Scaling Limits for PDE-Based Simulation

Paul F. Fischer*‡ Katherine Heisey† Misun Min‡

We analyze algorithm/architecture performance characteristics that have a direct impact on the scalability of present-day and future turbulent flow simulations on large-scale parallel computers.

I. Introduction

Parallel computing is founded on the principle that, given enough work for a given problem, one can subdivide the computation across P processors and realize an effective P-fold reduction in time to solution. On today's architectures, any PDE-based or particle-based simulation that uses a billion gridpoints or particles can easily be distributed across two compute nodes and run in half the time—for essentially the same power—compared with running on just a single node. This computational scenario, running a problem of *fixed size* in half the time on two processors or nearly one-Pth the time on P processors, is termed *strong scaling* and is the focus of this paper. Specifically, we explore the basic question of how far one can scale a given problem, defined by its computational resolution n (e.g., the number of gridpoints), when using P processors.

The relevance of the strong scaling question is expressed succinctly in the equation

$$S_P = \eta P S_1, \tag{1}$$

which relates the speed (i.e., the inverse time-to-solution) for solving a given problem to the number of processors, P. Here, S_P is the speed (in Mflops, say) when using P processors and $\eta = \eta(P, n)$ is the parallel efficiency. Although (1) is effectively nothing more than the definition of η, it in fact expresses a deep point, namely, that parallel computing can deliver a **multiplicative increase in performance when using P processors**. This statement holds whether the processors are traditional cores, multicore-nodes, or accelerators. In the current post-frequency-scaling era where clock rates are no longer increasing, the multiplicative effect of the distributed parallel computing model is the *only* mechanism we have for increasing the speed of calculations by factors of 1000s. Thus, it is vital that future architectures and algorithms continue to support this pathway to high-performance computing (HPC).

We illustrate the multiplicative effect by considering the strong-scale performance of the incompressible flow solver Nek5000 applied to a thermal-hydraulics flow problem in the domain depicted in the top panel of Fig. 1. This is a complex domain with hundreds of rods separated by helically wrapped spacer wires. The spatial discretization is based on the spectral-element method.[1,2] The mesh comprises 3 million spectral elements of order N=9, for a total of $n = 2$ billion gridpoints. Navier-Stokes timestepping is based on semi-implicit timesteppers that use multilevel-preconditioned GMRES for the pressure solve and Jacobi-preconditioned iteration for the viscous update. The lower panel of Fig. 1 shows the wall-clock time per step and parallel efficiency on Mira, the IBM Blue Gene/Q at the Argonne Leadership Computing Facility. The timing curves reflect the use of either one or two MPI ranks per core, with two ranks per core outperforming a single rank because of BG/Q's hardware support for threads. The parallel efficiency graph, plotted for the two-ranks-per-core case, shows that efficiency is unity out to P=262,144 MPI ranks (131,072 cores), drops to 0.8 for P=524,288, and is 0.6 for P=1,048,576.

Before proceeding with parallel performance analysis, we make several comments about the simulation results of Fig. 1. First, the discretization is based on an efficient spectral element method (SEM) that requires only $O(n)$ storage and $O(nN)$ work for Nth-order spatial approximations.[1,3] (Here, N=9.) For

*CS and MechSE Depts., Univ. of Illinois, Urbana-Champaign
†Dept. of Neurosience, Washington University of St. Louis
‡Mathematics and Computer Science Div., Argonne National Laboratory

turbulent channel flow, the spectral element method requires an order of magnitude fewer points than 2nd-order schemes and the cost per grid point is the same.[4] Thus, the SEM realizes a given accuracy with minimal data movement. Moreover, while the local operators are dense (but never formed), the solution is in C^0 and the data exchanges between element interfaces are thus communication-minimal—only one plane of data is exchanged, as would be the case for a standard finite element or second-order finite difference scheme. Finally, the computationally-intensive pressure Poisson problem is solved using highly-tuned multigrid solvers that require only 15 iterations per step for the problem shown.

While the scaling behavior illustrated in Fig. 1 appears reasonable, it raises several questions. Is it the best possible? What, exactly, is *good*? Can this performance be sustained on an exascale architecture? For what problem sizes, n? In the following sections, we address these questions through modeling and analysis of several algorithms on different architectures.

