Characterizing Parallel Scaling of Scientific Applications using IPM

Nicholas J. Wright, Wayne Pfeiffer, and Allan Snavely

Performance Modeling and Characterization Lab

San Diego Supercomputer Center

9500 Gilman Drive, La Jolla, CA 92093-0505

Abstract

Scientific applications will have to scale to many thousands of processor cores to reach petascale. Therefore it is crucial to understand the factors that affect their scalability. Here we examine the strong scaling of four representative codes that exhibit different behaviors on four machines. We demonstrate the efficiency and analytic power of our performance monitoring tool, IPM, to understand the scaling properties of these codes in terms of their communication and computation components. Our analysis reveals complexities that would not become apparent from simple scaling plots; we disambiguate application from machine bottlenecks and attribute bottlenecks to communication or computation routines; we show by these examples how one could generate such a study of any application and benefit from comparing the result to the work already done. We evaluate the prospects for using extrapolation of results at low concurrencies as a method of performance prediction at higher concurrencies.

1. Introduction

Moore's Law continues unabated; the number of transistors on a chip still doubles every eighteen months. However, because of power constraints, clock speeds remain approximately constant. Hence to harness increasing amounts of computational power, scientific codes must achieve higher and higher levels of concurrency. The petascale era will be one of massive parallelism.

It is therefore important to understand the factors affecting the scaling behavior of scientific codes so that the capabilities of petascale machines can be fully exploited. Scaling behavior depends upon both the algorithm the code is using and the machine characteristics. For example, a code could have a serial bottleneck that only begins to manifest itself above a certain concurrency, or the machine could have a bandwidth that is too low or a latency that is too high to keep up with the computation. Understanding code performance in the petascale era will require knowing whether any scalability limit that has been reached is because of some intrinsic property of the machine or because of the algorithm. That is, before removing a bottleneck, one has to first know its location and cause. If one performs scaling scans on a few machines and records only the performance (run time), it is very difficult from that to understand *why* the performance is the way it is. Advanced performance modeling techniques can reveal the why but are laborious to produce; we show by example a middle path that is lightweight in the data acquisition phase and relatively simple in the analysis phase that can answer crucial questions such as "what is the primary scaling bottleneck on this application/machine combination?"

For this study we measured the scaling characteristics of four codes (DNS, MILC, PARATEC, and WRF) on four different machines (DataStar, Franklin, Lonestar, and Ranger). For each code we ran a *strong* scaling scan, in which the problem is fixed and the core count is varied, since this gives us most insight into scalability behavior. (This is in contrast to a *weak* scaling scan, in which the problem size is increased concurrently with the core count to keep the work per core roughly the same.) By measuring the performance using IPM (the Integrated Performance Monitoring tool) [1], we are able to analyze separately, with very low overhead (< 5%), the computation and communication characteristics of each code. IPM allows us to see the time spent in individual MPI routines, the sizes of the messages sent, and to classify the communication pattern. The profiles obtained from IPM are comparable *across machines*, which gives us insight as to why each code scales as it does on each machine. Such knowledge is very useful for users trying to understand the scaling behavior of their codes as well as for centers looking to procure machines that enable good scalability.

Many previous studies have been performed that examine the performance of scientific codes on high-performance computers. One of the most comprehensive, recent studies is that of Oliker, *et al.* [2], which examined the performance of six codes on five different architectures. Performance evaluations of the Cray XT3 and XT4 architectures were also presented recently [3, 4]. These two papers contain results obtained on other architectures as well, but their focus was mostly upon evaluation of the XT architecture itself. Another recent performance comparison was performed by Hoisie, *et al.* [5] who studied the BG/L, Purple, and Red Storm machines. Other studies looking at scalability for the purposes of determining bottlenecks have been published. A recent example is Coarfa, *et al.* [6], who uses their HPCtoolkit software in order to determine the location of scalability bottlenecks at the subroutine level. Although this work provides interesting information about where the scalability bottlenecks are located, it does not address the underlying causes of the issues identified.

One project that has similar aims to ours is Prophesy [7]. It aims to analyze and model the performance of a scientific application using a database to aid in gaining the needed insights. In one mode of operation it aims to model the performance of scientific applications using performance analysis and measurements and then makes performance projections based upon such analysis. In order to do this the scientific code of interest is analyzed by hand in order to identify *kernels*. The runtime for each kernel is then measured and an equation based upon the coupling between the kernels is produced. While this method has impressive accuracy for simple benchmarks [7b] its applicability to real scientific codes is limited by the need for a human to define the kernels. For example, for an application like WRF that has >100K lines of source code such analysis is completely unfeasible. The Prophesy system has also been used in a slightly different mode of operation in order to determine a performance model of the MILC application [7c]. In this case the runtime at a variety of core counts (8-2048) was fit to a polynomial expression in order to determine the coefficients of the expansion and predict the runtime at 4096 cores. The resulting model was reasonable at predicting the runtimes although no justification for the expression used was presented and it is difficult to see how it can be tied to the characteristics of the machine in question or to provide any information that allows the comparison of performance across machines. In contrast, the primary focus of this paper is to describe a practical method of lightweight analysis that one can do with reasonable effort on largescale codes and machines to identify location and cause, as well as possible ameliorations, of scalability bottlenecks. Specifically we focus on whether the machine or the code is the primary bottleneck to scalability.

This paper is organized as follows, Section 2 describes the procedures we used to collect the measurements, Sections 3 and 4 give a description of the architectures of the four machines and the applications used in the study respectively, Section 5 describes our results and Section 6 is the discussion and conclusions.

2. Procedure

Each of the four codes was run on all four machines using IPM to gather performance information. IPM is a portable profiling infrastructure for parallel codes initially developed at Lawrence Berkeley National Laboratory. It combines information from multiple sources -- hardware counters, the profiling interface to MPI (P-MPI) and the operating system -- into a single job level profile that spans all tasks to provide a performance summary of the computation and communication in a parallel program. IPM is easy to use (at most relinking is required), has extremely low overhead and is scalable, which means that it is ideal for this work, as it gives us a method of measuring performance in production and at scale without significantly perturbing the application's performance. Additionally, by analyzing the data flow, IPM can distinguish the communication pattern exhibited by the code, which gives insight into the demands that the code places upon the machine it is running on. IPM also records the sizes of the messages that are sent. This allows us to determine if we are in the bandwidth or latency limit at particular core count, as well as to look at the variation in message size and total amount of data sent as function of core count. All of this data provides us with information about the scalability of the code.

IPM provides a breakdown of the run time in terms of time spent computing and time spent in MPI, with the latter in terms of the individual MPI calls made. We associate the time spent in MPI with the time spent communicating. There is not, however, a perfect correspondence between these two quantities, since factors such as overlapping computation and communication can cloud the distinction. In cases where such effects are important we will highlight them. All timings reported correspond to the average across all MPI tasks. In the presence of load imbalance or cases in which different MPI tasks are doing different things, there is a possibility that this can be misleading. Again, in cases where such effects are important we will highlight them.

We initially made a single run for each combination of code, machine, and core count in a scan. When a run seemed unusually slow, we repeated it to ensure that we achieved optimal performance. Since we were always performing strong scaling scans, a questionable result could often be identified by comparison with the results obtained at adjacent core counts. On the newer machines used in this study, significant variation between run times was sometimes observed from run to run, occasionally by as much as 50%. Our analysis of the IPM data always indicated that these slowdowns were in the communication time, presumably because of network contention from other user jobs or the operating system.

