High Performance Geometric Multigrid: A Supercomputer Benchmark & Metric

Mark Adams and Sam Williams
with
Jed Brown, John Shalf, Erich Strohmaier, Brian Van Straalen

ISC, Frankfurt Germany, 21 June 2016

HPGMG Project Design Goals

1. General benchmarking efforts and provide compact benchmark codes to aid engineers & centers to design & deploy well balanced machines
   - Benchmark code with sensitivities to machine metrics that correlate well with applications
     - Presented some data from LLNL at ISC '14
2. HPGMG-FV: Supercomputer ranking metric
   - A specification and packaging built from Sam's finite volume code base

HPGMG

HPGMG-FV design

- Compact stand alone: C + MPI (+OMP, CUDA)
- Conceptually simple: solve Ax = b with multigrid
- Finite Volume, 3D non-constant coef. Laplacian
- Non-iterative (full) geometric multigrid solver
- Metric: equations (N) solved / sec
  - Map to flops/sec: 1200N (not exact nor well defined)
    - HPL uses $2N^3$ map

Stable HPGMG-FV metric

- This year transitioned to 4th order accurate discretization
- First "official" 4th order list, stable specification
- Sensitive to more machine parameters than 2nd order:
  1. MPI message rates: 3x messages/op-apply
  2. MPI bandwidth: 2x message sizes (two ghost cell layers)
  3. More pressure on cache: ~2x working set size
  4. Large stencil, many data streams
  5. Wide range of message sizes: coarse grids are smaller
  6. Full MG: more small messages (more coarse grid visits)
- More floating point intensive (flop/byte ~1)
  - With respect to "book-end" strategy of HPL-HPCG
    - We try to be in the middle

HPGMG

Maintainable, durable (+ dynamic range)

- Experience TOP500, HPL, other benchmarking initiatives
  - Simplicity key
- Need unambiguous specification
  - Can not afford to micro-manage or adjudicate each submission
- Dynamic range, or strong scaling, important to applications
  - But Constraints interfere with incorporating dynamic range in metric

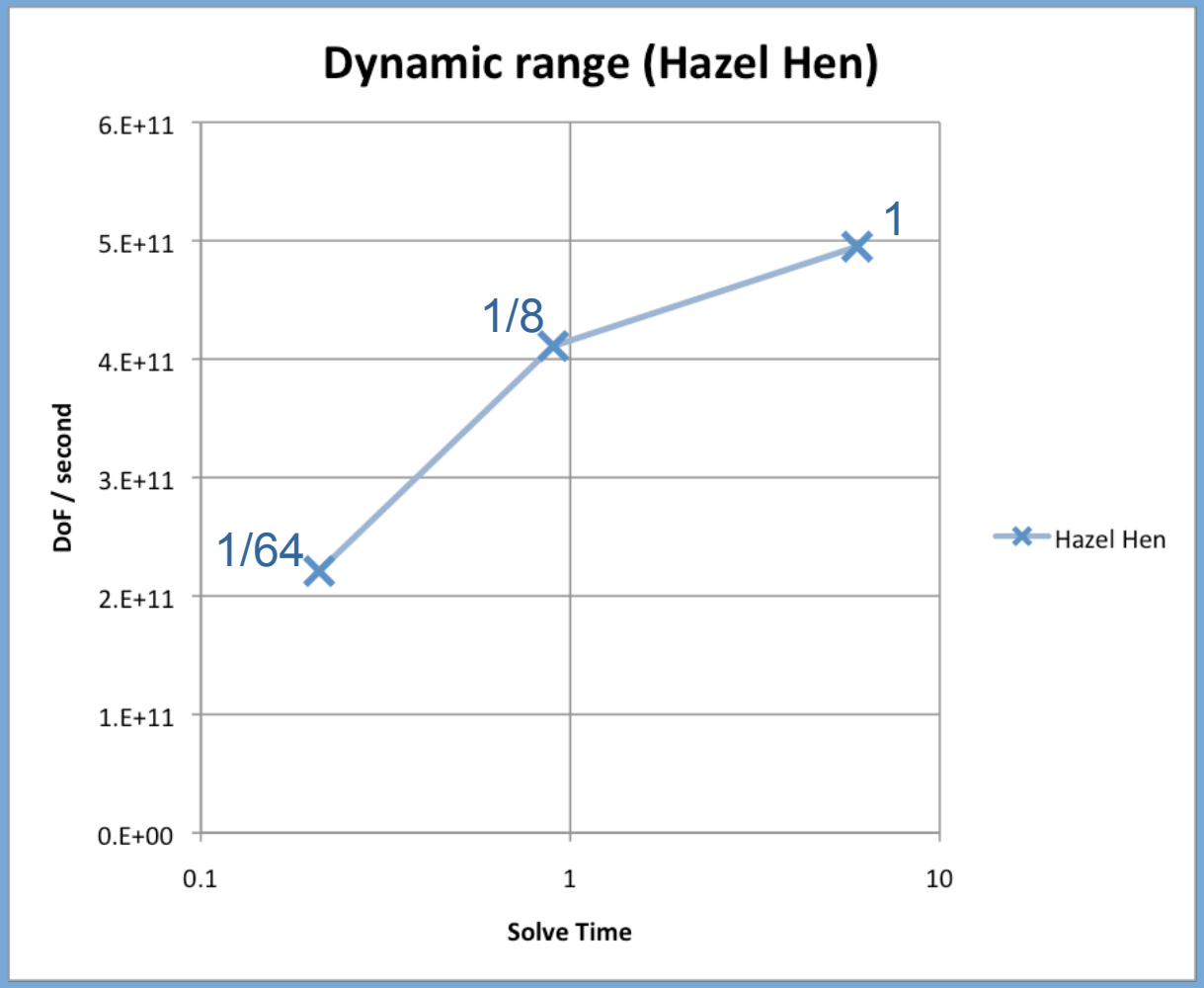
Criterion	HPGMG-FV	HPL	HPCG
Architecture Free	✔	✔	Minimum 25% “main memory” usage
Scale/PM Free	✔	✔	Gauss-Seidel only on “sub domains”
Math/algorithm fully specified	✔	✔ (!Str)	Vertex order not specified for G-S, affects convergence, affects metric
Dynamic range	1, 1/8, 1/64	$R_{½}$	

