Nek Deep Dive

E. Merzari, K. Heisey, P.F. Fischer

Nuclear Engineering Division
Argonne National Laboratory

&

Mathematics and Computation Division
Argonne National Laboratory

9700 S. Cass Avenue, Lemont, IL 60439
Outline

- **Background (10 min)**
  - Navier-Stokes equations
  - Why high-order for turbulence
  - “High-order accuracy at low-order cost”
  - Overview of the N-S Solver

- **Nekbone App (15 min)**
  - Overview
  - Current tests

- **Overview of Methods (25 min)**
  - SEM
  - Pressure Solver in Nek
  - Nekbone/Nek differences
  - Scaling issues

Three-dimensional rendering of a Nek5000 simulation of the MASLWR experiment
Three codes are focus of CESAR research

**Computational Fluid Dynamics**
- Incompressible Navier-Stokes
- Spectral Elements
  - High FLOP/load ratios
  - Nearest neighbor
  - Bulk synchronous
  - Low memory per node required for scalability
  - Global AllReduce latency key

**Neutron Transport (approach 1)**
- Boltzmann
  - Tabulated exponential
  - Complex parallelization
  - Load balancing issues
  - High FLOP/s rate

**Neutron Transport (approach 2)**
- Stochastic (Monte Carlo)
  - Data and Domain Decomposition
  - Load dominated
  - Branch heavy
  - Highly parallelizable in particle space
  - Poor locality in x-section and tally space
  - Low FLOP/s rate
  - Performance hot spot

**Mini-Apps, Kernels**
Nek - Principal Application

Incompressible Navier-Stokes equations

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

\[\nabla \cdot u = 0\]

- **Reynolds number** $Re > \sim 1000$
  - *small amount of diffusion*
  - *highly nonlinear* (small scale structures result, range of scales widens for increasing $Re$)

- Excellent Approximation for a variety of reactor problems: Incompressibility is an excellent approximation to single-phase reactor flows.

- **Very General Applicability**: Many systems of engineering interest (ex. Combustion Engines, Heat Exchangers), Blood Flow, etc..
Spectral Element Method & Transport Problems

- Variational method, similar to Finite Element, using \( GL \) quadrature.

- Domain partitioned into \( E \) high-order quadrilateral (or hexahedral) elements (decomposition may be nonconforming - localized refinement) . Converges exponentially fast with \( N \)

- Trial and test functions represented as \( N \) th-order tensor-product polynomials within each element. \((N: 4 \rightarrow 15, \text{typ.}, \text{but } N = 1\text{-}100+ \text{also works.}) n \sim EN^3 \text{gridpoints in 3D, } EN^2 \text{gridpoints in 2D.}\)

- These problems are particularly challenging because, unlike diffusion, where:

\[
\frac{\partial u}{\partial t} = \nu \nabla^2 u \quad \rightarrow \quad \hat{u}_k(t) \sim e^{-\nu k^2 t}
\]

implies rapid decay of high wavenumber \( (k) \) components (and errors), the high-\( k \) components and errors in advection-dominated problems persist.

- Turbulence provides a classic example of this phenomena, Excellent performance of SEM. High-order methods can efficiently deliver small dispersion errors.

Spectral element simulations of turbulent pipe flow
Benefit of High Order Methods

Comparison for DNS in channel flow: standard test case in turbulence research

- SEM & FV have the same cost per gridpoint
- For same accuracy, over several metrics, **FV needs 8-10 times as many points as 7th-order SEM**

From Sprague et al., 2010
Remember: These are useful flops!

