Scalable Simulations of Multiscale Physics

Paul Fischer,Katie Heisey, Misun Min
and CESAR* Team
Argonne National Laboratory

*DOE Center for Exascale Simulation of Advanced Reactors

Overview

- Look at scalability limits for grid-based PDE solvers.
- Outward looking:
  - Internode vs. Intranode
- Current situation is not encouraging:
  - End of Moore's Law = End of running faster ☹
- Finer granularity: $n/P \rightarrow 0$ is key.

A PDE Example: Incompressible Navier-Stokes Eqns.

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla p + \frac{1}{Re} \nabla^2 \mathbf{u}$$

$$\nabla \cdot \mathbf{u} = 0$$

- Key algorithmic / architectural issues:
  - Unsteady evolution implies many timesteps, significant reuse of preconditioners, data partitioning, etc.
  - Div u = 0 implies long-range global coupling at each timestep, multiple time-scales → iterative solvers; communication intensive
  - Small dissipation → large number of scales → large number of gridpoints for high Reynolds number Re

Examples: Highly Unstructured Graphs

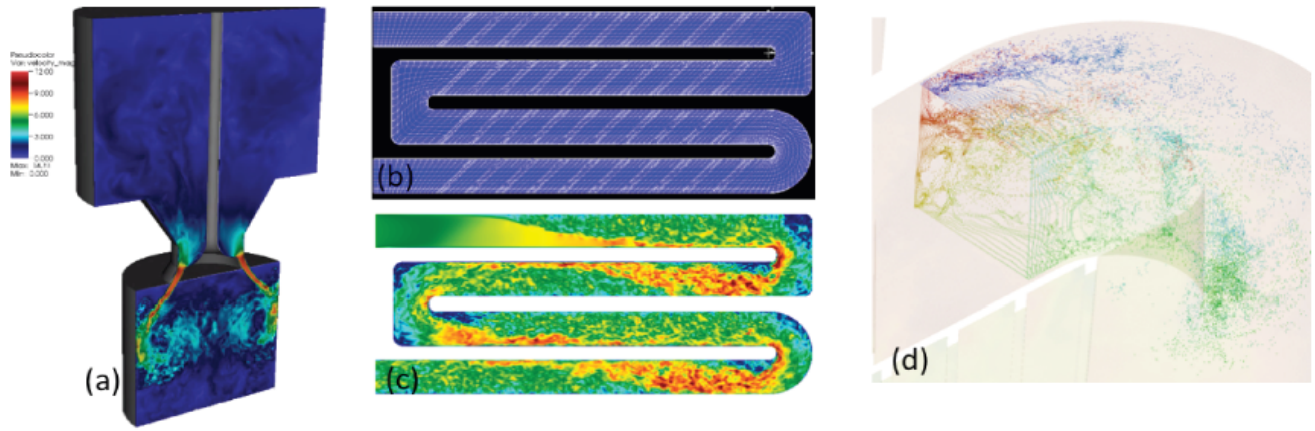


Fig. 3 Recent SE flow simulations: (a) velocity distribution during the intake stroke in an SE-ALE simulation for the benchmark cylinder configuration of [64], (courtesy M. Schmitt, ETHZ); (b) model geometry of Booten et al. for a ribbed turbine-blade cooling passage [29]; (c) snapshot of velocity magnitude in passage midplane; (d) close up of Lagrangian particles in rounded turn of Booten geometry.

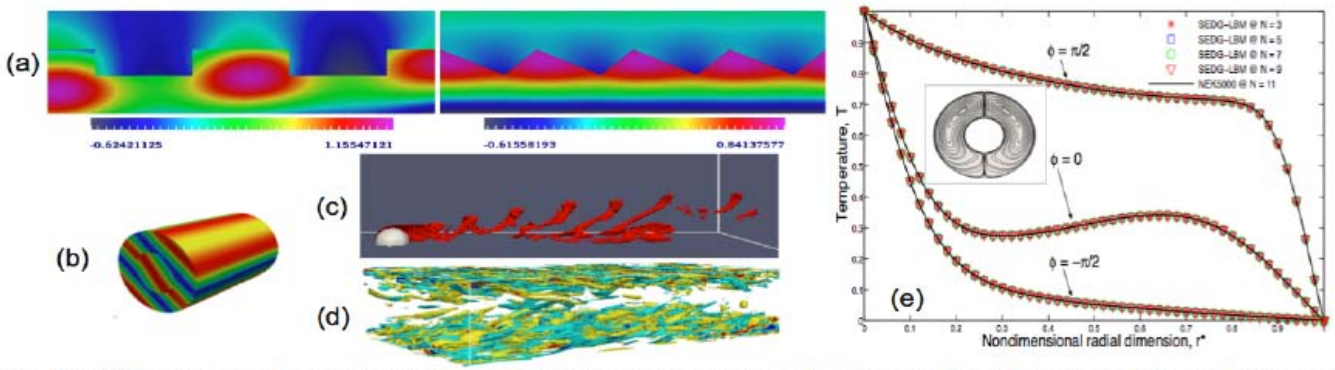


Fig. 5 (a) SE quasi-periodic scattering solutions for oblique incidence in double-layer media with square and sawtooth grooves. SE transparent boundary operator is applied on the top and bottom boundaries; (b) SEDG Schrödinger solution in layered cylinders; (c) SEDG lattice Boltzmann (LB) method: vortices contour for flows past a hemisphere with $Re = 5000$ and $Ma = 0.01$; (d) SEDG-LB turbulent model for channel flows; and (e) SEDG thermal LB model: temperature profiles at different angles for natural convection flows in an annulus, validated with Nek5000 results.

Influence of Scaling on Discretization

Large problem sizes enabled by peta- and exascale computers allow propagation of small features (size ¸) over distances L >> ¸. If speed ~ 1, then t_{final} ~ L/ ¸ >> 1.

- Dispersion errors accumulate linearly with time:

  error ~ |*correct speed – numerical speed*| * *t* (for each wavenumber)

  → $error_{t\_final}$ ~ (*L* / ¸) * | numerical dispersion error |

- For fixed final error ε_ϕ, require numerical dispersion error ~ (¸/L)ε_f, << 1.

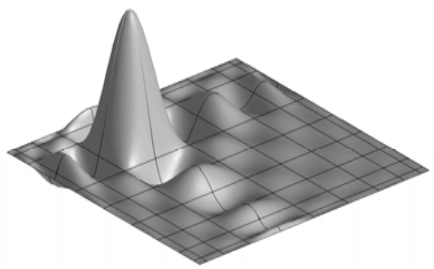
High-order methods can efficiently deliver small dispersion errors.

(Kreiss & Oliger 72, Gottlieb et al. 2007)

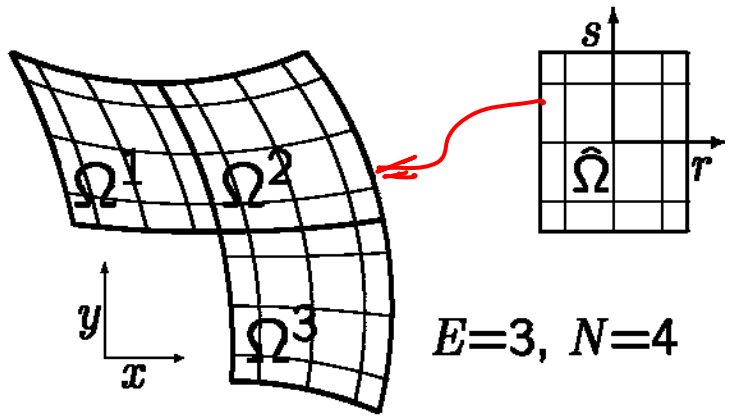
Spectral Element Method

(Patera 84, Maday & Patera 89)

- Variational method, similar to FEM, using *GL* quadrature.
- Domain partitioned into E high-order hexahedral elements
- Trial and test functions represented as N th-order tensor-product polynomials within each element. (N ~ 4 -- 15, typ.)
  - $n \sim EN^3$ gridpoints in 3D
  - Fast tensor-product-based operator evaluation: $O(n)$ storage, $O(nN)$ work
- Converges *exponentially fast* with N for smooth solutions.