HPGMG

HPGMG-FV ranking, June '16

#	Site	Sys	Arch	10^9 1*	DOF 1/8	Se 1/64	MPI *the	OMP metric	#GPU	# HPL	#HP CG
1	DOE/ ANL/USA	Mira	IB I- BGQ	500	313	107	9K	64	0	6	~6
		Mira	ase	395	286	107	49 K	64	0		
2	HLRS/ Germany	Hazel Hen	Cray XC40	495	411	221	15	12	0	9	~7
3	DOE/ ORNL/US	Titan	Cray XK7	440	163	39	16k	4	1	3	~4
4	KAUST/ SA	Sha. II	Cray C40	326	287	175	12	16	0	10	~10
5	DOE/ NER/USA	Edison	Cray X 30	296	246	127	1 K	12	0	49	~16
6	CSCS Swiss	Piz Daint	Cra XC30	153	69	19	4K	8	1	8	~9
8	HLRS/G	NEC	SX- AC		1.8	5	256	1	0	-	-

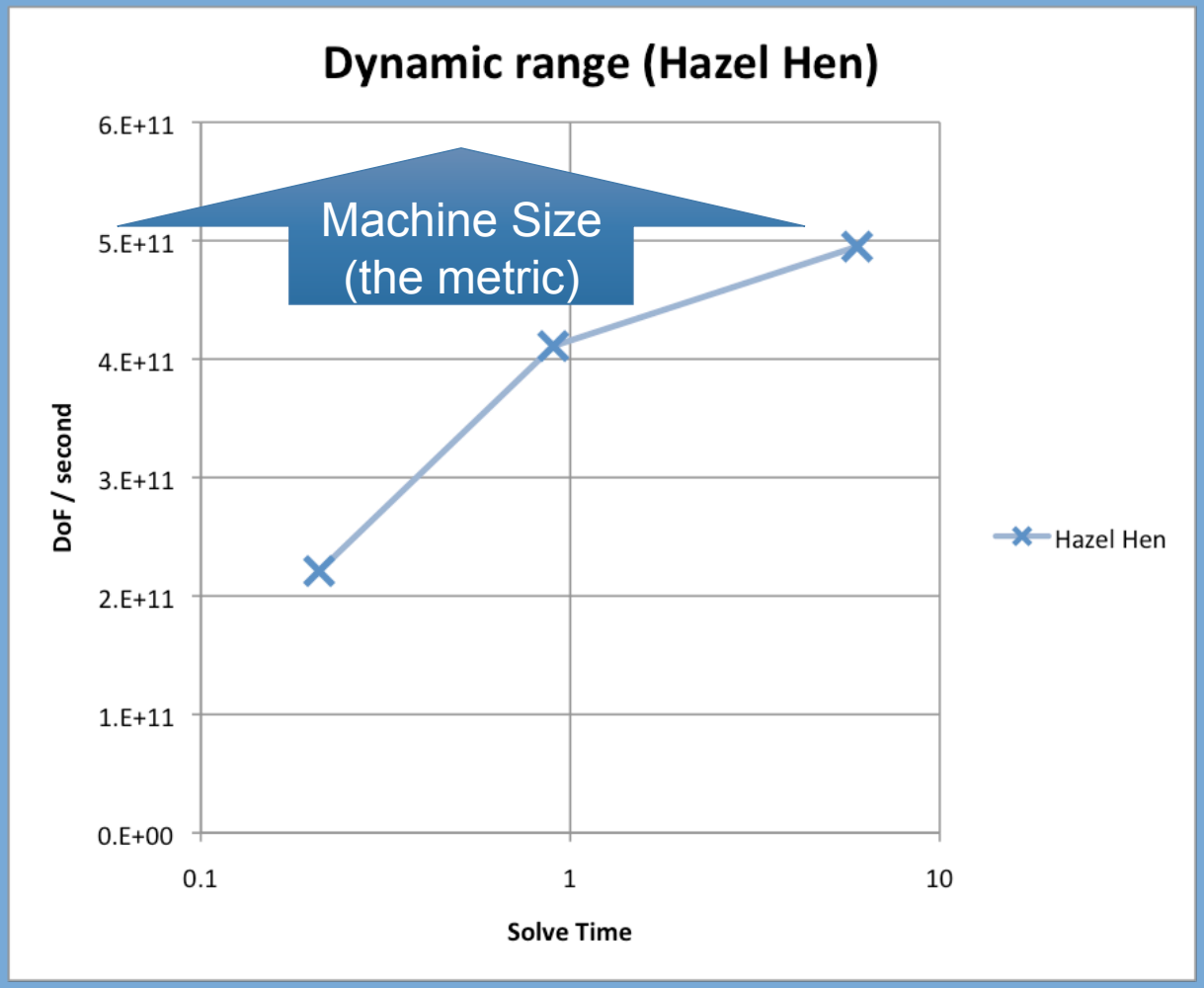
Turn around time: DoF/sec vs. Time



- Dynamic range:
  - Same concurrency
  - Reduce problem size
  - Flat lines perfect
- Strong scaling:
  - Same problem size
  - Increase concurrency
- Weak scaling:
  - Increase both