Additional runs for each combination of code, machine, and core count in a scan using half of the cores per node were made. These results gave us additional insight into the factors affecting scalability as they are sensitive to a different balance of machine parameters. For example, these runs have half of the demand upon the network interface in each node as compared to the fully loaded run and half the memory bandwidth demands. The importance of having a low-overhead tool like IPM should be apparent from the number of scans.

For each code we ran a "standard" benchmark problem designed to run in a reasonable amount of wall-clock time. These benchmarks are representative of production problems, but use fewer time steps or iterations. Some of the standard problems also include problem initialization. Since this can limit scalability compared to production problems involving many more time steps or iterations, we excluded the initialization time from all of the problems using a feature of IPM that allowed us to collect MPI data from only specific regions of the code.

3. Computer Systems

The key characteristics of the four computers used in our study are summarized in Table 1. Included are four processor types from the dominant vendors, a wide range of cores per node, and three types of interconnect.

DataStar is a heterogeneous cluster with IBM Power4+ processors at the San Diego Supercomputer Center. The primary compute partition contains 272 8-way nodes connected via a two-link network adaptor to an IBM Federation switch, which has a fat-tree topology. All of the results reported here on 1,024 cores or less were obtained on 1.5-GHz processors, while the results on 2,048 cores used a mix of 1.5- and 1.7-GHz processors.

Table 1. Key characteristics of computer systems

System name	Location	Processor	Clock speed (GHz)	Max flops/ clock	Peak Gflop/s/ core	Cores / node	Node	Interconnect
DataStar	SDSC	IBM Power4+	1.5/1.7	4	6.0/6.8	8	IBM p655	Custom fat tree
Franklin	LBNL/ NERSC	AMD Opteron	2.6	2	5.2	2	Cray XT4	Custom 3D torus
Lonestar	TACC	Intel Xeon (Woodcrest)	2.66	4	10.6	4	Dell Power-Edge 1955	InfiniBand SDR fat tree
Ranger	TACC	AMD Opteron (Barcelona)	2.0	4	4.0	16	SunBlade x6420	InfiniBand SDR fat tree

Franklin is a Cray XT4 system at Lawrence Berkeley National Laboratory (LBNL). At the time the measurements were made, Franklin had dual-core Opteron processors running at 2.6 GHz and was running compute node Linux. Each node contains one dual-core processor and is connected to a 3D-torus network arranged in a 17x24x24 topology.

Lonestar is an Intel Xeon 5100 (Woodcrest) system at the Texas Advanced Computing Center (TACC) with 1,300 nodes connected with 2 layers of InfiniBand SDR switches. There are 25 leaf switches in the first layer and 4 core switches in the second layer. Each node contains two dual-core Intel Xeon processors running at 2.66GHz.

Ranger is an AMD Opteron (Barcelona) system at TACC with 3,936 nodes connected with two Sun Constellation InfiniBand switches, each of which has 3,456 ports and a full Clos fat-tree topology. Each node contains four quad-core processors running at 2.0 GHz and a single SDR InfiniBand adaptor.

4. Scientific Applications and Problems

The four applications selected for study span a representative range of algorithms and scaling behavior as summarized in Table 2. They also consume large amounts of time at NSF and DOE supercomputer centers.

Table 2. Key characteristics of applications

Name	Solution domain	Numerical methods	Computation speed scaling	Dominant communication type and MPI calls
DNS	Direct numerical simulation of turbulence	3D FFTs in space with time stepping	Linear	Collective: MPI_Alltoallv
MILC	Quantum chromodynamics	Conjugate gradient iteration, sparse matrix-vector operations	Superlinear	Point-to-point on 4D space-time lattice: MPI_Wait + collective: MPI_Allreduce
PARATEC	Electronic structure calculation	Conjugate gradient iteration, 3D FFTs	Linear	Collective: MPI_Allreduce & MPI_Wait
WRF	Weather prediction	2D finite differences in space with time stepping	Sublinear	Point-to-point on 2D mesh: MPI_Wait

DNS ([8,9]) is a code for Direct Numerical Simulation of turbulent flows that is routinely used to simulate turbulent mixing at high Reynolds and Schmidt numbers. The basic numerical method is pseudospectral in space and second-order explicit predictor-corrector in time. The code is written in Fortran, uses MPI for communication, and can be run with either 1D or 2D processor decomposition. The most computationally intensive part involves 3D FFTs, which include collective all-to-all processor exchanges for data transposes. The subsidiary 1D FFTs are done with ESSL on DataStar and FFTW on the other computers.

The benchmark problem considered here is a segment of a turbulent flow DNS simulation run for 10 time steps using a 1024^3 grid and a 2D processor decomposition. I/O (in addition to initialization) is excluded from the benchmark.

MILC [10] is a code to compute properties of elementary particles subject to the strong force as described by quantum chromodynamics (QCD). The computational domain is a 4D space-time lattice. Most of the computational work is spent doing a conjugate gradient inversion of the fermion Dirac operator. This entails repeated calls to a kernel that multiplies a large sparse matrix by a vector. Communication is mostly from point-to-point exchange of ghost-cell data on the 4D lattice.

The benchmark problem considered here is the NERSC medium problem (excluding initialization) It performs 1,375 conjugate gradient iterations on a 32^4 lattice.

Optimized source code was used on DataStar in the form of assembly-coded matrix-vector routines (which are part of the distributed code). On the other three computers the default C version was used.

"**PARATEC** performs ab initio quantum-mechanical total energy calculations using pseudopotentials and a plane wave basis set." [11] The folded spectrum method described by Canning, *et al.* [11b] is used and entails conjugate gradient iteration to minimize a function whose evaluation requires 3D FFTs. These FFTs require an all-to-all transpose, similar to that done in DNS, except that the implementation is in terms of point-to-point MPI routines.

PARATEC is written in Fortran 90 and uses several standard libraries. Of special note is IBM's ESSL on DataStar, AMD's ACML on Ranger and Franklin, and Intel's MKL on Lonestar. FFTW was used for performing 1D FFTs on all the machines except DataStar. Since much of the computation time (typically 60%) is spent in these routines, PARATEC runs at a high percentage of peak on most systems until communication performance becomes limiting.

The benchmark problem considered here is the NERSC large problem (excluding initialization). It performs 10 conjugate gradient iterations for a system consisting of 686 silicon atoms in the diamond structure. We improved communication performance by setting the parameter number_bands_fft to 20 in the input file. Using this larger value instead of the default value of 1 allows messages to be aggregated to reduce communication time and improve scaling. The computed results remain the same, since this input change affects only how communication is done.

