HPGMG-FV

Samuel Williams

Lawrence Berkeley National Laboratory

SWWilliams@lbl.gov

Outline

- Introduction to Multigrid
  - Cell-Centered Multigrid
  - U/V-Cycles
  - issues in AMR
  - Multigrid Performance Challenges
- HPGMG-FV Overview
- HPGMG-FV Deep dive
- CoDesign Questions

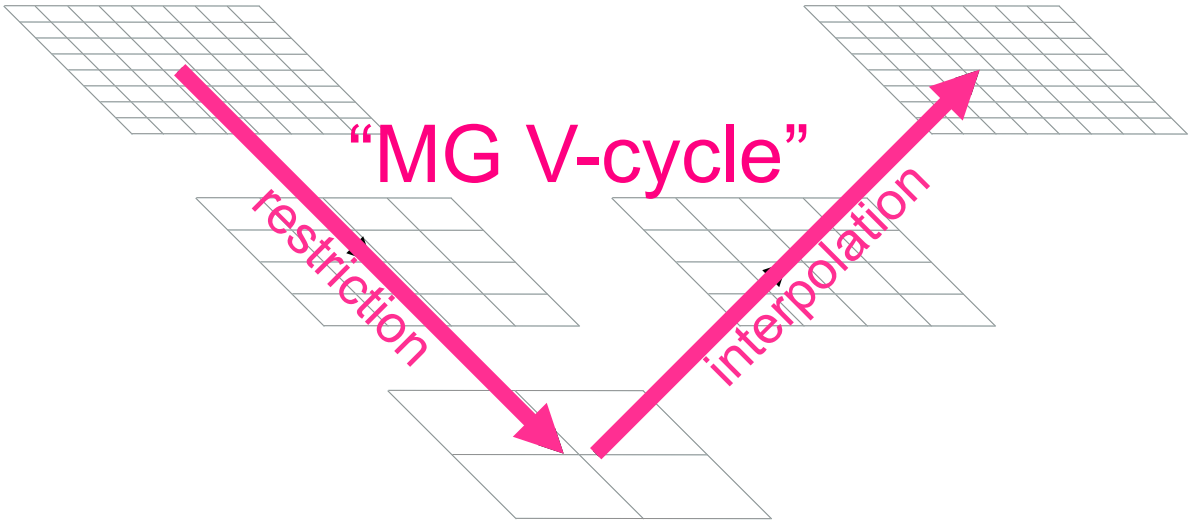
Introduction

- Linear Solvers (Ax=b) are ubiquitous in scientific computing

- Multigrid solves elliptic PDEs (Ax=b) using a hierarchical (recursive) approach.
  - solution (correction) to hard problem is expressed in terms of solution to an easier problem
  - Provides **O(N) computational complexity** where N is number of unknowns
  - AMR applications like LMC (Combustion Co-Design Center) might perform a MG solve for **every chemical species** (NH_4, CO_2, …) on **every AMR level**.
  - Performance (setup time, solve time, scalability, and memory usage) can be critical

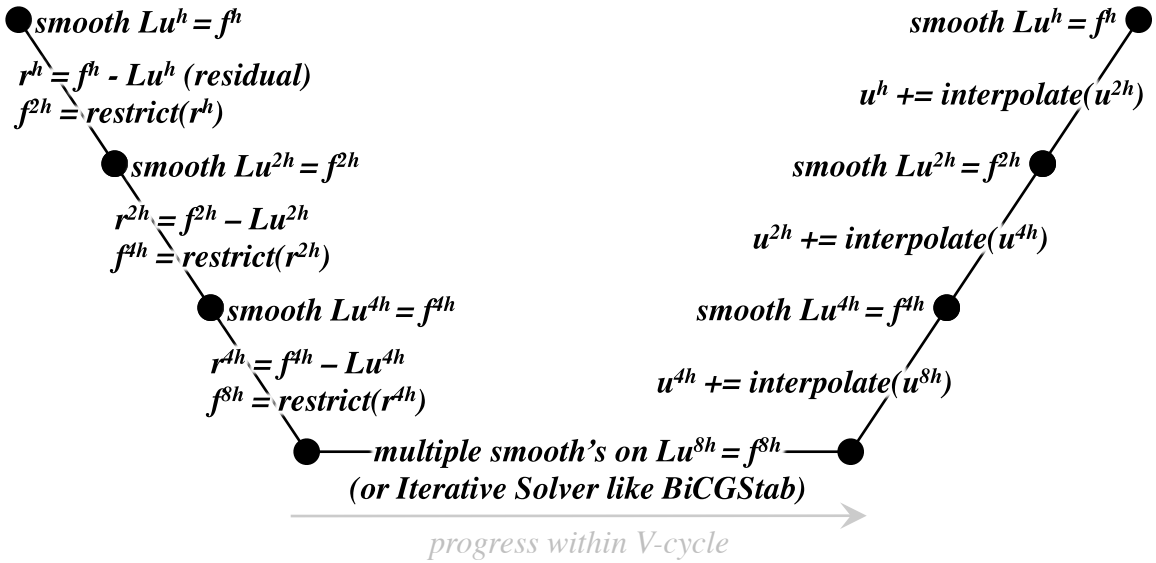
Geometric Multigrid

- Geometric Multigrid is specialization in which the linear operator (A) is simply a **stencil on a structured grid** (i.e. *matrix-free*)
- Inter-grid operations are recast in terms of stencils based on the underlying numerical method (e.g. cell-centered finite volume)
- Extremely fast/efficient…
  - O(N) computational complexity (#flops)
  - O(N) DRAM data movement (#bytes)
  - $O(N^{0.66})$ MPI data movement



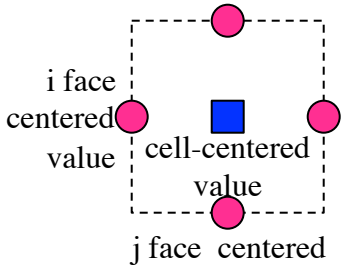
Geometric Multigrid

- Geometric Multigrid is specialization in which the linear operator (A) is simply a **stencil on a structured grid** (i.e. *matrix-free*)
- Inter-grid operations are recast in terms of stencils based on the underlying numerical method (e.g. cell-centered finite volume)
- Extremely fast/efficient…
  - O(N) computational complexity (#flop's)
  - O(N) DRAM data movement
  - O($N^{0.66}$) MPI data movement

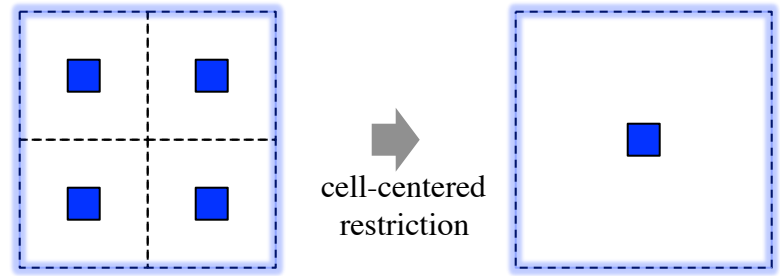


Cell-Centered MG

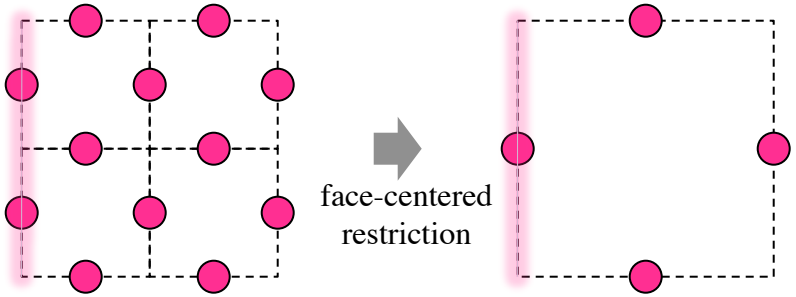
- Values can represent…
  - cell averages (cell-centered)
  - face averages (face-centered)



- Restriction/Prolongation can be either cell- or face-centered.
- In piecewise constant restriction, coarse grid elements are the average value of the region covered by fine grid elements

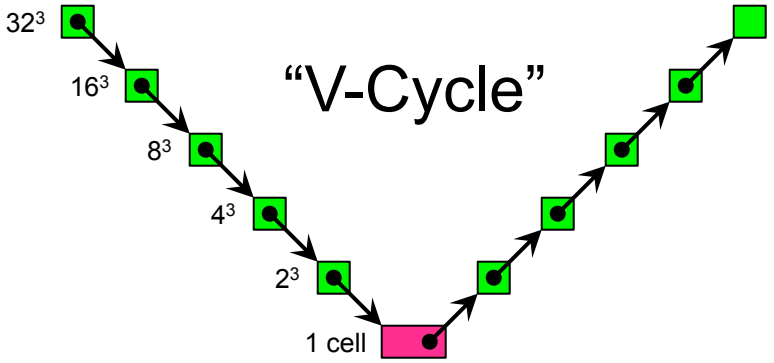


- Solutions variables are usually cell-centered, **but boundary values exist on cell faces (face-centered)**
  - enforcing a homogeneous Dirichlet boundary condition is not simply forcing the ghost cells to zero.
  - Rather one has to select a value for each ghost cell that allows one to interpolate to zero on the face.

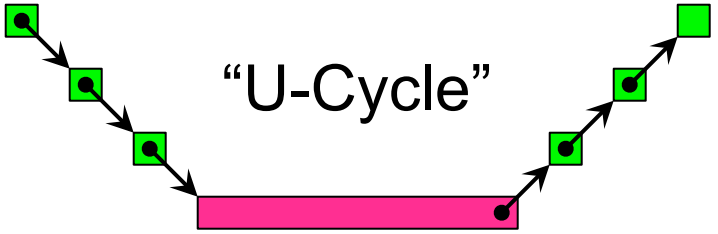


"V-Cycle" vs. "U-Cycle"

- Ideally, one should be able to restrict the global problem down to a small coarse grid problem on a single node.
  = true "V-Cycle"
- In distributed memory, this approach requires a tree-like **agglomeration** in which subdomains are restricted and combined onto a subset of the nodes.

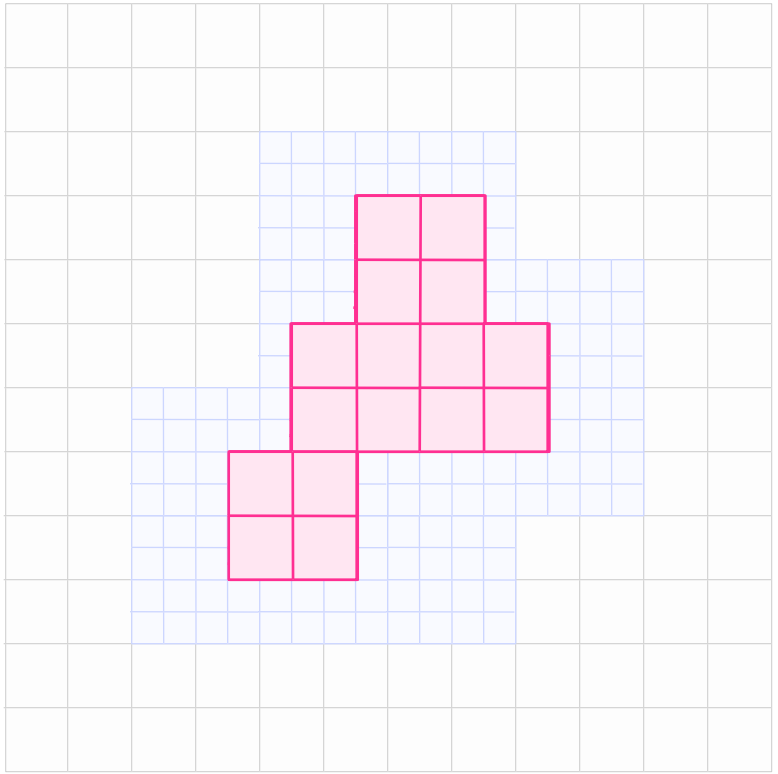


- However, realties of complex geometries (**e.g. AMR**), boundary conditions, and expediency result in MG solvers often terminating restriction early (e.g. only perform local restriction)
  = "U-Cycle"
- Unfortunately, the resultant coarse grids solves can be large and distributed and often use solvers with **superlinear computational complexity**.