Two computing scenarios are of interest to the HPC community: *bounded* and *unbounded* resources. The bounded case is typical of computing on local clusters. One uses all the resources on the cluster. This scenario bounds P by the compute resources; consequently, the granularity n/P usually is large (e.g., a half-billion in the preceding example, if only two compute nodes are available.) Our attention will focus on the unbounded resource case, which corresponds to large computing facilities with 10^4–10^6 cores. Here, one rarely uses the entire machine. Our question is "Why not?" We note that most HPC centers encourage submission of large jobs, which often get favorable queue priorities. Moreover, if one can strong-scale a simulation from P processes to $2P$, the turnaround time drops by a factor of 2, with a corresponding increase in productivity. Furthermore, since power is essentially fixed under the strong-scaling model, there is no increase (or decrease) in power consumption if P is increased. Why, then, do most people not use the entire (or a significant fraction of the) machine at HPC centers? The answer is the reduction in efficiency. The objective of this study is to quantify, under suitable simplifying assumptions, the reasons for this performance drop-off and to identify potential mitigation strategies as we move forward with future HPC architectures.

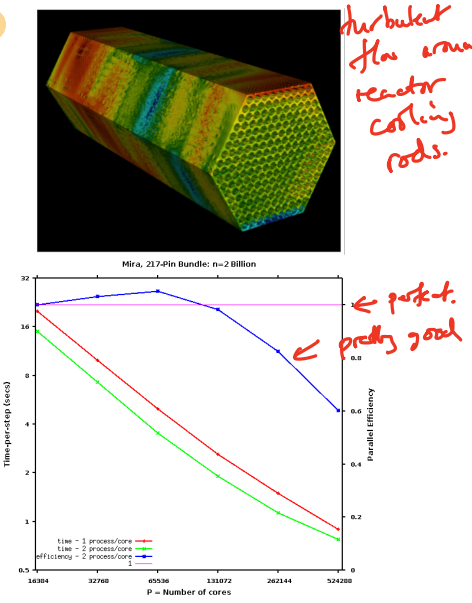

Figure 1. Strong-scaled Navier-Stokes results using one (red) or two (green) MPI ranks/core on Mira for n=2 billion.

The outline of the remainder of the paper is as follows. In Section II, we introduce computational complexity estimates for several algorithms used to solve the Poisson equation in $\mathbb{R}^3$ and examine the scalability limits of each. In Section III, we analyze the strong-scaling potential of GPGPUs based on a recent performance study for time-dependent electromagnetics simulations. In Section IV, we summarize our ideas and discuss our findings.

II. Modeling Multi-CPU Performance

Computational complexity estimates for parallel computing have been well established since before the introduction of the first distributed-memory parallel computers. A comprehensive summary of algorithms and complexity estimates can be found in the text by Fox *et al.*[5] and in contemporary papers on parallel CFD..[2,6] emergent principle across all the applications is the surface-to-volume effect. This principle says that the work per process scales as the local volume, or $\sim C_w\, n/P$ in the current context, while the communication scales as $C_c\,(n/P)^\gamma$, with $\gamma = (d-1)/d$ and d being the dimension of the problem. For PDEs, d is typically the spatial dimension, whereas for matrix-oriented operations, such as Gaussian elimination, d would be two. One can expect order-unity efficiency when $C_w n/P > C_c(n/P)^\gamma$. To quantify this principle in the context of today's architectures, we consider as a model problem the solution of the Poisson equation in $\mathbb{R}^3$ using a 7-point finite-difference stencil. We investigate the expected scalability of several algorithms on up to a billion cores.

To begin, we introduce two parallel performance models for the time T required to solve a problem of size n on P processors. The models reflect the situations where computation and communication are either

overlapped or not.

$$\text{Nonoverlapping:} \quad T(n,P) = T_a/P + T_c(n,P) + c_0 \tag{2}$$

$$\text{Overlapping:} \quad T(n,P) = \max[T_a/P, T_c(n,P)] + c_0 \tag{3}$$

Here, T_a reflects time spent on the parallelizable work for a single processor, T_c is the communication overhead, and c_0 represents non-parallelizable work or other overhead such a data motion. We note that the multiplicative effect of parallel computing (i.e., order unity efficiency) is realized only when

$$T_a/P \gg T_c(n,P) + c_0,$$

which is a manifestation of Amdahl's law. Our interest is in quantifying the (P,n) parameter space where we can expect performance to be good. A reasonable breakpoint in either the overlapping or nonoverlapping case is where communication is subdominant to the time spent doing useful work. That is,

$$T_a/P \geq T_c(n,P) + c_0. \tag{4}$$

We will take equality in (4) to be the breakpoint for any particular algorithm/architecture coupling. For our CPU-based analysis, we will ignore c_0, since it typically will be small compared with the communication overhead.

A. Interprocessor Communication Costs

The next component required for the complexity analysis is a model for interprocessor communication costs. The linear model

$$t_c(m) = \alpha^* + \beta^* m \tag{5}$$

is well suited for the present analysis. Here, m is the number of 64-bit words transferred in a single message between two processors, α^* is the internode latency in seconds, and β^* is the inverse-bandwidth in seconds/word. We consider a nondimensional version of (5),

$$t_c(m) = (\alpha + \beta m)\, t_a, \tag{6}$$

where $t_a = 1/S_1$ (seconds) is the inverse of the flop rate observed for the given algorithm on the computer in question, in the absence of communication.

