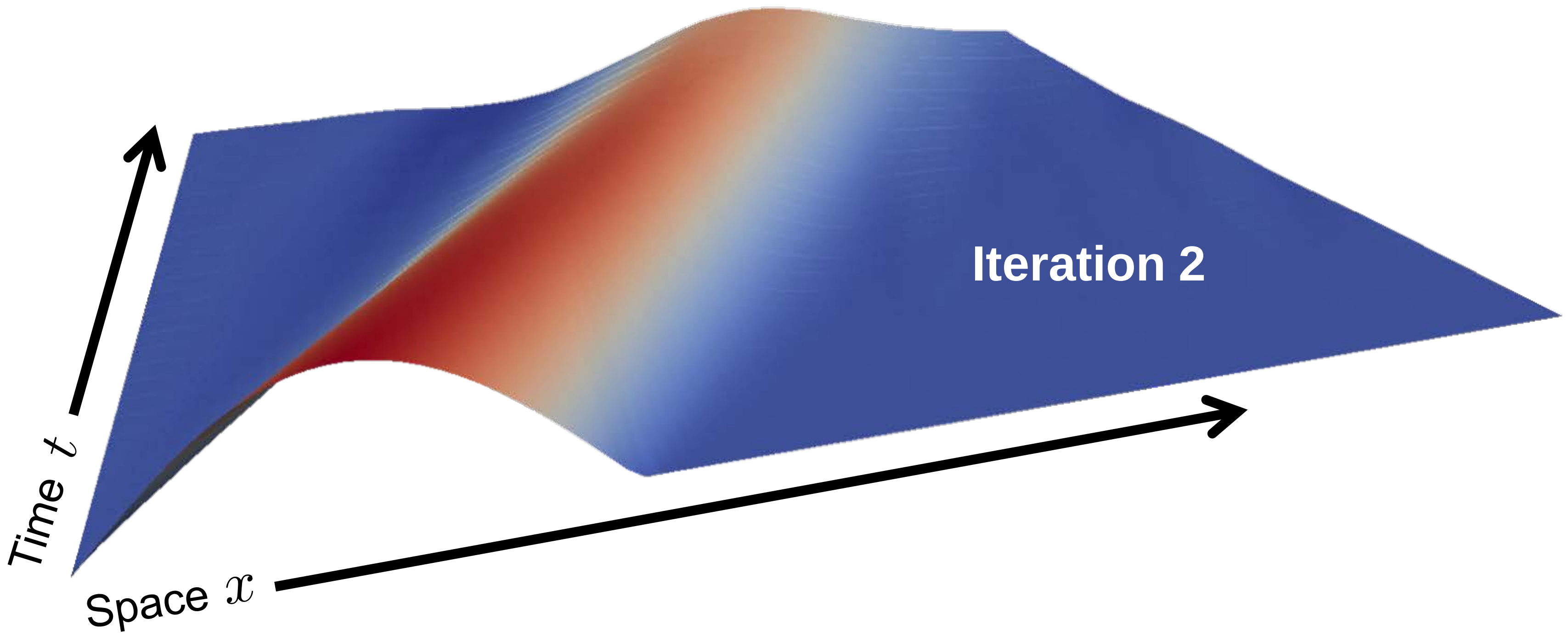
Multigrid-in-time converges to the serial space-time solution in parallel

- Simple advection equation, $u_t = -cu_x$
- Multilevel structure allows for fast data propagation