Multigrid in Adaptive Mesh Refinement Applications

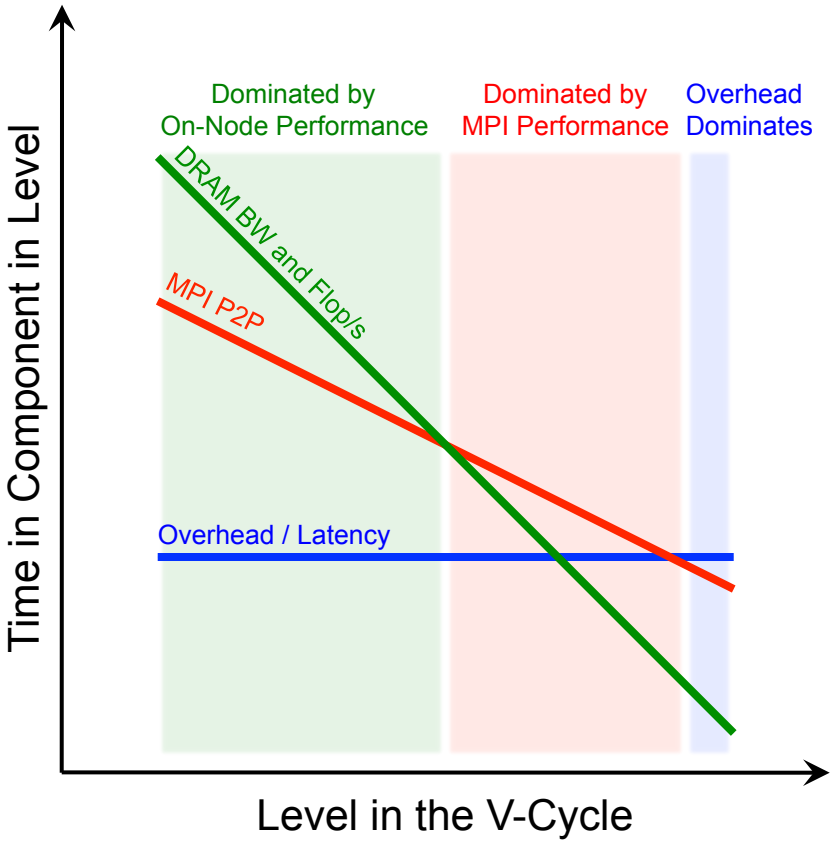
- Start with a coarse AMR level (coarse grid spacing)
- Add progressively finer AMR levels as needed (**observe they can be irregularly shaped**)
- One often performs a MG solve one one AMR level at a time.
- Unfortunately, one can reach a point where **further geometric restriction is not possible.**
- To solve this potentially large **coarse grid ("bottom") problem**, there are a number of approaches:
  - Direct solver (slow, hard, but works)
  - Point Relaxation (slow, easy)
  - Algebraic Multigrid (**Chombo/PETSc**)
  - Use iterative Solver like BiCGStab (**BoxLib**)



Multigrid Performance Challenges

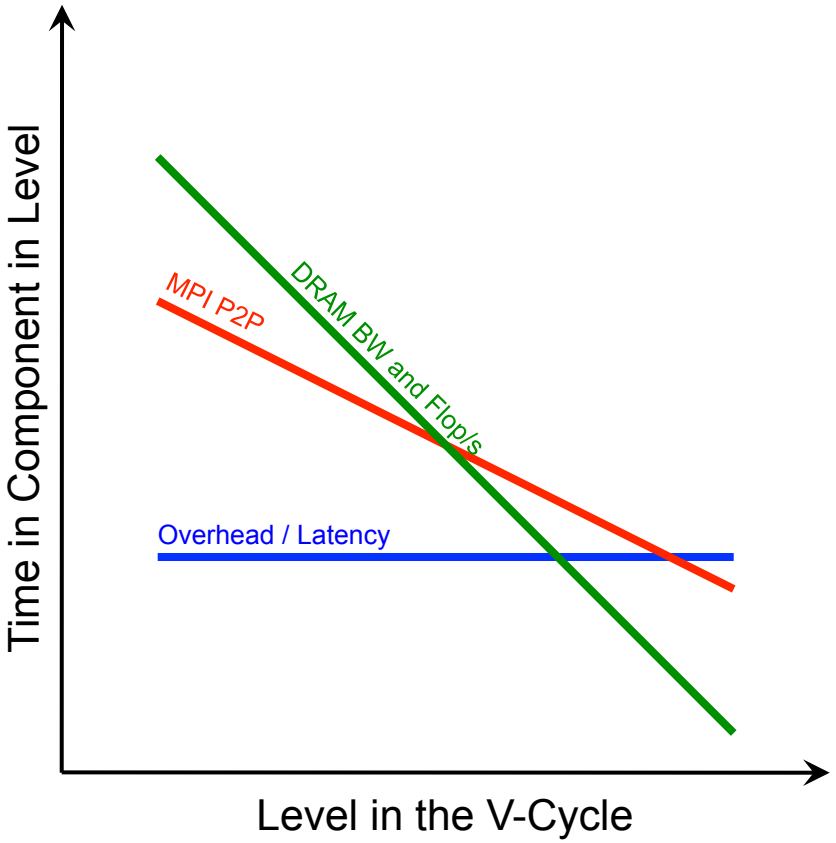
Ideal Performance

- Nominally, multigrid has three components that affect performance
  - **DRAM data movement** and flop's to perform each stencil
  - **MPI data movement** for halo/ghost zone exchanges
  - **latency/overhead** for each operation
- These are constrained by
  - **DRAM and flop rates**
  - **MPI P2P Bandwidth**
  - **MPI overhead, OpenMP/CUDA overheads, etc…**
- The time spent in each of these varies with level in the v-cycle
  - coarse grids have ⅛ the volume (number of cells), but ¼ the surface area (MPI message size)



Faster Machines?

- If one just increases DRAM bandwidth by 10x, then the code becomes increasingly **dominated by MPI P2P communication**
- If one improves just DRAM and MPI bandwidth, the code will eventually be **dominated by CUDA, OpenMP, and MPI overheads**.
- Unfortunately, the overheads are hit O(logN) times.
- Thus, if overhead dominates (flops and bytes are free), then **MGSolve Time looks like O(logN)**
- **Co-Design for MG requires a balanced scaling of flop/s, GB/s, memory capacities, and overheads**.



HPGMG-FV Overview

HPGMG is Available on BitBucket

- There are two versions of HPGMG
  - **HPGMG-FV: Finite Volume, thread-friendly, memory and network intensive**
  - HPGMG-FE: Finite Element, flat MPI, compute intensive, built on PETSc
  - both are geometric MG
  - both use Full Multigrid (FMG)
  - both are available from https://bitbucket.org/hpgmg/hpgmg/
  - **HPGMG-FV is in the finite-volume/source subdirectory.**

- By default, HPGMG-FV is configured for Top500 benchmarking evaluations.
- **However, when using HPGMG-FV for CoDesign, one should…**
  - use the helmholtz operator (-DUSE_HELMHOLTZ)
  - compared Chebyshev vs. GSRB challenges (-DUSE_CHEBY vs. -DUSE_GSRB)
  - start a few smallish boxes per process (e.g. **8 x 64^3** by running **mpirun –n# […] ./run 6 8**)
  - explore performance as one varies box size (e.g. 32^3->128^3) and number (e.g. 8->64 boxes)
  - run with at least one process per NUMA node (per GPU)
  - use more than one process (more than one GPU) to quantify impact of communication

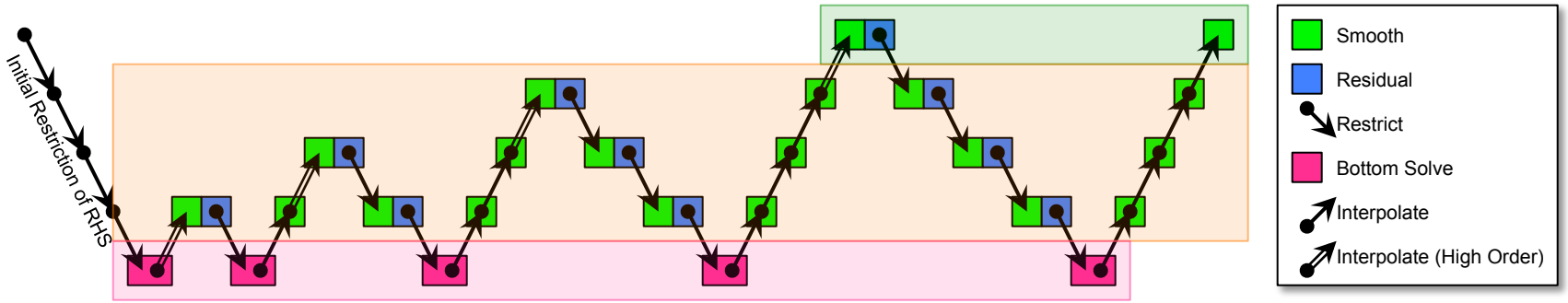
- I will try and post up-to-date online documentation and notes to…
  http://crd.lbl.gov/departments/computer-science/performance-and-algorithms-research/research/hpgmg/

HPGMG-FV

- **portable MPI + OpenMP** (no SIMD intrinsics)
- Based on **true distributed V-Cycles** that allow restriction of a trillion cells distributed across 100K processes down to one cell (total).
  - Once a process runs out of cells (falls beneath a threshold), the next restriction will shuffle data onto a subset of the processes in a tunable tree-like agglomeration structure
  - **Unfortunately, restriction/prolongation are now distributed operations** in which locally restricted/interpolated grids may need to be sent en masse to another process (inter-level communication)
- Data decomposition is on a level-by-level basis (rather than static)
  - proxies the irregular decompositions which may emerge on an AMR level
  - allows for easy extension with any user-defined domain decomposition
    (recursive bisection ~= Z-Mort, specialized recursive variant, and lexicographical)
  - allows for a **truly heterogeneous implementation** (not yet implemented) in which fine grids are run on accelerators and coarse grids on host processor
- Configurable for U-Cycles, V-Cycles, or **F-Cycles (FMG)**
- In addition to GSRB, there are **Chebyshev**, SymGS, weighted-Jacobi, and L1-Jacobi smoothers
- Implements both **homogenous Dirichlet** and periodic BC's
- Configurable bottom solver including **BiCGStab**, CG, and s-step variants.

FMG

- HPGMG-FV implements Full Multigrid (FMG).
- FMG uses an F-Cycle with a V-Cycle at each level.
- No iterating. One global reduction (to calculate the final residual)
- **Essentially, an O(N) direct solver (discretization error in 1 pass)**



- Fine grids (those in slow "capacity" memory) are accessed only twice
- Coarser grids (those that have progressively smaller working sets) are accessed progressively more
- **Same routines are used many times with highly varied working sets**
- Coarsest grids are likely latency-limited (**run on host?**)
- FMG sends $O(\log^2(P))$ messages (**needs low overhead communication**)
- Stresses many aspect of the system (memory hierarchy, network, compute, threading overheads, heterogeneity, …)

HPGMG-FV detailed timing….