HPGMG

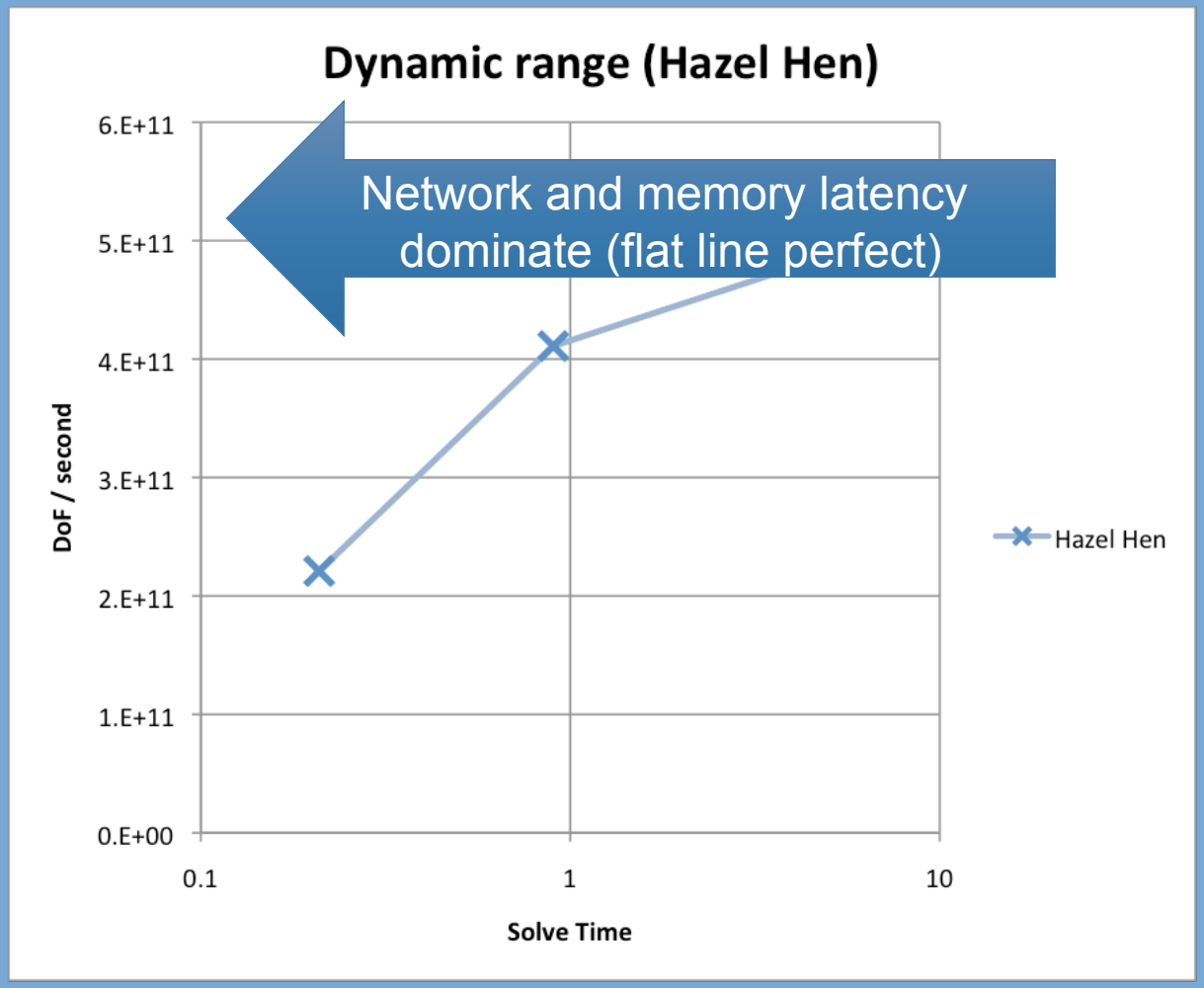
Turn around time: DoF/sec vs. Time



- Dynamic range:
  - Same concurrency
  - Reduce problem size
  - Flat lines perfect
- Strong scaling:
  - Same problem size
  - Increase concurrency
- Weak scaling:
  - Increase both

