



Birds of a Feather:
High performance geometric multigrid (HPGMG)
proposal for a new Top500 benchmark

Mark Adams (LBNL)
Jed Brown (ANL,CU)
John Shalf (LBNL)
Brian Van Straalen (LBNL)
Erich Strohmaier (LBNL)
Sam Williams (LBNL)



HPGMG Birds of a Feather Outline

- Introduction – HPGMG benchmark (10 min)
- HPGMG-FV, Sam Williams (10 min)
- HPGMG-FE, Jed Brown (10 min)
- Users
 - Mitsuhsa Sato, Tsukuba/RIKEN (5 min)
 - Muthu M Baskaran, Reservoir Labs (5 min)
- Panel discussion (20 min)
- Please fill out short survey: hpgmg.org/survey



Motivation & Proposal for new Top500 metric

- **HPL not relevant metric for contemporary computation**
 - Applications now exploit structure - $O(1)$ work / word
 - HPL poor match contemporary computation (but good some)
- Addition of new metric – HPCG – Top500 list, demonstrates
 - **Recognition of this fact and a willingness to address it**
 - HPCG: Conjugant gradient solver with *stored matrix*
 - *Stored matrix* solves ran at 25% of peak 20+ years (RS6000)
 - Now run at < 2% of peak (BGQ)
 - ***This trend likely to continue – very unlikely to reverse***
- High Performance Geometric MultiGrid – **HPGMG**
 - **Matrix free** kernels, *~10-20% theoretical peak*
 - Complements HPL (~90% peak) & HPCG (~1% peak)
 - Global solve with optimal algorithm & fast kernels
 - **Stresses network** – leading cause application lack scalability

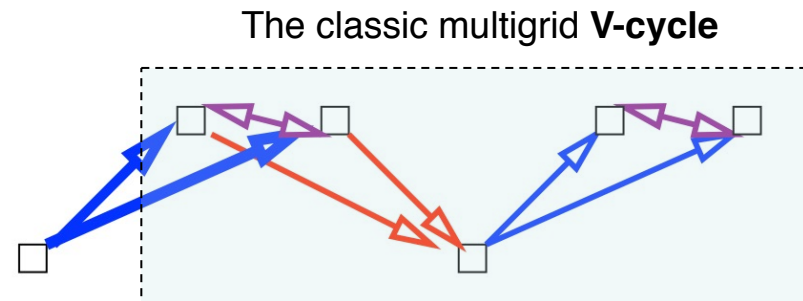
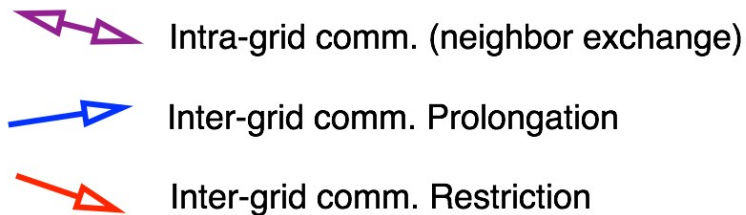


HPGMG Benchmark definition

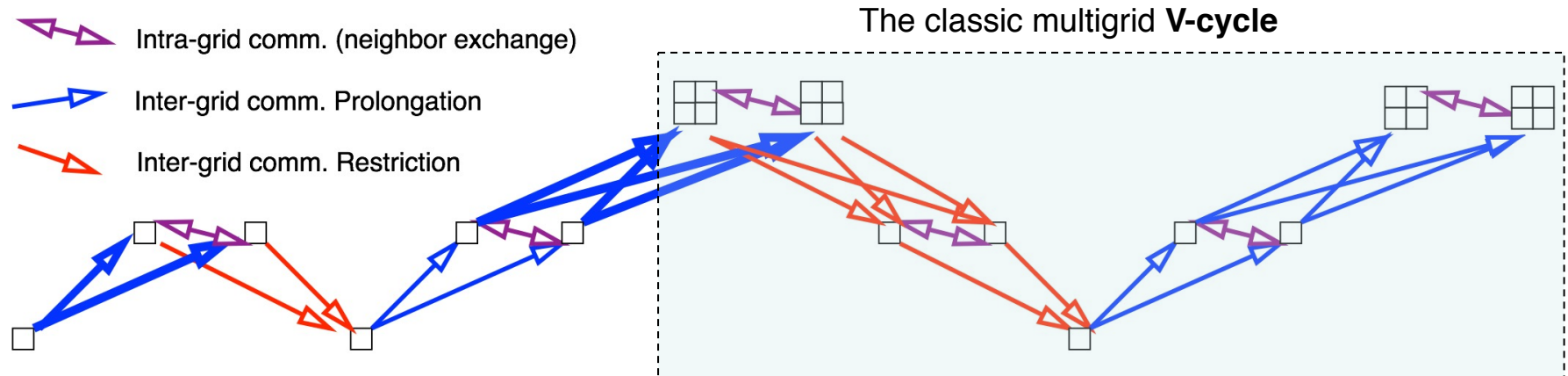
- **High Performance (Full) Geometric MultiGrid**
 - Solve of discretized Laplacian logically regular grid
 - *Non-iterative, asymptotically exact solver* $O(N)$ work complexity
 - **One FMG iteration reduces error scale discretization err**
 - *Built-in verification* that is oblivious to floating point
- **Metric: Q = number of equations solved per second**
 - Map to flops, like HPL's $2N^3$ flops/eq (w/o Strassen, etc.)
 - *Perhaps functional of Q for dynamic range (pressure network)*
- **Unlike HPL, HPGMG can fill whole machine and run in < 1 minute**
 - Community wants benchmark to run for few hours
 - Collect auxiliary data (eg, speedup studies) research & analysis



Parallel full multigrid – 2 processes



- Start at coarsest grid, accurate solve
 - usually “local” – little parallelism available
- Refine grid, split, and populate processes
 - Exponentially increasing parallelism
- Two types of inter-grid comm: **Restriction** & **Prolongation**
- **Intra-grid comm, nearest neighbor comm.**
- FMG: for all levels C to F:
 - new grid, **High Order Proj.**; **V-cycle**



- Continue to: refine, split, populate (*not shown*)
 - **Build a “fuzzy” tree**
- Continue with local refinement (*one more level shown*)
- $O(N)$ work/data algorithm - time & energy efficient
- $O(\log^2(N))$ PRAM complexity, computational depth, ...
- MG hierarchy representative many algorithms
 - Trees efficient global communication