Time within a MG level by function

Total Time by function

	0	1	2	3	4	5	6	7	8	9	
box dimension	128^3	64^3	32^3	16^3	8^3	8^3	8^3	4^3	2^3	9^3	total
smooth	0.083160	0.009769	0.002024	0.000753	0.000592	0.000711	0.000833	0.001602	0.001382	0.000000	0.100826
residual	0.018734	0.000940	0.000204	0.000088	0.000073	0.000087	0.000102	0.000181	0.000158	0.000155	0.020721
applyOp	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.001907	0.001907	
BLAS1	0.004149	0.000115	0.000057	0.000053	0.000064	0.000069	0.000082	0.000206	0.000197	0.014692	0.019984
BLAS3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
Boundary Conditions	0.000000	0.000308	0.000080	0.000017	0.000005	0.000005	0.000005	0.000013	0.000014	0.000011	0.000458
Restriction	0.000922	0.000350	0.000297	0.000141	0.000435	0.000363	0.000445	0.000603	0.000790	0.000000	0.004346
local restriction	0.000915	0.000342	0.000288	0.000130	0.000032	0.000037	0.000042	0.000129	0.000146	0.000000	0.002062
pack MPI buffers	0.000001	0.000001	0.000000	0.000001	0.000001	0.000001	0.000001	0.000001	0.000001	0.000000	0.000007
unpack MPI buffers	0.000001	0.000001	0.000001	0.000001	0.000095	0.000106	0.000124	0.000140	0.000224	0.000000	0.000694
MPI_Isend	0.000001	0.000000	0.000001	0.000001	0.000001	0.000001	0.000001	0.000001	0.000001	0.000000	0.000007
MPI_Irecv	0.000001	0.000001	0.000001	0.000001	0.000035	0.000045	0.000061	0.000056	0.000063	0.000000	0.000263
MPI_Waitall	0.000000	0.000001	0.000001	0.000001	0.000263	0.000164	0.000205	0.000263	0.000340	0.000000	0.001239
Interpolation	0.002921	0.001742	0.001107	0.000369	0.000499	0.000579	0.000741	0.000631	0.000740	0.000000	0.009329
local interpolation	0.002916	0.001735	0.001098	0.000358	0.000068	0.000077	0.000085	0.000137	0.000147	0.000000	0.006621
pack MPI buffers	0.000000	0.000000	0.000001	0.000001	0.000157	0.000179	0.000202	0.000147	0.000238	0.000000	0.000926
unpack MPI buffers	0.000000	0.000000	0.000001	0.000001	0.000001	0.000001	0.000001	0.000001	0.000002	0.000000	0.000009
MPI_Isend	0.000000	0.000000	0.000001	0.000001	0.000131	0.000154	0.000196	0.000154	0.000185	0.000000	0.000822
MPI_Irecv	0.000000	0.000000	0.000001	0.000001	0.000001	0.000001	0.000001	0.000001	0.000001	0.000000	0.000007
MPI_Waitall	0.000000	0.000001	0.000001	0.000001	0.000132	0.000155	0.000241	0.000176	0.000150	0.000000	0.000856
Ghost Zone Exchange	0.010486	0.005997	0.003671	0.003480	0.003963	0.004767	0.005602	0.007449	0.007796	0.002098	0.055309
local exchange	0.000003	0.000003	0.000004	0.000005	0.000006	0.000007	0.000008	0.001059	0.001659	0.001838	0.004589
pack MPI buffers	0.001327	0.000467	0.000442	0.000518	0.000624	0.000743	0.000863	0.000991	0.001208	0.000026	0.007210
unpack MPI buffers	0.000473	0.000455	0.000485	0.000593	0.000738	0.000878	0.001019	0.001130	0.001331	0.000025	0.007125
MPI_Isend	0.000302	0.000339	0.000450	0.000781	0.000937	0.001143	0.001334	0.001515	0.001190	0.000018	0.008009
MPI_Irecv	0.000093	0.000096	0.000140	0.000165	0.000210	0.000250	0.000299	0.000313	0.000257	0.000012	0.001835
MPI_Waitall	0.008260	0.004603	0.002103	0.001355	0.001370	0.001656	0.001970	0.002306	0.002008	0.000011	0.025641
MPI_collectives	0.001312	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.002378	0.003691
Total by level	0.122319	0.018799	0.007384	0.004927	0.005706	0.006680	0.008064	0.010724	0.010967	0.021933	0.217503

smooth on finest level is only 38% of solve time

Time in an operation by level

demonstrates performance of MPICH's MPI_Comm_split/dup is broken at scale !

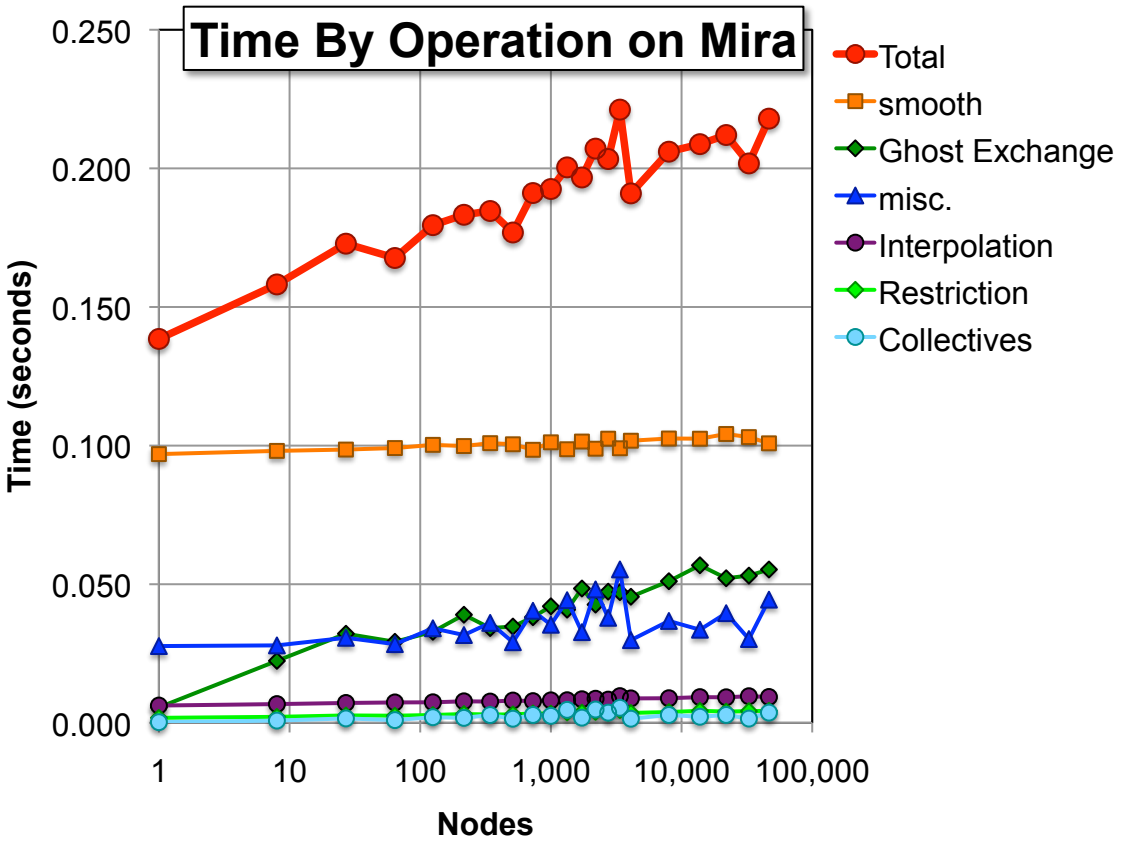
```
Total time in MGBuild    225.675795 seconds
Total time in MGSolve      0.217941 seconds
   number of v-cycles             1
Bottom solver iterations         70

          Performance      4.489e+11 DOF/s
```

Used to verify implementation… error should be 2nd order

```
calculating error...
 h =  2.170138888888889e-04  ||error|| =  4.595122248560908e-11
```

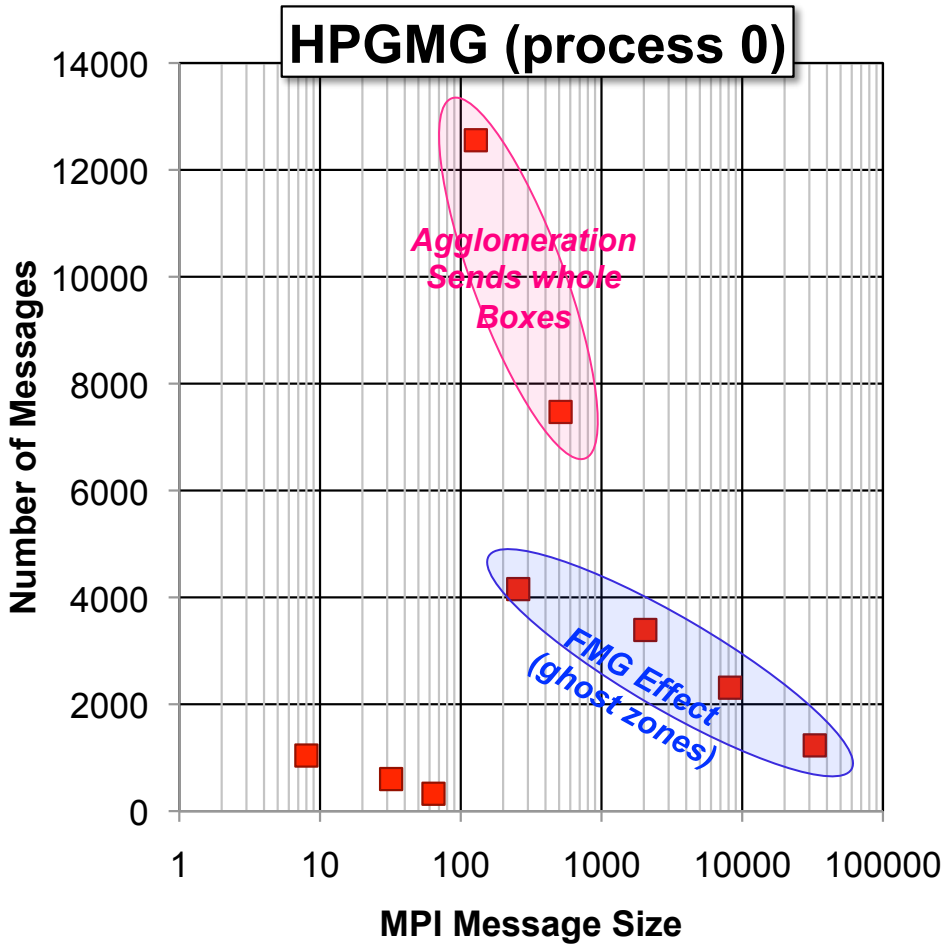
Breakdown of Time



- Breakdown of time on Mira…:
- **smooth** time is roughly constant.
- **misc…**
  - Residual and BLAS1 operations
  - relatively constant (residual is large %)
  - but sees some variation for odd coarse grid problem sizes (15^3) due to variable # of BLAS1 in BiCGStab iterations
- **Ghost zone** exchange time steadily increases with scale…
  - topology not exploited in job scheduler or MPI as mapping processes in AMR codes is difficult
- **Collectives…**
  - Just 1 global collective for the residual
  - many local collectives on the coarse grid solve
  - Collectives on on BGQ are fast
- **Restriction/Interpolation…**
  - despite requiring inter-level communication are still very fast
  - Interpolation sends 8x more data

HPGMG-FV (message sizes)

- In FMG,
  - large messages only occur in only the last V-Cycle
  - **smaller messages are more frequent** as FMG performs progressively more small V-Cycles
  - agglomeration causes a spike in message counts when you reach the agglomeration threshold
  - eventually all cells are on one node and the number of messages is small.
- **Performance on small messages (overhead/latency) can be critical**



data collected on BGQ w/HPM

HPGMG-FV Deep Dive

Coordinates

- Coordinates can either be discrete of continuous

- Continuous coordinates
  - labeled x,y,z by convention
  - represent coordinates in space
  - are used to evaluate continuous functions
  - are independent of multigrid/AMR level

- Discrete coordinates
  - labeled i,j,k by convention
  - access array elements
  - define array sizes
  - are a function of the current MG/AMR level and grid spacing h…

    (x,y,z) = (i*h,j*h,k*h) for some grid spacing h

Boxes

(The Quanta for Domain Decomposition)