The communication constants are measured by running ping-pong tests with MPI for varying values of m between rank 0 and rank k. Figure (2) shows typical ping-pong results for $k = 15, \ldots, 511$. (Lower values of k correspond to intranode communication and are omitted here for clarity.) Each time point represents an average of anywhere from 4 to a 1,000 tests, with 1,000 trials being used for the shorter messages. The model curve (5) is plotted as a dashed line for each case. We see that the XK7 has the lowest minimal times and highest peak bandwidth but that the timings are noisy despite the averaging, whereas the BG timings are essentially noise free. Moreover, a weakness of the model (5) is revealed by the plots, particularly for BG/Q, namely, that the model underpredicts communication by roughly a factor of 2 in the important range $m \approx 10$–10^4. The underprediction is platform dependent and could readily be incorporated into the complexity model. The curves also indicate that on BG/Q there would be merit in changing the shift points to support longer messages before shifting to the three-trip message protocol. The challenge on BG/Q is that one must set aside enough buffer space to accommodate the potentiality of a million unsolicited inbound messages (one from each MPI rank). In practice, this condition almost never occurs, however, and exploiting mechanisms such as the Eager message protocol allows this size to be adjusted.

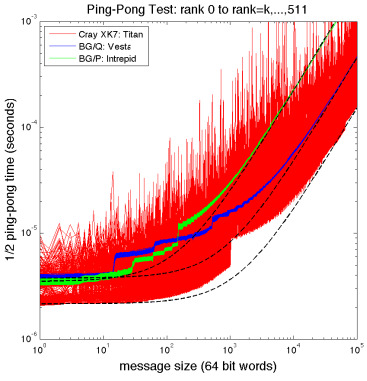

Figure 2. Ping-pong times for Cray XK7 (Titan), IBM BG/P (Intrepid), and IBM BG/Q (Mira).

Table 1 presents a list of machine parameters measured over the past several decades. The arithmetic times, t_a, are based on the matrix-matrix product performance for sets of noncached matrices of order

Year	t_a (μs)	αt_a (μs)	βt_a (μs/wd)	α	β	m_2	machine
1986	50	5960	64	119.2	1.28	93	Intel iPSC-1 (286)
1987	0.333	5960	64	17898	192	93	Intel iPSC-1/VX
1988	10	938	2.8	93.8	0.28	335	Intel iPSC-2 (386)
1989	0.25	938	2.8	3752	11.2	335	Intel iPSC-2/VX
1990	0.1	80	2.8	800	28	29	Intel iPSC-i860
1991	0.1	60	0.8	600	8	75	Intel Delta
1992	0.066	50	0.15	760	2.3	333	Intel Paragon
1995	0.02	60	0.27	3000	13.5	222	IBM SP2 (BU96)
1996	0.016	30	0.02	1875	1.25	1500	ASCI Red 333
1998	0.006	14	0.06	2333	10	233	SGI Origin 2000
1999	0.005	20	0.04	4000	8	500	Cray T3E/450
2005	0.002	4	0.026	2000	13	154	BGL/ANL
2008	0.0017	3.5	0.022	2060	13	160	BGP/ANL
2011	0.0007	2.5	0.002	3570	2.87	1250	Cray Xe6 (KTH)
2012	0.0007	3.8	0.0045	5430	6.43	845	BGQ/ANL
2015	0.0004	2.2	0.0015	5500	3.75	1467	Cray XK7

Table 1. Measured machine-dependent parameters

$N = 10$, chosen to reflect the computational load for the spectral-element method. The communication times are based on half the round-trip times for point-to-point ping-pong tests taken over a large number of processor pairings on each platform. The latency values are based on $t_c(1)$ and taken to be the maximum observed between any two pairings. On most machines, message exchanges between two MPI ranks not on the same node are only weakly dependent on node placement. We remark that these timings are for isolated ping-pong tests and do not reflect network contention. For domain-decomposition-based approaches to the solution of PDEs, contention is generally not an issue: in the large n/P limit, significant message traffic communication is dominated by work; in the small n/P limit, communication is dominated by internode latency.

We have also listed in Table 1 the parameter

$$m_2 \quad := \quad \alpha^*/\beta^*, \tag{7}$$

which, according to the linear model, is the size of message that would take twice as long to transmit as a one-word message. From an algorithmic design and analysis standpoint, m_2 is a convenient delimiter between the short- and long-message limits. If there are several messages of length $m < m_2$, it is clearly beneficial to agglomerate them together into one longer message, if possible.