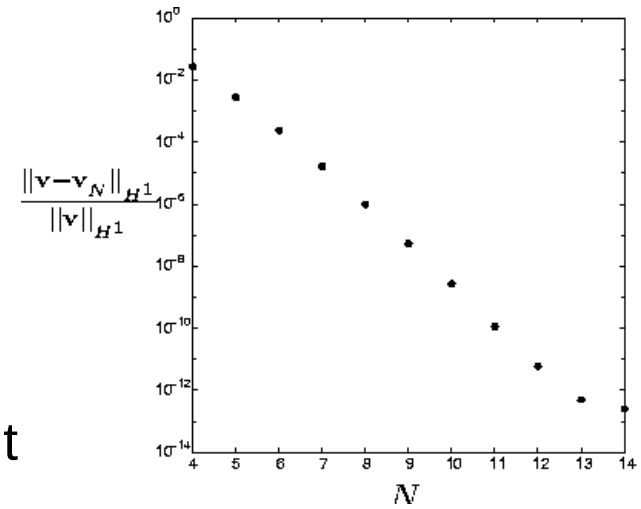
2D basis function, N=10

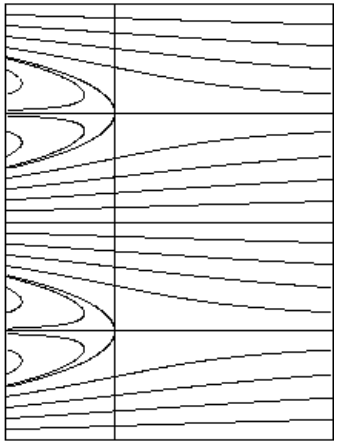


Spectral Element Convergence: Exponential with N

Exact Navier-Stokes Solution (Kovazsnay '48)

- 4 orders-of-magnitude error reduction when doubling the resolution in each direction





- Benefits realized through tight data-coupling.

$$
\begin{aligned}
v_x &= 1 - e^{\lambda x}\cos 2\pi y \\
v_y &= \frac{\lambda}{2\pi} e^{\lambda x}\sin 2\pi y \\
\lambda &:= \frac{Re}{2} - \sqrt{\frac{Re^2}{4} + 4\pi^2}
\end{aligned}
$$

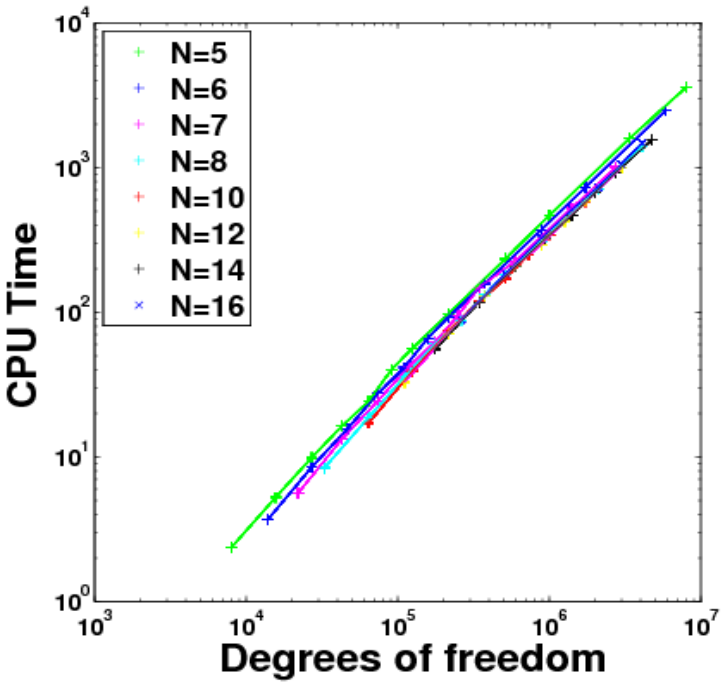
- For a given error,
  - Reduced number of gridpoints
  - Reduced memory footprint.
  - Reduced data movement.

Impact of Order On Costs

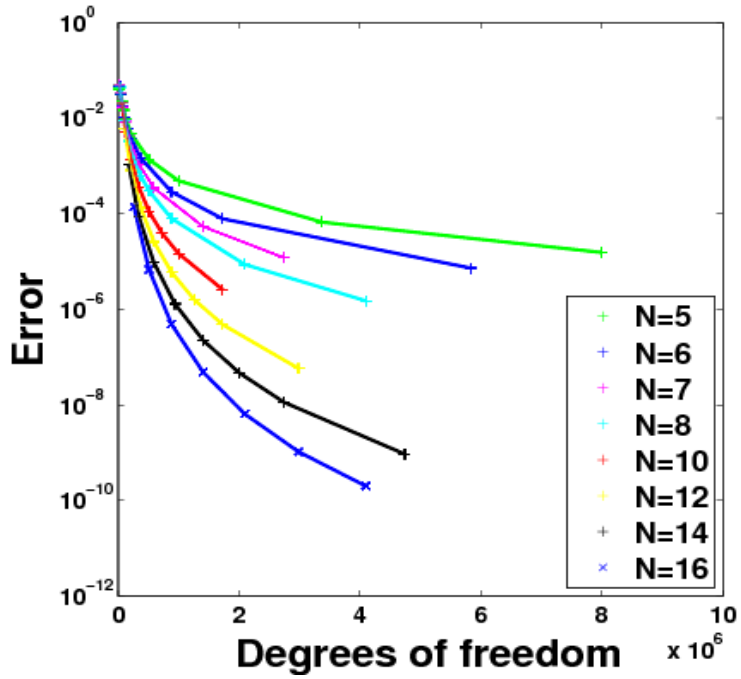
- For SEM, **memory** scales as number of gridpoints, n.
- Work scales as nN, but is in form of (fast) matrix-matrix products.
- Time scales as n

Time-Domain Maxwell's SEDG Simulation Misun Min, ANL

CPU time vs. #dofs, varying N.