WRF (Weather Research and Forecast) is a weather modeling system developed at NCAR. [12] WRF provides a limited-area model of the atmosphere for mesoscale research and operational numerical weather prediction. The WRF model represents the atmosphere as a number of variables of state discretized over regular Cartesian grids. The model solution is computed using an explicit high-order Runge-Kutta time-split integration scheme in the two horizontal dimensions with an implicit solver in the vertical. Since WRF domains are decomposed over processors in the two horizontal dimensions only, interprocessor communication is between-neighbor on most supercomputer topologies. In this work we use Version 2.1.2 of WRF and the TI-07 large benchmark. The latter models the Continental U.S. with 2.5 km resolution on a 1501 x 1201 x 35 grid and runs for 9 simulated hours with a time step of 15 seconds. Each time step involves 36 halo exchanges and a total of 144 nearest-neighbor exchanges (assuming aggregation). We ran with parallel I/O switched on (one output file per MPI task). This significantly improved the scalability on all computers as compared to the default serial I/O.

5. Scaling Results

The strong scaling behavior of each application on the various computers can be seen from the four plots in Figure 1. Each plot shows the speed per core versus number of cores with the curves normalized to the DataStar speed at the lowest core count. A flat curve indicates ideal, linear scaling, and a higher curve is faster. Speed is just proportional to the inverse run time, and the DataStar run time used for the normalization is listed in each plot.

Plots for some of the applications show clear differences in scalability between the computers. For example, Figures 1b and 1c show that DataStar scales much better than Lonestar for the MILC medium and PARATEC large problems. The difference in scalability is so large that DataStar runs faster than Lonestar at high core counts,

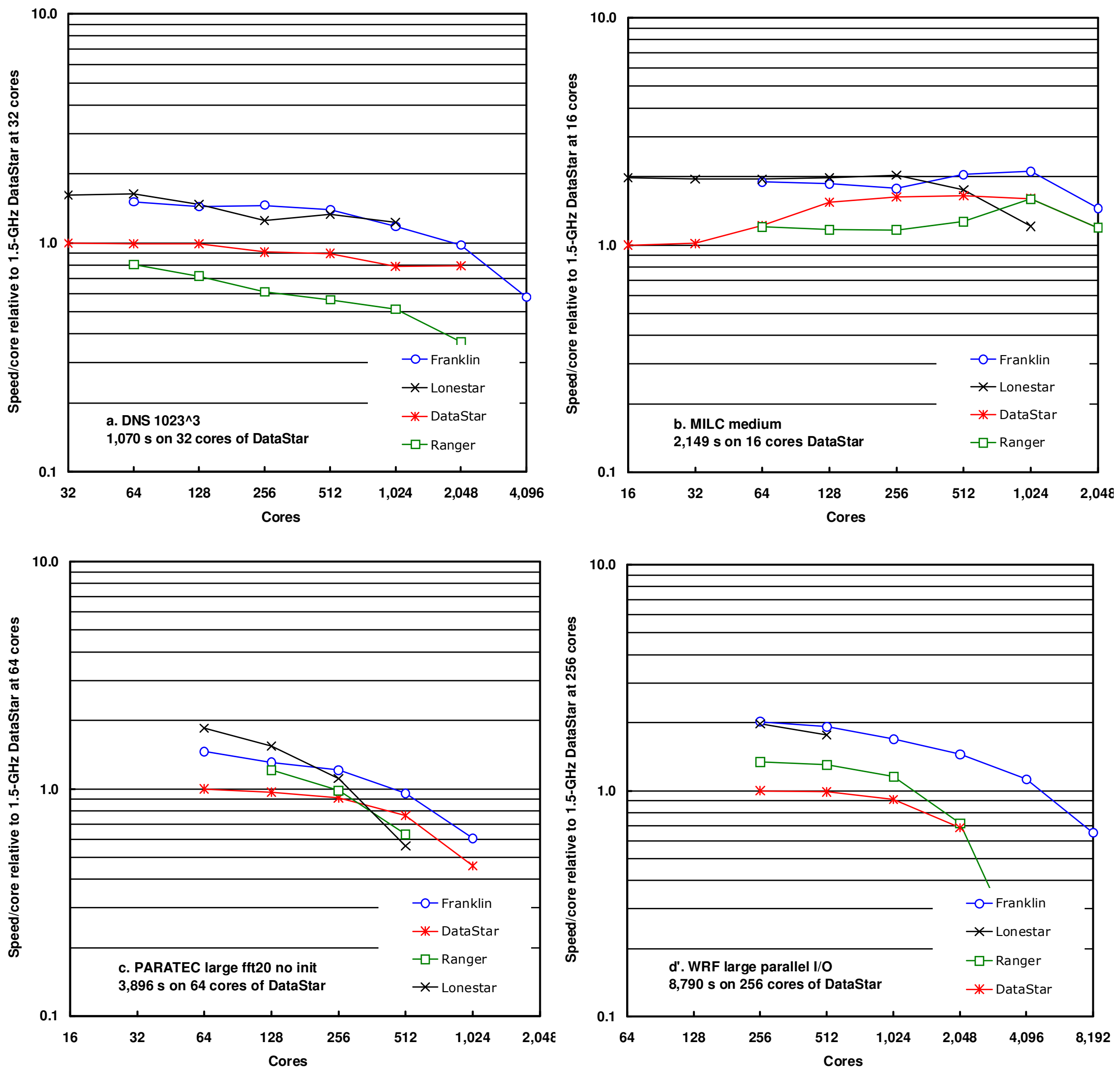

Figure 1. Scaling of relative speeds per core on four computers for each application.

whereas the reverse is true at low core counts. The natural question remains why?

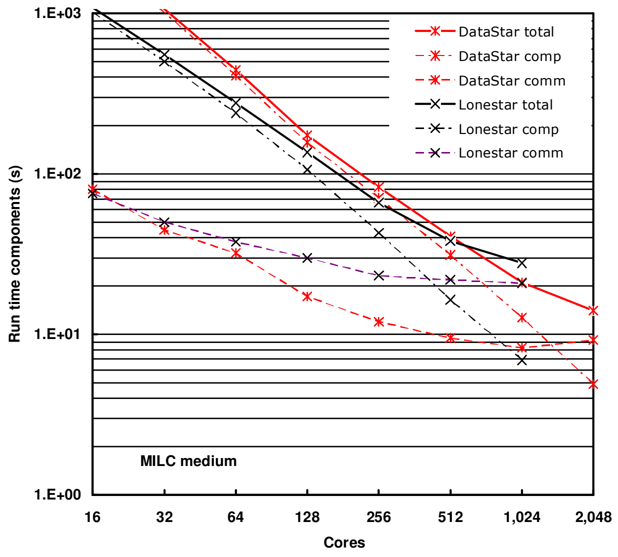

Figure 2. Comparison of computation and communication time scalings for the MILC medium problem on DataStar and Lonestar.

IPM can help us understand the underlying behavior that causes differences in scalability by separating the computation and communication times for codes where all communication is done by MPI. Specifically, IPM measures the time spent in MPI, which can be subtracted from the total time to give the computation time.

As an example consider Figure 2 showing the separate computation and communication times along with the total run times of the MILC medium problem on both DataStar and Lonestar. As is generally the case, each communication time curve decreases more slowly than its corresponding computation time curve. Eventually the curves cross, and strong scaling ends soon thereafter. This cross-over occurs below 512 cores on Lonestar, but above 1,024 cores on DataStar, reflecting the better network on DataStar. As a result, the total run time curves for DataStar and Lonestar cross just above 512 cores, consistent with the behavior shown in Figure 1b. Thus IPM allows one to attribute DataStar performance overtaking that of Lonestar at 1,024 cores to a superior network.