HPGMG design principles

- *Stick to HPL formula*: Conceptually simple linear equation solve
 - Conceptually simple but parallel multigrid codes not trivial
- **Proxy application interest to HPC** (w/ minimal “complexity budget”)
 - Balanced stress of machine
 - Memory system, network, floating point ... more later
 - Computational pattern (fuzzy) tree – common HPC paradigm
- **Easy to administer** with small staff for 500+ applications
 - Minimal legislation – low order prolongation for example
 - Easy build, few deps: C + MPI + any kernel language/libraries
 - And again a conceptually simple design: solve $Ax = b$
- **Users optimize**: compilers, one primary kernel, secondary kernels, ... communication, ...
 - as far down as new programming models (Muthu)



HPGMG design requirements

- **Scale Free specification:** Solution independent parallel strategy
 - Do not punish/reward fine/coarse grain parallelism mathematically
 - Do not want to adjudicate “sub domain” nor complicate spec ...
- **Architecture Free specification:**
 - No “cache”, “main/shared memory”, “CPU”, etc., in spec.
- **Stresses interconnect** – global tree w/ non-trivial software kernels
 - **Hard global problem implicitly demands good end-to-end eng.**
 - “Extensive” metric (Strohmaier): not just sum of node performance
- **Remain relevant indefinitely (optimal algo & arch/scale free spec)**
 - *Increasingly effective proxy as more apps use efficient algorithms*
- **Balanced exercise of machine:**
 - Memory system, floating point, network, ...
 - But still compute oriented – no I/O (that is another benchmark)



HPGMG-FE & HPGMG-FV

- HPGMG-FV & HPGMG-FV excellent parallel multigrid impls & w/ great kernels
 - *Sam and Jed are awesome.*
 - Both stress *memory bandwidth, network, local caches, and vectorization*
- **HPGMG-FV:**
 - Long vector length, lower arithmetic intensity
 - ~9.6% peak flop rate on SuperMUC
 - ~60% parallel efficiency on SuperMUC (filling ~2% of memory)
 - MPI+OpenMP optimized, runs well on x86, MIC, SPARC VIIIfx (K) & BGQ
 - **Bottleneck: memory bandwidth & network**
- **HPGMG-FE:**
 - Short vector length, higher arithmetic intensity
 - ~15% peak flop rate on SuperMUC
 - ~78% parallel efficiency on SuperMUC (filling ~2% of memory)
 - More flops, higher flop rates, better scaling: b/c more kernel time
 - **Bottleneck: local caches & vectorization**
 - *Add dynamic scale to metric to stress network more effectively*



HPGMG-FV Performance

Sam Williams (LBNL)
SWWilliams@lbl.gov

SC14 HPGMG BoF – November 19 2014 12:15PM



HPGMG-FV

- **Finite Volume Method**
 - variable coefficient stencils
 - homogeneous Dirichlet boundary condition via linear interpolation
 - moderate arithmetic intensity ~ 0.25 - 1.0 flops/byte depending on optimizations
- **Optimized MPI+OpenMP implementation of HPGMG-FV**
 - delivers nearly equal performance per core compared to flat MPI
 - Faster coarse-grid solves on multi-/manycore processors
 - Mitigates some non-scalable MPI routines
- **User defined domain decomposition with multiple subdomains (boxes) per process**
 - improves load balancing
 - improves scalability



HPGMG-FV

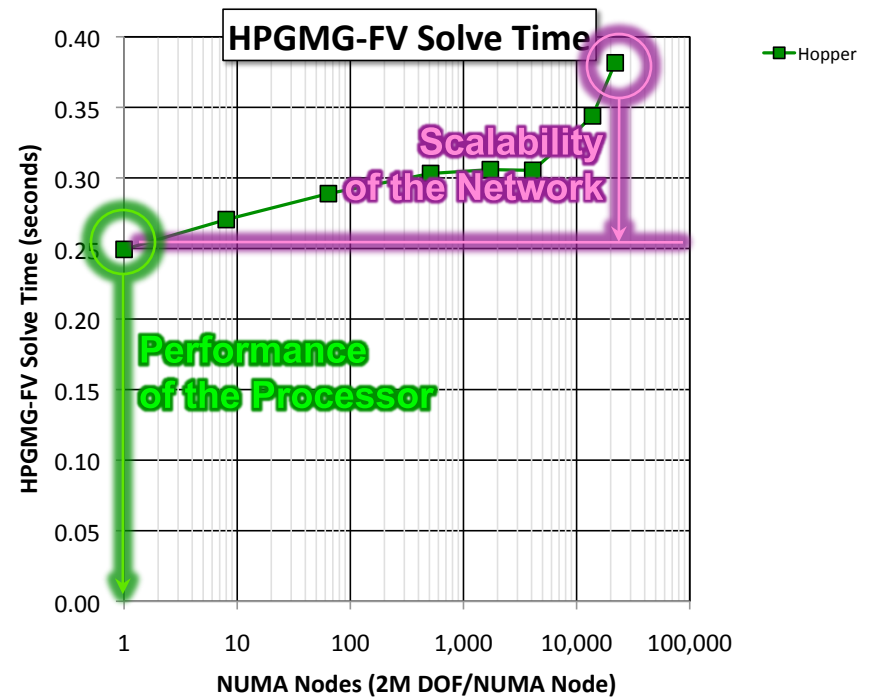
- In this talk, we will examine weak scaling HPGMG-FV performance on various systems
 - Fix the problem size to 2M DOF/NUMA node* (= 128^3)
 - This provides apples-to-apples comparisons as at a given scale, all machines will solve the same problem in the same way
 - Allows us to quantify HPGMG performance as function of...
 - Processor performance
 - DRAM bandwidth
 - Network topology (Torus, Tree, Dragonfly, etc...)
 - etc...
- We plot time-to-solution as a function of concurrency
 - Flat curve is perfect scaling (better networks)
 - Lower curves show better performance (faster nodes)

*Note, the cost and power per NUMA node will vary by perhaps a factor of 2x



3D Torus (Gemini)