Error vs. #dofs, varying N



Periodic Box; 32 nodes, each with a 2.4 GHz Pentium Xeon

Strong Scaling: 1 Million MPI Ranks

217 Pin Problem, N=9, E=3e6:

– 2 billion points

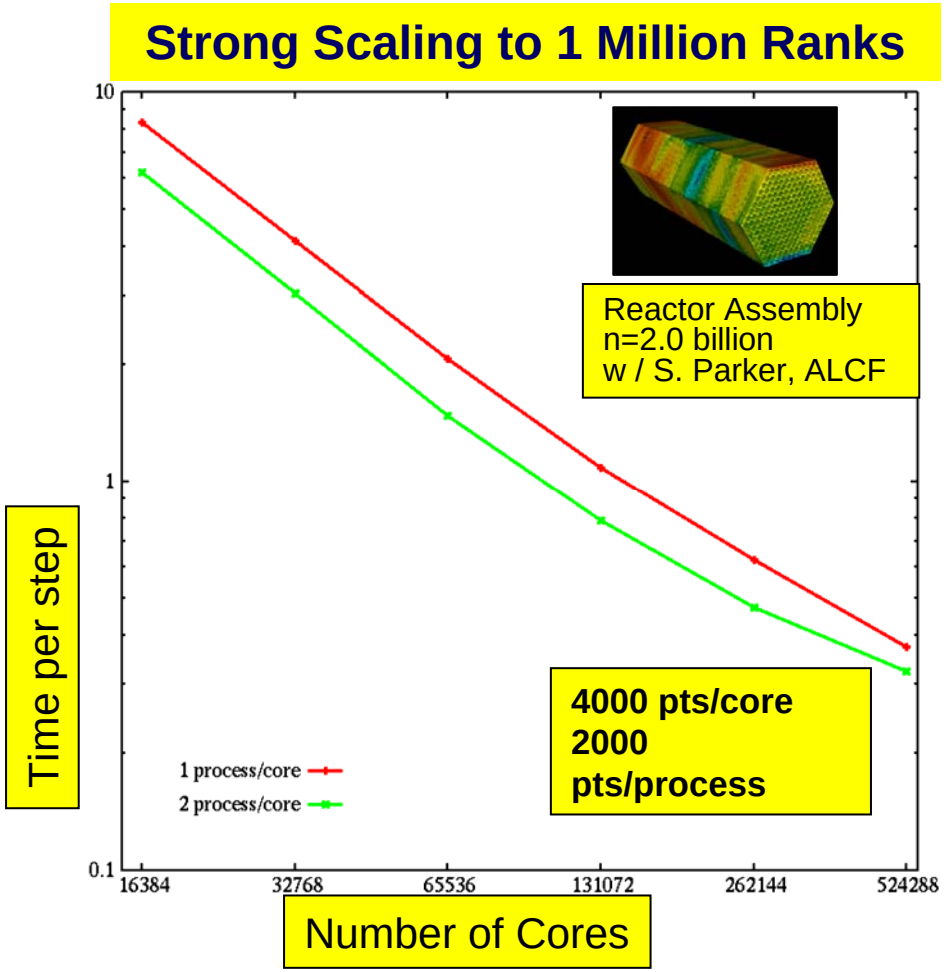
– BGQ – 524288 cores

- 1 or 2 ranks per core

– 60% parallel efficiency at 1 million processes

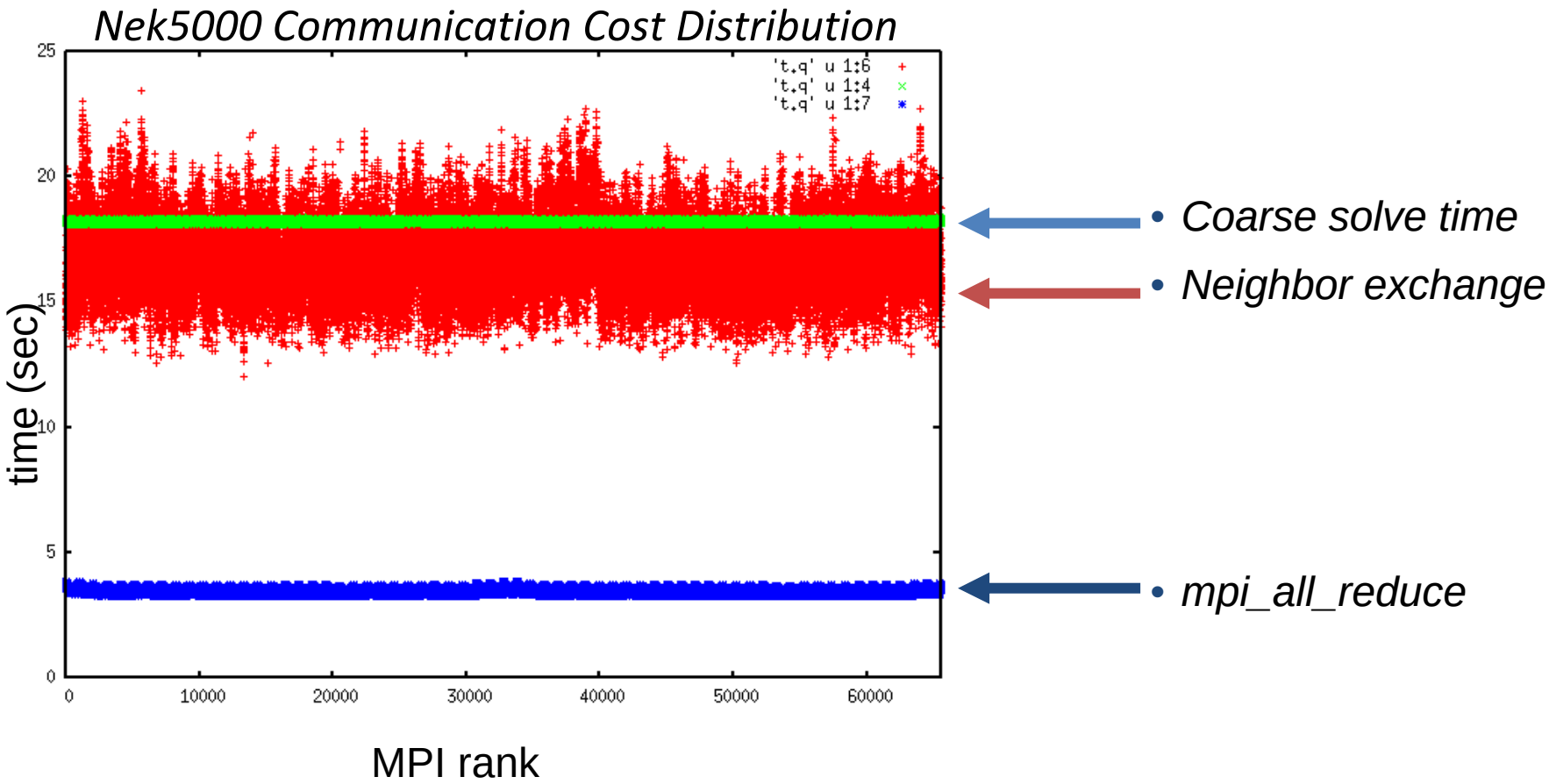
– 2000 points/rank

– A mixture of CG / multigrid



Nek/BGP Communication Cost Distribution vs Rank

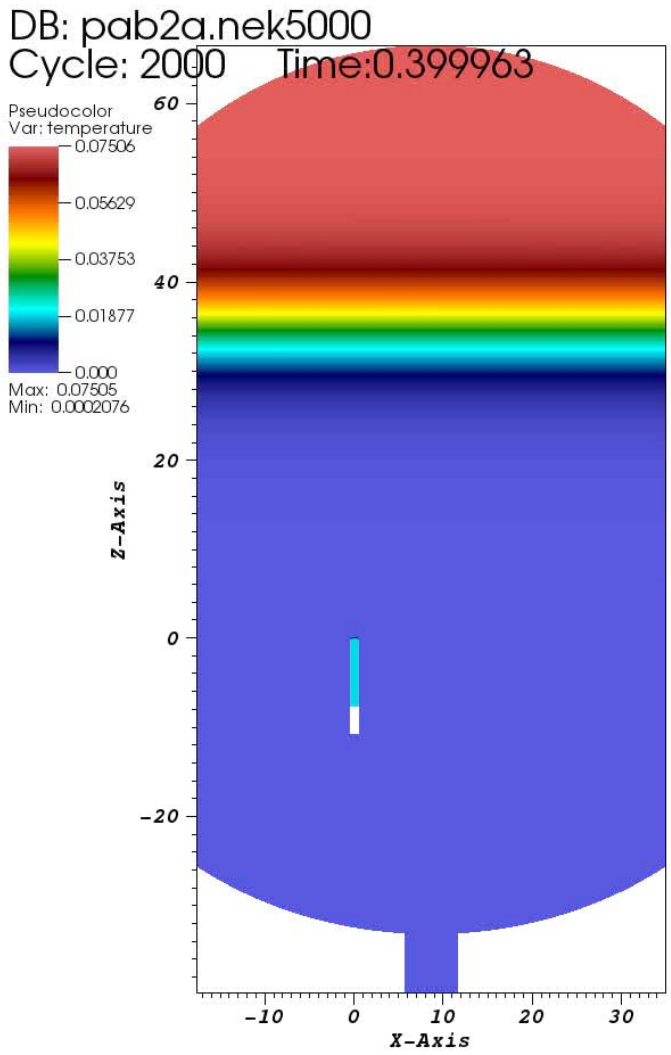
- Billion-point 217-pin bundle simulation on P=65536



- Neighbor vs. all_reduce: 50α vs 4α (4α, not 16 x 4α)

We Are Not Running Faster

- *Panda Benchmark – Nek5000*
  - *E=190,000 elements*
  - *N=7*
  - *n ~ EN^3 = 62 million*
  - *P ~ n / 3000 ~ 16384 MPI ranks*
- *Very long time integrations*
  - *1 month of wall clock time*
- *n/P ~ 3000*



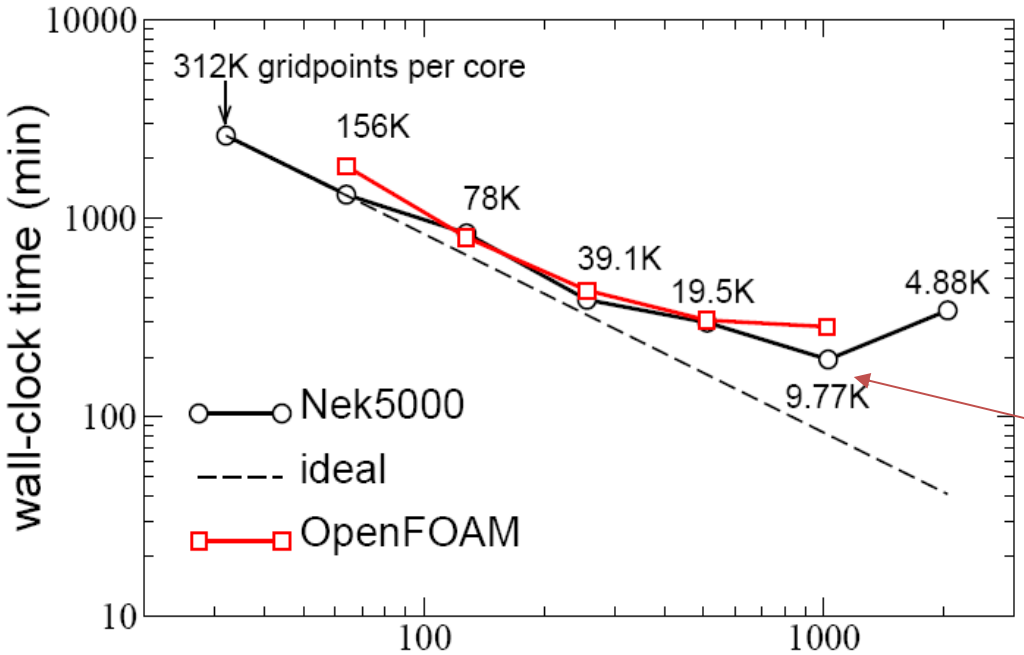
How Can a User Control Runtime?