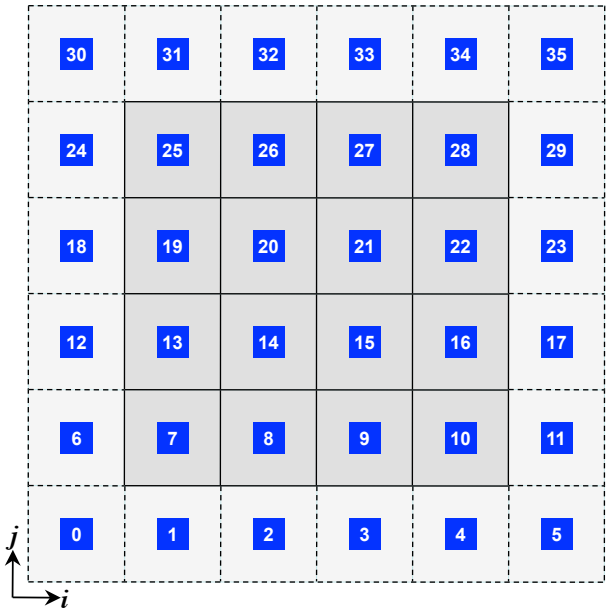
- **box is a cubical region of space at a particular grid spacing h**
  - has a i,j,k discrete coordinate of its lower coordinate
  - discrete i,j,k maps to continuous coordinates x,y,z = ih,jh,kh
  - boxes have a dimension '**dim**', but have additional '**ghosts**'-deep ghost zones (halo) which replicates data from neighboring boxes.
  - boxes can have some array padding to facilitate SIMDization/alignment
  - **jStride, kStride, and volume** are defined to facilitate indexing in the presence of deep ghost zones with complex padding for alignment

- Boxes have 'numVectors' **vectors** (e.g. solution, RHS, D^{-1}, etc…) each comprising double-precision values over the region of space
  - box->vectors[id] returns a pointer to a 3D double precision array
  - this data can represent either cell-centered –or– face-centered data.
  - box->vectors[id][0] is the first **ghost zone** element !!
  - **many routines perform some pointer arithmetic to create a pointer to the first non-ghost zone element.**

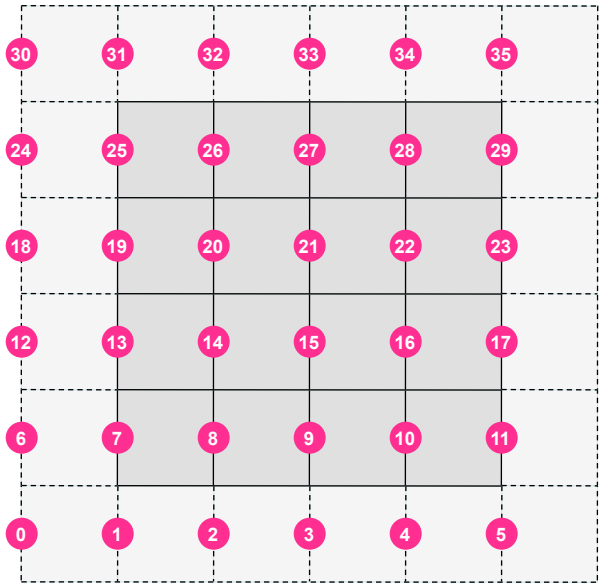
Boxes

(Cell- vs. Face-Centered Data Layout)

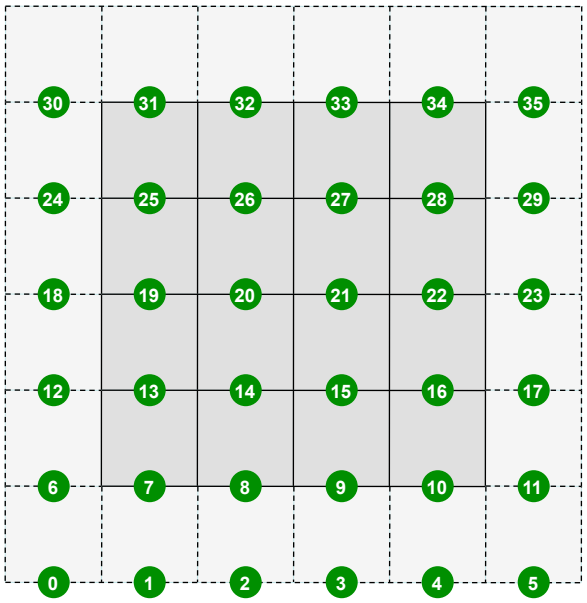
- Nominally, face-centered data can get by with smaller array dimensions (no need for face-centered ghost data)
- However, for simplicity and to facilitate indexing, HPGMG always uses the same number of elements for both cell- or face-centered (=array padding)
- Thus we have different 3D arrays (#'s are offsets from the base pointer)
- Although data is always stored in separate arrays…



Cell-centered data
(e.g. x[ ])



Face-centered data
(e.g. beta_i[ ])

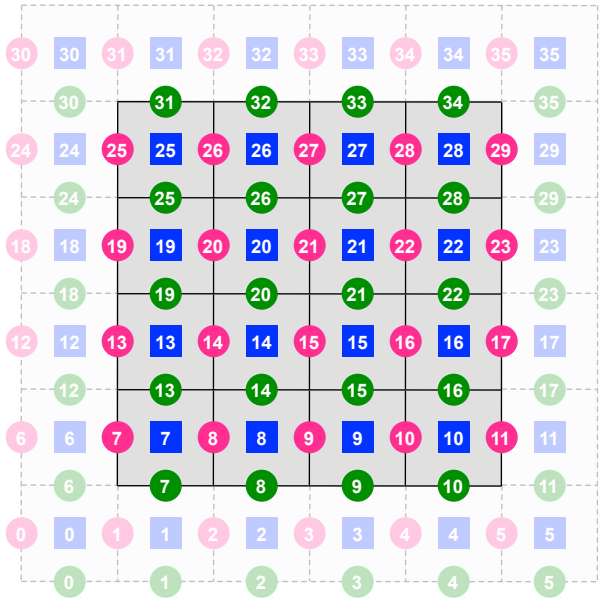


Face-centered data
(e.g. beta_j[ ])

Boxes

(Cell- vs. Face-Centered Data Layout)

- However, conceptually, it represents different quantities in the same region of space…

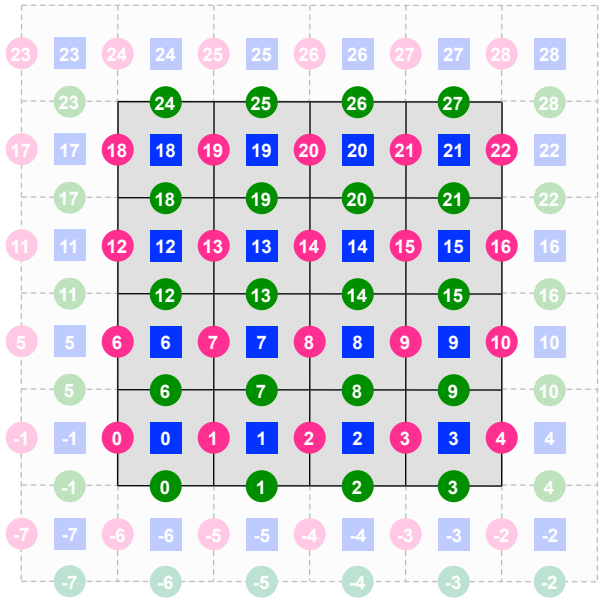


Boxes

(Cell- vs. Face-Centered Data Layout)

- With a little pointer arithmetic, the first **non-ghost zone** cell or lower faces of that cell have coordinates (0,0,0)
- e.g.

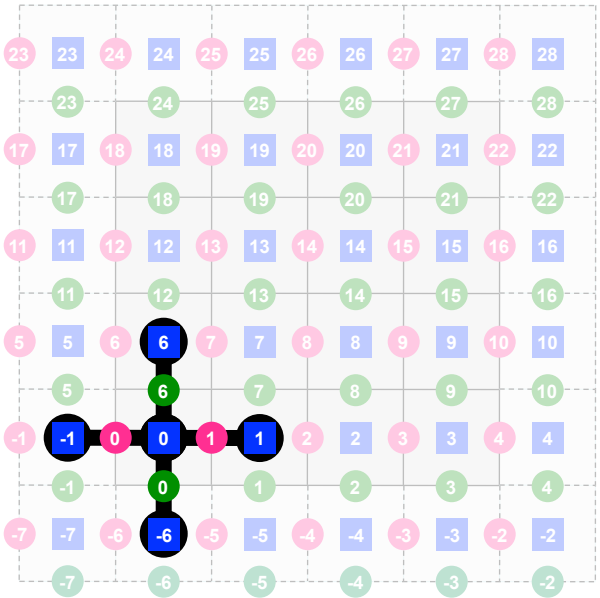
```
const double * __restrict__ rhs = level->my_boxes[box].vectors[rhs_id] + ghosts*(1+jStride+kStride);
```



Boxes

(Cell- vs. Face-Centered Data Layout)

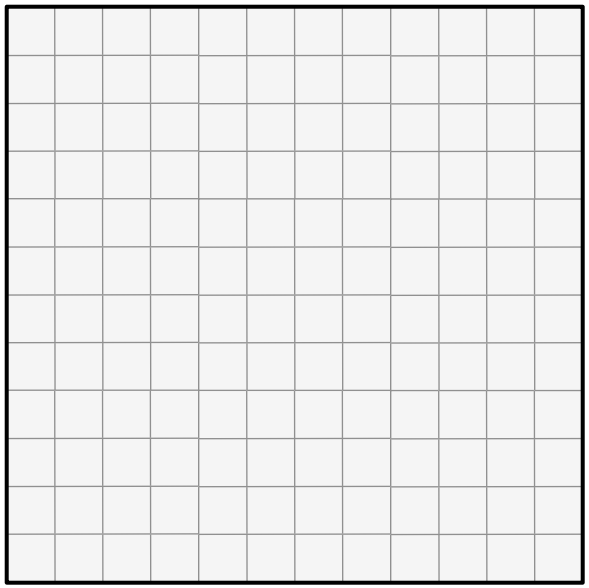
- A variable-coefficient 7-point stencil has asymmetric coefficient indexing
- observe that a stencil at x[ijk=0] uses…
  - beta_i[0] and beta_i[1]
  - beta_j[0] and beta_j[6]
  - x[-1], x[1], x[-6], x[6], and x[0]



Levels
(the domain at a grid spacing h)

- HPGMG Creates a hierarchy of '**levels**'
  - each level is (currently) a cubical domain partitioned into cubical boxes
  - each level has a unique grid spacing 'h' which differs by a factor of 2x from its coarse and fine neighboring levels
  - boxes can either be smaller, the same size, or larger on coarser levels (but total number of cells is always 8x less)
- Boxes are distributed among processes
  - in **level.c**, a 3D array is created and populated with the MPI rank of the owner of each box.
  - All communication routines are built using this 3D array
  - Researchers can replace the existing domain decomposition options (**populate this 3D array**) with something more sophisticated **without changing any other code**
- On a given level, a process can have any number of boxes (even none)
  - **Not all processes have the same number of boxes (load imbalance, fewer boxes deep in the v-cycle)**
  - An '**active**' process is a process that has work on the current or deeper levels
  - Inactive processes drop out (complex MPI DAG)
  - HPGMG creates a subcommunicator for each level to minimize any global communication

example for illustration purposes…

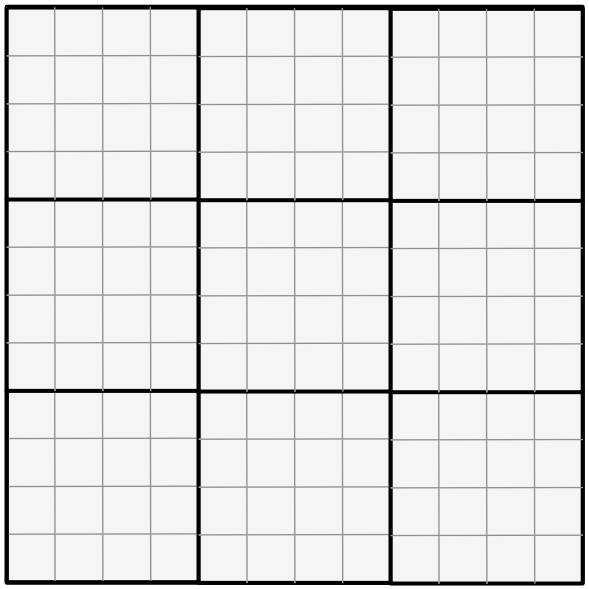


start with a 12x12 level

Levels
(the domain at a grid spacing h)