- **Exponential convergence:**
  - e.g., doubling the number of points in each direction leads to $10^4$ reduction in error, vs. 4x for 2nd-order schemes.
  - **Minimal numerical dispersion** – *essential for exascale problems.*

\[
\frac{\|v-v_N\|_{H^2}}{|v|_{H^2}}
\]

\[
\begin{align*}

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{align*}
\]

*Exact Navier-Stokes solution, Kovazsnay(1948)*
Overview of the Navier-Stokes Solver in Nek

\[ \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla p + \frac{1}{Re} \nabla^2 \mathbf{u} \]
\[ \nabla \cdot \mathbf{u} = 0 \]

- Semi implicit treatment of the nonlinear term: explicit \((k \text{ th-order backward difference formula / extrapolation } \quad (k = 2 \text{ or } 3) \) - \(k\) th-order characteristics)

- Leads to linear Stokes problem: pressure/viscous decoupling:
  - 3 Helmholtz solves for velocity \(\text{("easy" w/ Jacobi-precond.CG)}\)
  - \((\text{consistent) Poisson equation for pressure [calls Helmholtz solver]}\)
  - SPD, Stiffest substep in Navier-Stokes time advancement, Most compute-intensive phase
  - Spectrally equivalent to SEM Laplacian, \(A\)

- Additional operations to ensure stability and turbulence resolution

Velocity, \(u\) in \(P_N\), continuous
Pressure, \(p\) in \(P_{N-2}\), discontinuous
or

Velocity, \(u\) in \(P_N\), continuous
Pressure, \(p\) in \(P_N\), discontinuous

\(Gauss-Lobatto\ Legendre\ points\ \( (P_N) \)
\(Gauss\ Legendre\ points\ \( (P_{N-2}) \)\)
Strong Scaling: Where We Are

- **Juelich's IBM BG/P Eugene (32768-262144 cores)**
  - 2.1M spectral elements with a polynomial order of N=15
  - fixed velocity/pressure iterations 5/50
  - full scale parallel efficiency = 0.71
  - peak application performance = 172.1TFlops (19.3% of peak)
  - elapsed time / timestep / gridpoint = 483 us (averaged over 20 timesteps)

- **ANL's IBM BG/P Intrepid (16384-163840 cores)**
  - 2.95M spectral elements with a polynomial order of N=7
  - fixed velocity/pressure iterations 8/20
  - full scale parallel efficiency = 0.7
  - peak application performance = 40.1TFlops (7.2% of peak)
  - elapsed time / timestep / gridpoint = 96 us (averaged over 20 timesteps)

- **Recently we were able to run with 10^6 MPI processes**
A Realistic Reactor Problem (Exascale)

• *Nek is not an App. It is a code. An App is when you apply Nek to a specific problem.*

• It makes no sense to consider solving most of current Petascale problems on an exascale machine. Reactor Problems are ideal for Exascale.

• Larger system (increased number of channels) but at SCALE (same a-dimensional numbers) with available experiments.

• The computational cost can be predicted in a straightforward manner using Petascale simulations of at SCALE facilities.

• ~ trillion points (tens of billion CPU hours needed in the best case)

• Exascale might also help perform parametric studies (hundreds of current petascale simulations) – e.g. to help study the effect of the heating rate on stability.
Nekbone
Nekbone: Introduction - 1

- **Goal:**
  - *Provide light-weight code to capture the principal hot kernel in Nek.*
  - *Elliptic solver:*

\[
H u^n := (I + \Delta t A) u^n = f^n
\]

- **Solves Helmholtz equation in a box using the spectral element method.** Currently, box is a 1D array of 3D elements or a box of 3D elements.
  - Design decision: simple, rather than many options.
  - Semi-Implicit formulation: Evaluation of a Poisson equation at every time step – most of computational cost in Nek is in the Poisson solve *(62% of a representative run)*
  - Also: any Nek5000 application that spends a majority of time in the temperature solver will very closely resemble a Nekbone test ran on a large, brick element count.

- Provides estimates of realizable flops, as well as internode latency and bandwidth measures.

- **Diagonally-preconditioned CG iteration** to solve the Poisson equation
  - Different from Nek: *Optimized Two level preconditioner in Nek vs. Diagonal preconditioner in Nekbone*
Current Focus on diagonally-preconditioned CG iteration, involving [More on this later]:

- Operator-evaluation: $w = Ap$
- Nearest-neighbor communication: $w = QQ^T w^*$
- All-reduce: $a = w^T p$

A $p$ kernel offers several opportunities for parallelism to be explored.