B. Computational Models

Armed with the parallel performance data from Table 1, we turn now to complexity estimates for solution of the Poisson equation in $\mathbb{R}^3$ discretized with a 7-point finite-difference stencil. We consider three solution strategies for the sparse linear system that results: Jacobi iteration, Jacobi-preconditioned conjugate gradient iteration, and geometric multigrid.

1. *Jacobi Iteration*

We begin with Jacobi iteration. To make the model somewhat realistic, we assume that the equation is discretized with variable coefficients that reflect, say, geometric deformation. The local update takes the form

$$u_i^k \quad = \quad = a_{ii}^{-1} \left(f_i - \sum_{j \in \mathcal{I}_i} a_{ij}\, u_j^{k-1} \right), \qquad i = 1, \ldots, \tilde{n}, \tag{8}$$

where $\tilde{n}$ =ceil(n/P) is the number of points local to a processor and $\mathcal{I}_i$ is the index set associated with gridpoint i. Assuming that the cardinality of $\mathcal{I}_i$ is six, the arithmetic time for a single Jacobi iteration (8) is

$$T_{aJ} \sim 14(n/P)\,t_a. \tag{9}$$

Assuming that the parallel work decomposition leads to perfect cubic arrays of data on each processor, a distributed-memory parallel implementation of this scheme requires near-neighbor exchanges of $m = (n/P)^{\frac{2}{3}}$ values for each of six faces.[7] The resulting communication complexity is

$$T_{cJ} \sim 6\,t_c(m) = 6\left(\alpha + (n/P)^{\frac{2}{3}}\right) t_a. \tag{10}$$

We now seek values of n/P for which communication is subdominant, that is,

$$\frac{T_{cJ}}{T_{aJ}} = \frac{6(\alpha + \beta(n/P)^{\frac{2}{3}})}{14n/P} \leq 1. \tag{11}$$

For BG/Q with $\alpha = 3750$ and $\beta = 2.86$, the inequality is satisfied when

$$n/P \geq 1700. \tag{12}$$

At this granularity, one expects the cost per iteration to be

$$T_{\min,J} \approx 2 \cdot 14 \cdot 1700 \cdot t_a. \tag{13}$$

Simply stated, this analysis indicates that Jacobi iteration, ostensibly one of the most scalable PDE solvers, will require approximately 1,700 gridpoints per processor on a machine with the same relative hardware characteristics as BG/Q. Moreover, from (11) we see that this result depends *only* on (n/P) and not on P itself. The time $T_{\min,J}$ is not necessarily the minimum time that could be realized; it is simply the breakpoint where one would expect parallel efficiency to diminish. (For (n/P) lower than this breakpoint, overall power utilization certainly must increase.)

We note that Jacobi iteration is not a scalable *algorithm*. The number of iterations must scale at least as the diameter of the grid, $k_{\max} \sim Cn^{\frac{1}{3}}$. If we were solving a convection-dominated problem, this value would be a reasonable prediction of the number of iterations. For the Poisson problem, the count is closer to $k_{\max} \sim Cn^{\frac{2}{3}}$. With n/P fixed by (11), we have

$$\text{Time to solution} \approx 2 \cdot 14 \cdot (n/P)n^{\frac{\gamma}{3}} = 2 \cdot 14 \cdot (n/P)^{1+\frac{\gamma}{3}} P^{\frac{\gamma}{3}}.$$

When (n/P) is fixed at its lower bound, as scalability limits dictate, the time to solution must scale as $P^{\frac{\gamma}{3}}$, with γ=1 or 2, depending on whether the problem is convection-dominated or Poisson-like. We comment that although Jacobi is a poor iterative solver for the Poisson equation, it is a reasonable model for explicit timestepping algorithms that would be used for advection or wave equations such as considered in Section III.

2. *Conjugate Gradient Iteration*

Jacobi-preconditioned conjugate gradient (CG) iteration finds the *best approximation* in the Krylov subspace spanned by the iterates of the Jacobi iteration and thus converges more rapidly than any other iteration covering the same space.[8] The costs for this optimality are twofold: an increase in work from 14 to 27 operations per gridpoint and the addition of two global vector reduction operations. If we assume that the vector reductions are performed with a contention-free binary fan-in/fan-out, the communication cost of each is

$$T_{\text{all-reduce}} = (2\log_2 P)\,\alpha\,t_a. \tag{14}$$

Balancing the communication and arithmetic costs for CG thus leads to the condition

$$\frac{T_{cCG}}{T_{aCG}} = \frac{6(\alpha + \beta(n/P)^{\frac{2}{3}}) + 4\,\alpha\log_2 P}{27n/P} \leq 1. \tag{15}$$