- HPGMG Creates a hierarchy of '**levels**'
  - each level is (currently) a cubical domain partitioned into cubical boxes
  - each level has a unique grid spacing 'h' which differs by a factor of 2x from its coarse and fine neighboring levels
  - boxes can either be smaller, the same size, or larger on coarser levels (but total number of cells is always 8x less)
- Boxes are distributed among processes
  - in **level.c**, a 3D array is created and populated with the MPI rank of the owner of each box.
  - All communication routines are built using this 3D array
  - Researchers can replace the existing domain decomposition options (**populate this 3D array**) with something more sophisticated **without changing any other code**
- On a given level, a process can have any number of boxes (even none)
  - **Not all processes have the same number of boxes (load imbalance, fewer boxes deep in the v-cycle)**
  - An '**active**' process is a process that has work on the current or deeper levels
  - Inactive processes drop out (complex MPI DAG)
  - HPGMG creates a subcommunicator for each level to minimize any global communication

example for illustration purposes…

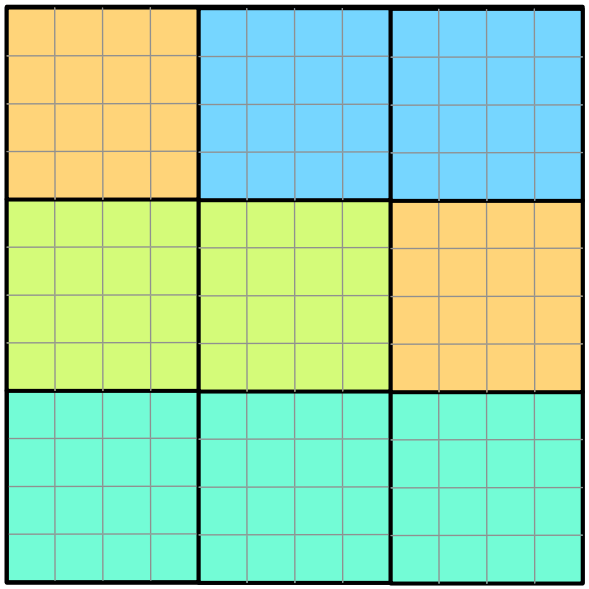


decompose into nine 4x4 boxes

Levels
(the domain at a grid spacing h)

- HPGMG Creates a hierarchy of '**levels**'
  - each level is (currently) a cubical domain partitioned into cubical boxes
  - each level has a unique grid spacing 'h' which differs by a factor of 2x from its coarse and fine neighboring levels
  - boxes can either be smaller, the same size, or larger on coarser levels (but total number of cells is always 8x less)
- Boxes are distributed among processes
  - in **level.c**, a 3D array is created and populated with the MPI rank of the owner of each box.
  - All communication routines are built using this 3D array
  - Researchers can replace the existing domain decomposition options (**populate this 3D array**) with something more sophisticated **without changing any other code**
- On a given level, a process can have any number of boxes (even none)
  - **Not all processes have the same number of boxes (load imbalance, fewer boxes deep in the v-cycle)**
  - An '**active**' process is a process that has work on the current or deeper levels
  - Inactive processes drop out (complex MPI DAG)
  - HPGMG creates a subcommunicator for each level to minimize any global communication

example for illustration purposes…

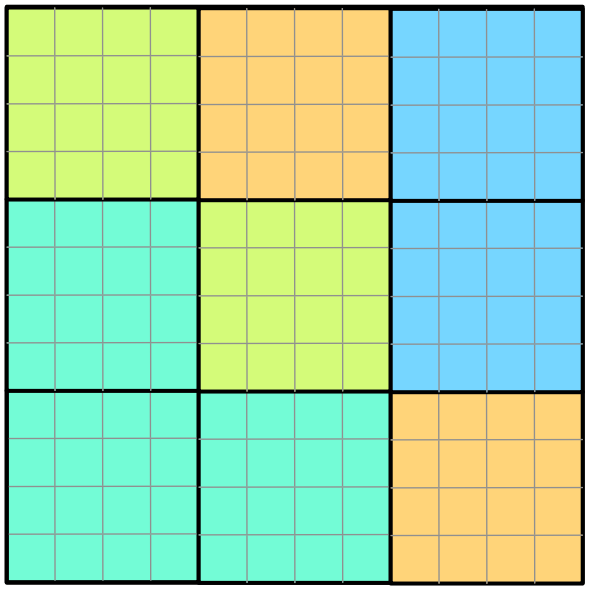


parallelize among 4 MPI processes (max=3) using a lexicographical ordering

Levels
(the domain at a grid spacing h)

- HPGMG Creates a hierarchy of '**levels**'
  - each level is (currently) a cubical domain partitioned into cubical boxes
  - each level has a unique grid spacing 'h' which differs by a factor of 2x from its coarse and fine neighboring levels
  - boxes can either be smaller, the same size, or larger on coarser levels (but total number of cells is always 8x less)
- Boxes are distributed among processes
  - in **level.c**, a 3D array is created and populated with the MPI rank of the owner of each box.
  - All communication routines are built using this 3D array
  - Researchers can replace the existing domain decomposition options (**populate this 3D array**) with something more sophisticated **without changing any other code**
- On a given level, a process can have any number of boxes (even none)
  - **Not all processes have the same number of boxes (load imbalance, fewer boxes deep in the v-cycle)**
  - An '**active**' process is a process that has work on the current or deeper levels
  - Inactive processes drop out (complex MPI DAG)
  - HPGMG creates a subcommunicator for each level to minimize any global communication

example for illustration purposes…



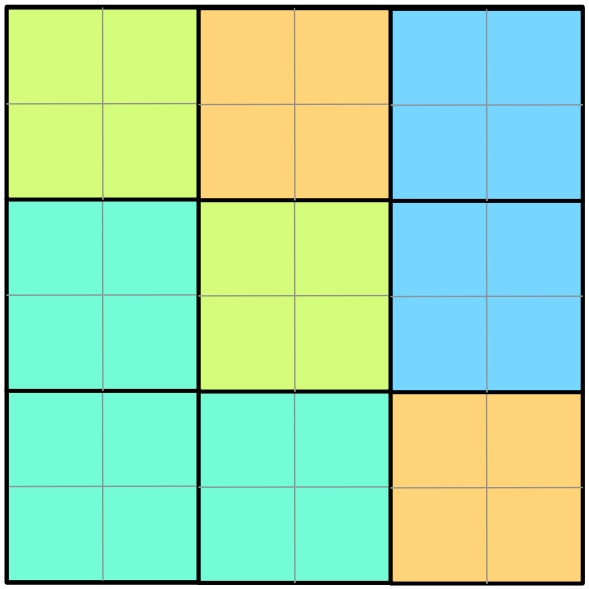
By default, the code uses a recursive bisection decomposition to form a SFC

Observe the complex communication pattern reminiscent of an AMR level.

Levels
(the domain at a grid spacing h)

- HPGMG Creates a hierarchy of '**levels**'
  - each level is (currently) a cubical domain partitioned into cubical boxes
  - each level has a unique grid spacing 'h' which differs by a factor of 2x from its coarse and fine neighboring levels
  - boxes can either be smaller, the same size, or larger on coarser levels (but total number of cells is always 8x less)
- Boxes are distributed among processes
  - in **level.c**, a 3D array is created and populated with the MPI rank of the owner of each box.
  - All communication routines are built using this 3D array
  - Researchers can replace the existing domain decomposition options (**populate this 3D array**) with something more sophisticated **without changing any other code**
- On a given level, a process can have any number of boxes (even none)
  - **Not all processes have the same number of boxes (load imbalance, fewer boxes deep in the v-cycle)**
  - An '**active**' process is a process that has work on the current or deeper levels
  - Inactive processes drop out (complex MPI DAG)
  - HPGMG creates a subcommunicator for each level to minimize any global communication

example for illustration purposes…

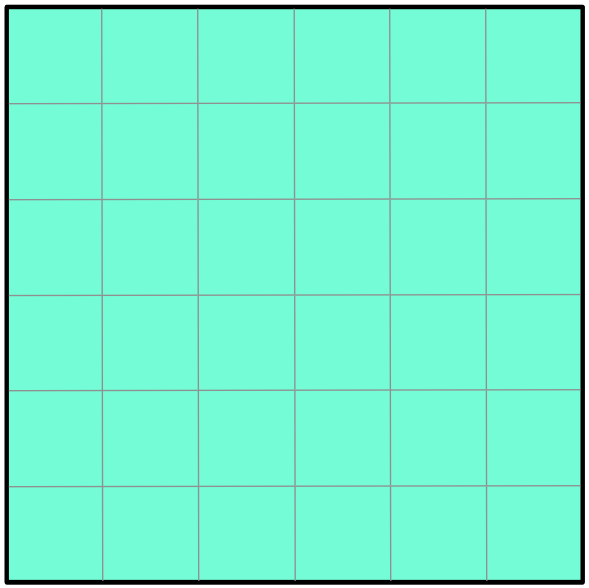


A coarser level (spacing=2h) may have smaller (2x2) boxes decomposed the same way…

Levels
(the domain at a grid spacing 2h)

- HPGMG Creates a hierarchy of '**levels**'
  - each level is (currently) a cubical domain partitioned into cubical boxes
  - each level has a unique grid spacing 'h' which differs by a factor of 2x from its coarse and fine neighboring levels
  - boxes can either be smaller, the same size, or larger on coarser levels (but total number of cells is always 8x less)
- Boxes are distributed among processes
  - in **level.c**, a 3D array is created and populated with the MPI rank of the owner of each box.
  - All communication routines are built using this 3D array
  - Researchers can replace the existing domain decomposition options (**populate this 3D array**) with something more sophisticated **without changing any other code**
- On a given level, a process can have any number of boxes (even none)
  - **Not all processes have the same number of boxes (load imbalance, fewer boxes deep in the v-cycle)**
  - An '**active**' process is a process that has work on the current or deeper levels
  - Inactive processes drop out (complex MPI DAG)
  - HPGMG creates a subcommunicator for each level to minimize any global communication

example for illustration purposes…



…or may have only one box owned by process 0.