Figures 3 and 4 separately characterize the computation and communication time scaling of each application on the various computers. Since computation times often exhibit ideal scaling and hence decrease as the inverse core count, in Figure 3 we plot the inverse computation time per core, i.e., speed per core (just as for the total time in Figure 1). This highlights the departure from the ideal as deviation from a flat curve. Because the communication times decrease more slowly and often reach an asymptote, or even increases, we plot those times with no normalization in Figure 4. In other words, in Figure 3, which is plotted in terms of performance, ‘ideal’ performance corresponds to a horizontal line, whereas in Figures 4 and 5, which are plotted in terms of time, the faster slope declines with increase in core count, the better the performance.

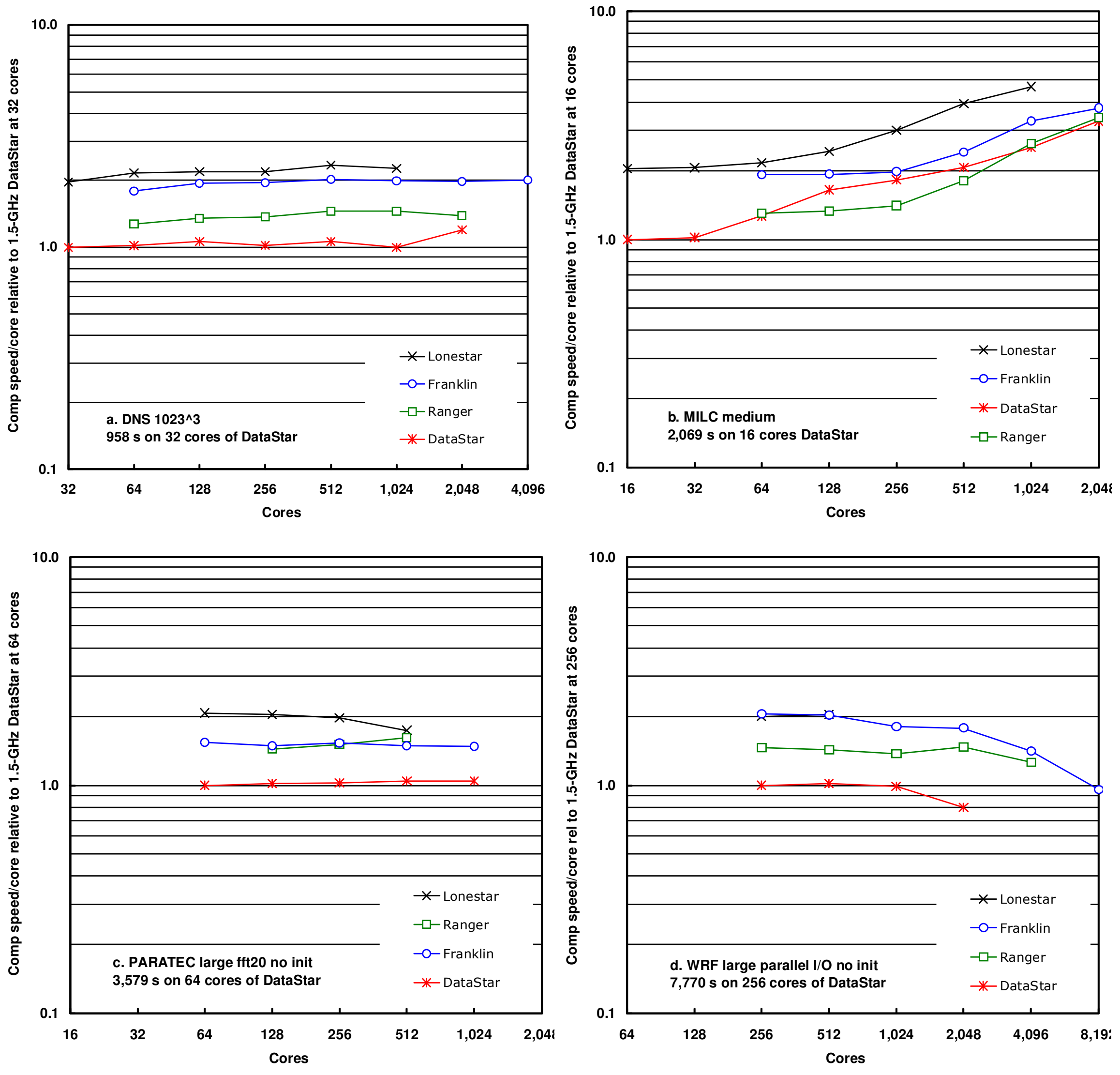

Figure 3. Scaling of relative computation speeds per core on four computers for each application. DNS and PARATEC are linear, MILC is superlinear, and WRF is sublinear, in computation speed .

Using IPM, the communication times can be further broken down into times for each type of MPI call. Figure 5 shows scaling plots of such MPI data for each application, but corresponding to only a single computer in each case. These plots show that communication in DNS and WRF is dominated by single types of MPI calls: Alltoallv in DNS and Wait in WRF. By contrast, more than one type of MPI call can be important in MILC and PARATEC depending upon the core count.

The time to execute a point-to-point message-passing operation can be approximated as follows:

$$t_m = m/b + t_{lat} \ , \tag{1}$$

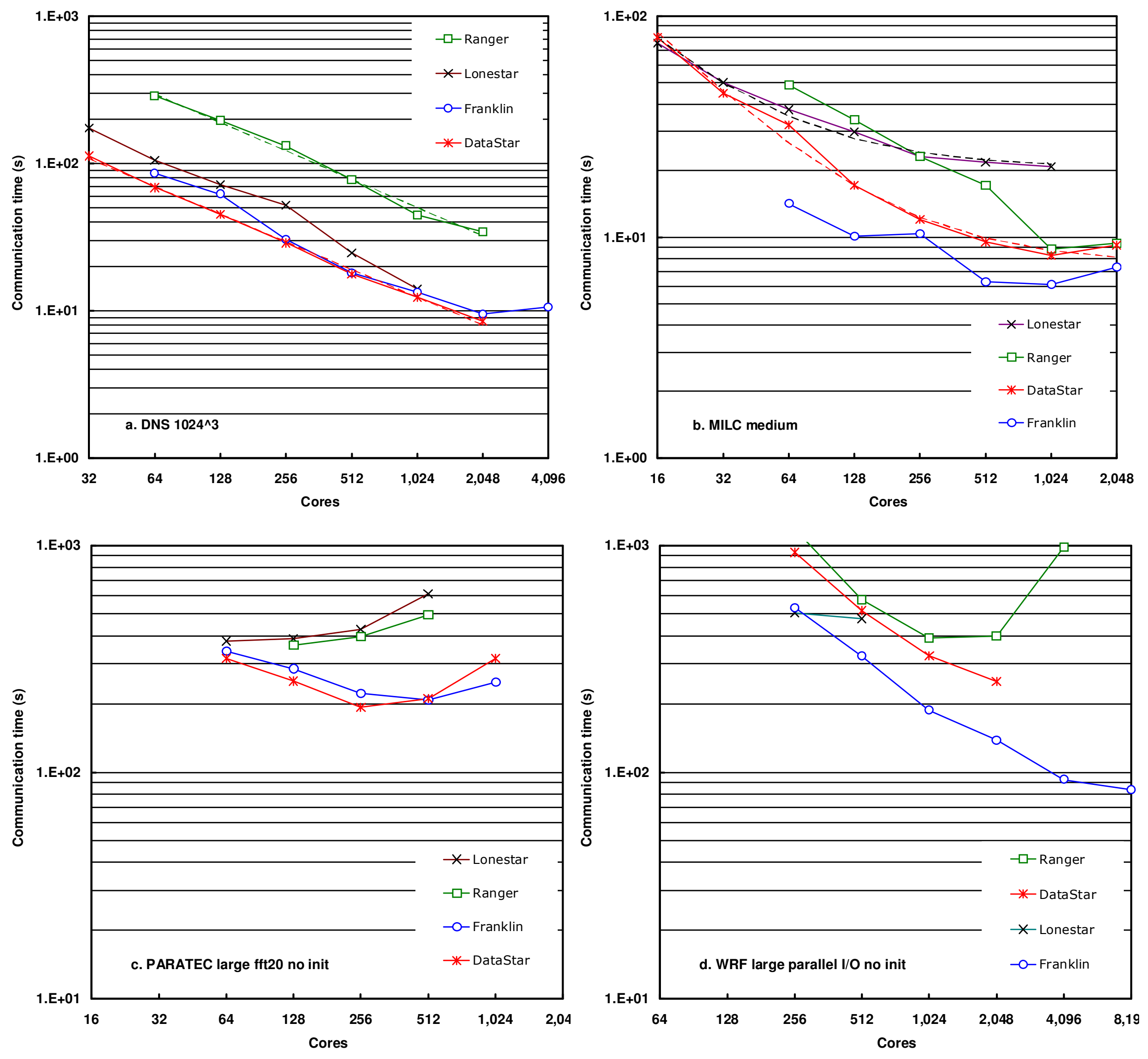

Figure 4. Scaling of communication times on four computers for each application. Solid lines are for measured data, while dashed lines in a. and b. are for fits. DNS obeys a power law, while other applications are sublinear, in communication speed scaling.

where m is the message size, b is the bandwidth, and t_{lat} is the latency. Then the wall-clock time to transmit n_m such messages shared equally among p processes (assumed equal to the number of cores) is

$$t_{comm} = \frac{n_m}{p} t_m = \frac{n_m m}{pb} + \frac{n_m}{p} t_{lat} \equiv \frac{d}{pb} + a t_{lat} \ . \tag{2}$$

Often it is the case that d, the total amount of data transmitted, and a, the number of operations per process, are constant or nearly so with respect to p. If the bandwidth and latency are also constant with respect to p, then Eq. (2) gives a simple, explicit expression for the p dependence of the communication time that has a characteristic

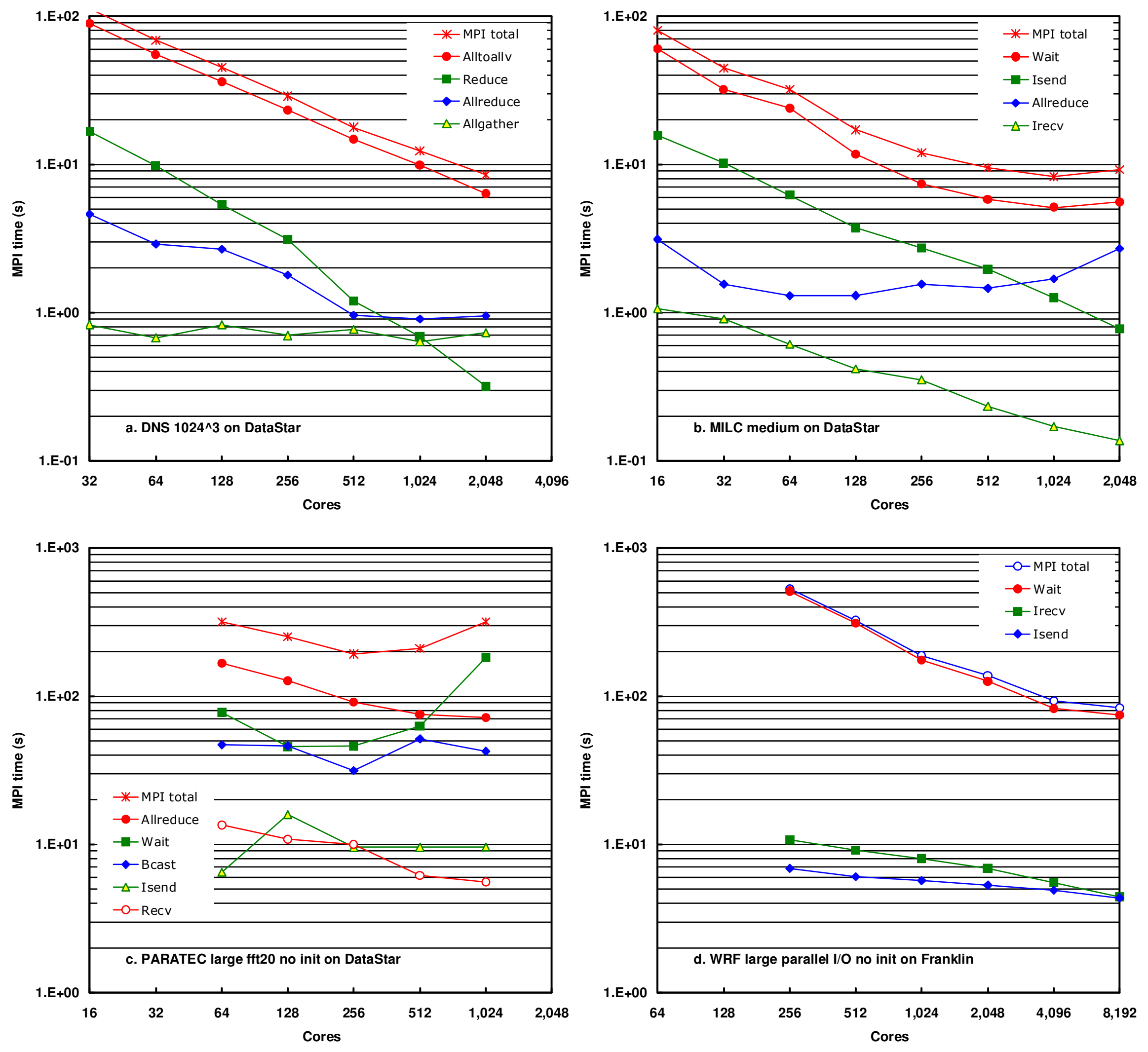

Figure 5. Scaling of communication times by major MPI calls on a single computer for each application.

shape similar to that shown for Lonestar in Figure 2. Eq. (2) also applies for collective communication with coefficients d and a that depend upon p (per p. 187 of [16]). Eq. (2) will be useful in our subsequent analysis.

5.1 DNS

Figure 1a shows that DataStar, Lonestar, and Franklin scale similarly for DNS up to 1,024 cores, whereas the scaling on Ranger is poorer. Also, there is a sizeable range in absolute performance, with Ranger being the slowest per core. Compared to Ranger on 1,024 cores, DataStar, Franklin, and Lonestar are 1.5x, 2.3x, and 2.4x faster, respectively.