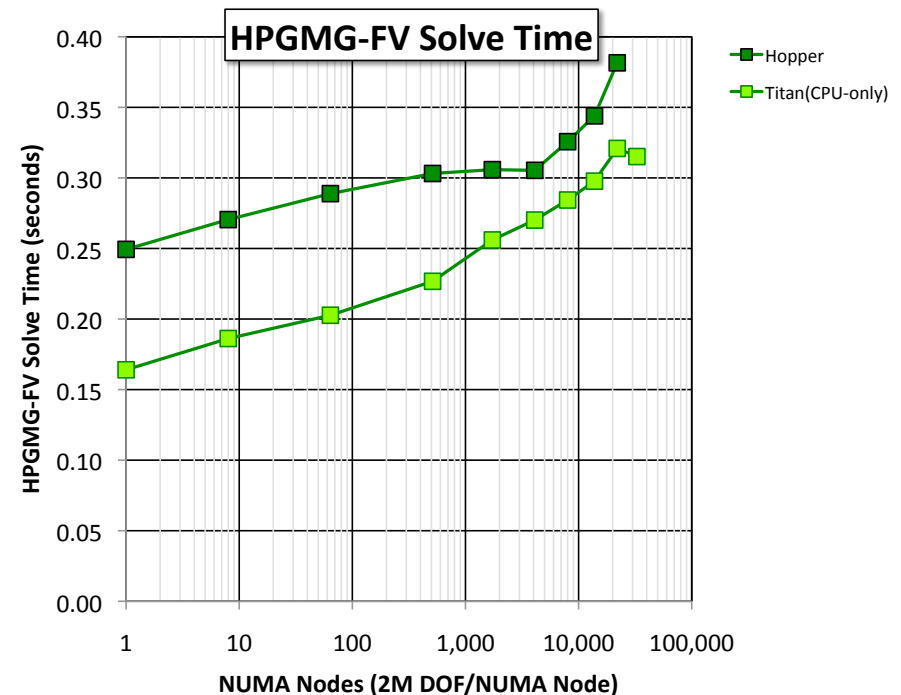
- Consider Hopper (Cray XE6)
 - single process multigrid solve time is fast (250ms)
 - performance degrades at scale
 - larger problem sizes mitigate this lack of scalability





3D Torus (Gemini)

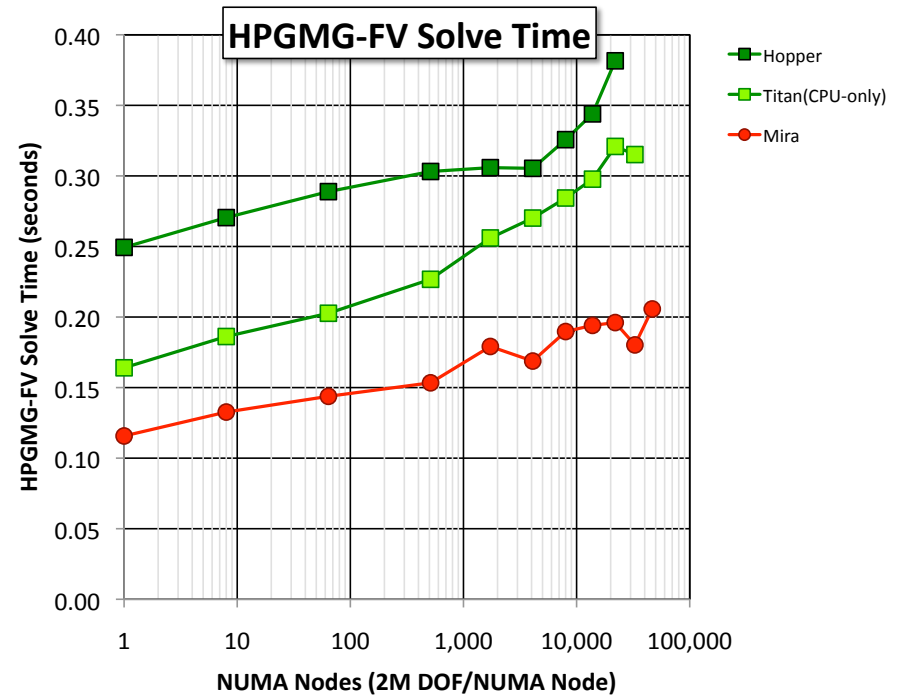
- Consider Hopper (Cray XE6)
 - single process multigrid solve time is fast (250ms)
 - performance degrades at scale
 - larger problem sizes mitigate this lack of scalability
- Titan (Cray XK7)
 - use only CPUs (same MPI+OpenMP)
 - delivered 50% better performance per socket and ~2x better overall performance
 - However, as Hopper and Titan both use Gemini, the network impeded performance at scale





Blue Gene/Q

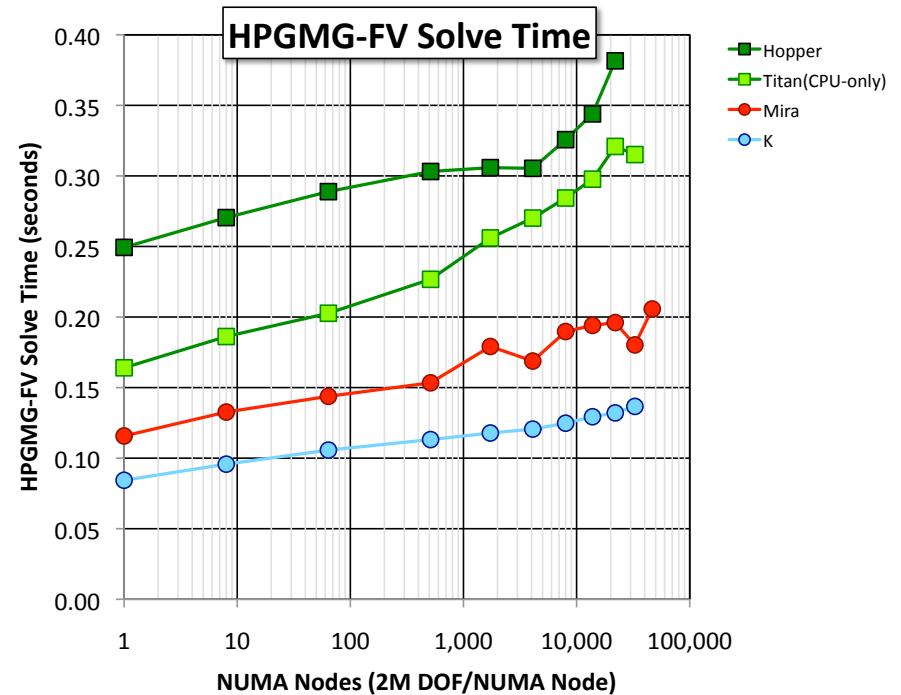
- Mira (Blue Gene/Q)
 - custom processor enabled better performance per socket
 - custom network (5D torus) enabled better scalability