- There is one ubiquitous knob: granularity
  - i.e., n / P
  - or, for a given problem size (n), vary P
  - Usually can choose P sufficiently small to amortize communication overhead and get good parallel efficiency
  - What are reasonable n/P values as we move to exascale ?

How Can a User Control Runtime?

Example

M. Sprague, NREL



Roll-over at
n/P ~ 10K

- Generally, one can reduce P to increase n/P
- ***Conversely, for a given P, what value of n will be required for good efficiency?***

Improving Scalability

- Q: Will this scaling continue as we move to exascale?
- Q: Is this the best we can do?

- What, exactly, is better, or even good?
  - Good node performance
  - Strong scaling to 10^6 or 10^9 processors.

- For most PDE solvers (and many other applications), strong scaling is ultimately limited by communication costs that do not go to zero as $n/P \rightarrow 0$:

$$t \sim c_1 \, n/P + c_2$$

How Exascale Differs from Cluster Computing

- Single jobs on communal HPC systems rarely use all compute resources:

  → Effectively, an infinite sea of cores.

  There is an incentive (*time to solution*) to spread the job as thin as possible, i.e., a premium is placed on fine-grained / strong-scale parallelism.

  → *As little work per node as possible*.

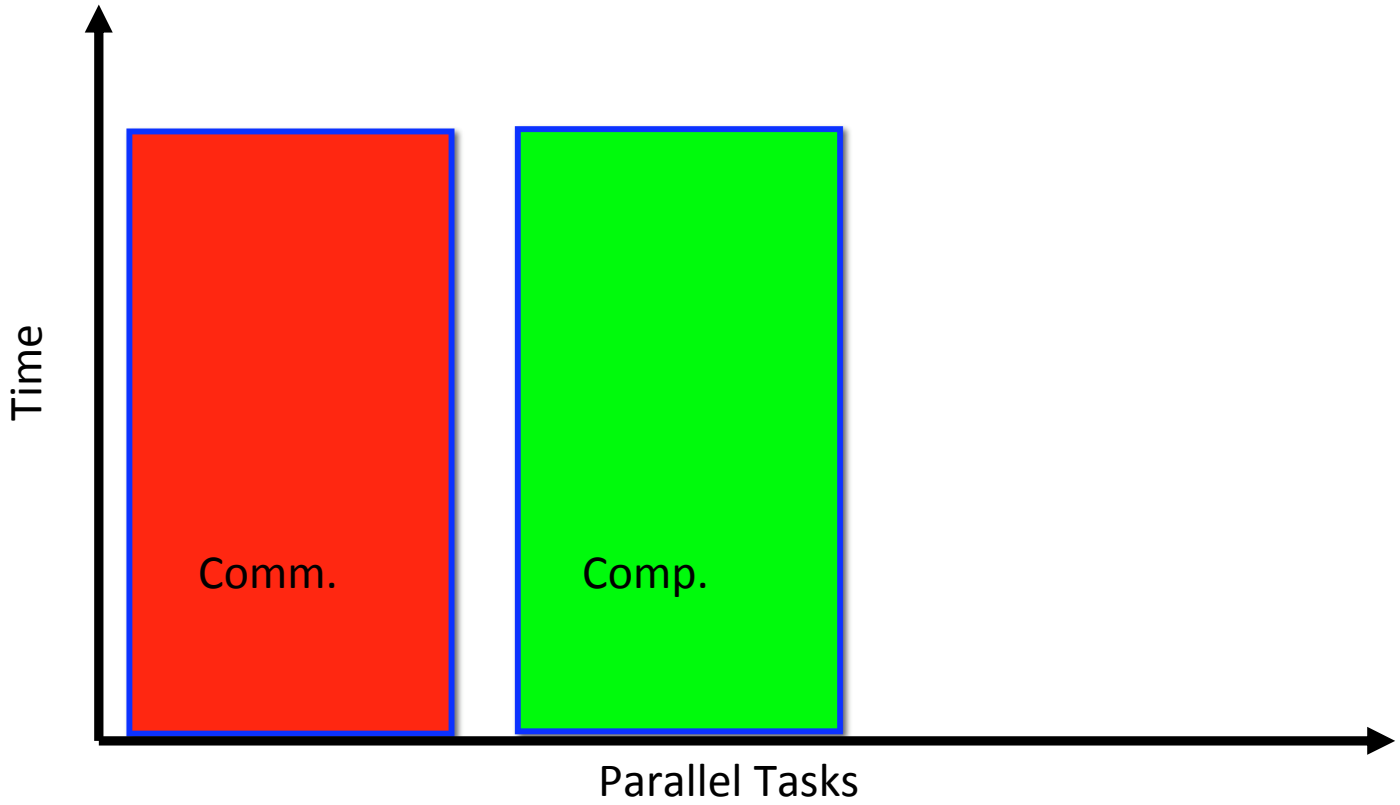
- *If we wish to run faster, we must optimize in this limit*

  → Need strategies to reduce *internode latency*.

  Result will be a broader array of applications and more effective use of exascale resources.

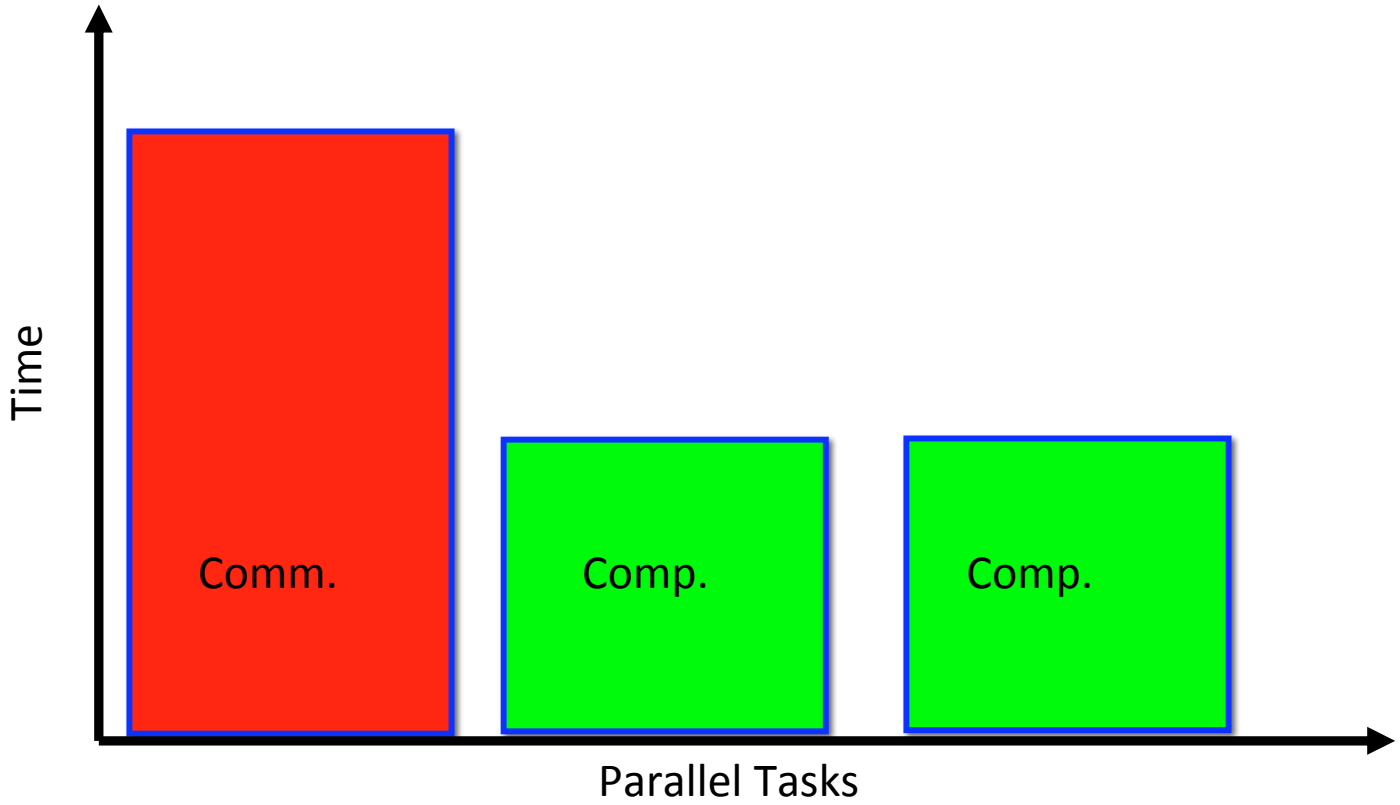
Multicore Does Not Imply Fine-Grained Parallelism

- Consider an operation with balanced communication / computation.
- Assume we can reduce the time spent on work through multicore.