Essentially: many matrix-matrix products

Tests run through a battery of problem sizes, $n = P \times E \times N^3$

- $P =$ number of cores
- $E =$ number of blocks (spectral elements) per core
- $N =$ block-size (polynomial order in each direction)
- Run on Intel, BG/P, and BG/Q
Nekbone: Initial Results on Xeon, BG/P, and BG/Q

- MFLOPS/core for full skeleton kernel, 400 tests in (E,N) space.
- Xeon shows cache dependencies (less data = more performance)
- BG/P and Q show less cache sensitivity.
- Max-performance block size (N):
  - Xeon: N=8 (2.3 GFLOPS),  BG/P: N=14 (.55 GFLOPS),  BG/Q: N=12 (.765 GFLOPS)
Representative Nek case vs. Nekbone

- The case is a natural convection case at high Rayleigh number (Ra > 10^{10}).
- Number of processors: 32768.
- Number of elements: 875520.

From this example, we see that the Helmholtz solve consumes 82% of the CPU time (75% in pressure, 7% in u,v,w, and T).

Of this, 20% is spent in the pre-conditioner, [which is not yet in nekbone, but will be, once the nekbone users are ready for it]

It makes more sense, however, to focus on the 63% of the time that is represented by the fairly simple nekbone kernel (essentially the CG iteration).

**Representative of the Envisioned Application**
Communication: Nek vs. Nekbone

- **Nearest neighbor Communication:** *Source of Load Imbalance*
- *Case-dependent!*
- One element N\times N\times N per core: 8 messages of size 1; 12 messages of size N; and 6 messages of size \(N^2\)
- Message volume is dominated by the 6 face exchanges.
- At fine granularity (latency dominates), each of this exchanges has a similar cost (depends on whether is exceeds \(m^2\))
- Nekbone has two modes of operation:
  - a string of elements in a 1 \times 1 \times E array (**communication-minimal**)
  - a block of elements E1 \times E2 \times E3, which has 26 neighbors (**representative**).
Complexity: Nek vs. Nekbone

How we plan to increase mini-app fidelity/complexity:

- Diagonally-preconditioned CG
- Addition of Schwarz smoother: increase in comm and syncronicity
- Addition of coarse grid solve: significant increase in global comm and syncronicity
- Switch to GMRES: increase in local memory traffic w/o much work
Nekbone - Summary

The principal challenge at exascale is going to be to boost or retain reasonable single-node performance. To leading order, this question is \textbf{NOT} a scaling question, it is a "hot kernel" question.

\textbf{Nekbone focuses on the hot kernel}

This high-level kernel is simplified, so that it's accessible to computer scientists (it's not the full app. after all).

In terms of flops and fidelity to data flow, it captures the essential points in the smallest piece of code.

It provides opportunity for optimization at multiple levels. From low to high, these are:

- \textbf{mxm kernels} (already heavily optimized for most platforms)
- \textbf{gradient kernels} (3 mxms on the same data)
- \textbf{grad-transpose} (3 mxms on different data, producing one output)
- \textbf{operator level} - calls to grad
- \textbf{solver level}:
  - calls to operator
  - calls to vector-vector ops
  - calls to all-reduce
  - \textbf{calls to gs}.