K (6D Torus/Mesh)

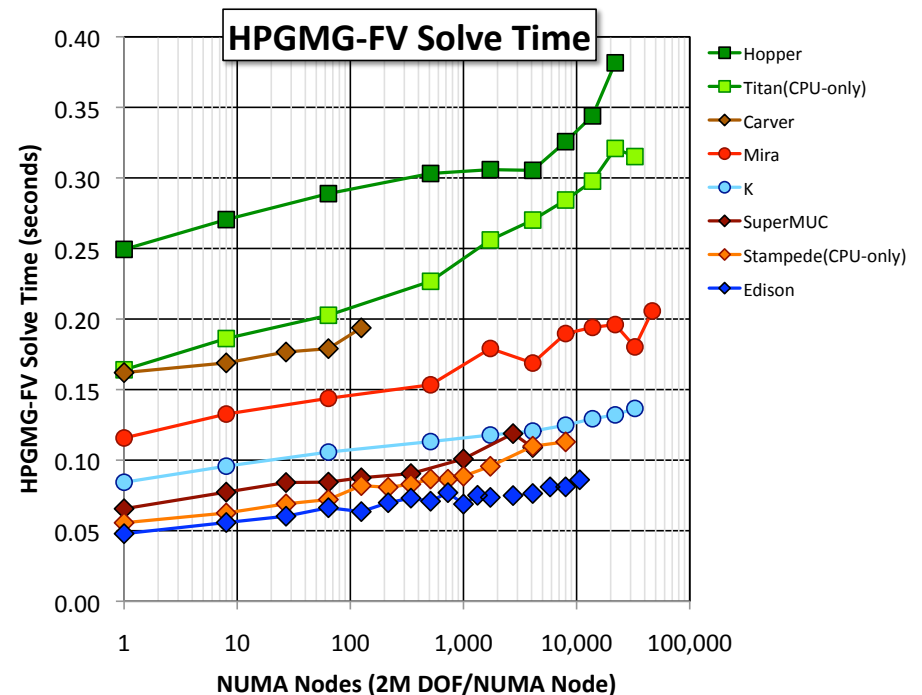
- K (Sparc VIIIfx) at RIKEN
 - less flops per proc than BGQ
 - more bandwidth per proc than BGQ = deliver better HPGMG-FV performance per node.
 - TOFU (6D) delivered similar (but smoother) scalability to BGQ





Fat Trees and Dragonfly

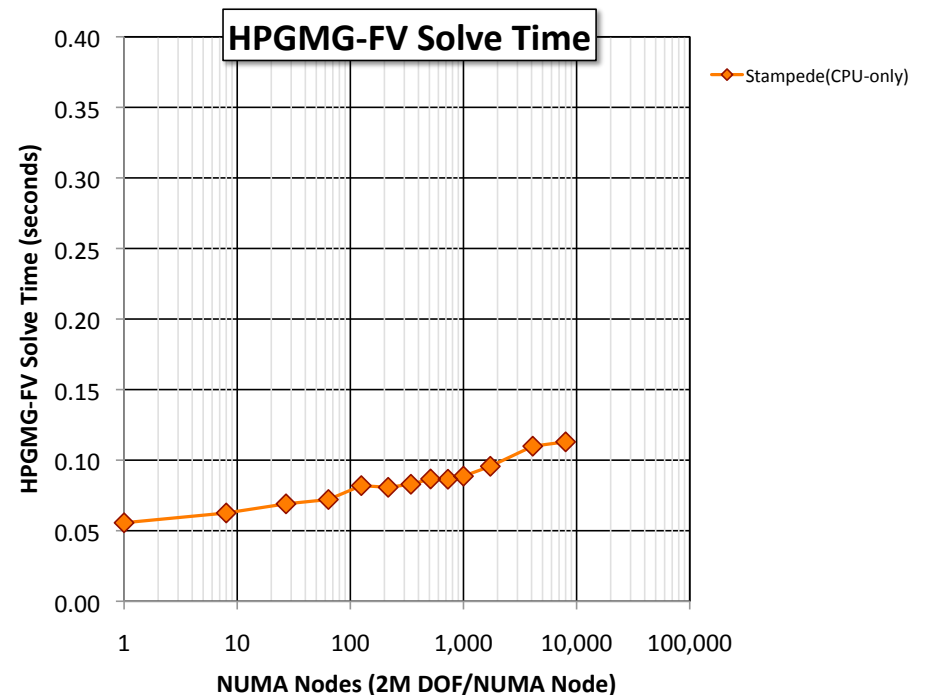
- Xeon processors (IVB, SNB, NHM) are common on the Top500 today
- Xeon performance defined by #cores, processor frequency, and memory frequency
- **Fat Trees saw degraded scaling beyond 1K sockets**
- XC30/Aries (Dragonfly), K (6D), and BGQ (5D) **continued to scale well from 1K-48K sockets**





Accelerators?