Multicore Does Not Imply Fine-Grained Parallelism

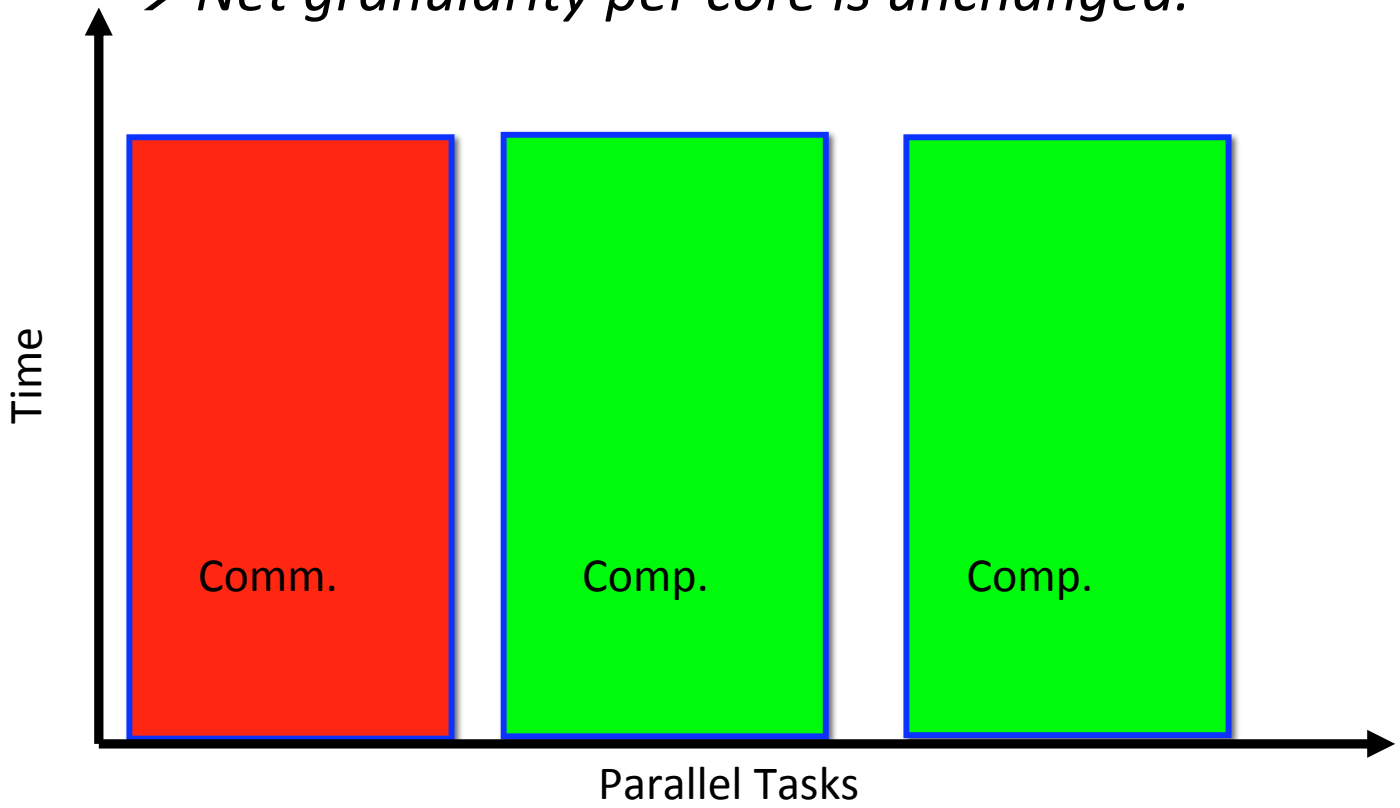
- Internode latency is not reduced by adding more cores.
- Simulation becomes communication dominated.



Multicore Does Not Imply Fine-Grained Parallelism

- To restore computation / communication balance, we must double the work per core*.

→ *Net granularity per core is unchanged.*

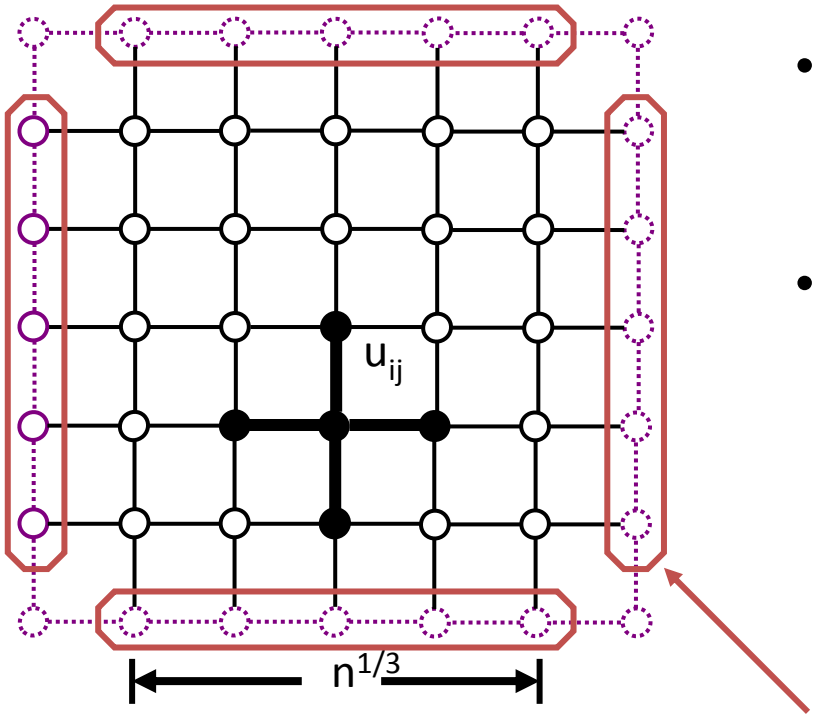


**We must also now increase bandwidth/node.*

Impact of Order on Costs

- With the assumption that cost is governed by number of gridpoints and weakly on polynomial approximation order, we study the scalability question in the context of familiar finite difference stencils, applied to the 3D Poisson problem.

Model Problem: Poisson with finite differences



processor p

data from neighbor
allows stencil update

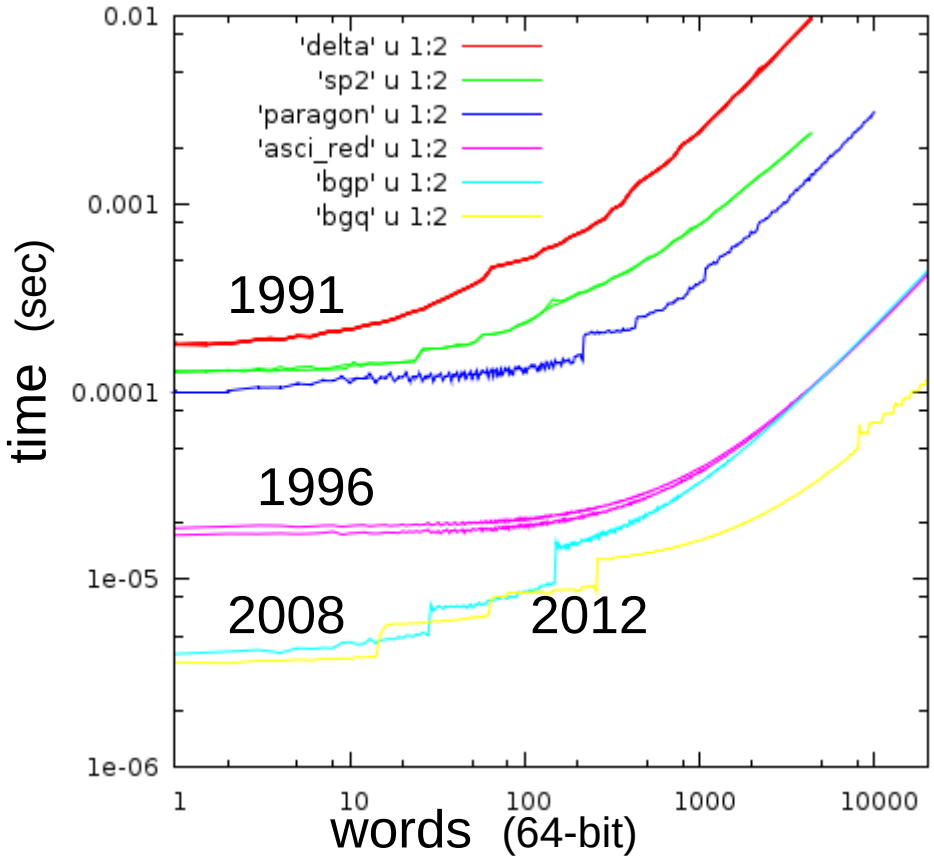
- We will consider complexity estimates for 3D Poisson with several iterative solvers.
- n/P points on each processor