Unlike Jacobi iteration, the CG complexity depends on P as well as (n/P). Since we are interested in exascale, we consider currently accessible values of P and those that could theoretically deliver exascale, that is, $P = 10^6$ and 10^9. For these cases, we find the strong-scale limit (15) is realized with the BG/Q parameters when

$$n/P \geq 12000, \qquad P = 10^6, \tag{16}$$

$$n/P \geq 17000, \qquad P = 10^9. \tag{17}$$

Here the nearly 3/2 increase in (n/P) results from the fact that $\log_2 10^6 \approx 20$ and $\log_2 10^9 \approx 30$.

The complexity increase resulting from the projective dot products in conjugate gradient iteration can be avoided through the use of Chebyshev iteration,[8] which has the same asymptotic complexity as CG but eliminates the need for vector reductions. In practice, however, we've found that Chebyshev typically results in a 10 to 15 percent increase in iteration count, even with optimally estimated eigenvalue ranges, and is therefore not of interest unless one is running at the critical (n/P) value, that is, in the range where all-reduce really dominates the total costs. Other choices, such as low-communication CG variants,[9] are also possible. On the BG series and some other forthcoming platforms, however, the $\log_2 P$ overhead is significantly reduced by having hardware support for all-reduce operations. Figure 3 shows the all-reduce times for processor counts $P = 16$, 32, 64, 128,..., 524288 (running one process per core) on the Argonne BG/Q, Mira. The times are for `mpi_allreduce` on vectors $\underline{v}^p$, which implements

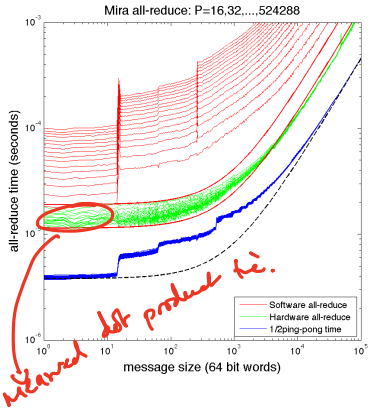

Figure 3. BG/Q `mpi_allreduce` times in software and hardware, along with 1/2 round-trip ping-pong times.

$$\underline{v} = \sum_{p=0}^{P-1} \underline{v}^p \tag{18}$$

and redistributes $\underline{v}$ to each processor p for $\underline{v}$ and $\underline{v}^p \in \mathbb{R}^m$. Figure 3 includes timings for (18) implemented in software and hardware. The software times are close to the model (14). By contrast, the hardware times are bounded by 3 to 5 times the ping-pong model (5). The dashed lines in the figure show this model (black) and this bracketing interval (red). A reasonable complexity bound for all-reduce is thus to replace (14) by

$$T_{\text{all-reduce}} = C_{ar}\,\alpha\,t_a, \tag{19}$$

where C_{ar}=3–5.

If we use (19) in the CG complexity estimate, we arrive at new granularity bounds deriving from the updated formula,

$$\frac{T_{cCG}}{T_{aCG}} = \frac{6(\alpha + \beta(n/P)^{\frac{2}{3}}) + 2C_{ar}\,\alpha}{27n/P} \leq 1, \tag{20}$$

which is once again independent of P. For BG/Q-based parameters with $C_{ar} = 5$, we find the inequality (20) is satisfied when

$$n/P \geq 2200, \tag{21}$$

which is almost as low as the point-Jacobi granularity limit (12) and remarkably close to the Navier-Stokes break-even point of Fig. 1.

3. *Geometric Multigrid*

Even with the best-fit property, CG iteration does not achieve order-independent convergence rates. A truly scalable Poisson solver requires a multilevel strategy. Here, we consider geometric multigrid as a model multilevel solver. In particular, we consider the following V-cycle.

```
for k = 1 : N_level − 1
    smooth residual
    exchange faces
    compute residual
    restrict residual
    reduce n by 2× in each direction
end

solve 1×1 system

for k = N_level − 1 : −1 : 1
    prolongate and add correction
    exchange faces
    increase n by 2× in each direction
end
```

For our analysis we assume that the load is initially balanced; that the number of points in each direction is enumerated as $0, 1, \ldots, n_d$, with $n_d = 2^{N_{level}}$; and that the number of processors is a power of 8. A detailed count of the operations, including the restriction, smoothing, and prolongation at each level, reveals that the total operation count per iteration for this V-cycle is $\sim 50n$ for n gridpoints. With the assumption of perfect load balance, the arithmetic time is therefore

$$T_{aMG} \sim 50(n/P)t_a. \tag{22}$$

Similarly, the communication complexity is

$$T_{cMG} \sim \left(8\,\alpha\,\log_2(n/P) + 30\,\beta\,(n/P)^{\frac{2}{3}} + 8\,\alpha\,\log_2 P\right) t_a. \tag{23}$$

In (23) we once again see a direct P dependence. Here, the $8\,\alpha\,\log_2 P$ term comes from the communication intensive coarse-grid solve phase of multigrid that commences with one degree of freedom per processor, restricts down to a single active processor (idling 7/8 of the active processors after each restriction), and prolongates back up to P active processors.