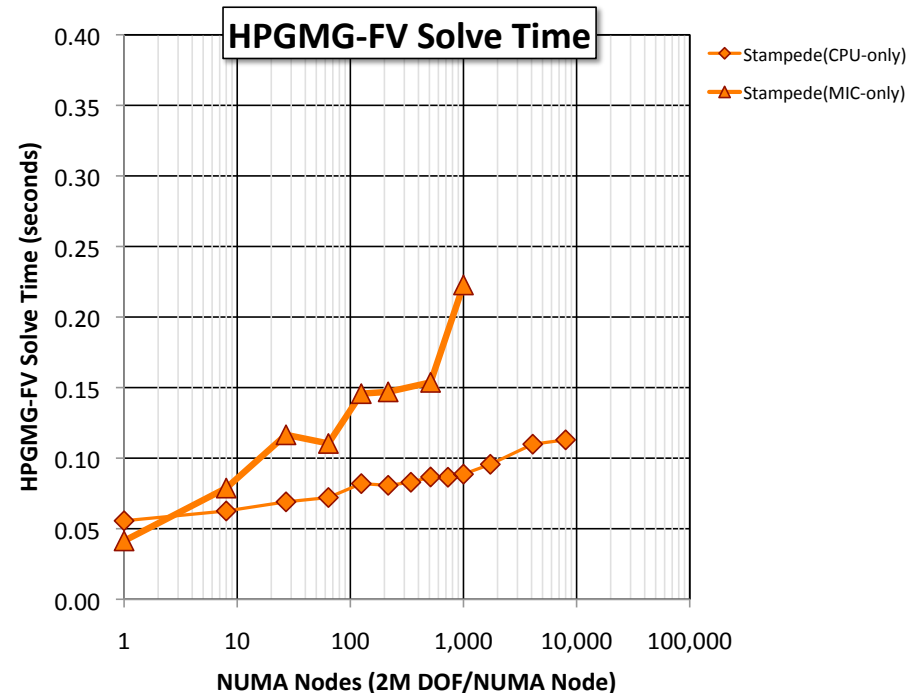
- Thus far, we have completely ignored accelerators like the Xeon Phi.
- Consider Stampede...
 - Xeon (SNB)
 - Fat Tree
 - Scaled reasonably well for this mid-sized problem
 - Each Stampede node also has a MIC
- How does HPGMG-FV on MIC (Xeon Phi) perform/scale?
 - using Stampede for CPU or MIC runs removes the network as a variable





MIC Scaling

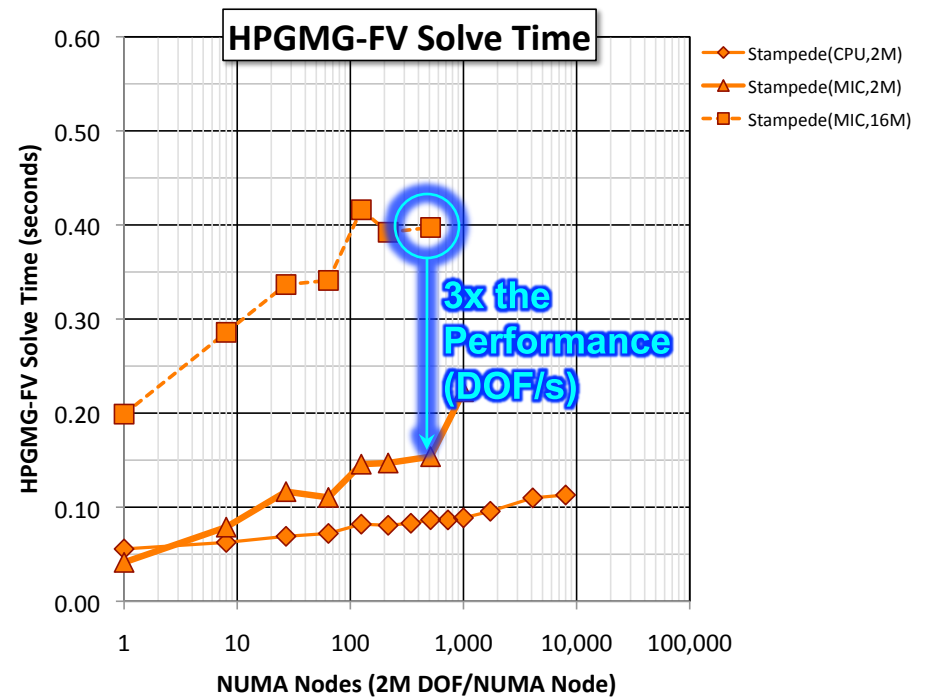
- **MIC can run HPGMG-FV without modification**
 - we tuned BLOCKCOPY_TILE_*
 - MIC delivers the best single node performance of any architecture
- However, with 2M DOF/proc, MIC did not weak scale well...
 - **very surprising as MIC used the same network as the CPU runs**
- Messaging overheads
 - **FMG sends $O(\log^2(M*P))$ messages**
 - nominally overhead is small compared to data movement.
 - However, overheads for MPI messaging on MIC can be >10x the overhead on CPUs





MIC Scaling

- When running on 512 MIC's, increasing the problem size by 8x **increases performance by 3x**
- Thus, larger problem sizes helped, but didn't rectify the issue





Maximum Performance

- We can increase the problem size on all machines to obtain maximum performance...

HPGMG-FV Rank	System	Site	Eq/s	Fraction of System	Parallelization MPI OMP	DOF per Process	Top500 Rank
1	Mira	Argonne	7.21E+11	100%	49152 64	16M	5
2	K	RIKEN	7.12E+11	73%	64000 8	2M	4
3	Edison	NERSC	3.85E+11	100%	131072 1	4M	18
4	Titan (CPU-only)	Oak Ridge	2.53E+11	88%	32768 8	16M	2
5	Stampede (CPU-only)	TACC	1.49E+11	64%	8192 8	2M	7
6	Hopper	NERSC	1.21E+11	86%	21952 6	2M	34
7	Piz Daint (CPU-only)	CSCS	1.02E+11	78%	4096 8	18M	6
8	SuperMUC	LRZ	7.13E+10	15%	2744 8	16M	12
9	Stampede (MIC-only)	TACC	2.16E+10	8%	512 180	16M	7
10	Peregrine (IVB-only)	NREL	1.08E+10	18%	512 12	2M	-
11	Carver	NERSC	1.35E+09	5%	125 4	2M	-
12	Babbage (MIC-only)	NERSC	8.24E+08	30%	27 180	16M	-

NOTES...

- We did not have the opportunity to run on >64K nodes or with >2M DOF on K.
- FV jobs >2744 nodes failed to launch on SuperMUC.
- Without a GPU implementation, GPU-accelerated machines were ranked lower.



What's Missing?

- MIC implementations / systems?
 - No data on TH2 (IVB-only and/or MIC-only)
 - No XC30's w/KNC (are Cray's overheads as high as Intel's?)
 - HPGMG-FV should be able to run in Symmetric Mode on Stampede (2CPU+1MIC)
 - One could write an offload version of HPGMG-FV for MIC that would sidestep some of the current communication inefficiencies.

- GPU implementations?
 - Will require a CUDA, OpenCL, or OpenACC port
 - Like MIC, HPGMG will challenge all aspects of GPU-accelerated systems.
 - However, there is the opportunity for truly heterogeneous implementations code is selectively run on either host or GPU

- Clouds?
 - Amazon EC2, Microsoft Azure, etc...
 - HPGMG would stress Cloud performance far more than other benchmarks.