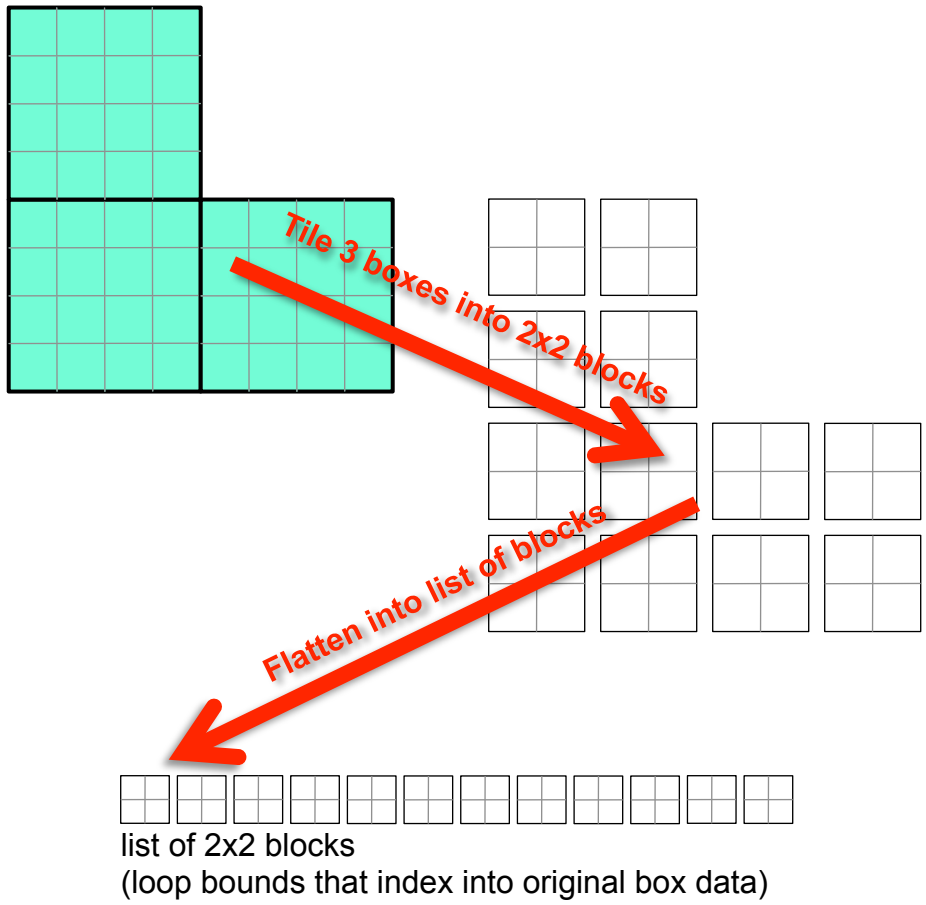
MPI processes 1, 2, and 3 would thus be inactive.

Blocks (use case #1)

flatten computation for efficient threading of operators

- HPGMG varies between coarse- and fine-grained parallelism
  - One can have anywhere from 0 to perhaps 64 boxes per process
  - **Due to agglomeration the active number of boxes varies over the course of a solve**
  - Depending on level in the v-cycle, **the size of each box varies exponentially**
  - Although there are many ways of expressing this parallelism in OpenMP (collapse(2), nested, tasks, etc…), performance portability across compilers is illusive
- '**Blocks**' are a tiling/flattening of the loop iteration space
  - take highly-variable 4 deep loop nest (box,i,j,k) and tile to create 4-deep loop nest (block,bi,bj,bk) of roughly equal block sizes
  - **blocks are just meta data !!!**

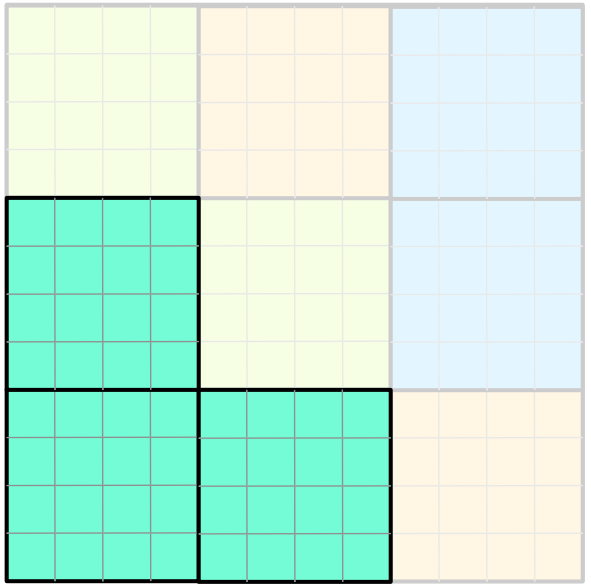
    They are an auxiliary array of loop bounds used to index into box data (actual FP data)
  - **Smaller blocks provide more coarse-grained TLP, but incur more overhead**
  - **Large blocks provide more fine TLP/SIMD**
  - A block has dimension and both read and write (source/dest) boxID, and i,j,k coordinates.



Blocks (use case #2)

Ghost Zone Exchanges

- Ghost zone exchanges can be difficult to implement efficiently
- For each box, you need to determine whether its neighbors are on- or off-node.
- If the latter, one should aggregate data together and minimize the number of messages (amortize MPI overheads)
- This process is complex/expensive.

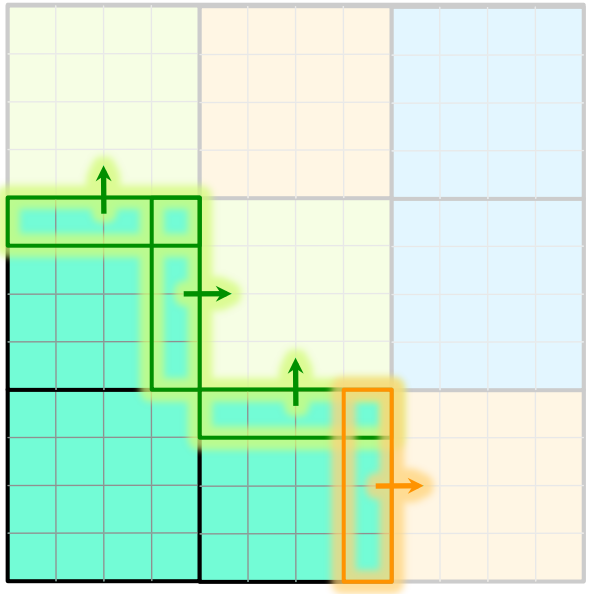


- HPGMG reuses the '**block**' mechanism to cache this traversal of meta data for fast replay
  - pack list (box->MPI buffer)
  - local list (box->box)
  - unpack list (MPI buffer->box)

Blocks (use case #2)

Ghost Zone Exchanges

- Ghost zone exchanges can be difficult to implement efficiently
- For each box, you need to determine whether its neighbors are on- or off-node.
- If the latter, one should aggregate data together and minimize the number of messages (amortize MPI overheads)
- This process is complex/expensive.

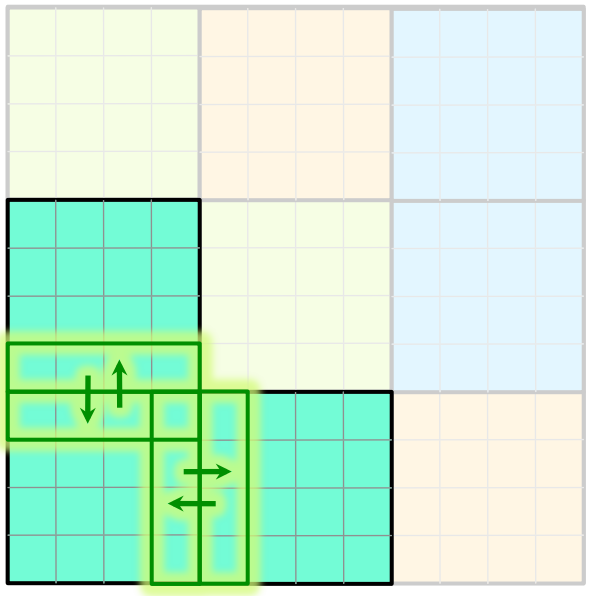


- HPGMG reuses the '**block**' mechanism to cache this traversal of meta data for fast replay
  - **pack list (box->MPI buffer)**
  - local list (box->box)
  - unpack list (MPI buffer->box)

Blocks (use case #2)

Ghost Zone Exchanges

- Ghost zone exchanges can be difficult to implement efficiently
- For each box, you need to determine whether its neighbors are on- or off-node.
- If the latter, one should aggregate data together and minimize the number of messages (amortize MPI overheads)
- This process is complex/expensive.

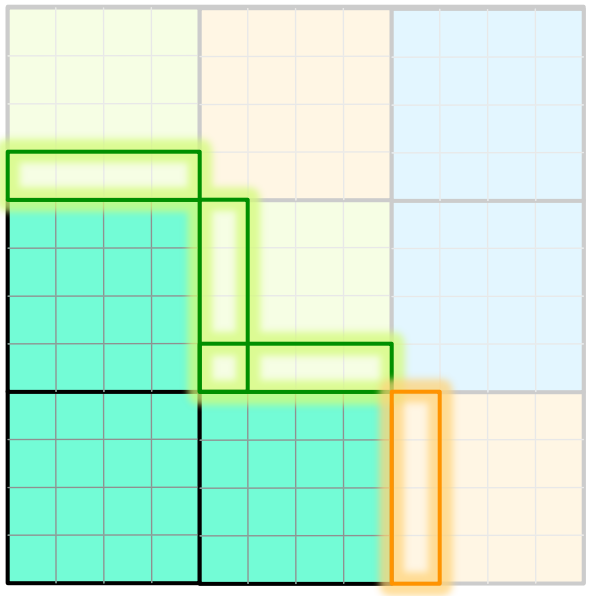


- HPGMG reuses the '**block**' mechanism to cache this traversal of meta data for fast replay
  - pack list (box->MPI buffer)
  - **local list (box->box)**
  - unpack list (MPI buffer->box)

Blocks (use case #2)

Ghost Zone Exchanges

- Ghost zone exchanges can be difficult to implement efficiently
- For each box, you need to determine whether its neighbors are on- or off-node.
- If the latter, one should aggregate data together and minimize the number of messages (amortize MPI overheads)
- This process is complex/expensive.

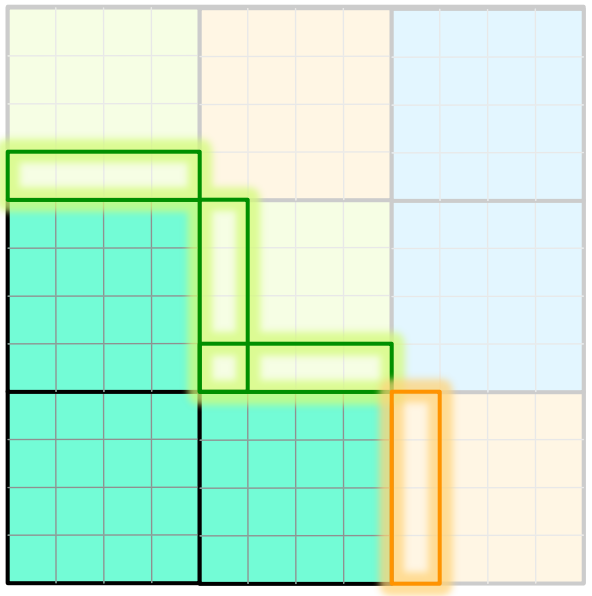


- HPGMG reuses the '**block**' mechanism to cache this traversal of meta data for fast replay
  - pack list (box->MPI buffer)
  - local list (box->box)
  - **unpack list (MPI buffer->box)**

Blocks (use case #2)

Ghost Zone Exchanges

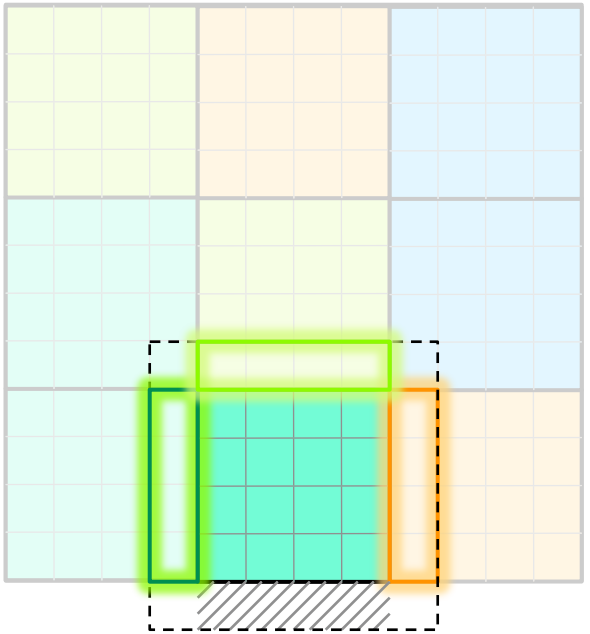
- Currently, HPGMG implements all distributed memory operations with MPI2.
- As part of the DEGAS (X-Stack) project, we are exploring **PGAS** variants of HPGMG
  - **UPC++ w/GASNet**
  - but MPI3 is a possibility
- In theory, one could extend the block meta data to allow a copy from local to remote memory.
  - P2P synchronization is still required
  - strided access pattern complicates matters (copy a 3D tile from a local 3D array to a remote 3D array).
  - would replace pack/local/unpack structure with PGAS put's that the runtime would differentiate