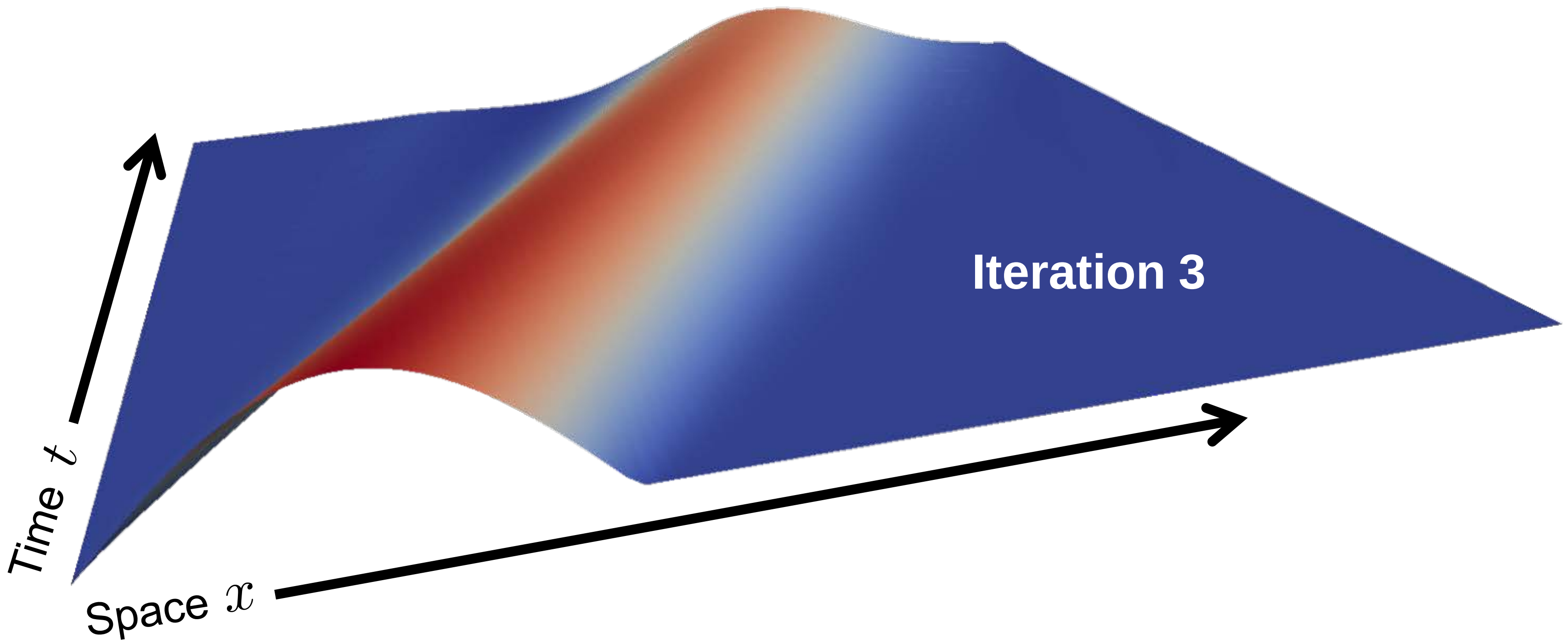
Multigrid-in-time converges to the serial space-time solution in parallel

- Simple advection equation, $u_t = -cu_x$
- Multilevel structure allows for fast data propagation



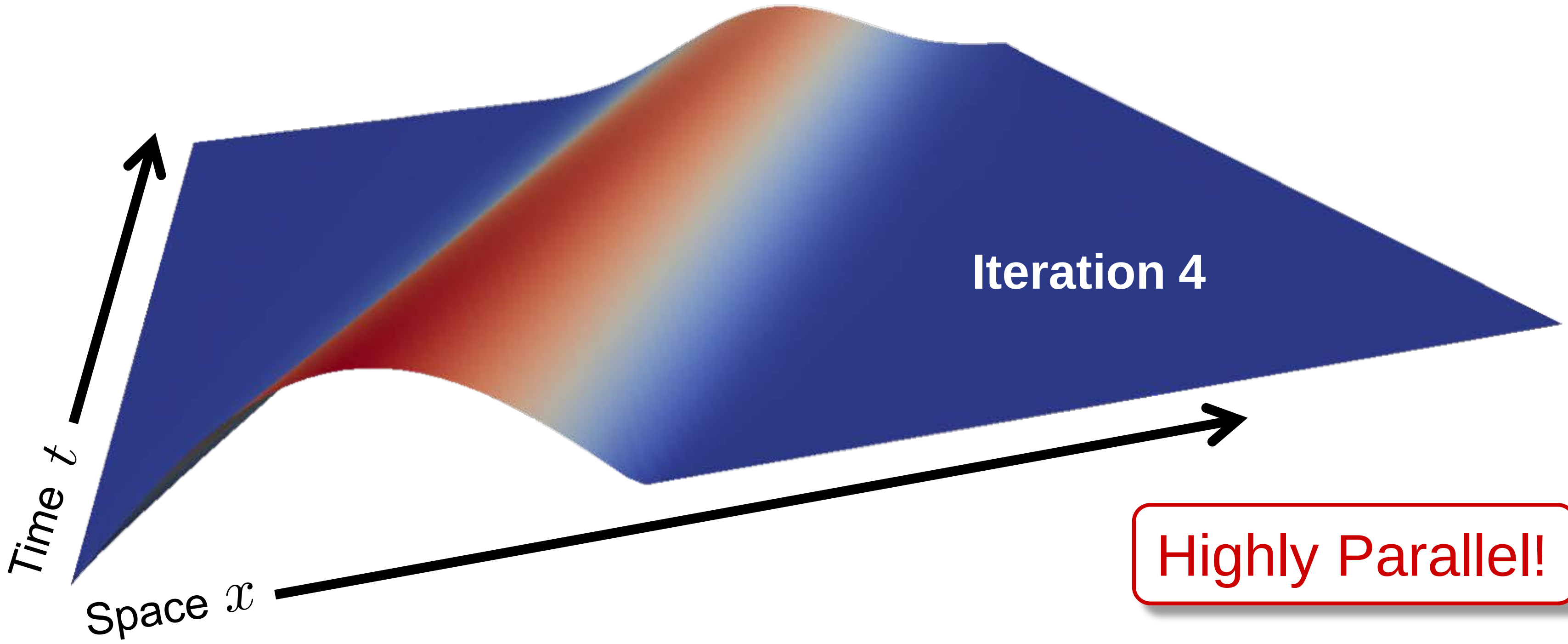
Multigrid-in-time converges to the serial space-time solution in parallel