Metric for Scalability

- P-processor solution time for n points:
  - T(P,n) = TA(P,n) + TC(P,n), ***or*** *nonoverlapping comm.*
  - T(P,n) = max (TA(P,n) , TC(P,n)) *overlapping comm.*
- As a metric for *scalable*, we seek conditions where $T_A > T_C$ *i.e., communication is subdominant*, with
  - $T_A(P,n) = T(1,n) / P$ the parallel work
  - $T_C(P,n)$ the total communication cost

    Assume linear message cost: $t_c(m) = (\alpha + \beta m) * t_a$
    - *m = number of 64-bit words*
    - t_a *= representative (observable) time for c=a*b*
    - α *= nondimensional latency* ($\alpha := \alpha^* / t_a$)
    - β *= nondimensional inverse-bandwidth* ($\beta := \beta^* / t_a$)

Linear Communication Model



Linear communication model :

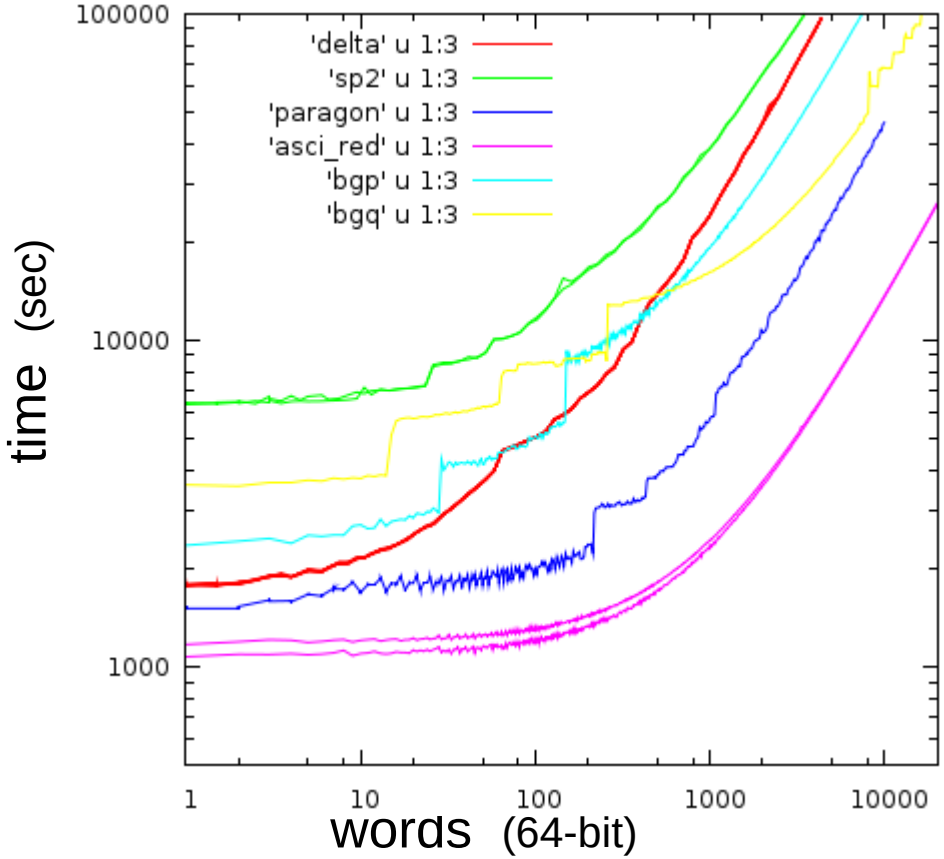
$t_c(m) = \alpha^* + \beta^* m$, m: 64-bit words

Nondimensionalize by t_a [c = a*b] :

$$t_c(m) = (\alpha + \beta m)\, t_a$$

$$\alpha = \alpha^* / t_a, \quad \beta = \beta^* / t_a$$

Linear Communication Model



Linear communication model :

$t_c(m) = \alpha^* + \beta^* m$, m: 64-bit words

Nondimensionalize by t_a [c = a*b] :

$t_c(m) = (\alpha + \beta m)\, t_a$

$\alpha = \alpha^* / t_a$, $\beta = \beta^* / t_a$

25 Years of Nondimensional Machine Parameters

YEAR	t_a (us)	α^*		β^*	α	β	m_2	MACHINE
1986	50.00	5960.	64	119.2	1.3	93		Intel iPSC-1 (286)
1987	.333	5960.	64	18060	192	93		Intel iPSC-1/VX
1988	10.00	938.	2.8	93.8	.28	335		Intel iPSC-2 (386)
1988	.250	938.	2.8	3752	11	335		Intel iPSC-2/VX
1990	.100	80.	2.8	800	28	29		Intel iPSC-i860
1991	.100	60.	.80	600	8	75		Intel Delta
1992	.066	50.	.15	758	2.3	330		Intel Paragon
1995	.020	60.	.27	3000	15	200		IBM SP2 (BU96)
1996	.016	30.	.02	1800	1.25	1500		ASCI Red 333
1998	.006	14.	.06	2300	10	230		SGI Origin 2000
1999	.005	20.	.04	4000	8	375		Cray T3E/450
2005	.002	4.	.026	2000	13	154		BGL/ANL
2008	.0017	4.	.021	2353	12.6	185		BGP/ANL
2011	.0007	2.5	.002	3570	3	1190		Cray Xe6 (KTH) [m2=24]
2012	.0010	4.	.005	5000	5	1000		BGQ/ANL

- $m_2 := \alpha / \beta$ ~ message size → twice cost of single-word message
- t_a based on matrix-matrix products of order 10—13

Complexity Models for Iterative Solvers

(see, e.g., Fox et al., '88)

- Point Jacobi iteration (7-point stencil, 3D):
  - Work: $T_{aJ} \sim 14\, n/P\, t_a$
  - Communication: $T_{cJ} \sim (6 + (n/P)^{2/3} (1/m_2))\, \alpha\, t_a$
- Conjugate gradient iteration (7-point stencil): (alt: *Chebyshev iteration*)
  - Work: $T_{aCG} \sim 27\, n/P\, t_a$
  - Communication: $T_{cCG} \sim T_{cJ} + 4 \log_2 P\, \alpha\, t_a$
- Geometric Multigrid:
  - Work: $T_{aMG} \sim 50\, n/P\, t_a$
  - Communication: $T_{cMG} \sim (8 \log_2 n/P + 30/m_2 (n/P)^{2/3} + 8 \log_2 P)\, \alpha\, t_a$

Scaling Estimates: Jacobi

- Q: How large must n/P be for $T_a \sim T_c$?

Scaling Estimates: Jacobi

- Q: How large must n/P be for $T_a \sim T_c$?

$$\frac{T_c}{T_a} = \frac{6\left(1 + \frac{1}{m_2}(n/P)^{2/3}\right)\alpha}{14\, n/P} \leq 1$$

$$\left.\begin{aligned} \alpha &= 2300 \\ \beta &= 12.6 \\ m_2 &= 185 \end{aligned}\right\} \text{BG/P parameters}$$

$$\boxed{(n/P) \approx 2000}$$

- Jacobi scaling is independent of P.
  - Of course, need occasional all_reduce to check convergence…
  - Also, ***not*** a scalable algorithm (but, similar to explicit timestepper)

Scaling Estimates: Conjugate Gradients (I)

$$\frac{T_c}{T_a} = \frac{6\left(1 + \frac{1}{m_2}(n/P)^{2/3} + 4\log_2 P\right)\alpha}{27\, n/P} \leq 1$$