Proceeding as in the previous cases, we establish the strong-scale limit for geometric multigrid as the point where

$$\frac{T_{cMG}}{T_{aMG}} = \frac{8\,\alpha\,\log_2(n/P) + 30\,\beta\,(n/P)^{\frac{2}{3}} + 8\,\alpha\,\log_2 P}{50n/P} \leq 1, \tag{24}$$

For $P = 10^6$ and 10^9 with BG/Q parameters we find the multigrid granularity limits:

$$n/P \geq 21000, \qquad P = 10^6, \tag{25}$$

$$n/P \geq 27000, \qquad P = 10^9. \tag{26}$$

We remark that (26) predicts that an exascale Poisson problem would require over 27 trillion gridpoints to realize reasonable parallel efficiency. The large increase over conjugate gradient iteration is primarily a result of the communication intensive coarse-grid solve. If this operation could be cast as a hardware-supported parallel prefix operation similar to the all-reduce support on BG, we speculate that the multigrid communication costs could be reduced to

$$T_{cMG} \sim \left(8\,\alpha\,\log_2(n/P) + 30\,\beta\,(n/P)^{\frac{2}{3}} + 4\,(5\,\alpha)\right) t_a, \tag{27}$$

which would lead to $(n/P) = 7500$ as the fine-grained multigrid limit for $P = 10^9$. This nearly fourfold increase in scalability would translate into a fourfold reduction in CPU time through the use of more processors (with no reduction in power). One can, in fact, cast multigrid as a sequence of prefix operations, even in the more general case of algebraic multigrid, as was demonstrated by Bell et al.[10] Identifying primitives that can be supported by the network interface card would be a potentially productive avenue for co-design in future-generation HPC systems.

Quite simple models do a good job predicting actual performance.

Gordon Bell prize
→ award for "most impressive work at flops".

⇒ all do strong scaling studies.

→ last 5 or 6 winners all had scaling efficiency down to ~ 2000 dofs/process.

⇒ We won't get that good in our python implementation.

C. Multi-CPU Summary

We briefly present here a few comments regarding factors influencing the preceding model-based analysis. The analysis makes several basic assumptions and is designed to allow rapid estimation of performance characteristics on distributed-memory architectures at scale. First, one of the most influential components of the estimates is arithmetic time, t_a, that is embedded in the definition of α and β. This timescale can easily change by significant factors through vectorization/optimization of the dominant computational kernels. Second, with machines such as the Cray XK7 (Titan), measuring even α^* and β^* is difficult because of the network noise. One of the advantages of BG/Q is that every partition of the processor set is convex, such that only the user's traffic traverses the partition. The use of convex partitions also permits the use of dedicated network resources for hardware-supported collective operations.

Several variants of the models can affect scalability. For example, the solution of vector-based systems of PDEs such as Maxwell's equations or the compressible Navier-Stokes equations allows the communication of multiple solution components (six in the case of Maxwell's and five for Navier-Stokes) in each nearest-neighbor data exchange. This agglomeration amortizes the message latency and allows for lower values of (n/P) than is possible with the scalar Poisson equation. We find, for example, that (n/P) can be as low as a few hundred when strong-scaling to $P > 10^5$ on BG/P with our spectral-element electromagnetics code NekCEM. Additional physics modules can also influence the strong-scale limit. For example, combustion simulations entail a significant amount of pointwise chemistry that can be strong-scaled with no communication overhead. This local work can amortize the communication overhead of the hydrodynamics and allow for finer granularity (lower (n/P)). Another factor that influences scalability in general-purpose finite-/spectral-element codes is whether one uses a continuous or discontinuous Galerkin (DG) formulation. In the strong-scale limit, DG has a distinct advantage in that each element communicates with at most 6 neighbors. By contrast, the continuous Galerkin method requires communication among edges and vertices, which can raise the number of messages up to anywhere from 26 to ≈ 50. Because these are all short messages ($< m_2$ in length), their costs are equal. Thus, we can expect DG to have a significant scaling advantage.