HPGMG

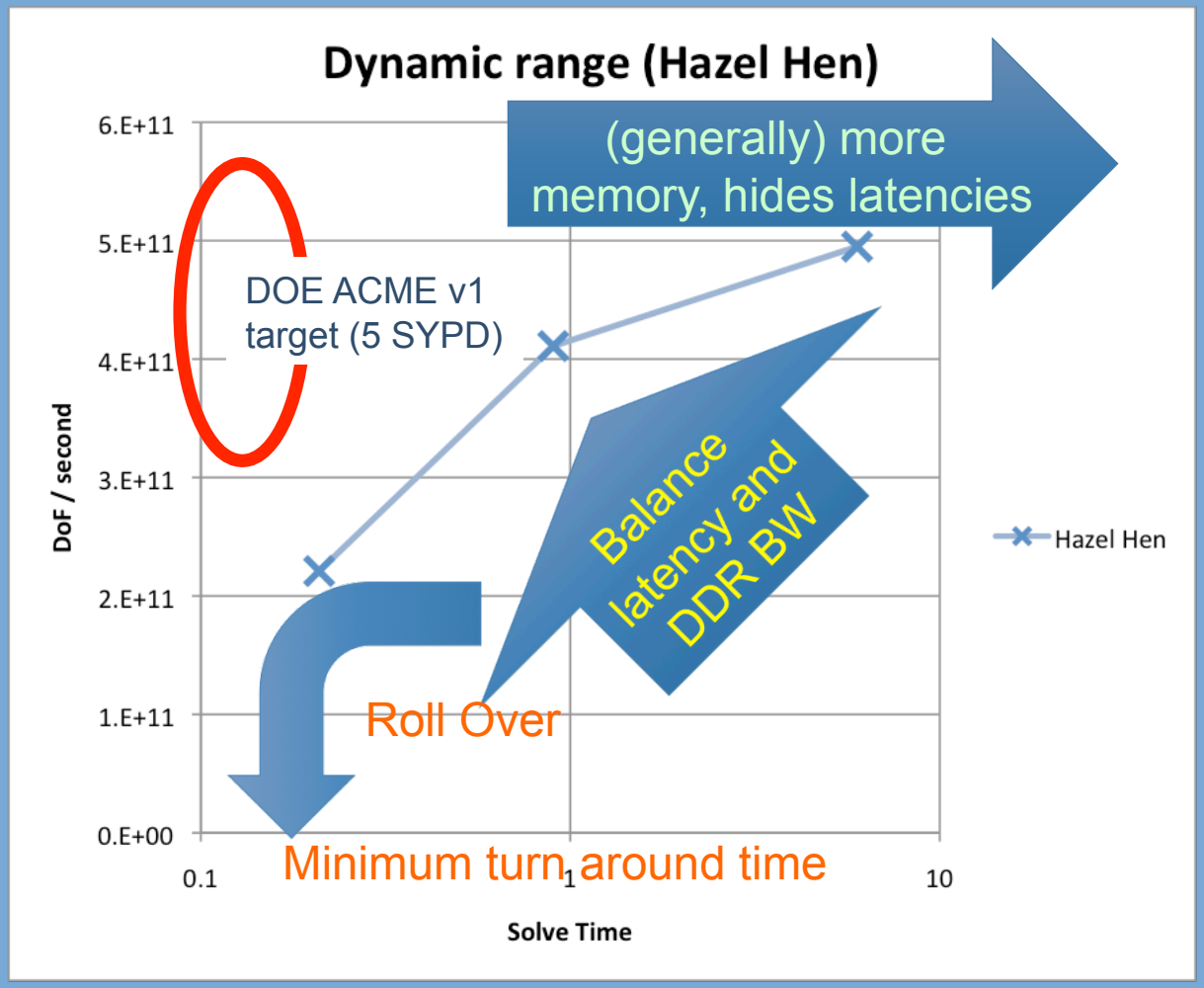
Turn around time: DoF/sec vs. Time



- Dynamic range:
  - Same concurrency
  - Reduce problem size
  - Flat lines perfect
- Strong scaling:
  - Same problem size
  - Increase concurrency
- Weak scaling:
  - Increase both

HPGMG

Turn around time: DoF/sec vs. Time



- STREAM proxy (usually)
  - On right
- Balance of latencies to memory capacity
- Roll-over
  - On left

HPGMG

Turn around time: DoF/sec vs. Time

- Dynamic range:
  - Same concurrency
  - Reduce problem size
- Strong scaling:
  - Same problem size
  - Increase concurrency
- Weak scaling:
  - Increase both

HPGMG

Consider a dynamic range metric

- Consider the sum for the three data points (dof/sec) and not the maximum
  - Convergent series, will roll over eventually

1. Hazel Hen 1130 (x10^9 dof/second)
2. Mira 920
3. Shaheen II 788
4. Edison 669
5. Titan 642
6. Piz Daint 240

Community buy-in & related efforts

1. Vendor buy-in
   - Nvidia optimized 2nd & 4th order for Kepler & Pascal
     - MPI + OpenMP + CUDA
   - Intel recently started KNL optimizing (Sam already started)
2. Sam Williams' GMG codes used in several efforts
     - HPGMG-FV is the ranking metric instantiation …
   - DOE FastForward2: selected as a 1/6 proxy apps
   - DOE DEGAS project:
     - investigated one-sided UPC++ implementations
     - PYGMG (python version HPGMG) with
     - SEJITS (selective just in time specialization)
   - DOE DTEC project: Halide stencil DSL of GMG
   - DOE Traleika Glacier project (Intel)
     - OCR dynamic task runtime (3 versions, low to higher level)
   - Other groups:
     - UCB, MIT, Rice, ORNL, SDSC, Riken, HLRS, KAUST, NREL

Submissions SC16 (hpgmg.org)

