Performance Modeling and Comparative Analysis of the MILC Lattice QCD Application su3_rmd

Greg Bauer
National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign
1205 W. Clark Street
Urbana, IL
gbauer@ncsa.illinois.edu

Steven Gottlieb
Physics Department
Indiana University
727 E. Third St.
Bloomington, IN
sg@iub.edu

Torsten Hoefler
National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign
1205 W. Clark Street
Urbana, IL
htor@illinois.edu

***Abstract*—Application performance modeling is an essential part of application and system development as HPC moves into the petascale and prepares for the exascale. However, performance modeling of parallel systems is a difficult task due to natural variations in measurements and noise effects. In this paper, we give a detailed example for a semi-analytical performance-modeling method applied to the ubiquitous HPC application su3_rmd from the lattice Quantum Chromodynamics field on a variety of parallel computing platforms. We apply statistical techniques that are well known in natural sciences to model the variance in the input system. Using a simple analytical model to capture the main characteristics of the code, such as numbers and sizes of passed messages and invocation counts of serial code blocks in conjunction with statistically sound curve-fitting methods, we develop an accurate performance model and use it to characterize application performance on various target architectures. Our fitting techniques allow us to characterize the variance of different performance observations on a given system and show the influence of noise from different sources. The techniques we developed can be applied to a wide class of bulk-synchronous applications. With this detailed example, we aim to motivate the scientific computing community to develop and use similar performance models for software development and maintenance.**

I. Introduction

We consider an application from theoretical high-energy physics as a detailed example for our semi-analytical modeling method in this paper. Quantum Chromodynamics (QCD) is the quantum field theory of Nature's strong force. Although some field theories such as Quantum Electrodynamics (QED) have a weak coupling and can be dealt with using perturbation theory, QCD has a much stronger coupling. Thus, many of the most interesting issues in QCD require a non-perturbative technique. Lattice quantum field theory, invented in the mid-1980s, is the most successful such technique. It is one of the original "Grand Challenge" problems in scientific computing.

The MIMD Lattice Computation (MILC) collaboration [5] has been working in lattice QCD for some twenty years, and has made a suite of codes for lattice QCD freely available. The collaboration uses 10s of millions of service units annually on NSF and DOE computers. The code has been used for hardware diagnostics on the Intel Paragon, for SPEC CPU2006 and SPEC MPI benchmarks, as part of the NERSC and NSF benchmarks, and as one of the applications whose performance needed to be analyzed for the NSF Tier 1 solicitation.

Briefly, the MILC application su3_rmd is used to create sample gauge configurations that are the starting point for many physics projects. We are using the version of the code that implements the R algorithm for improved staggered quarks. Although this is no longer the most efficient algorithm, it is one that has been benchmarked on many computers and the major kernels are very similar, if not identical, to those used by more efficient algorithms such as rational hybrid molecular dynamics algorithm (RHMD) with Monte Carlo [6]. The essential data types in this work are three component complex vectors that represent quarks (matter fields), and 3×3 complex unitary matrices that represent the gluons (force carriers). The quark fields are defined on a 4-dimensional grid of space time points. The gluon variables are defined on the "links" joining grid points. The most time consuming kernel in production runs is the conjugate gradient solver that determines how the motion of the quarks is affected by the gluons. The system is parallelized by decomposing the grid, usually in all four dimensions. Most communications involve point-to-point communication with the neighboring processors in a 4-dimensional grid. However, the conjugate gradient solver also requires global summations.

In general, performance modeling can either be done analytically, that is, based on first principles, or empirically. *Analytic performance modeling* (e.g., [1]) involves counting the number of basic operations (e.g., floating point operations, memory accesses, or integer operations), developing, and parametrizing performance models based on those. *Empirical performance modeling* (or black-box modeling, e.g., [4]) is practiced by observing the performance of a code for a certain combination of input parameters and machine settings, and forming performance expectations for different input parameters and architectures based on those observations.