The relative speeds of the different computers can be better understood by looking at Figures 3a and 4a, which separately plot the computation and communication scalings. These plots clearly show that Ranger's overall slowness arises from its much slower communication speed (i.e., its higher communication time). As can be seen from Figure 5a, the communication time for DNS is dominated by the collective MPI_Alltoallv routine, so this routine must run much slower on Ranger.

Now let us look at the individual computation and communication time scalings for DNS. Since initialization and I/O are not included in the benchmark problem, we expect that the computation speed will exhibit nearly ideal, linear scaling with p, so the computation speed per core should be constant. This is indeed the case for all of the computers, as can be seen in Figure 3a. At 1,024 cores, the computation speeds on Lonestar, Franklin, and Ranger are 2.3x, 2.0x, and 1.5x faster, respectively, than on DataStar.

As we showed in a recent paper [14], application run times can often be correlated with microbenchmark metrics in the HPC Challenge benchmark set [15]. For DNS the computation time is well correlated with the clock speed, rather than flop speed or memory bandwidth, except on Lonestar, which runs faster than its clock speed alone would predict.

Turning to communication, the time scaling for DataStar is fit extremely well empirically by a power law, $1/p^{0.62}$, which is shown as the dashed line in Figure 4a. Similar power laws approximate the data for Ranger (as can be seen in Figure 4a) as well as for Franklin and Lonestar, though the data for the latter vary more from their fits. The first row of Table 3 lists the power-law exponents for all four fits, with the outlier at 4,096 cores on Franklin excluded.

Table 3. Exponents and bandwidths for power-law fits to DNS communication times

Quantity	DataStar	Franklin	Lonestar	Ranger
t_{comm} exponent	-0.624	-0.665	-0.711	-0.638
1/(p*RR bandwidth) exponent	-0.654	-0.742	-0.770	-0.830
Effective bandwidth (MB/s) at 1,024 cores	109	100	95	30
RR bandwidth (MB/s) at 1,024 cores	91	93	82	34
Effective bw / RR bw at 1,024 cores	1.20	1.07	1.15	0.89

Pekurovsky, et al. [13] derived upper and lower bounds for the communication time scaling. Both decrease roughly as $1/p^{2/3}$, with some oscillation for power-of-2 core counts. The exponents found empirically are thus in good agreement with the model exponent.

Calls to MPI_Alltoallv with two different message sizes account for most of the communication time (75% or more at all core counts on all four computers). Since these message sizes are very large – 131 kB or 4.2 MB at 1,024 cores – communication bandwidth, not latency is limiting (as assumed in Pekurovsky's model). We can thus compare the observed communication time scaling with that suggested by the first, bandwidth-limited term in Eq. (2). Since d, the total amount of data transmitted, is constant, our power-law fits imply that the effective communication bandwidth decreases with core count, e.g., as $1/p^{0.38}$ for DataStar.

From IPM the amount of data being transferred by MPI_Alltoallv can be obtained, which allows the effective communication bandwidth to be determined. For DataStar, this bandwidth is 136 MB/s at 1,024 cores for the MPI_Alltoallv time alone or 109 MB/s (i.e., 20% lower) when the entire communication time for all types of MPI communication is considered. Effective bandwidths for all of the computers at 1,024 cores are listed in Table 3.

For DNS the communication time is highly correlated with the inverse of the random ring (RR) bandwidth,[1],which quantifies the effect of congestion on a heavily loaded interconnect (ie. time α $1/RR_{BW}$) . Indeed, the RR bandwidths are well fit by power laws, and the exponents for the fits to the inverse product of p times the RR bandwidth for each computer are listed in Table 3. These exponents are close to those for the communication time fits, especially for DataStar.

The measured value for the random ring bandwidth on 1,024 cores of DataStar is 91 MB/s. This is 1.5x lower than the MPI_Alltoallv bandwidth and 1.2x lower than the effective bandwidth for all communication. Hence MPI_Alltoallv on DataStar appears to have somewhat less congestion than random ring communication, which is reasonable since the communication pattern of MPI_Alltoallv is more regular. For Franklin at 4,096 cores, Lonestar at 256 cores, and Ranger at 2,048 cores, the MPI_Alltoallv bandwidths are actually slightly lower than the random ring bandwidths. This suggests a greater effect of congestion or a poorer MPI implementation in these cases.

Often we are interested in predicting performance from measurements at lower core counts to higher core counts. Looking at the measured data in Figure 4a, we see that power-law fits to the first four communication times on DataStar and Ranger will give good extrapolations to higher core counts. By contrast, such extrapolations for the other computers are poorer.

Another prediction of interest is from one computer to another. One way of doing this is to use the random ring bandwidth as the predictor, similar to the method used in Ref. [14]. This works well, since the ratio of the effective

[1] The random ring bandwidth is measured by the HPCC benchmark suite [15].

bandwidth to the random ring bandwidth is relatively constant across the computers and core counts, as suggested by the data on 1,024 cores in the last row of Table 3. Outliers are Franklin at 4,096 cores, Lonestar at 256 cores, and Ranger at 1,024 cores, which have relatively low MPI_Alltoallv bandwidths, as noted previously.

DNS is a good candidate for petascale; its reliance on collectives rather than point-to-point communications is perhaps its primary scaling bottleneck. This issue could be ameliorated by using non-blocking collectives, which would allow the overlap of computation and communication.

5.2 MILC

Figure 1b shows that the overall speed of each computer scales differently for MILC. DataStar exhibits significant superlinear speedup from 32 to 128 cores due to better cache utilization and only shows decreasing scalability at 2,048 cores. By contrast, Lonestar shows no superlinear speedup and exhibits decreasing scalability at 512 cores. Franklin and Ranger show smaller amounts of superlinear speedup and at higher core counts than DataStar and then exhibit decreasing scalability at 2,048 cores, just as DataStar does. This superlinear phenomena is well known in the QCD community and occurs for many QCD codes. (For example see Ref. [17].)

As for relative speeds, Lonestar is the fastest at low core counts, being 2x as fast as DataStar there, but the slowest at high core counts because of its poorer scalability. On 1,024 cores, Franklin is 1.7x faster than Lonestar, while DataStar and Ranger are both 1.3x faster.

Figure 3b shows the scaling of the computation speed inferred from the IPM communication time data. Very large superlinear speedups are observed on all of the computers, though the core count of their onset varies. These speedups are due to improved memory access and cache utilization as the memory footprint per core decreases. This behavior is consistent with the correlation between computation speed and memory bandwidth reported in [14] for the MILC medium problem run on 64 cores of a more extensive set of computers.

Communication in MILC is primarily point-to-point on a 4D lattice. The resulting MPI time is mostly for MPI_Wait following calls to the non-blocking MPI_Isend and MPI_Irecv routines. The scaling of the associated MPI time distribution is shown in Figure 5b for DataStar. Although the time for MPI_Wait dominates, this dominance of a single MPI routine is not as great as that for DNS or WRF. On 2,048 cores of Franklin, the time for MPI_Allreduce slightly exceeds that for MPI_Wait.