- Welcome submissions, collaborations & questions
  - Contact hpgmg-forum@hpgmg.org
  - Visit hpgmg.org
  - Submission instructions: hpgmg.org/fv
- Next list release at SC16 BoF, more architectures:
  - Accelerators: TH-2A, …
  - Cluster of commodity processors, infiniband
    - SuperMUC (all 17 islands, only have data with 4), …
  - K: want full scale 4^{th} order data and G-S optimization
  - More, THL …
- Our repositories
  - OpenMP: bitbucket.org/hpgmg/hpgmg
  - CUDA: bitbucket.org/nsakharnykh/hpgmg-cuda

HPGMG

Thank you

https://hpgmg.org/
https://bitbucket.org/hpgmg/hpgmg
Mark Adams
Jed Brown
John Shalf
Erich Strohmaier
Brian Van Straalen
Sam Williams

ISC, Frankfurt Germany, 21 June 2016

HPGMG

…. More back ground

4 architectures on list

1) Nvidia GPUs

#	Site	Systm	Arch.	10^9 x h*	DOF/ 2h	sec 4h	MPI	OMP	GPU	HPL rank
1	DOE/ANL/ USA	Mira	IBM-BGQ	**500**	313	107	49K	64	0	5
		Mira	(Base)	395	286	107	49K	64	0	
2	HLRS/ Germany	Hazel Hen	Cray XC40	**495**	411	221	15K	12	0	8
3	DOE/ ORNL/US	Titan	Cray XK7	**440**	163	39	16K	4	1	2
4	KAUST/SA	Sha. II	Cray XC40	**326**	287	**175**	12K	16	0	9
5	DOE/NER/ USA	Edison	Cray XC30	**296**	246	127	11K	12	0	40
6	CSCS Swiss	Piz Daint	Cray XC30	**153**	69	19	4K	8	1	7
8	HLRS/ Germany	NEC	SX-ACE	**3.3**	1.8	.75	256	1	0	-

* *The metric*

2) Light-weight in-order cores

#	Site	Systm	Arch.	10^9 x h*	DOF/ 2h	sec 4h	MPI	OMP	GPU	HPL rank
1	DOE/ANL/ USA	Mira	IBM-BGQ	**500**	313	107	49K	64	0	5
		Mira	(Base)	395	286	107	49K	64	0	
2	HLRS/ Germany	Hazel Hen	Cray XC40	**495**	411	221	15K	12	0	8
3	DOE/ ORNL/US	Titan	Cray XK7	**440**	163	39	16K	4	1	2
4	KAUST/SA	Sha. II	Cray XC40	**326**	287	**175**	12K	16	0	9
5	DOE/NER/ USA	Edison	Cray XC30	**296**	246	127	11K	12	0	40
6	CSCS Swiss	Piz Daint	Cray XC30	**153**	69	19	4K	8	1	7
8	HLRS/ Germany	NEC	SX-ACE	**3.3**	1.8	.75	256	1	0	-

* *The metric*

3) Cray – Xeon processors

#	Site	Systm	Arch.	10^9 x h*	DOF/ 2h	sec 4h	MPI	OMP	GPU	HPL rank
1	DOE/ANL/ USA	Mira	IBM-BGQ	**500**	313	107	49K	64	0	5
		Mira	(Base)	395	286	107	49K	64	0	
2	HLRS/ Germany	Hazel Hen	Cray XC40	**495**	411	221	15K	12	0	8
3	DOE/ ORNL/US	Titan	Cray XK7	**440**	163	39	16K	4	1	2
4	KAUST/SA	Sha. II	Cray XC40	**326**	287	**175**	12K	16	0	9
5	DOE/NER/ USA	Edison	Cray XC30	**296**	246	127	11K	12	0	40
6	CSCS Swiss	Piz Daint	Cray XC30	**153**	69	19	4K	8	1	7
8	HLRS/ Germany	NEC	SX-ACE	**3.3**	1.8	.75	256	1	0	-

* *The metric*

4) NEC vector architecture

#	Site	Systm	Arch.	10^9 x h*	DOF/ 2h	sec 4h	MPI	OMP	GPU	HPL rank
1	DOE/ANL/ USA	Mira	IBM-BGQ	**500**	313	107	49K	64	0	5
		Mira	(Base)	395	286	107	49K	64	0	
2	HLRS/ Germany	Hazel Hen	Cray XC40	**495**	411	221	15K	12	0	8
3	DOE/ ORNL/US	Titan	Cray XK7	**440**	163	39	16K	4	1	2
4	KAUST/SA	Sha. II	Cray XC40	**326**	287	**175**	12K	16	0	9
5	DOE/NER/ USA	Edison	Cray XC30	**296**	246	127	11K	12	0	40
6	CSCS Swiss	Piz Daint	Cray XC30	**153**	69	19	4K	8	1	7
8	HLRS/ Germany	NEC	SX-ACE	**3.3**	1.8	.75	256	1	0	-