- Simple advection equation, $u_t = -cu_x$
- Multilevel structure allows for fast data propagation



Multigrid-in-time converges to the serial space-time solution in parallel

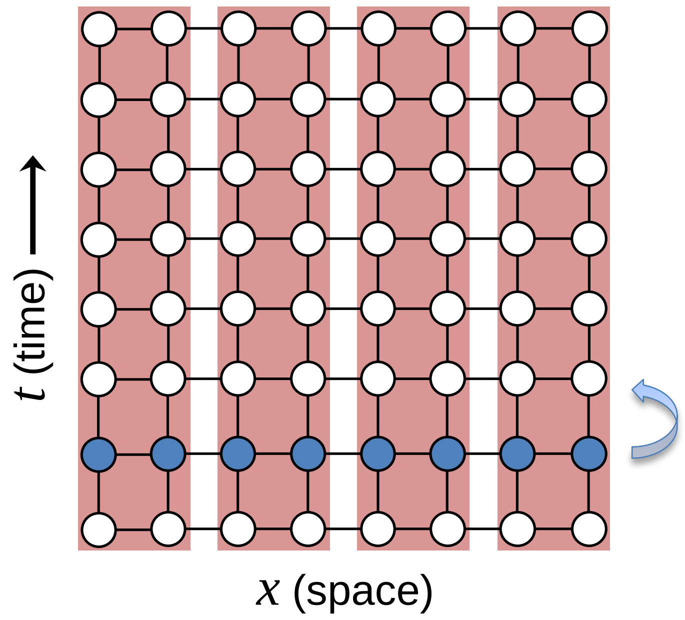
- Simple advection equation, $u_t = -cu_x$
- Already very close to the solution



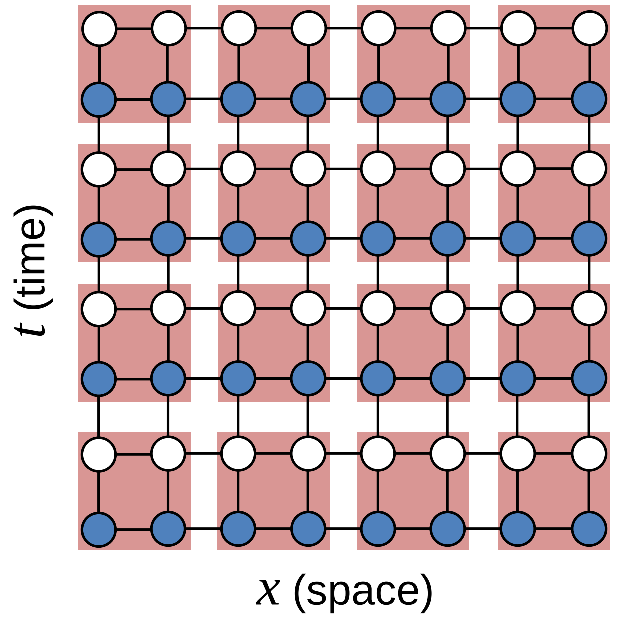
Highly Parallel!

Significantly more parallel resources can be exploited with multigrid in time

Serial time stepping



Multigrid in time



- ⊖ Parallelize in space only
- ⊕ Store only one time step

- ⊕ Parallelize in space and time
- ⊖ Store several time steps

It's useful to view the time integration problem as a large block matrix system

- General one-step method

$$\boldsymbol{u}_i = \Phi_i(\boldsymbol{u}_{i-1}) + \boldsymbol{g}_i, \quad i = 1, 2, \ldots, N$$

- Linear setting: time marching = block forward solve
  - $O(N)$ direct method, but sequential

$$A\mathbf{u} \equiv \begin{pmatrix} I & & & \\ -\Phi & I & & \\ & \ddots & \ddots & \\ & & -\Phi & I \end{pmatrix} \begin{pmatrix} \boldsymbol{u}_0 \\ \boldsymbol{u}_1 \\ \vdots \\ \boldsymbol{u}_N \end{pmatrix} = \begin{pmatrix} \boldsymbol{g}_0 \\ \boldsymbol{g}_1 \\ \vdots \\ \boldsymbol{g}_N \end{pmatrix} \equiv \mathbf{g}$$