III. Modeling Multi-GPU Performance

General purpose graphical processing units (GPUs) offer the potential for significant performance, cost, and power benefits over traditional CPUs and are at the heart of several current and future HPC systems. GPUs introduce (or, extend, rather) a second level of parallelism within each node. The strategy of GPUs is to attain performance by having hundreds of independent fine-grained tasks that are either executing or waiting on memory requests within the node. With enough tasks, all function units can be busy and one realizes a 100 % efficiency at the node level. Of paramount importance in the multi-GPU context is to know *how many tasks* are required to attain this efficiency, because that is the number that will set the granularity limit in an HPC setting. In this section, we analyze performance data taken from a recent OpenACC port of NekCEM to a multi-GPU implementation[11] and use it to illustrate some of the concepts introduced in the preceding sections.

NekCEM is an explicit time-marching code to solve Maxwell's equations, shown here in source-free form.

$$\epsilon \frac{\partial \mathbf{E}}{\partial t} = \nabla \times \mathbf{H}, \qquad \mu \frac{\partial \mathbf{H}}{\partial t} = -\nabla \times \mathbf{E}, \tag{28}$$

The spectral-element/discontinuous Galerkin (SEDG) formulation results in a diagonal mass matrix, so time-advancement of these equations has a complexity that is quite similar to Jacobi-preconditioned conjugate gradient iteration. A major difference is that one can amortize the nearest-neighbor communication costs by updating the surface flux terms for all six components of the vector-field pair $(\mathbf{E}, \mathbf{H})$ in a single pass.

Figure 4 shows performance results for the OpenACC/GPU-based variant of NekCEM developed in Otten et al.[11] Timing runs are presented for the Cray XK7, Titan, using one GPU per node. Also shown in panels (b) and (c) are multi-CPU runs using 1, 4, 8, and 16 cores per node on Titan and on the IBM BG/Q, Vesta, for P=1, 2, 4,...,128 cores (one rank per core). We remark that the OpenACC port was highly successful, with a single GPU sustaining performance equivalent to 40 CPUs on Titan. Moreover, the GPU power usage was about 40% of that required by the all-CPU simulation on the same platform.

To understand the data in Figure 4 we first note that a *column* of dots corresponds to classic strong scaling. As one moves down a vertical line, the problem size n is fixed and P doubles from curve to curve. On the other hand, a *row* of dots corresponds to weak scaling. Starting from the left and moving right, the

processor counts and problem sizes double. If the points are on the same horizontal line (i.e., have the same runtime) then we have perfect weak scaling, as is the case for $n/P > 10^5$ in Figure 4(a). We see that the largest problems (large, fixed, n) realize a two-fold reduction in solution time with each doubling of processor count over the range of P considered.

For the GPU-only (i.e., P=1) case, the strong scale limit corresponds to $n/P \approx 10^5$. In the absence of communication overhead, it is clear that the GPU has another limiter—the c_0 term in (2) is no longer negligible. This behavior is understandable given that there is a high degree of parallelism internal to the GPU. Those resources are not exploited unless there is sufficient work on the GPU. (We remark that, for the single GPU case, there is little data traffic between the GPU and the host.) For the GPU performance curves in Fig. 4 we have indicated a *strong-scale limit* line where the performance diminishes with small n/P.

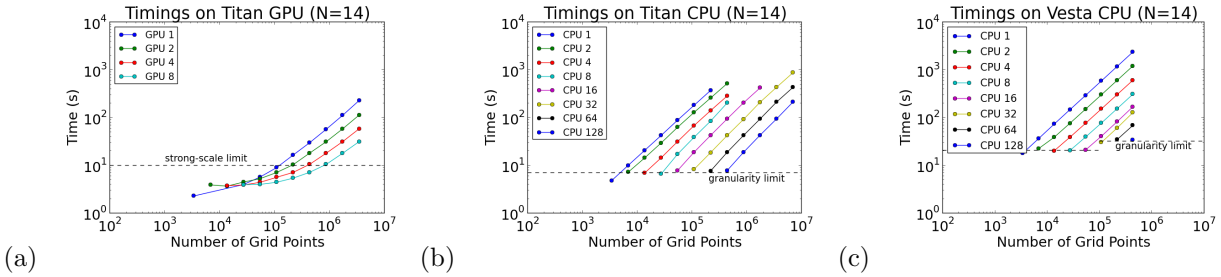

Figure 4. Timings on different number of GPUs and CPU cores on OLCF Titan and ALCF BG/Q Vesta; 1000 timestep runs with the number of grid points $n = E(N+1)^3$ increased with $E = 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048$ and $N = 14$.