It includes access to two essential types of communication
Methods Overview
A little more about the SEM (Nek & Nekbone)

- **Nodal basis:**
  
  \[ u(x, y)|_{\Omega^e} = \sum_{i=0}^{N} \sum_{j=0}^{N} u_{ij}^e h_i(r) h_j(s) \]

  \[ h_i(r) \in \mathcal{P}_N(r), \quad h_i(\xi_j) = \delta_{ij} \]

- \( \xi_j \) = Gauss-Lobatto-Legendre quadrature points:
  - stability (not uniformly distributed points)
  - allows pointwise quadrature (for most operators...)
  - easy to implement BCs and \( C^0 \) continuity

- **Local tensor-product form** (2D), allows derivatives to be evaluated as matrix-matrix products:
  
  \[
  \frac{\partial u}{\partial r} \bigg|_{\xi_i, \xi_j} = \sum_{p=0}^{N} u_{pj} \frac{dh_p}{dr} \bigg|_{\xi_i} = \sum_p \bar{D}_{ip} u_{pj} =: D_r u
  \]

- **Memory access scales only as** \( O(n) \)
  (even in general domains)

- **Work scales as** \( N*O(n) \), but CPU time is weakly dependent on \( N \)

- **Tensor Product:** “Matrix Free”

- **For the Stiffness Matrix** \( A \) evaluations:
  1. Operation count is \( O(N^4) \)
  2. Memory access is \( 7 \times \) number of points
  3. Work is dominated by (fast) matrix-matrix products involving \( Dr, Ds \), etc.
Evaluation of $a(v,u)$ in $\mathbb{R}^3$

In 3D, $I$ is given by,

$$I \approx a_N(v,u) = v^T \left( \begin{array}{c} D_r \\ D_s \\ D_t \end{array} \right)^T \left[ \begin{array}{ccc} G_{11} & G_{12} & G_{13} \\ G_{12} & G_{22} & G_{23} \\ G_{13} & G_{23} & G_{33} \end{array} \right] \left( \begin{array}{c} D_r \\ D_s \\ D_t \end{array} \right) u$$

$$= v^T A u,$$

with

$$(G_{ij})_{lmn} := \rho_l \rho_m \rho_n J_{lmn} \sum_{k=1}^{3} \left( \frac{\partial r_i}{\partial x_k} \frac{\partial r_j}{\partial x_k} \right)_{mnl}.$$

- Look at the memory access costs: only $7(N+1)^3$ to evaluate $Au$.
- However, if we store $A$, the cost is $(N+1)^6$ (per element!)
- Recall, there are now $(N+1)^3$ unknowns in $u$, or in $u^e$ in the multi-element case.
SEM Operator Evaluation (Nekbone & Nek)

- Spectral element coefficients stored on element basis ($u_L$ not $u$)

$$w = Au = Q^T A_L Q u, \quad w_L := Qw, \quad u_L := Qu$$

$$w_L = QQ^T A_L u_L$$

- Decouples complex physics –and computation - ($A_L$) from communication ($QQ^T$)

- Communication is required, and the communication pattern must be established a priori (for performance): set-up phase, $gs\_setup()$, and execute phase, $gs()$
Parallel Structure (Nek & Nekbone)

- Elements are assigned in ascending order to each processor
Communication Kernel:
General Purpose Gather-Scatter (Nek & Nekbone)

- Handled in an abstract way, simple interface.
- **Given index sets:**
  
  | proc 0: global_num = { 1, 9, 7, 2, 5, 1, 8 } |
  | proc 1: global_num = { 2, 1, 3, 4, 6, 10, 11, 12, 15 } |

  On each processor: \( gs\_handle = gs\_setup(\text{global\_num}, n, \text{comm}) \)

- In an `execute()` phase, exchange and sum:
  
  | proc 0: \( u = \{ u_1, u_9, u_7, u_2, u_5, u_1, u_8 \} \) |
  | proc 1: \( u = \{ u_2, u_1, u_3, u_4, u_6, u_{10}, u_{11}, u_{12}, u_{15} \} \) |

  On each processor: \( \text{call gs}(u, gs\_handle) \)
Pressure Solution Strategy (Nek ONLY)

1. Projection: compute best approximation from previous time steps
   - Compute $p^*$ in span\{ $p^{n-1}$, $p^{n-2}$, $\ldots$, $p^{n-l}$ \} through straightforward projection.
   - Typically a 2-fold savings in Navier-Stokes solution time.
   - Cost: 1 (or 2) matvecs in $E$ per timestep