HPGMG-FV

- In the last 6 months...
 - We wrote HPGMG-FV from scratch
 - Produced a high-performance, portable MPI+OpenMP implementation
 - Evaluated on a variety of systems (addressing a number of issues)
 - Demonstrated scalability to 64K nodes on K and 48K nodes on Mira

- In this talk, we examined systems which varied...
 - DRAM bandwidth
 - Processor Performance
 - Network Bandwidth/Topology
 - MPI overheads

- We demonstrated...
 - Improving any one aspect of the system showed limited performance benefits.
 - In order to significantly improve HPGMG-FV performance, one must improve multiple/all aspects of the system.



HPGMG-FE Performance

Jed Brown (ANL)
jedbrown@mcs.anl.gov

SC14 HPGMG BoF – November 19 2014 12:15PM



Acknowledgments

SC14 HPGMG BoF – November 19 2014 12:15PM



Acknowledgments

- Mitsuhsa Sato, Takenori Shimosaka (RIKEN)
- Muthu Baskaran (Reservoir Labs)
- Bill Barth, Dan Stanzione (TACC)
- Nick Wright, Katie Antypas, Helen He (NERSC)
- Ray Grout (NREL)
- Thomas Schulthess, Benjamin Cumming (CSCS)

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.



Thank You !

- Please visit <http://hpgmg.org>
- Fill out the Survey <http://hpgmg.org/survey>
- Please download and evaluate HPGMG
- Send us feedback
- Remember to sign up and post questions to the mailing list: HPGMG-Forum@hpgmg.org



Thank you

- Visit us on the web: hpgmg.org
- Download the code
- Take the short survey: hpgmg.org/survey
- Thanks to speakers
 - Mitsuhsa Sato
 - Muthu Baskaran
- Thank you for your attention
 - the HPGMG team
 - Mark Adams (LBNL)
 - Jed Brown (ANL, CU)
 - John Shalf (LBNL)
 - Brian Van Straalen (LBNL)
 - Erich Strohmaier (LBNL)
 - Sam Williams (LBNL)



Multigrid Tree++ & Nearest Neighbor Communication Patterns



FMG starts with accurate solve on coarsest grid

Usually local (ie, no message passing)

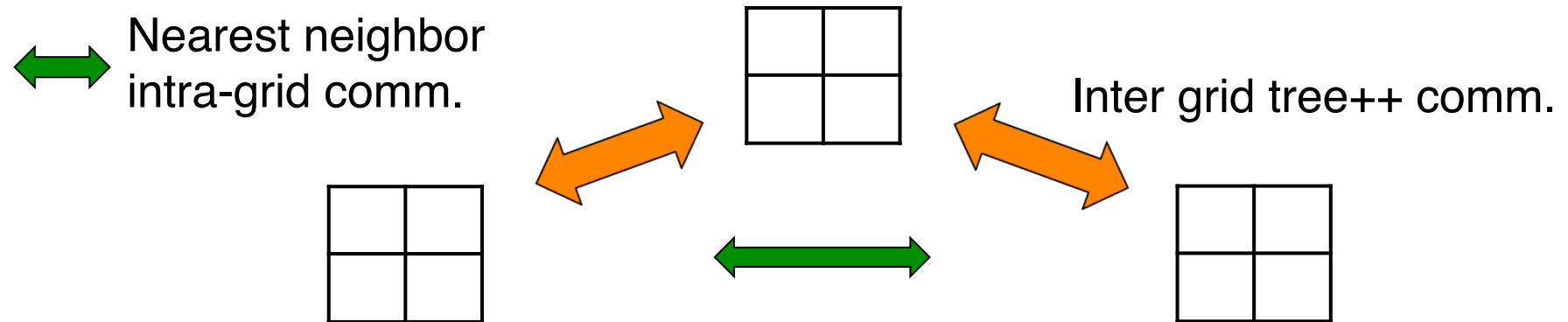
Very little parallelism

Still an $O(N)$ algorithm with polylog computational depth

Parallelism exponentially increases



Multigrid Tree++ & Nearest Neighbor Communication Patterns



Refine grid split processes, building tree

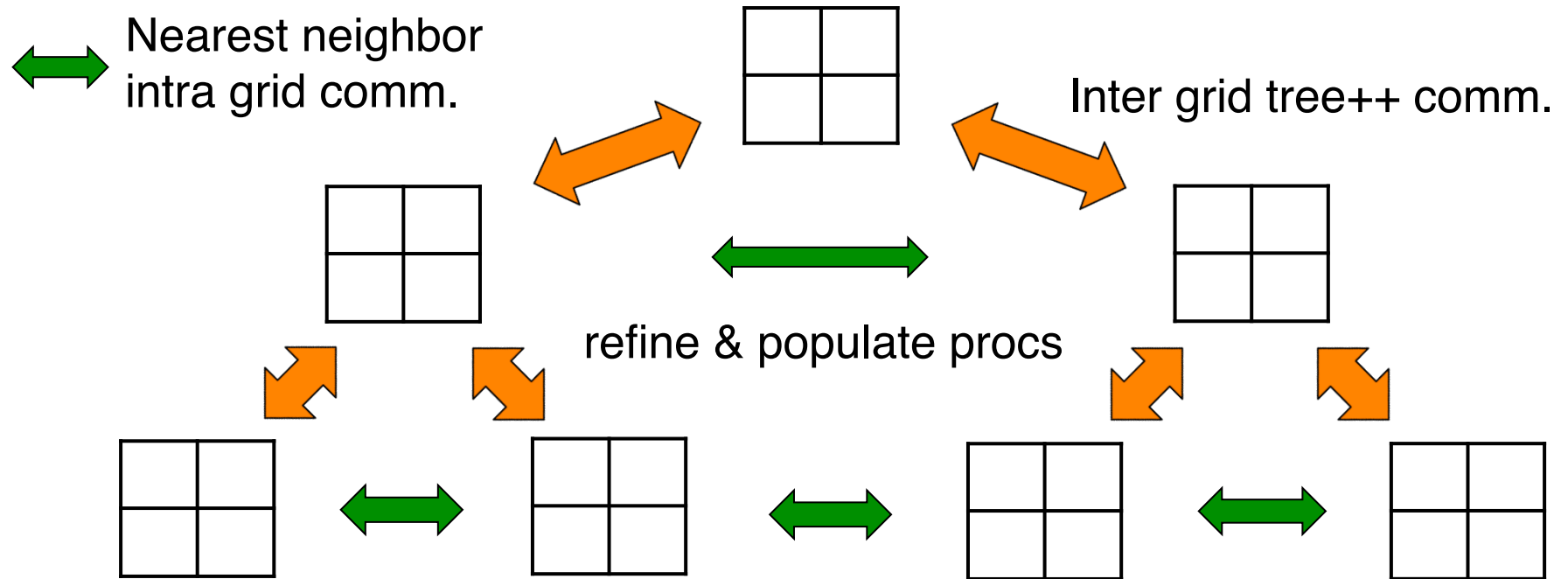
Two types of inter-grid transfers: restriction & prolongation
intra-grid communication, nearest neighbor exchange