In this work, we combine analytic and empirical methods and use *semi-analytical* (also called *semi-empirical*) modeling [9], [14] to build our models. This technique models the performance of basic code blocks empirically and composes them analytically to form an application performance model. Similar techniques are routinely used for performance analysis

of large-scale parallel codes (e.g., [13], [16]). We also discuss how we can vary the levels of analytical and empirical modeling in our technique.

The MILC code has long had flop counts for the major kernels and there are flags that can be set at compile time to print the time and flop rate for each of those kernel calls. These performance numbers are very useful for monitoring running jobs and have frequently been helpful to determine system problems. MILC has also used single node benchmarks of the conjugate gradient (CG) phase and microbenchmark results for point-to-point communication to predict whether it will be possible to overlap message passing with computation in the CG. Several studies analyze and compare the performance of MILC on different architectures [17], [18].

However, we go quite a bit further here. We designed a detailed semi-analytical performance model that reflects the composition of the serial kernels to the overall application, provides detailed message counts, and models performance variations. This model enables us to express the performance of the code on different architectures with a small set of parameters. This concise representation of the overall performance can be used to compute the time to solution on a specific architecture, compare architectures, extrapolate to larger process counts or lattice sizes, and much more [9].

Our work shows that the performance of the MILC su3_rmd code can be captured mathematically and concisely by rigorous application of semi-analytical performance modeling principles. This enables a simple baseline performance reporting method that can be used to compare practically relevant performance characteristics across various publications and systems. This small number of parameters, which characterizes the su3_rmd performance on each system, increases the **comparability**, **reproducibility**, and **readability** of microbenchmark results and performance or speedup plots published in scientific papers and technical reports.

Concisely, the contributions of our work are the following: an analytic performance model for MILC su3_rmd, a description of the semi-analytical modeling methodology that enables simple modeling of bulk-synchronous applications, a discussion of the necessary statistical methods to ensure good model fits in the presence of system noise, and finally a concise, model-based performance comparison of MILC su3_rmd on various architectures.

II. The Performance Model

We use a semi-analytical performance modeling technique and apply the six modeling steps as described in [9]:

- **A1** identify input parameters that influence runtime
- **A2** identify application kernels
- **A3** determine communication pattern
- **A4** determine communication/computation overlap
- **E1** determine sequential baseline
- **E2** determine communication parameters

We now discuss in detail how to apply the above steps to the MILC Lattice QCD application su3_rmd version 7.6.3.

A. Step A1: Identify all Critical Parameters

Table I lists relevant input parameters and briefly describes their influence on the runtime. Some variables, such as `nflavors1` or `nflavors2` are fixed by a particular type of calculation and are thus assumed to be part of the algorithm. The best way to gather those parameters is from the documentation or from a domain expert[1].

In our model, we differentiate between two different types of parameters—*simple* and *complex* parameters. Simple parameters have direct arithmetic relations to the runtime. For example, an input parameter, which defines the number of loop iterations, may have a linear influence on the runtime of an application or kernel. Complex parameters also influence the runtime but their influence cannot be expressed with simple arithmetic expressions. For example, an error bound for a conjugate gradient influences the number of iterations and thus the runtime. However, the actual number of iterations depends upon other input parameters such as the quark masses and beta. It thus is often necessary to rely on *experience* of the application scientists in order to predict complex parameters. In order to handle such parameters in MILC, we introduce a new *meta* parameter—`niters` that models the number of iterations. The value of `niters` is based on the experience of domain scientists.

B. Step A2: Identify all Kernels

Finding the performance-critical application kernels is a key component of semi-analytical modeling. The empirically determined runtimes of those kernels serve as the basis for the composed analytical model. We thus start with a serial performance model for each kernel. We then use this model to compose a full semi-analytical serial performance model and then extend it to a parallel performance model.

1) Performance Model for each Kernel: A kernel can either be a function or a code block inside a function. A useful way to identify those blocks is to analyze the call-graph of a representative run (cf. [9]).

Choosing the right level of abstraction for modeling is very important. The modeler needs to decide where to cut subtrees in the call graph, i.e., combine them into a single term and he needs to determine functions to ignore. Our modeling strategy ignores several functions in the callgraph, e.g., `trace_su3`. This can only be done after ensuring that those functions do not consume significant time **and** scale asymptotically much slower (with all input parameters) than other modeled functions. This is the first of a number of necessary trade-offs to balance the complexity of the model with its accuracy.