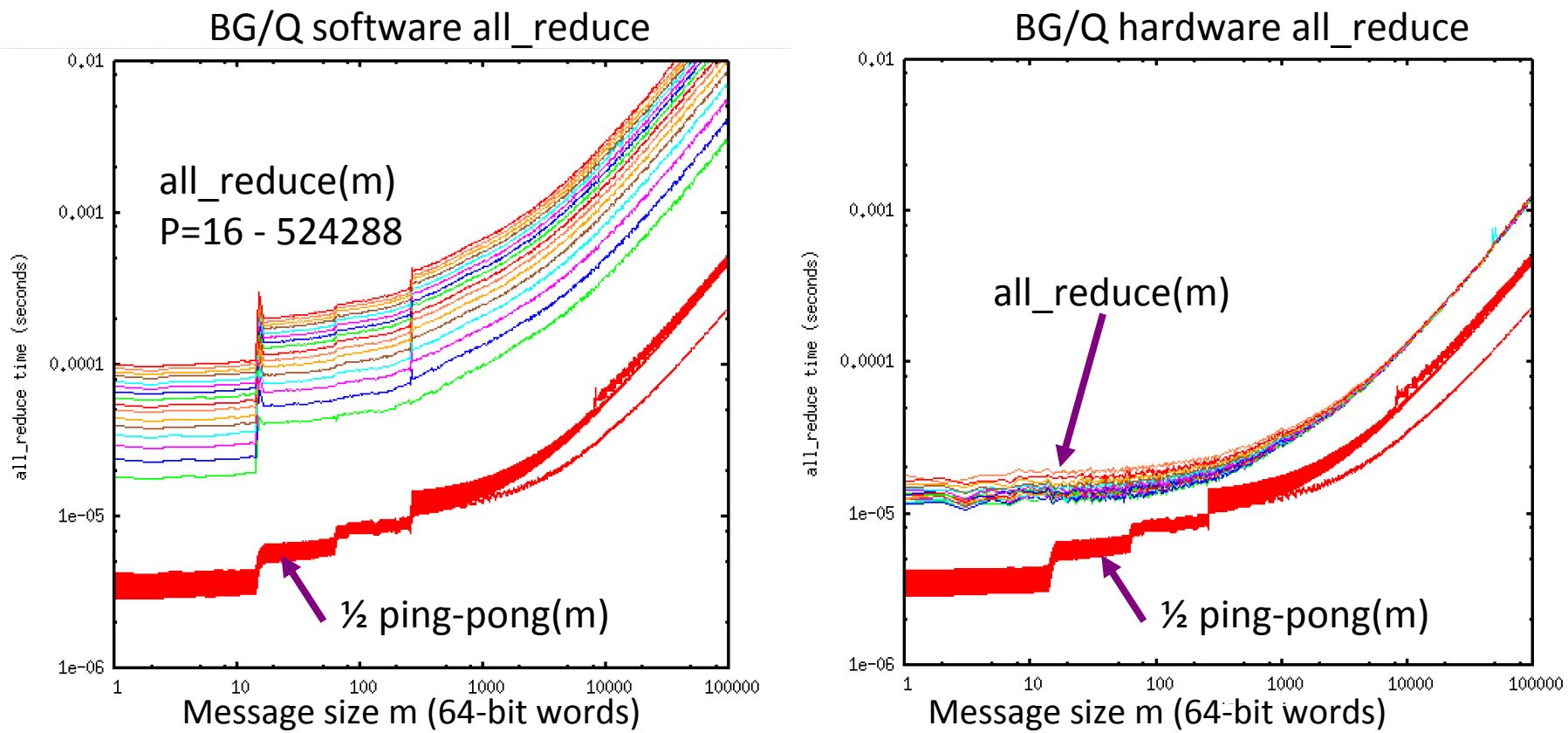
$$P = 10^6, \qquad \log_2 P = 20, \qquad \boxed{(n/P) \approx 8500}$$

$$P = 10^9, \qquad \log_2 P = 30, \qquad \boxed{(n/P) \approx 12000}$$

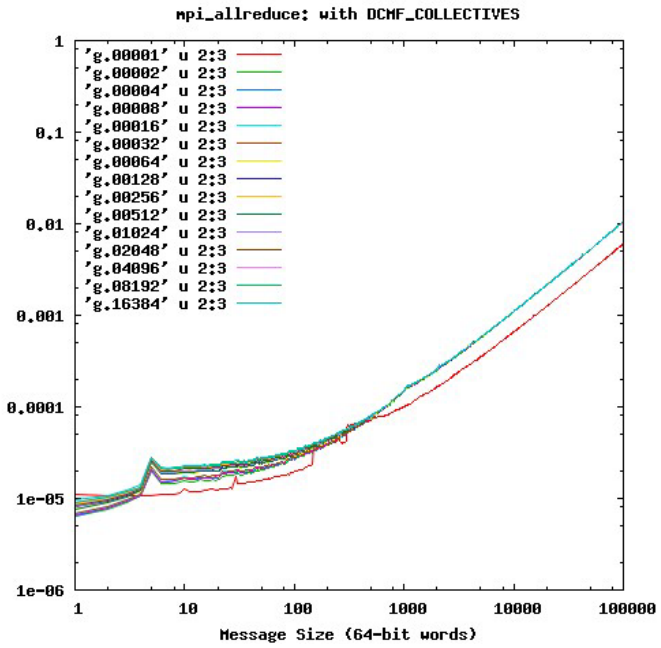
- The inner-products in CG, which give it its optimality, drive up the minimal effective granularity because of the log P scaling of all_reduce.
- On BG/L, /P, /Q, however, all_reduce is effectively P-independent.

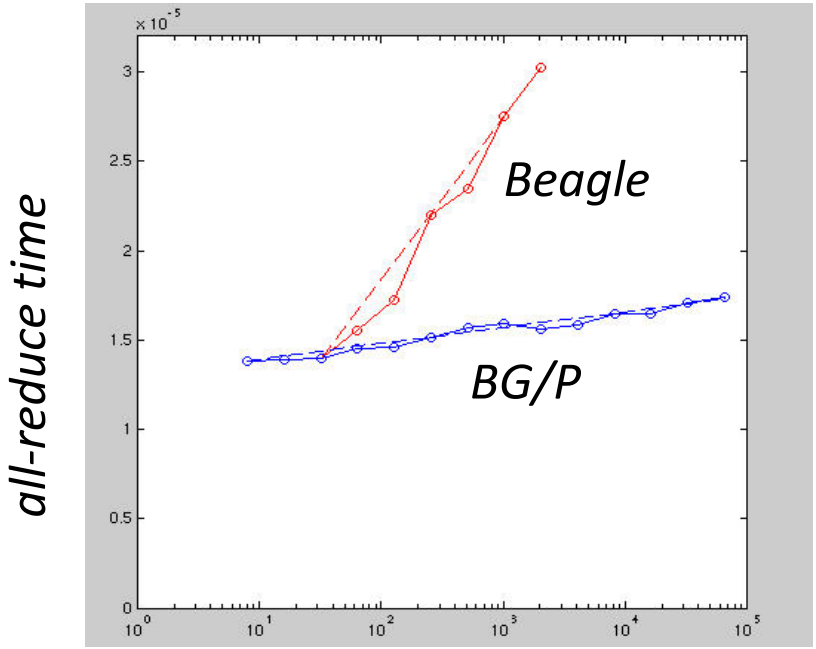
Eliminating log P term in CG

- On BG/L, /P, /Q, all_reduce is nearly ***P-independent***.
- For P=524288, all_reduce(1) is only 4α !



Not All Platforms Support Fast Collectives





P = 1—65,536

- In addition to all_reduce on network-interface card, processors must be allocated on convex subnets – not helter-skelter.
- At least one vendor was incredulous that this could possibly be beneficial. (They were worried about cycle utilization…)

Eliminating log P term in CG

2 x 4

$$\frac{T_c}{T_a} = \frac{6\left(1 + \frac{1}{m_2}(n/P)^{2/3} + 4\log_2 P\right)\alpha}{27\,n/P} \leq 1$$

$$\boxed{n/P \approx 1200}$$

- On BG/L, /P, /Q, CG is effectively P-independent because of hardware supported all_reduce.
- In this (admittedly simple) exascale model, net result is a 10x improvement in granularity (n/P=1200 vs. 12,000).

  → 10x faster run, but no reduction in power consumption.