3. Preconditioned CG or GMRES to solve

   $$ E \mathbf{D} \mathbf{p} = g^n - E \mathbf{p}^* $$

   Preconditioning is divided in two steps.
Overlapping Schwarz Preconditioning for Pressure (Nek ONLY)

\[ z = P^{-1} r = R_0^T A_0^{-1} R_0 r + \sum_{e=1}^{E} R_{o,e}^T A_{o,e}^{-1} R_{o,e} r \]

- \( A_{o,e} \) - low-order FEM Laplacian stiffness matrix on overlapping domain for each spectral element \( k \) (Orszag, Canuto & Quarteroni, Deville & Mund, Casarin)
- \( R_{o,e} \) - Boolean restriction matrix enumerating nodes within overlapping domain \( e \)
- \( A_0 \) - FEM Laplacian stiffness matrix on coarse mesh \( \sim E \times E \)
- \( R_0^T \) - Interpolation matrix from coarse to fine mesh

2D: \[ A = (B_y \otimes A_x + A_y \otimes B_x), \quad S^T A S = \Lambda, \quad S^T B S = I \]
\[ A^{-1} = (S_y \otimes S_x) (I \otimes \Lambda_x + \Lambda_y \otimes I)^{-1} (S_y^T \otimes S_x^T). \]

**NOTE:** \( B_x, B_y \), lumped 1D mass matrices (conditioning)

**Op. Count:** \( W = 8KN^3 \) \( \text{ (vs. } 4KN^3 \text{ for band solve)} \)

**Storage:** \( S = O(KN^2) \) \( \text{ (vs. } KN^3 \text{ for band solve)} \)

**NOTE:** \( S_y \otimes S_x u = S_x U S_y^T \) \( \text{ (matrix-matrix product)} \)

- Local and nearest neighbors
- **Exploit local tensor-product structure**
- Fast diagonalization method (FDM) - local solve cost is \( \sim 4d KN^{(d+1)} \)
  (Lynch et al 64)
Coarse-Grid Solver (Nek ONLY) - 1

**Designed for rapid solution performance – preprocessing cost not a constraint.**

- Uses coarse/fine (C-F) AMG
- Energy minimal coarse-to-fine interpolation weights
- Communication exploits \texttt{gs()} library

Break-Down of Navier-Stokes Solution Time for \( n=120 \) M, \( n_c = 417000 \)

<table>
<thead>
<tr>
<th>Case/( P )</th>
<th>Total</th>
<th>( QQ^T )</th>
<th>Coarse</th>
<th>all_reduce()</th>
</tr>
</thead>
<tbody>
<tr>
<td>x4096</td>
<td>1994</td>
<td>125</td>
<td>1180</td>
<td>1.2</td>
</tr>
<tr>
<td>a4096</td>
<td>1112</td>
<td>125</td>
<td>192</td>
<td>1.4</td>
</tr>
<tr>
<td>b4096</td>
<td>846</td>
<td>126</td>
<td>25</td>
<td>1.0</td>
</tr>
<tr>
<td>8192</td>
<td>460</td>
<td>88</td>
<td>22</td>
<td>1.0</td>
</tr>
<tr>
<td>16384</td>
<td>266</td>
<td>64</td>
<td>20</td>
<td>1.0</td>
</tr>
</tbody>
</table>

\( XX^T \) is a fast parallel coarse solver, AMG is necessary

Pairwise+all\_reduce Pwise/all\_red/crys.rtr

**coarse (red) and fine (blue) points**
AMG Statistics for $n_c = 417600$

Communication stencil width ($\sim \text{nnz}/n_c$ or $\text{nnz}/n_f$) grows at coarse levels.

- pairwise exchange strategy is \textit{latency dominated} for large stencils
- Therefore, rewrite $gs()$ to switch to either \texttt{crystal\_router} or \texttt{all\_reduce}, if faster.