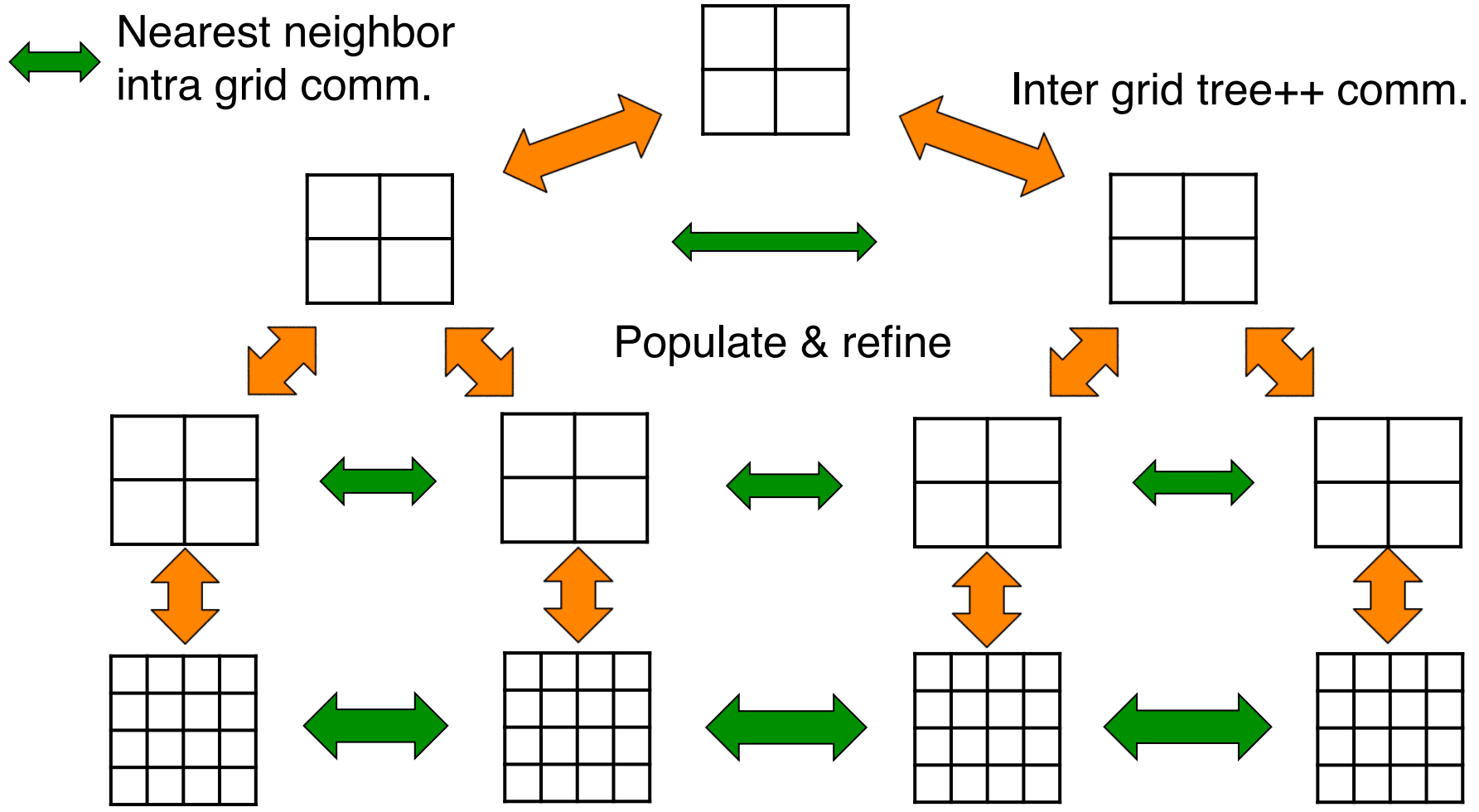
Multigrid Tree++ & Nearest Neighbor Communication Patterns



FMG goes back to coarse grid after each new level
Results in $O(\log^2(N))$ computational depth or PRAM complexity
theoretical minimum is $O(\log(N))$
 $O(N)$ work and memory movement – time & energy efficient