We expect that the total MPI time, apart from that due to MPI_Allreduce, can be approximated by Eq. (2) with nearly constant coefficients. By contrast, the MPI_Allreduce time is associated with small messages and so should be given by only the latency term in Eq. (2), with the coefficient a increasing logarithmically with p (per p. 187 of [16]).

Proceeding empirically, we fit Eq. (2) with constant coefficients to the measured computation times on DataStar and Lonestar, the computers with the most and smoothest data. We also weighted the fit assuming that each measurement had the same relative error as described in [14]. This led to the fits shown in Figure 4b, which are quite good.

Further investigation, however, revealed two complicating effects. First, we examined the detailed IPM output and found that d, the total amount of data transmitted, is not constant, but grows roughly as $p^{0.26}$. Second, by changing the location of the MPI_Wait calls in the code, we showed that there is substantial, beneficial overlap of communication with computation. Indeed, we found that on DataStar 20 to 25% and on Lonestar 25 to 35%, of the time taken to send the messages was being hidden by the overlap.

Finally, we looked at whether we could use our empirical fitting procedure to extrapolate the communication time from the first four core counts on DataStar and Lonestar. We found the extrapolation to be very good for DataStar, but somewhat poorer for Lonestar.

MILC is a strong candidate for petascale. To ensure good scaling it will be essential that the point-to-point messages stay above the latency limit and the time in MPI_Allreduce does not become prohibitively expensive.

5.3 PARATEC

PARATEC exhibits the poorest scalability of the benchmarks considered, especially on Lonestar and Ranger, as can be seen by comparing Figure 1c with 1a, 1b, and 1d. Figures 3c and 4c show that the poor scalability is entirely due to communication. Indeed, the communication times on all four computers eventually increase with core count.

Examination of Figure 5c shows that the growth in communication arises from the MPI_Wait routine. This is called after MPI_Isend and MPI_Recv, and all three routines effectively perform a collective MPI_Alltoallv, similar to that in DNS. However, the messages sizes in PARATEC are relatively small (a few kB, even after setting number_bands_fft = 20), so the latency term in Eq. (2) is limiting rather than the bandwidth term. Increasing number_bands_fft to a higher value might improve scalability somewhat (at the cost of more memory usage for buffering MPI data). More promising ways to improve scalability, though, are to solve a larger problem and look for additional opportunities for parallelism.

5.4. WRF

Figure 1d shows that WRF scales similarly on all the machines up to 1,024 cores. There is, however, a sizeable range in absolute performance. Lonestar and the Franklin XT4 perform approximately equally and are 1.5x faster than Ranger and 2x faster than Datastar. At 2,048 and 4,096 cores, the performance on Ranger begins to decrease rapidly, whereas Franklin has reasonable performance all the way to 8,192 cores.

The relative speeds of the different computers can be better understood by looking at Figures 3d and 4d, which separately plot the computation speed and communication time. Figure 3d clearly shows that part of Datastar's overall slowness arises because it has the slowest processors: the compute speed on Ranger is 1.5x faster, and Franklin and Lonestar are 2x faster. The relative communication performance of Datastar is much better however; in fact it outperforms Ranger at every core count.

Figure 5d shows that almost all the communication time in WRF on Franklin (and the other three machines) is spent in MPI_Wait. This agrees with the description of the code given earlier: WRF predominately communicates using point-to-point messaging, and the significant amount of time spend in MPI_Wait is simply a reflection of that. The poor scalability of Ranger at 2,048 cores and above is clearly because of the poor performance of interconnect, which gets dramatically worse at 2,048 cores. Since the communication pattern is so simple, involving no collectives and almost the minimum possible for contention, it is perhaps a little surprising that the scalability of Ranger is so poor, even in comparison to DataStar. The communication performance of Franklin is much better than the other machines for WRF and is clearly the reason for its superior overall scaling performance.

Results obtained on half of the cores per node show identical communication performance to those using all the cores on every machine. This indicates that there is no significant contention for on-node network resources in WRF, which seems reasonable given the communication pattern. Computation times are approximately 1.06x to 1.16x faster using half the cores at the lowest core count. This factor decreases with increasing core count until it becomes unity at 4,096 cores on Franklin. This is simply a reflection of the decreased competition for memory bandwidth as the size of the data objects being worked on gets smaller.

Examination of the communication time scaling for WRF on Franklin in Figure 5d shows that it does not behave 'ideally', as $1/p$, but decreases to a plateau. The obvious explanation for this behavior is that the message size has reached the latency limit. However, examination of the message size distribution produced by IPM shows that this is not the case. At 8,192 tasks the smallest message is 3 kB. In fact, the average amount of data sent per

MPI task essentially follows a power law: $d/p \sim 1/p^{0.4}$. The deviation of the exponent from the ideal, $1/p$ case is because WRF uses a ghost-cell region that is five grid points wide, which slows the decrease in average message size. At 4,096 cores the message size in one of the dimensions reaches the minimum allowed (because of the ghost-cells), so very little decrease in communication time is observed between 4,096 and 8,192 cores. In effect what is happening is that WRF reaches its scalability limit at 4,096 cores, before reaching the latency limit of the machine. The ghost-cell region is also what is causing the compute performance to drop (Figure 3d) as there is some increased compute cost associated with the presence of the ghost cells.

WRF scaling performance is also influenced by load imbalance. At 256 cores the load is almost perfectly distributed. However, the load imbalance increases with core count until at 8,192 cores the MPI task with the least computational work spends almost 20% longer waiting for messages to arrive than the one with the most work. This is another factor in the plateau effect in the communication and computation time scaling plots.

We found no obvious correlation between the communication performance of WRF and any of the HPCC MPI benchmarks, presumably because of the complications induced by the overlap of computation and communication. Using an equation of the form of Eq. 2. we have found that fits can be obtained to the lowest four data points for the purposes of extrapolation of the compute and communication components. The resulting predictions are reasonably accurate (at worst 15% error).

WRF is also a strong candidate for petascale; the primary scaling issue identified is simply running out of parallel work. By utilizing parallelization over the third dimension, which is already implemented via OpenMP for some systems, even more parallelism can be extracted to ameliorate this issue somewhat. Additionally, tweaks to the code to resolve the load-balancing issues would also increase the scaling efficiency.

6. Discussion and Conclusions

Perfect scalability is not really a practically achievable goal, because algorithmic and machine-specific limitations will always eventually come into play.

What is important when scaling to petascale is to be able to identify the location and causes of scaling bottlenecks. There are many reasons for such bottlenecks. In this work we examined the strong scaling behavior of four benchmark problems on four machines. From looking at Figure 1, which simply shows the performance of each code on each machine, it would be difficult to elucidate the reasons for the scaling behavior of each of the codes on each of the machines. However, using IPM and relatively straight-forward analysis, we were able to obtain insight into *why* the codes scale as they do on each machine. The reasons we determined can be placed into three general classes:

- Superlinear scaling of computation speed. This was exhibited by MILC and is due to the data set moving into cache as the core count increases.
- Sublinear scaling of computation speed. This was exhibited by WRF and is due to algorithmic limitations.
- Sublinear scaling of communication speed. This was exhibited by every code in some form or another and was either due to the machine or some algorithmic limitation within the code.