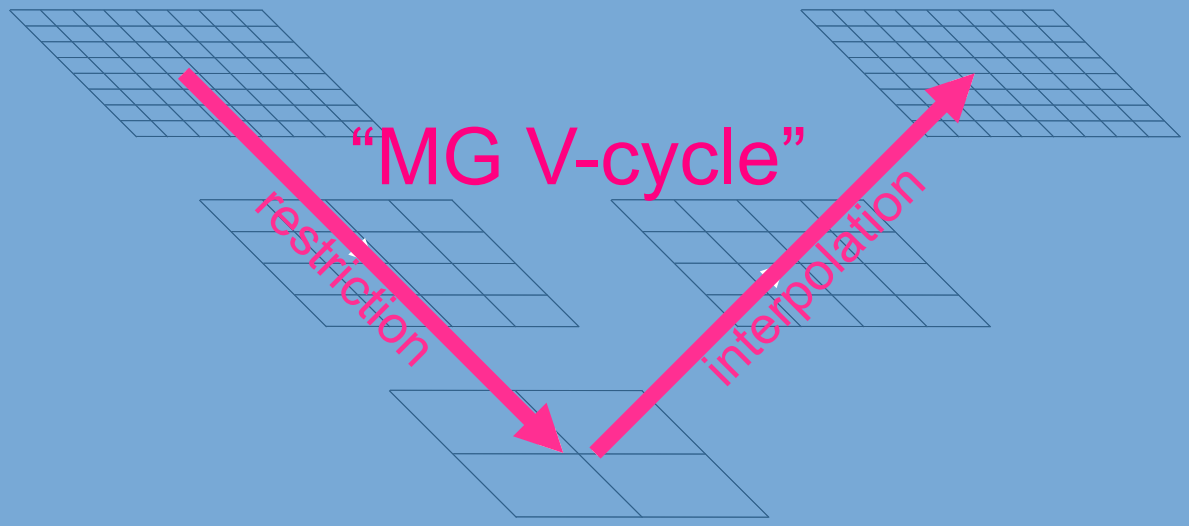
Dynamic range, DDR BW, cache size

#	Site	Systm	Arch.	10^9 x h*	DOF/ 2h	sec 4h	MPI	OMP	GPU	HPL rank
1	DOE/ANL/ USA	Mira	IBM-BGQ	**500**	313	107	49K	64	0	5
		Mira	(Base)	395	286	107	49K	64	0	
2	HLRS/ Germany	Hazel Hen	Cray XC40	**495**	411	221	15K	12	0	8
3	DOE/ ORNL/US	Titan	Cray XK7	**440**	163	39	16K	4	1	2
4	KAUST/SA	Sha. II	Cray XC40	**326**	287	**175**	12K	16	0	9
5	DOE/NER/ USA	Edison	Cray XC30	**296**	246	127	11K	12	0	**40**
6	CSCS Swiss	Piz Daint	Cray XC30	**153**	69	19	4K	8	1	7
8	HLRS/ Germany	NEC	SX-ACE	**3.3**	1.8	.75	256	1	0	-

* *The metric*

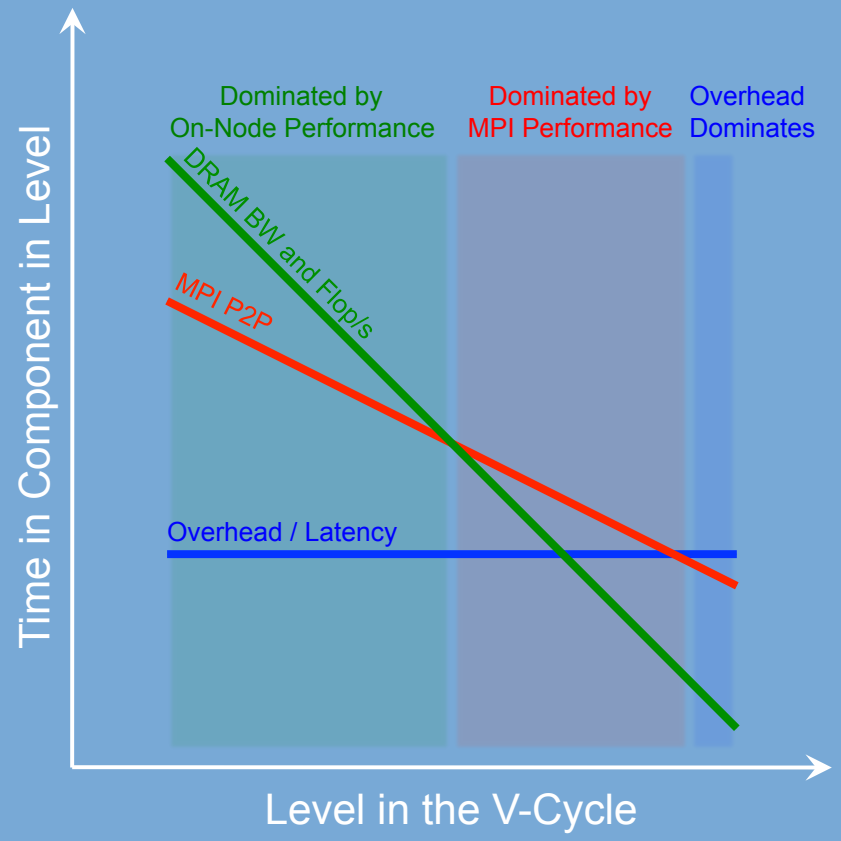
Geometric Multigrid

- Extremely fast/efficient…
  - O(N) computational complexity (#flops)
  - O(N) DRAM data movement (#bytes)
  - O($N^{0.66}$) MPI data movement