<table>
<thead>
<tr>
<th>Level</th>
<th>$n$</th>
<th>$n_c/n$</th>
<th>$\rho_{f,m=1}$</th>
<th>$m$</th>
<th>$\gamma$</th>
<th>$\rho(E_{mg})$</th>
<th>$\frac{\text{nnz}(W)}{n_c}$</th>
<th>$\frac{\text{nnz}(A_{ff})}{n_f}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>417600</td>
<td>0.36</td>
<td>0.80</td>
<td>3</td>
<td>?</td>
<td>0.67</td>
<td>8.3</td>
<td>9.4</td>
</tr>
<tr>
<td>2</td>
<td>151248</td>
<td>0.44</td>
<td>0.61</td>
<td>2</td>
<td>?</td>
<td>0.63</td>
<td>7.9</td>
<td>22.0</td>
</tr>
<tr>
<td>3</td>
<td>66887</td>
<td>0.43</td>
<td>0.59</td>
<td>2</td>
<td>?</td>
<td>0.60</td>
<td>8.8</td>
<td>30.8</td>
</tr>
<tr>
<td>4</td>
<td>28862</td>
<td>0.33</td>
<td>0.62</td>
<td>2</td>
<td>?</td>
<td>0.57</td>
<td>12.8</td>
<td>41.5</td>
</tr>
<tr>
<td>5</td>
<td>9471</td>
<td>0.22</td>
<td>0.68</td>
<td>3</td>
<td>?</td>
<td>0.55</td>
<td>21.4</td>
<td>32.8</td>
</tr>
<tr>
<td>6</td>
<td>2116</td>
<td>0.18</td>
<td>0.67</td>
<td>2</td>
<td>0.42</td>
<td>0.51</td>
<td>36.4</td>
<td>86.6</td>
</tr>
<tr>
<td>7</td>
<td>390</td>
<td>0.20</td>
<td>0.60</td>
<td>2</td>
<td>0.42</td>
<td>0.48</td>
<td>35.8</td>
<td>53.8</td>
</tr>
<tr>
<td>8</td>
<td>79</td>
<td>0.16</td>
<td>0.72</td>
<td>3</td>
<td>0.61</td>
<td>0.46</td>
<td>33.3</td>
<td>43.8</td>
</tr>
<tr>
<td>9</td>
<td>13</td>
<td>0.23</td>
<td>0.62</td>
<td>2</td>
<td>0.39</td>
<td>0.35</td>
<td>9.3</td>
<td>10.0</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>0.33</td>
<td>0.44</td>
<td>2</td>
<td>0</td>
<td>0.11</td>
<td>2.0</td>
<td>2.0</td>
</tr>
</tbody>
</table>
gs() times $P=131,000$ on BG/P

crystal_router and all_reduce $> 5-10 \times$ faster than pairwise in many cases

# nontrivial connections

gs() time (seconds) vs. # nontrivial connections

red – pairwise

green – cr()

blue – all_reduce
Issues
Communication Costs - 1

Linear communication model
\[ t_c(m) = (\alpha + \beta m) t_a \quad m = \text{message size (64-bit words)} \]