Scaling Estimates: Geometric Multigrid (I)

$$\frac{T_c}{T_a} = \frac{\left(8\log_2 n/P + \frac{30}{m_2}(n/P)^{2/3} + 8\log_2 P\right)\alpha}{50\, n/P} \leq 1$$

$$n/P\ (P = 10^3) \approx 13{,}000$$

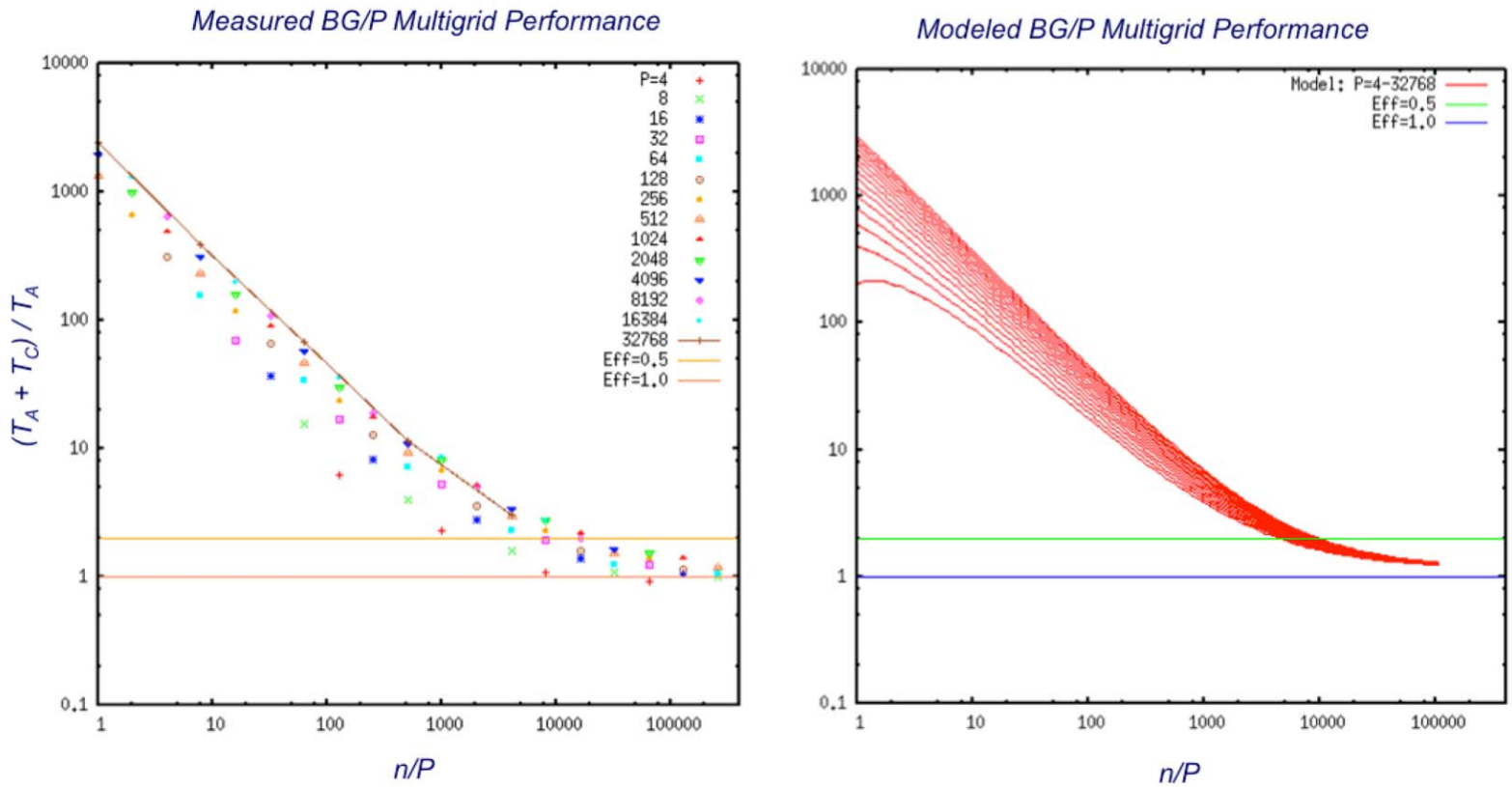
$$n/P\ (P = 10^6) \approx 17{,}000$$

$$n/P\ (P = 10^9) \approx 22{,}000$$

- In this case, granularity is relatively high because of the $8 \log_2 P$ term, which is associated with the coarse solve in MG.
- Replacing $8\,\alpha \log_2 P$ with 16α yields $n / P \sim 9000$, which is > 2x gain in scalability.
  - Such gains could be realized through hardware support in the network interface card (NIC) for scan / reduce operations.
  - Further savings might be possible by reducing the first term.

MG-Lite Code for Testing

- We have built a light-weight scalable MG code for model test and development which demonstrates the validity of the model for this analysis.



e 1.1: Left: Measured scalability for 3D geometric multigrid, $(T_A + T_C)/T_A$ as a function of (n
arying processor counts, P. Right: Modeled scalability for 3D geometric multigrid using 1.1.

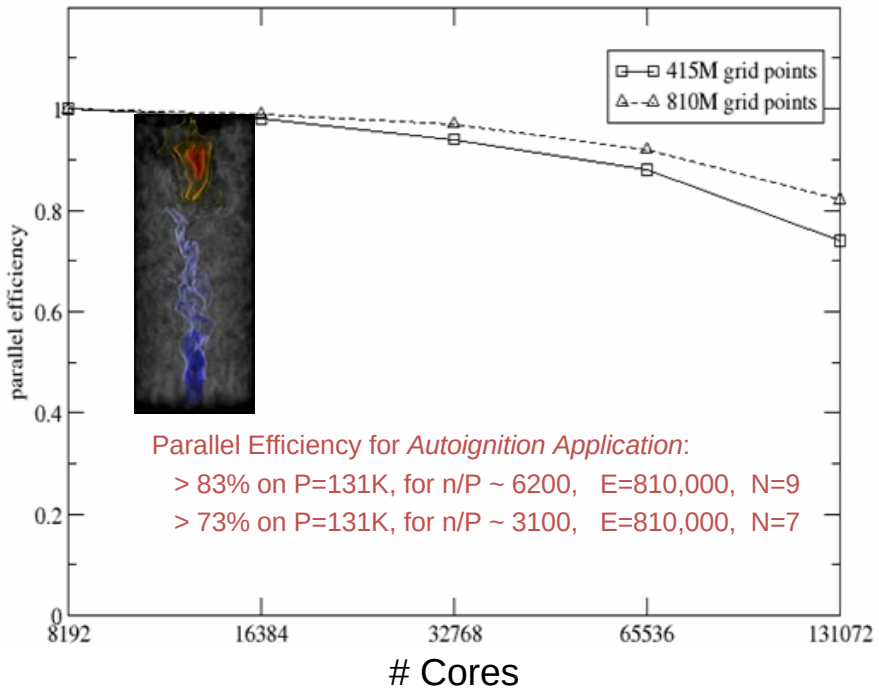
Scaling for More Complex Physics

Stefan Kerkemeier ETHZ / ANL

- More complex local physics helps realize finer granularity (but still, not faster runs).

- Production combustion and reactor simulations on ALCF BG/P demonstrate scaling to P=131072 with $\eta > .7$ for $n/P \approx 3000$.

BG/P Strong Scaling: P=8192 – 131072



Scaling for More Complex Physics

- Any time that communication can be organized into fewer messages there is a potential gain for reduced latency and thus finer granularity.

- Examples:
  - Time-domain Maxwell's equations
    - 6 fields to update, 6 fluxes exchanged in a single go
    - Also, DG implies 6 neighbors per element, not 26 (!)

  - Compressible Navier-Stokes / Euler
    - 5 fields to update in flux vector

- In these cases, the savings is *real.*

What Might Be Done?

- Must reduce internode latency.
  - MPI Lite?
    - Reduced instruction set (say, 8-10 operations)
    - Drop support for in-order message arrival
    - Strongly tag with sequential tags issued similar to communicators (to avoid hashing)
  - PGAS at the communication interface?
    - ***Nek5000 has NO MPI calls.***
      - » ***a = dot(u,v,n)***
      - » ***call gs(handle,u)***

- It's also clear that MPI Lite must be scalable
  - e.g., not built on mpi_alltoall(), etc.
  - we prefer generalized alltomany built on crystal-router (Fox et al. '88)

What Might Be Done?

Recast organized operations in term of a ***few scan/reduce operations to be supported in hardware on the NIC.***

- Such operations were part-and-parcel of the CM5 programming model (World's Fastest Computer in the early 90s).
- This approach has recently been effectively applied in the development of AMG for GPUs by Bell, Dalton, & Olson (2010).
- It's been shown to be viable for a host of science and graph-based applications (in particular, Blelloch and co-workers).
- Think of scan/reduce operations as BLAS is to LINPACK / LAPACK.

Conclusions

- Given present-day hardware:
  - *Fixed clock rate*
  - *Fixed communication/computation costs*

  we can expect n/P ~ 2000—10000 as granularity bound.

- A factor of 10 (optimistically) might be realized through hardware-supported collectives that reduce *organized* latency-bound operations.

- MPI-Lite or other schemes for reducing internode latency?

- Such gains would translate into reduced wall-clock time, but no change in power (unless we also reduce clock rates).

Some Observations, this meeting

- Anton approach – significantly reduced latencies

- In a post-Moore's-Law era, there is opportunity for customized architectures (J. Shalf).
  - (Recall GF-11 and other similar one-off architectures.)

Thank You!