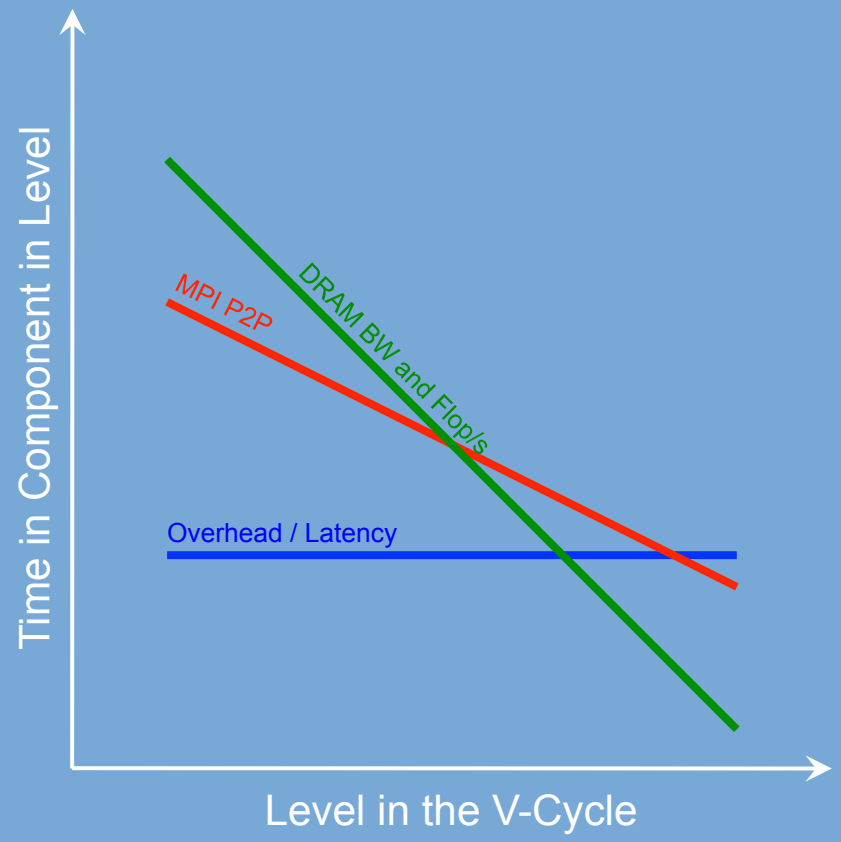
Ideal Performance

- 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 (MPI when it matters)
- 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?

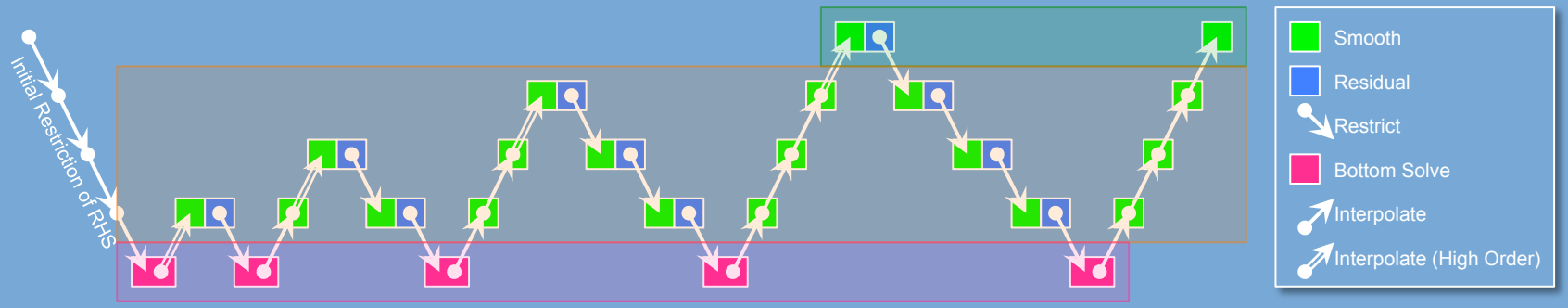
- If one just increases DRAM bandwidth by 10x, then the code becomes increasingly **dominated by MPI P2P communication**
- If one improves just DRAM and MPI bandwidth, the code will eventually be **dominated by CUDA, OpenMP, and MPI overheads**.
- Unfortunately, the overheads are hit O(logN) times.
- 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

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

HPGMG-FV detailed timing….

Time within a MG level by function

Total Time by function

box dimension	0 128^3	1 64^3	2 32^3	3 16^3	4 8^3	5 8^3	6 8^3	7 4^3	8 2^3	9 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	0.000000	0.001907	0.001907
BLAS1	0.004649	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