Blocks (use case #3)

Boundary Conditions

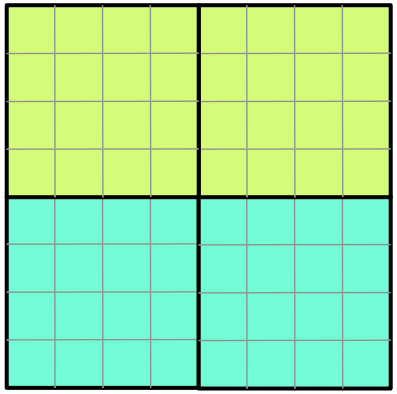
- Boxes on a domain boundary (outer boxes) require enforcement of a boundary condition (currently linear homogeneous Dirichlet)
- Currently the code has two options for Dirichlet Boundary Conditions…
  - naïve approach (**nearly serial**)
  - **fused with the stencil (overkill)**
- I am re-implementing the naïve approach to leverage the block concept
  - facilitates use of massive TLP for BC's
  - will facilitate implementation of high-order boundary conditions
  - **due to data dependencies, BC's must occur after ghost zone exchange completes**

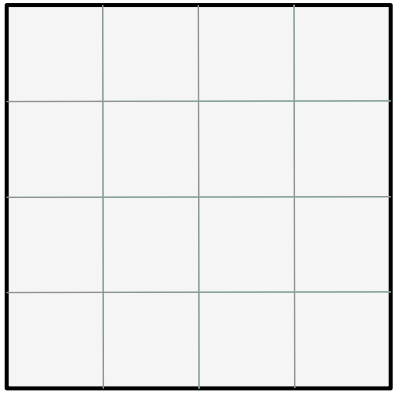


Blocks (use case #4)

Restriction and Interpolation

- HPGMG leverages the block mechanism for distributed restriction and interpolation (prolongation)
- However, rather than just block copy (ghost zones), operations can be block restrict or block interpolation

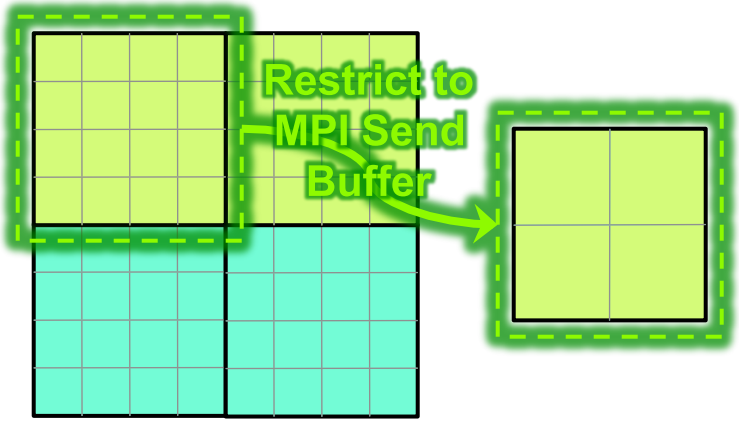


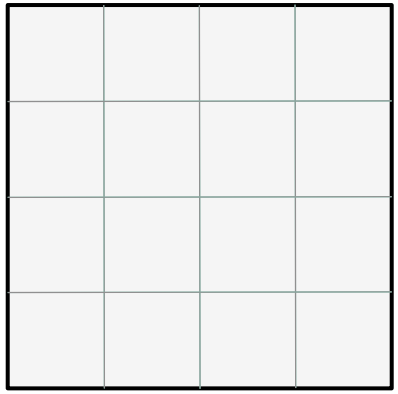


Blocks (use case #4)

Restriction and Interpolation

- HPGMG leverages the block mechanism for distributed restriction and interpolation (prolongation)
- However, rather than just block copy (ghost zones), operations can be block restrict or block interpolation
- For restriction…
  - **pack list (restrict box->MPI buffer)**
  - local list (restrict box->box)
  - unpack list (copy MPI buffer->box)

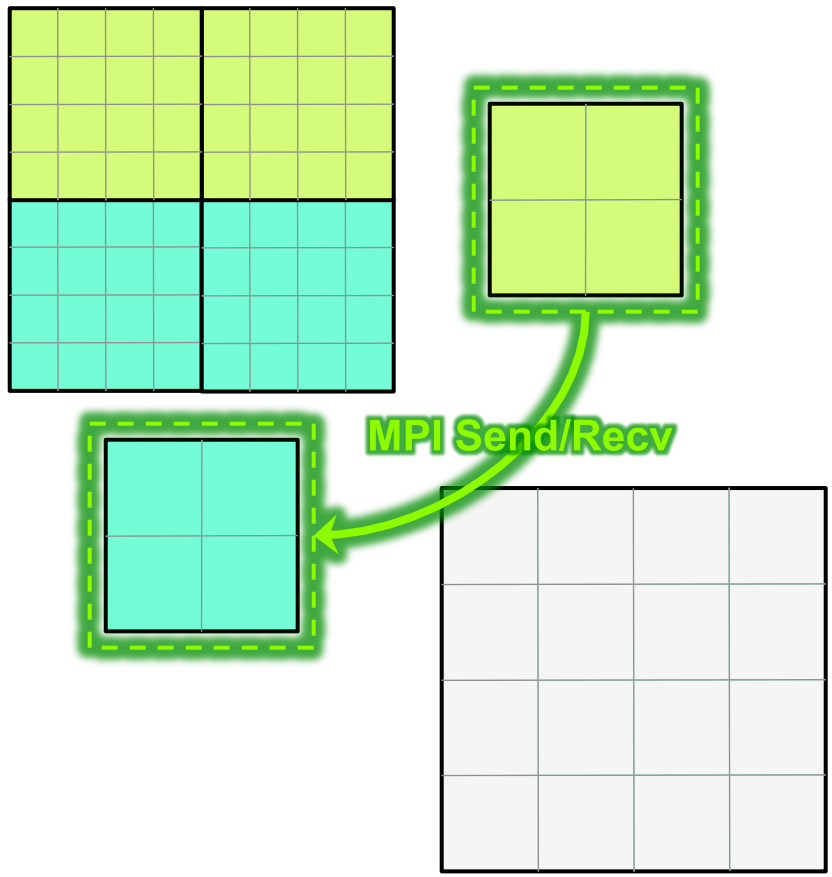




Blocks (use case #4)

Restriction and Interpolation

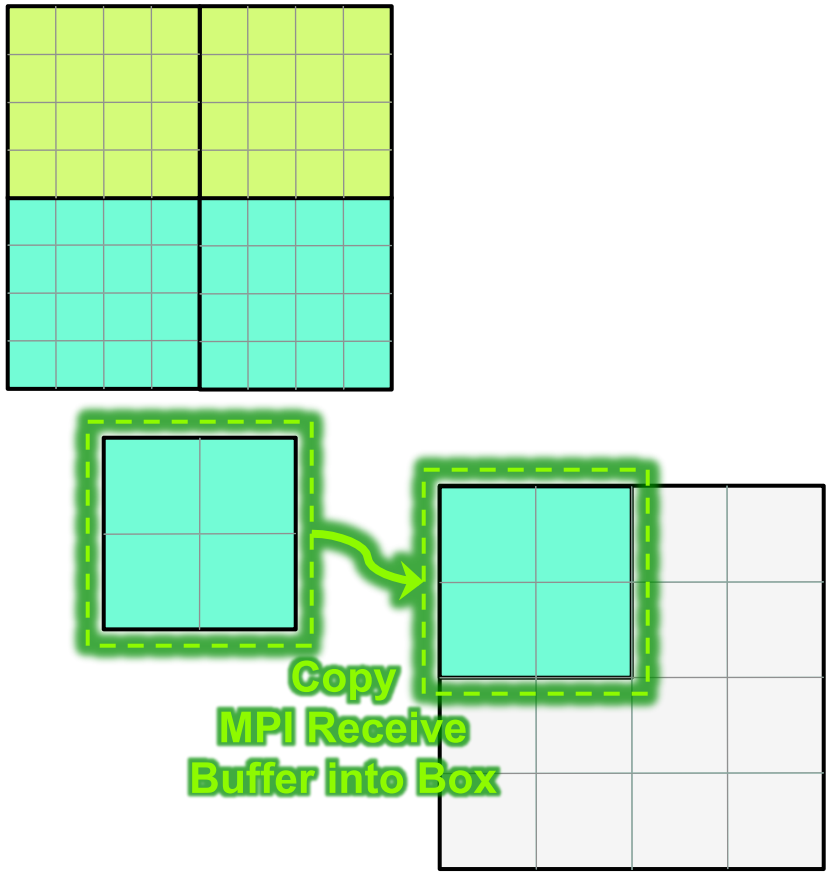
- HPGMG leverages the block mechanism for distributed restriction and interpolation (prolongation)
- However, rather than just block copy (ghost zones), operations can be block restrict or block interpolation
- For restriction…
  - **pack list (restrict box->MPI buffer)**
  - local list (restrict box->box)
  - unpack list (copy MPI buffer->box)



Blocks (use case #4)

Restriction and Interpolation

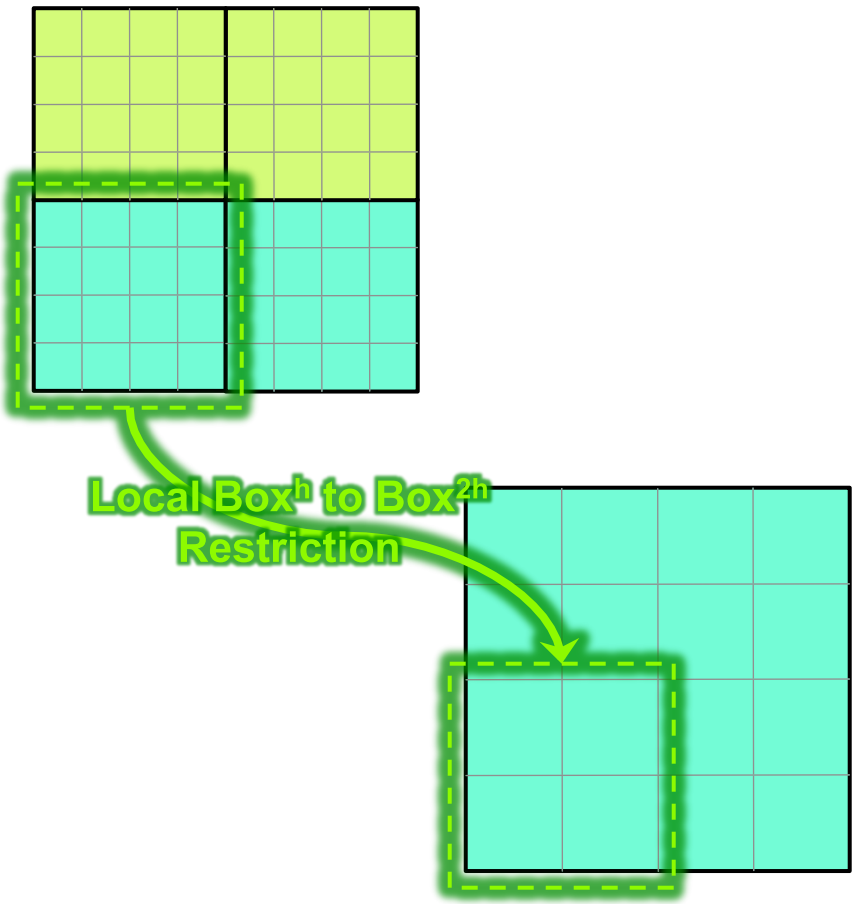
- HPGMG leverages the block mechanism for distributed restriction and interpolation (prolongation)
- However, rather than just block copy (ghost zones), operations can be block restrict or block interpolation
- For restriction…
  - pack list (restrict box->MPI buffer)
  - local list (restrict box->box)
  - **unpack list (copy MPI buffer->box)**