Perhaps the most important of these is the last one, as was probably to be expected, scalability is *always* limited by communication. In fact increased computational performance is often not of significant benefit to scalability. A machine with fast processors and a relatively poorly performing interconnect is always limited by communication at high concurrencies particularly for strong scaling. Similarly, doubling the performance of the compute portion of the code by tuning may not be as beneficial as doubling the communication performance, if one is aiming for high concurrencies using strong scaling. In a broader sense this is an argument for a balanced compute configuration, one in which the interconnect is able to keep up with the demands placed upon it by the processors. In order to reach the petascale such a configuration is essential. In fact one could imagine obtaining IPM profiles for a representative range of applications within a workload and using that information to procure a system with the correct balance to enable the desired amount of scalability.

The scaling performance of communication is influenced significantly by two factors: the capabilities of the machine and the algorithm the code uses. As these examples show, there are often significant differences between the communication performance of different interconnects. Comparing IPM data between machines allows a user to determine that in fact it is the machine that is the limiting factor, not their code, a piece of information that could save lots of wasted effort.

Algorithmic limitations can come in several forms. Probably the primary cause is collective communication operations, as typified in this study by DNS. By definition these will often show worse scaling than $1/p$, which is the basis for the frequently expressed advice to application developers to avoid collectives as much as possible. Unfortunately, because of the mathematical form of the underlying equations being solved this is not always possible. The other common limitation that we observed for both MILC and WRF is the growth in the total amount of data communicated as concurrency increases.

One frequently used technique to achieve better performance is to use overlapping computation and communication, which has the effect of hiding the time taken to perform the communication task. This and load imbalance complicate the analysis performed here and imply that the bandwidths we report are effective ones that are not necessarily close to those expected theoretically.

In order to reach the concurrencies required at the petascale, tens to hundreds of thousands, it will not be feasible to take the problem sizes of today and strong scale. However, simply increasing a problem size until the desired core count is achieved, using weak scaling, may not address a scientifically interesting problem. In other words, to reach the petascale a *mixture* of strong and weak scaling may be required. Obviously the most effective approach is to strong scale as far as possible, then increase the problem size if the science merits it.

To summarize, we examined the scalability performance of four applications on four machines. Using IPM, we found that scalability was always limited by communication, either because of algorithmic limitations, due to the way in which the code works, or limitations intrinsic to the machine itself.

7. Acknowledgments

We would like to thank Laura Carrington for valuable discussions and Andrew Canning, John Michalakes, and Dmitry Pekurovsky for their advice on interpreting PARATEC, WRF, and DNS output, respectively. This research was supported in part by the National Science Foundation under award #0721397 and through TeraGrid resources provided by TACC and SDSC. Computer time at NERSC was provided by DOE.

8. References

[1] IPM: Integrated Performance Monitoring, http://ipm-hpc.sourceforge.net.

[2] L. Oliker, A. Canning, J. Carter, C. Iancu, M. Lijewski, S. Kamil, J. Shalf, H. Shan, E. Strohmaier, S. Ethier, and T. Goodale, "Scientific Application Performance on Candidate PetaScale Platforms," *International Parallel & Distributed Processing Symposium (IPDPS) 2007.*

[3] S.R. Alam, R.F. Barrett, M.R. Fahey, J.A. Kuehn, J.M. Larkin, R. Sankaran, and P.H. Worley, "Cray XT4: An Early Evaluation for Petascale Scientific Simulation," in *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC07),* Reno, NV, November 10-16, 2007.

[4] S.R. Alam, R.F. Barrett, M.R. Fahey, J.A. Kuehn, O.E. Messer, R.T. Mills, P.C. Roth, J.S. Vetter, and P.H. Worley, "An Evaluation of the ORNL Cray XT3," *International Journal for High Performance Computer Applications*. Feb 2008; vol. 22: pp. 52 - 80.

[5] A. Hoisie, G. Johnson, D.J. Kerbyson, M. Lang, and S. Pakin, "A Performance Comparison through Benchmarking and Modeling of Three Leading Supercomputers: Blue Gene/L, Red Storm, and Purple," in *Proceedings of the IEEE/ACM Conference on Supercomputing (SC'06)*, Tampa, Florida, November 2006.

[6] C. Coarfa, J. Mellor-Crummey, N. Froyd, and Y. Dotsenko, "Scalability analysis of SPMD codes using expectations," in *Proceedings of the 21st Annual international Conference on Supercomputing* (Seattle, Washington, June 17 - 21, 2007). ICS '07. ACM Press, New York, NY, 13-22.

[7] a) V. Taylor, X. Wu, and R. Stevens, "Prophesy: An Infrastructure for Performance Analysis and Modeling of Parallel and Grid Applications," *ACM SIGMETRICS Performance Evaluation Review*, Volume 30, Issue 4, March 2003. b) V. Taylor, X. Wu, J. Geisler, and R. Stevens, "Using Kernel Couplings to Predict Parallel Application Performance," *Proc. of the 11th IEEE International Symposium on High-Performance Distributed Computing (HPDC 2002)*, Edinburgh, Scotland, July 24-26, 2002. c) X. Wu and V.Taylor "Performance Analysis and Modeling of the SciDAC MILC Code on Four Large-scale Clusters" Technical report Texas A&M University.

[8] P.K. Yeung and S.B. Pope, "An algorithm for tracking fluid particles in numerical simulations of homogeneous turbulence," *J. Computational Physics*, v. 79, 373-416 (1988).

[9] P.K. Yeung, D.A. Donzis, and K.R. Sreenivasan, "High-Reynolds-number simulation of turbulent mixing," *Physics of Fluids*, v. 17, 081703 (2005).

[10] The MILC Code (version: 6.20sep02), http://www.physics.utah.edu/~detar/milc/milcv6.html.

[11] a) PARAllel Total Energy Code (PARATEC), http://www.nersc.gov/projects/paratec.b) A. Canning, L.W. Wang, A. Williamson, and A. Zunger, "Parallel Empirical Pseudopotential Electronic Structure Calculations for Million Atom Systems," *J. Computational Physics,* v. 160, 24-41 (2000).

[12] WRF Model Users Site, http://www.mmm.ucar.edu/wrf/users.

[13] D. Pekurovsky, P.K. Yeung, D. Donzis, S. Kumar, W. Pfeiffer, and G. Chukkapalli, "Scalability of a pseudospectral DNS turbulence code with 2D domain decomposition on Power4+/Federation and Blue Gene systems," *Proc. ScicomP12,* Boulder, Colorado (2006), http://www.cisl.ucar.edu/info/scicomp/acceptedabstracts.html#pekurovsky.

[14] W. Pfeiffer and N.J. Wright, "Modeling and Predicting Application Performance on Parallel Computers Using HPC Challenge Benchmarks," 22nd IEEE International Parallel and Distributed Processing Symposium, Hyatt Regency Hotel, Miami, FL, April 14-18, 2008.

[15] HPC Challenge Benchmark, http://icl.cs.utk.edu/hpcc/index.html.

[16] A. Grama, A. Gupta, G. Karypis, and V. Kumar, *Introduction to Parallel Computing (Second Edition),* Addison Wesley (2003).

[17] T. Wettig "Performance of machines for lattice QCD simulations" XXIIIrd International Symposium on Lattice Field Theory 25-30 July 2005 Trinity College, Dublin, Ireland.