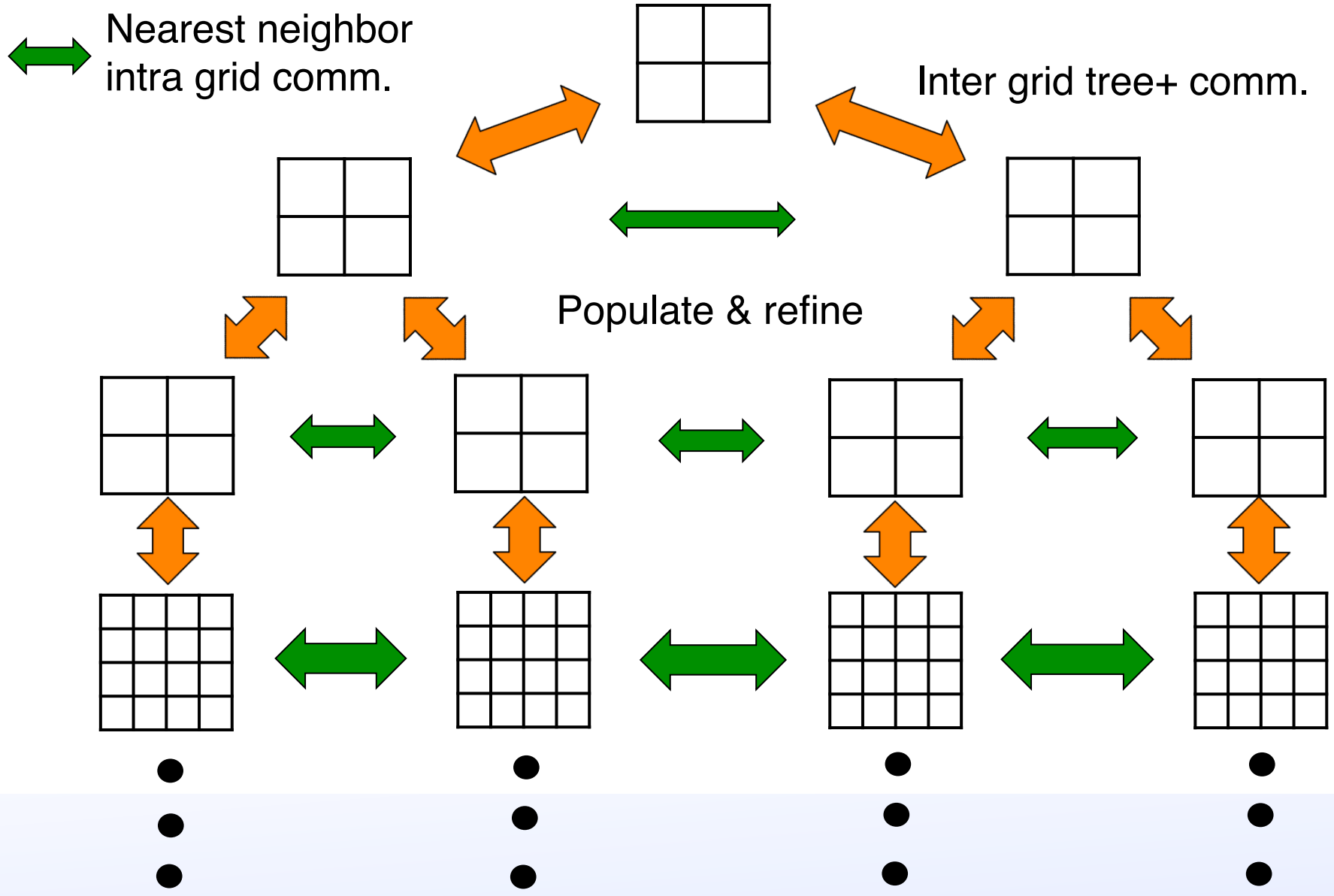
Down, make new grid, up, down, building up a tree ...



Processes fully populated – continue local refinement







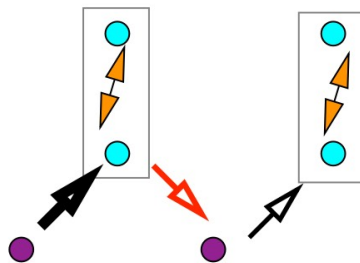
Multigrid Tree+ & Nearest Neighbor Communication Patterns





Parallel full multigrid – 1D, 2 processes

-  Intra-grid communication
-  Inter-grid high order prolongation
-  Inter-grid prolongation communication
-  Inter-grid restriction

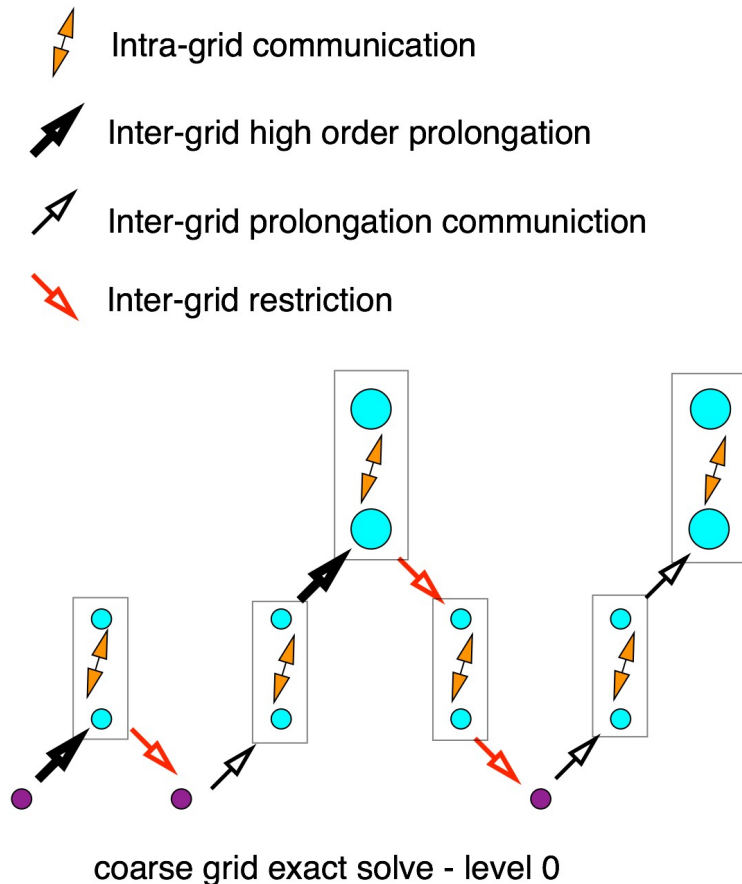


coarse grid exact solve - level 0

- Start at coarsest grid
 - Exact solve on small grid
 - Usually local
 - Little parallelism
- Refine grid & populate processes
 - Exponentially increasing parallelism
- Two types of inter-grid communication
 - **Restriction** & prolongation
- **Intra-grid communication, nearest neighbor exchange**



Parallel full multigrid – 2 processes



- Continue populating processes
- Until desired local grid size
 - Continue with local refinement
- Results in $O(N)$ work/data algo
 - Time & energy efficient
- $O(\log^2(N))$ PRAM complexity
 - computational depth, calls to kernels, communication steps, ...
- FFT provably better PRAM
 - $O(\log(N))$ PRAM complexity
 - $O(\log(N))$ work – tiny constant
 - *Not just asymptotically exact*
 - Lots of data movement
 - FFT not like anything else
- Multigrid hierarchy more representative many apps & algos