<table>
<thead>
<tr>
<th>YEAR</th>
<th>( t_a ) (us)</th>
<th>( \alpha^* )</th>
<th>( \beta^* )</th>
<th>( \alpha )</th>
<th>( \beta )</th>
<th>( m_2 )</th>
<th>MACHINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1986</td>
<td>50.00</td>
<td>5960.</td>
<td>64</td>
<td>119.2</td>
<td>1.3</td>
<td>93</td>
<td>Intel iPSC-1 (286)</td>
</tr>
<tr>
<td>1987</td>
<td>.333</td>
<td>5960.</td>
<td>64</td>
<td>18060</td>
<td>192</td>
<td>93</td>
<td>Intel iPSC-1/VX</td>
</tr>
<tr>
<td>1988</td>
<td>10.00</td>
<td>938.</td>
<td>2.8</td>
<td>93.8</td>
<td>.28</td>
<td>335</td>
<td>Intel iPSC-2 (386)</td>
</tr>
<tr>
<td>1988</td>
<td>.250</td>
<td>938.</td>
<td>2.8</td>
<td>3752</td>
<td>11</td>
<td>335</td>
<td>Intel iPSC-2/VX</td>
</tr>
<tr>
<td>1990</td>
<td>.100</td>
<td>80.</td>
<td>2.8</td>
<td>800</td>
<td>28</td>
<td>29</td>
<td>Intel iPSC-i860</td>
</tr>
<tr>
<td>1991</td>
<td>.100</td>
<td>60.</td>
<td>.80</td>
<td>600</td>
<td>8</td>
<td>75</td>
<td>Intel Delta</td>
</tr>
<tr>
<td>1992</td>
<td>.066</td>
<td>50.</td>
<td>.15</td>
<td>758</td>
<td>2.3</td>
<td>330</td>
<td>Intel Paragon</td>
</tr>
<tr>
<td>1995</td>
<td>.020</td>
<td>60.</td>
<td>.27</td>
<td>3000</td>
<td>15</td>
<td>200</td>
<td>IBM SP2 (BU96)</td>
</tr>
<tr>
<td>1996</td>
<td>.016</td>
<td>30.</td>
<td>.02</td>
<td>1800</td>
<td>1.25</td>
<td>1500</td>
<td>ASCI Red 333</td>
</tr>
<tr>
<td>1999</td>
<td>.005</td>
<td>20.</td>
<td>.04</td>
<td>4000</td>
<td>8</td>
<td>375</td>
<td>Cray T3E/450</td>
</tr>
<tr>
<td>2008</td>
<td>.002</td>
<td>3.5.</td>
<td>.02</td>
<td>1750</td>
<td>10</td>
<td>175</td>
<td>BGP/ANL</td>
</tr>
</tbody>
</table>

- \( m_2 := \alpha / \beta \sim \text{message size} \rightarrow \text{twice cost of single-word} \)
- \( t_a \) based on matrix-matrix products of order 10 – 13
- At Fine granularity, code not constrained by internode bandwidth
- Latency is everything.
Granularity Example: Standard Multigrid

- Computational complexity per V-cycle for the model problem:
  - $T_A = 50 N/P \ t_a$

- Communication overhead:
  - $T_C := \left[ (8 \alpha \log_2 N/P + 30\beta (N/P)^{2/3} + 8 \alpha \log_2 P \right) / [50 N/P \ t_a ]$

  \begin{align*}
  \text{Restrict, smooth, prolongate} & \quad \text{Coarse grid solve} \\
  - T_C := T_l + T_b + T_g < 1 \text{ to balance communication / computation}
  \end{align*}

- $T_l := 8 \alpha \log_2 N/P \ / \ (50 N/P)$
- $T_b := 30 (N/P)^{2/3} \ / \ (50 N/P)$
- $T_g := 8 \alpha \log_2 P \ / \ (50 N/P)$

- > 15000 pts/proc for $P=10^9$ (BGP)
  - ~24 MB/proc (Nek)
- > 15 trillion points total (24 PB)

  - Dominated by coarse-grid solve
  - Dominated by intra-node latency
Communication Costs - 2

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

- Coarse solve time
- Neighbor exchange
- mpi_all_reduce

Neighbor vs. all_reduce: $50\alpha$ vs $4\alpha$ ($4\alpha$, not $16 \times 4\alpha$)
Communication Costs - 3
Raw ping-pong & all_reduce times

A major advance with BG/L and P is that all_reduce does not scale as $\alpha \log P$. 
How Can a User/Developer Control Communication Cost?

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

- Assume BGP latency, \( \alpha^* = 3 \) usec
- Assume 100x increase in node-to-node bandwidth
- 15 \times 10^6 \) pts/node for P=10^9 (BGP)
- Dominated by intra-node latency
- More than 15 trillion points total? (estimated by MG computational model)