We note that the strong-scale limit is not observed for the CPU-only cases (graphs (b) and (c) in Fig. 4). These cases continue to scale well down to the point of one element per core, which is the natural granularity limit for the SEDG formulation. The lower-bound runtime is indicated by the granularity-limit line in the plots. In the present case, the GPU outperforms the CPU, but Otten et al.[11] show that the CPU-based simulations can outperform the GPU from a pure speed standpoint for the case of N=7, where the granularity limit of the SEDG formulation is reduced to 343 points per core. Although the use of more cores allows the CPU-based simulations to be faster, it does not alter their overall energy consumption, which remains at 2.5× that of the GPU-based runs. That NekCEM can sustain strong scalability to this level of granularity is not surprising given the analysis of the preceding section. The code is DG-based and thus requires communication with at most six neighbors per element. Moreover, the implementation exploits the exchange of multiple (six) vector components for each nearest-neighbor update, thus amortizing internode latency over more work.

IV. Conclusion

We have observed the scaling performance of two production codes, Nek5000 and NekCEM. These codes have scaled to over a million MPI ranks out to granularity limits of $(n/P) = 2000$ and 343, respectively. For Nek5000, one has a mixture of Jacobi-preconditioned CG iterations (for diffusion), explicit timestepping (for advection) and multigrid-preconditioned CG (for the pressure). The observed 60% parallel efficiency for $P = 10^6$ is in keeping with the respective granularity estimates for Jacobi and multigrid iteration of $(n/P) = 1700$ and 21000. For NekCEM, we would anticipate strong scaling out to $(n/P) \approx 1700/6$ because communication costs are amortized over six times the work per timestep. Remarkably, this scaling is observed on both the IBM BG/Q and the Cray XK7.

The foregoing analysis demonstrates that the granularity limits observed for Nek5000 are nearly optimal for the given architecture charateristics and that the degree of scalability is *not* strongly tied to the underlying discretization, but results from fundamental work/communication balances over a range of different solution strategies. The analysis further demonstates that scalable solution strategies for a CFD simulation at exascale would require about 10 trillion gridpoints to make effective use of the entire machine.

We argue that continued performance gains on HPC platforms will need to address strong-scale parallelism, which means that performance developments must focus on *reducing* the problem size per node if one

hopes to reduce time-to-solution. In particular, lowering the $n_{1/2}$ for GPU-based nodes could provide opportunities for significant performance gains in the unbounded-resource scenario characteristic of large parallel computing centers. In addition, we recommend hardware support for parallel prefix operations above and beyond all-reduce so that more sophisticated solvers such as multigrid may be implemented at speed.

Acknowledgments

This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357 . This research used resources of the Oak Ridge Leadership Computing 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 and resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

References

[1]Patera, A., "A spectral element method for fluid dynamics : laminar flow in a channel expansion," *J. Comput. Phys.*, Vol. 54, 1984, pp. 468–488.

[2]Fischer, P. and Patera, A., "Parallel Spectral Element Solution of the Stokes Problem," *J. Comput. Phys.*, Vol. 92, 1991, pp. 380–421.

[3]Deville, M., Fischer, P., and Mund, E., *High-order methods for incompressible fluid flow*, Cambridge University Press, Cambridge, 2002.

[4]Sprague, M., Churchfield, M., Purkayastha, A., Moriarty, P., and Lee, S., "A comparison of Nek5000 and OpenFOAM for DNS of turbulent channel flow," *Nek5000 Users Meeting*, Argonne National Laboratory, 2010.

[5]Fox, G. C., Johnson, M. A., Lyzenga, G. A., Otto, S. W., Salmon, J. K., and Walker, D. W., *Solving Problems on Concurrent Processors*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[6]Fischer, P. and Patera, A., "Parallel Simulation of Viscous Incompressible Flows," *Ann. Rev. Fluid Mech. 1994*, Vol. 26, 1994, pp. 483–528.

[7]Gropp, W., Lusk, E., and Thakur, R., *Using MPI-2: Advanced Features of the Message-Passing Interface*, MIT Press, Cambridge, MA, 1999.

[8]Golub, G. and Loan, C. V., *Matrix Computations*, Johns Hopkins University Press, Baltimore, 1996.

[9]Hoemmen, M., *Communication-Avoiding Krylov Subspace Methods*, Ph.D. thesis, University of California, Berkeley, 2010, Berkeley, California.

[10]Bell, N., Dalton, S., and Olson, L., "Exposing Fine-Grained Parallelism in Algebraic Multigrid Methods," *SIAM Journal on Scientific Computing*, Vol. 34, No. 4, 2012, pp. C123–C152.

[11]Otten, M., Gong, J., Mametjanov, A., Vose, A., Levesque, J., Fischer, P., and Min, M., "An MPI/OpenACC Implementation of a High Order Electromagnetics Solver with GPUDirect Communication," Submitted to Int. J. High Perf. Comput. Appl.