Blocks (use case #4)

Restriction and Interpolation

- HPGMG leverages the block mechanism for distributed restriction and interpolation (prolongation)
- However, rather than just block copy (ghost zones), operations can be block restrict or block interpolation
- For restriction…
  - pack list (restrict box->MPI buffer)
  - **local list (restrict box->box)**
  - unpack list (copy MPI buffer->box)
- Interpolation is simply the reverse of this process…
  - pack list (interpolate box->MPI buffer)
  - local list (interpolate box->box)
  - unpack list (copy MPI buffer->box)



Potential Issues with Blocks

- Currently, blocks can the following operations
  - box -> box/vector
  - box/vector -> pointer
  - pointer -> box/vector
  - pointer -> pointer
- block data structure include a box ID or a pointer (double*)
- use of a pointer is a nonissue for unified CPU memory architectures

- **If the presence of a pointer in the block data type is a problem, let me know and I can easily change it to an integer (int bufID)**

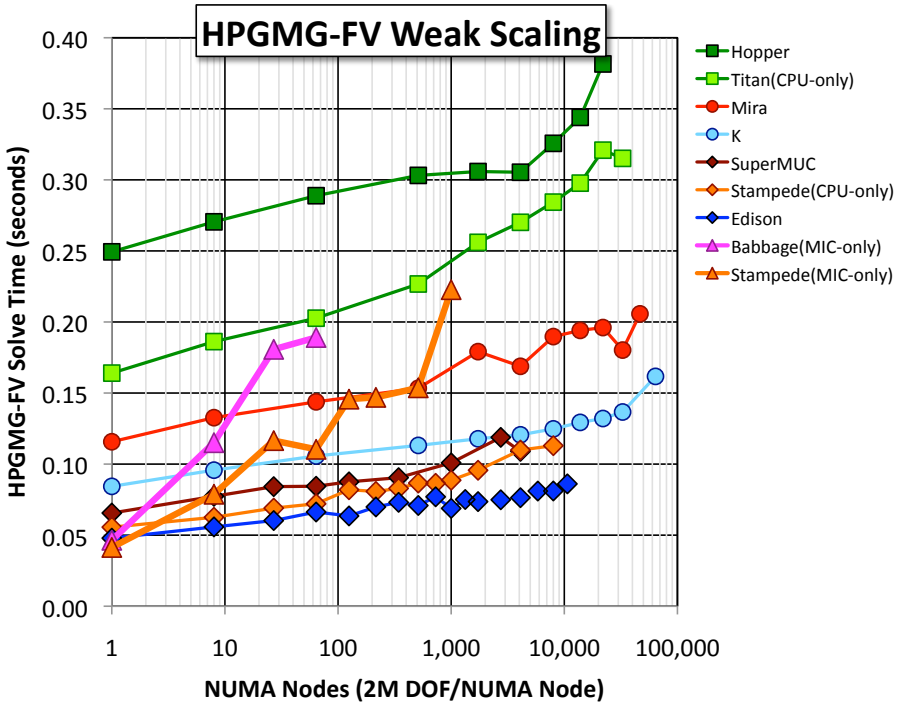
Threading Lists of Blocks

(Currently OpenMP, alternate possibilities)

- HPGMG currently uses OpenMP to thread across the list of blocks
- HPGMG leverages **c99's _Pragma( )** to hide OpenMP pragmas inside preprocessor macros…
  - allows programmers to change the macro once and have it affect all operators in the code (rather than changing a hundred routines every time one changes OpenMP usage)
  - currently, there are three flavors…
    - parallel execution of the list (smoothers, vector-vector)
    - parallel execution of the list w/sum reduction (dot products)
    - parallel execution of the list w/max reduction (max/inf norms)
  - … all are implemented with variants of `#pragma omp parallel for`
  - Moreover, there are three levels of threading…
    - none (flat MPI is different than OMP_NUM_THREADS=1)
    - OpenMP 2.x (no max reductions). XL/C runs in this mode due to a compiler issue with _Pragma() and the max reduction
    - OpenMP 3.x (max reductions for norm calculations)
- In theory, one could modify these to explore other programming models…
  - block becomes a **task in OpenMP3's task model**
  - block becomes a **team of threads in OpenMP4's distribute**
  - block becomes a **gang of workers in OpenACC**
  - block becomes a **thread block in CUDA**

Performance Expectations

- Performance on a single-node should be roughly STREAM-limited
- Performance at scale is dependent on the network and MPI implementation.
- Given M memory per node and P Processes, HPGMG should…
  - move O(M) data to/from DRAM
  - send O($\log^2$(M*P)) messages
  - perform 1 global MPI_Allreduce
  - perform many local MPI_Allreduce's
- Consider…
  - vanilla MPI+OpenMP with one process per socket (numa node)
  - tuned BLOCKCOPY_TILE* on MIC/BGQ
  - aprun –n[#] [affinity] ./hpgmg 7 1
  - weak scaled to # numa nodes (flat is perfect)
  - single process solve times should be <50ms on most intel processors (SNB/IVB/MIC)
  - If single node solve time is significantly greater than 50ms, something is wrong
- **This figure provides a means of comparing your performance & speedup to today's conventional approaches**



Co-Design Questions

HPGMG Configuration for CoDesign

- Nominally, HPGMG is configured as a Top500 benchmark
- For Co-Design purposes, a few additional compiler flags are required.

- use the helmholtz operator (**-DUSE_HELMHOLTZ**)
- compare Chebyshev & GSRB (**-DUSE_CHEBY** vs. **-DUSE_GSRB**)
- start a few smallish boxes per process (e.g. **8 x 64^3**) by running
  mpirun –n# […] ./run 6 8

Does Exascale Enable Strong Scaling or Consolidation?

- run with at least one process per NUMA node (per GPU)
- **use more than one process (more than one GPU) to quantify impact of communication**
- note, HPGMG's structure never stabilizes. It continually adapts to process count and problem size

- **how does performance vary as one varies box size (e.g. 32^3→128^3) and number (e.g. 8→64 boxes)**

mpirun –n# […] ./run 5 8	vs.	mpirun –n# […] ./run 5 64
mpirun –n# […] ./run 6 8	vs.	mpirun –n# […] ./run 6 64
mpirun –n# […] ./run 7 8	vs.	mpirun –n# […] ./run 7 64

- This provides insights to the CoDesign center as to whether algorithmic changes are required/sufficient (e.g. shift LMC to a 4D parallelization scheme)

How do we efficiently and succinctly manage locality?

- active working sets vary by factors of 8x
- there can be >7 levels in MG (**working sets vary from GB's to KB's**)
- With an AMM like the X-Stack program's target of 10 levels of software-controlled memory, different levels are lit up as the algorithm progresses…

  L1, L2,L1,L2, L3,L2,L1,L2,L3, L4,L3,L2,L1,L2,L3,L4, … L8,L7,L6,L5,L4,L3,L2,L1,L2,L3,L4,L5,L6,L7,L8

- **How does one concisely orchestrate data movement through the memory hierarchy?**
  - the same routine (same pragmas?) can be called with either a GB working set or a KB working sets
  - the arrays touched (double*) can be different for the same routine

How sensitive is exascale to operations with limited parallelism?

- MG's computational complexity is premised on the assumption that N/8 flops requires N/8 time.
  - N+N/8+N/64… = O(N) flops ~ O(N) time
- Today, the performance of MIC/GPU processors decreases substantially when parallelism falls below a certain threshold (underutilization)
- If time ceases to be tied to N but saturates at some constant, then
  - N+N/8+N/8+N/8+…N/8 ~ O(Nlog(N))

- **Does your FastForward processor performance on coarse (coasrer) grids impede overall multigrid performance?**
  - Are there architectural features you can exploit to avoid this?
  - If so, how do you succinctly specialize code to exploit them?
    (i.e. do we really have to write every routine twice?)
  - Are there other approaches to ensure coarse grid operations are not a bottleneck?

Acknowledgements

- All authors from Lawrence Berkeley National Laboratory were supported by the DOE Office of Advanced Scientific Computing Research under contract number DE-AC02-05CH11231.
- This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.
- This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357.
- This research used resources of the Oak Ridge Leadership Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

Questions?

SWWilliams@lbl.gov

Backup Slides

Combustion Proxy Apps

- HPGMG-FV
  - proxies MG solves in LMC
  - 6K lines of C
  - allows true distributed V-Cycles
  - boundary conditions generalized (currently Dirichlet or periodic)
  - multiple smoother options (GSRB, weighted and L1 Jacobi, Chebyshev, symgs)
  - multiple coarse grid solver options (point relaxation, BiCGStab, CABiCGStab,…)
  - true Full Multigrid (FMG) implementation
  - allows for easy(?) ports to alternate programming models
  - allows for easy exploration of high-order operators

- AMR_EXP_Parabolic
  - Explicit AMR (no MG)
  - proxies AMR issues in LMC with subcycling in time
  - Fortran
  - requires BoxLib

Memory Capacity Issues

- In AMR MG Combustion codes, you need a separate field/component/vector for each chemical species (NH_4, CO_2, …) on each AMR level
- As such, given today's memory constraints, the **size of each process's subdomain might be small** (64^3…128^3)
- Future machines may have 10x more memory than today's…
  - 100GB of fast memory
  - 1TB of slow memory
- **Why not run larger problems to amortize inefficiencies?**
  - Application scientists would prefer to use it for new physics or chemistry.
    e.g. increase the number of chemical species from 20 to 100
  - AMR codes could use the memory selectively (where needed) with deeper AMR hierarchies.
  - The memory hierarchy can be used to prioritize the active working set…
    e.g. fit the MG solve on current species of the current AMR level in fast memory

- **If performance is not feasible, we need to know soon as significant changes to LMC would be required to increase on-node parallelism**

Choice of Smoother

- In the manycore era, the choice of smoother:
  - must balance mathematical (convergence) and architectural constraints (TLP/SIMD/BW).
  - may see up to a 100x performance hit without threading on a Xeon Phi (MIC)
- Using HPGMG-FV we observed differences in performance among smoothers…
  - GSRB and w-Jacobi were the **easiest to use**
  - SYMGS required fewer total smooths, but its **performance per smooth was very poor**.
  - Based on Rob/Ulrike's paper, L1 Jacobi was made as fast as w-Jacobi
  - **Chebyshev was fastest** in the net (smooth was little slower, but required fewer smooths)
  - Unfortunately, Chebyshev is a bit twitchy as it **needs eigenvalue estimates**.

	SYMGS (blocked)	Gauss-Seidel Red-Black	Chebyshev Polynomial	weighted Jacobi	L1 Jacobi
Convergence	very good (2 SYMGS)	good (2-3 GSRB)	very good Degree 2 or 4	slow (8+ smooths)	slow (8+ smooths)
Requirements (in addition to D^{-1})		N/A to high-order operators	spectral properties of the operator	not necessarily stable	L1 norm
Threading?	extremely difficult	yes	yes	yes	yes
SIMD?	extremely difficult	inefficient (stride-2)	yes	yes	yes

GMG vs. AMG

GMG

- uses a structured grid
- operator (A) is a stencil
  - variable coefficient finite volume stencil requires 32 bytes per stencil
  - same is true for 27pt or higher order
- R and P are defined geometrically based on properties of the underlying numerical method.
- constructs the coarse grid operator directly as if it were a fine grid
- decomposition, communication, and optimization are straightforward.
- works very well for many problems of interest.
- when it fails, try alternate bottom solve or use AMG.

AMG

- uses a arbitrary graph
- operator (A) is a sparse matrix
  - assembled matrix requires ~12 bytes and a FMA per point in the stencil
  - low compute intensity regardless of discretization
- R and P are constructed based on the graph and the operator.
- constructs the coarse grid operator using the $A^{2h} = R^h A^h P^h$ triple product (expensive)
- good decomposition becomes a graph partitioning problem
- more general approach, but performance optimization is challenging