Time in an operation by level

smooth on finest level is only 38% of solve time

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
```

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.

HPGMG

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

Cell-Centered MG

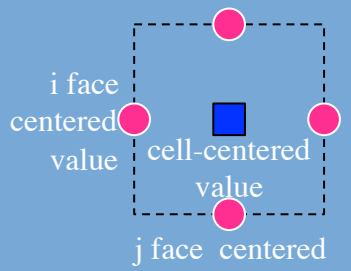
- Values can represent…
  - cell averages (cell-centered)
  - face averages (face-centered)

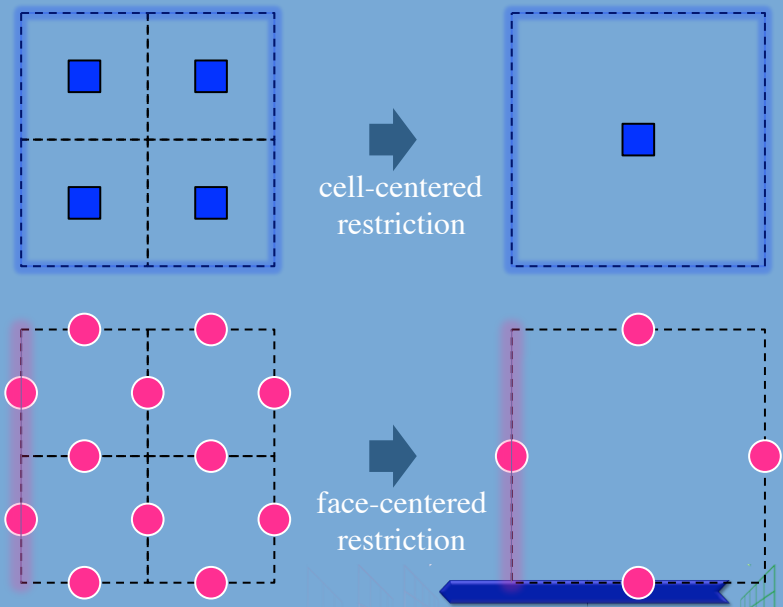
❖ Restriction/Prolongation can be either cell- or face-centered.

❖ In piecewise constant restriction, coarse grid elements are the average value of the region covered by fine grid elements

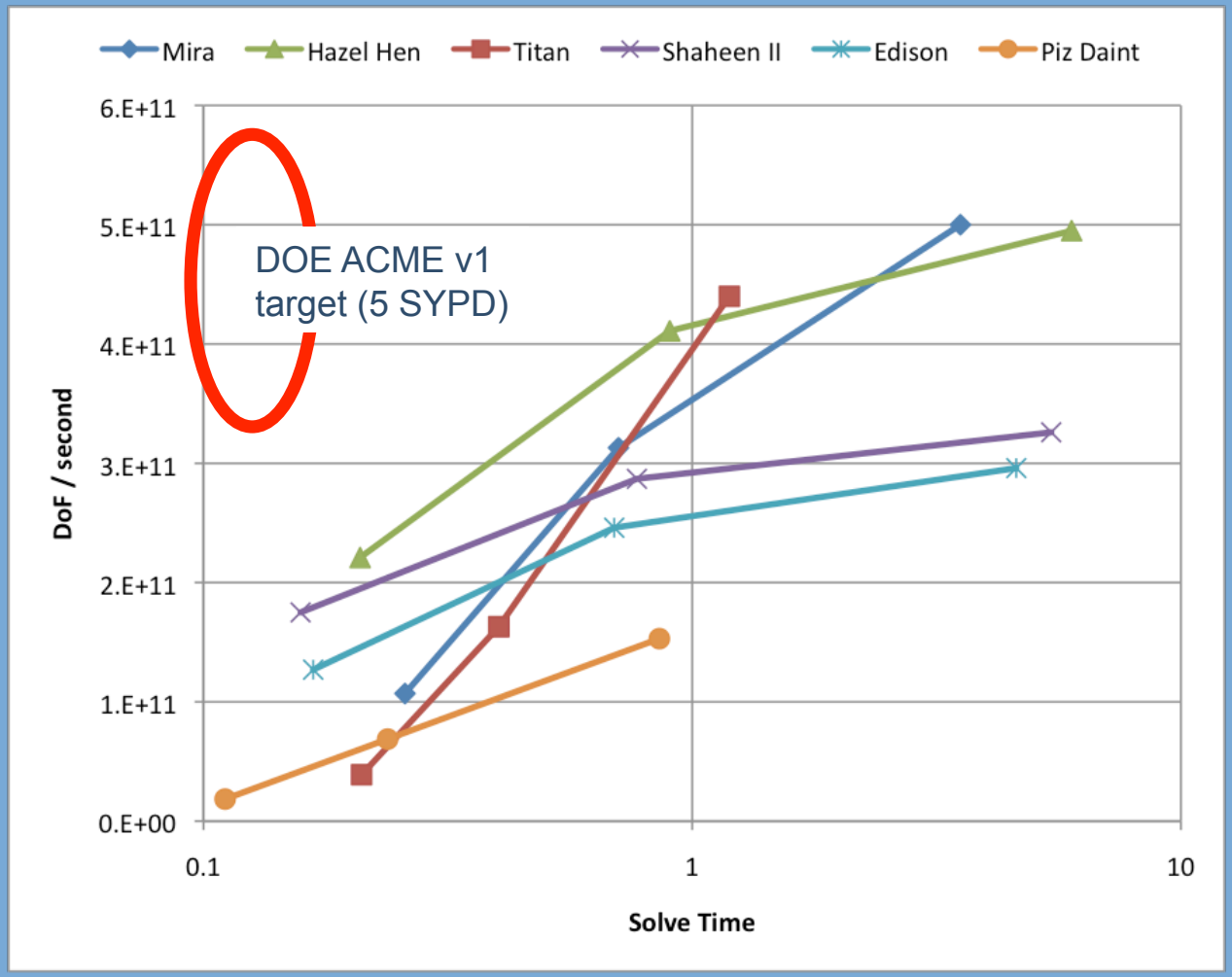
❖ Solutions variables are usually cell-centered, **but boundary values exist on cell faces (face-centered)**

- 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.





Strong scaling: DoF/sec vs Time



- Turn around time
- Flat lines = perfect strong scaling

HPGMG