- The MGRIT approach is based on multigrid reduction (MGR) methods (approximate cyclic reduction)
  - $O(N)$ iterative method, but highly parallel
  - Non-intrusive – user only provides time integrator Φ

The MGRIT approach builds as much as possible on existing codes and technologies

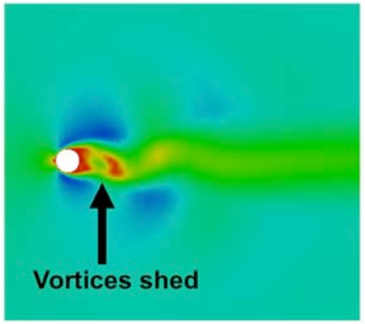
- Combines algorithm development, theory, and software proof-of-principle

- Goal: Create concurrency in the time dimension
- Non-intrusive, with unchanged time discretization
  - Implicit, explicit, multistep, multistage, …
- Converges to same solution as sequential time stepping
- Extends to nonlinear problems with FAS formulation

$$\begin{pmatrix} I & & & \\ -\Phi & I & & \\ & \ddots & \ddots & \\ & & -\Phi & I \end{pmatrix}$$

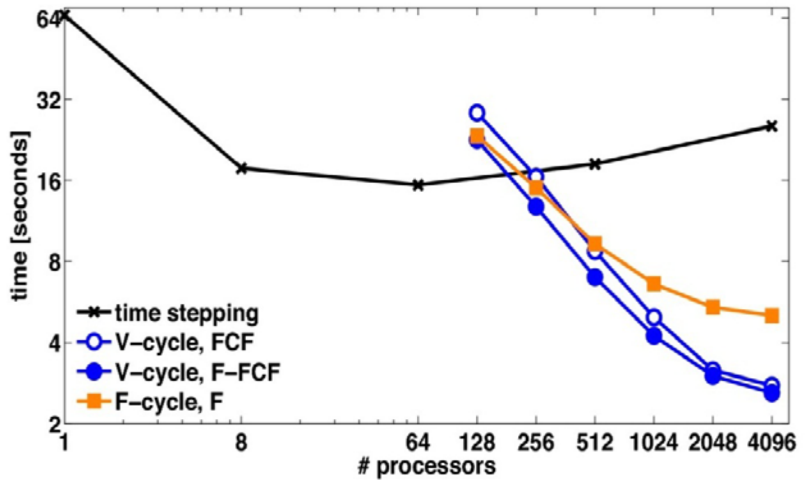
- XBraid is our open source implementation of MGRIT
  - User defines two objects and writes several wrapper routines (Step)
  - Only stores *C*-points to minimize storage

- Many active research topics, applications, and codes
  - Adaptivity in space and time, moving meshes, BDF methods, …
  - Linear/nonlinear diffusion, advection, fluids, power grid, elasticity, …
  - MFEM, hypre, Strand2D, Cart3D, LifeV, CHeart, GridDyn



Parallel speedups can be significant, but in an unconventional way

- Parallel time integration is driven entirely by hardware
  - Time stepping is already *O*(*N*)
- Useful only beyond some scale
  - There is a crossover point
  - Sometimes need significantly more parallelism just to break even
  - Achievable efficiency is determined by the space-time discretization and degree of intrusiveness



3D Heat Equation: 33^3 x 4097, 8 procs in space, 6x speedup

- The more time steps, the more speedup potential
  - Applications that require lots of time steps benefit first
  - Speedups (so far) up to 49x on 100K cores

Nearly 50 years of research exists but has only scratched the surface

- Earliest work goes back to 1964 by Nievergelt
  - Led to multiple shooting methods, Keller (1968)
- Space-time multigrid methods for parabolic problems
  - Hackbusch (1984); Horton (1992); Horton and Vandewalle (1995)
  - The latter is one of the first optimal & fully parallelizable methods to date
- Parareal was introduced by Lions, Maday, and Turinici in 2001
  - Probably the most widely studied method
  - Gander and Vandewalle (2007) show that parareal is a two-level FAS multigrid method
- Discretization specific work includes
  - Minion, Williams (2008, 2010) – PFASST, spectral deferred correction, FAS
  - DeSterck, Manteuffel, McCormick, Olson (2004, 2006) – FOSLS
- Research on these methods continues to ramp up!
  - Ruprecht, Krause, Speck, Emmett, Langer, … this is not an exhaustive list
- Recent review: Gander (2015), "50 years of time parallel time integration"

CASC

Center for Applied Scientific Computing

Lawrence Livermore National Laboratory

Thank You!

Disclaimer
This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.