The MILC code has five performance-critical kernels/functions that account for most of the time: (1) `load_longlinks()` (**LL**), (2) `load_fatlinks()` (**FL**), (3) `ks_congrad()` (**CG**), (4) `imp_gauge_force()` (**GF**), and (5) `eo_fermion_force_twoterms()` (**FF**).

It is well known that today's CPU architectures are too complex to derive performance models of a code from simple

[1]One of the authors of this paper, S.G., is a co-author of MILC.

parameter_name	simple	complex	comment
P	x		number of PEs (intrinsic parameter)
nx, ny, nz, nt	x		size in x, y, z, t dimension
warms, trajecs	x		warmup rounds and trajectories (outer loop)
traj_between_meas	x		measurement "frequency" (called `meas` for brevity)
steps_per_trajectory	x		number of "steps" in each trajectory (`steps` for brevity)
beta, mass1, mass2, error_for_propagator		x	physics parameters that influence convergence of the CG
max_cg_iterations		x	limits the conjugate gradient iterations

TABLE I
PERFORMANCE-CRITICAL MILC INPUT PARAMETERS

operation counts. Thus, we use semi-analytic modeling where we establish an asymptotically tight analytic model and derive the constants with curve fitting. The work required to execute each kernel scales linearly with the number of lattice sites. Let $L_x = \texttt{nx}$, $L_y = \texttt{ny}$, $L_z = \texttt{nz}$, and $L_t = \texttt{nt}$ be the sizes of the dimensions of the lattice on each process and $V = L_x \cdot L_y \cdot L_z \cdot L_t$ and let $\mathcal{B} \in \{LL, FL, CG, GF, FF\}$ identify each kernel. Assuming the linear relation, our model for each kernel is a simple function of the form $T(V) = a + b \cdot V$ where a is the constant overhead and b is the cost per lattice site.

However, if the local volume is small, most or all of the data will reside in the CPU cache, and if the local volume is large, much of the data will need to be fetched from memory, slowing the calculation. The parameters a and b will thus be different depending on V. We thus extend our analytical model to consider two levels of memory hierarchy. We also simplify the model by assuming zero constant overhead ($a = 0$). Lattice sites in the first level are computed at a rate of b_1 per site and sites in the second level are computed with the rate b_2. Our modeled cache can hold $s(\mathcal{B})$ data elements.

$$T(\mathcal{B}, V) = b_1(\mathcal{B}) \cdot min\{s(\mathcal{B}), V\} + b_2(\mathcal{B}) \cdot max\{0, V - s(\mathcal{B})\} \quad (1)$$

Most existing CPU architectures have deeper cache hierarchies, however, we found in our study that a single hierarchy is sufficient to model the performance of MILC accurately (see Section III-B).

C. Combined Serial Analytic Performance Model

We investigated the call graph and source code to determine the number of calls to each kernel as function of the parameters listed in Table I. The analytic model for the total serial computation time is shown in Equation (2):

$$\begin{aligned} T_{ser}(V) \quad = \quad & (\texttt{trajecs} + \texttt{warms}) \cdot \texttt{steps} \cdot [T(FF,V) + T(GF,V) + \\ & 3(T(LL,V) + T(FL,V))] + \\ & \left\lfloor \frac{\texttt{trajecs}}{\texttt{meas}} \right\rfloor [T(LL,V) + T(FL,V)] + \texttt{niters} \cdot T(CG,V) \end{aligned} \quad (2)$$

The variable `niters` represents the total number of conjugate gradient iterations for light and heavy quarks and models the effect of all complex input parameters.

D. Step A3: Determine Communication Pattern

MILC uses point-to-point and collective (allreduce) communication. Collective communication is performed at the end of each conjugate gradient iteration on the whole set of P processes.

1) Point-to-point Pattern: Point-to-point communication is used to implement nearest-neighbor halo exchange which depends on the domain decomposition scheme. The 4-dimensional domain can be decomposed in multiple different ways. MILC's 4-d balanced domain decomposition scheme results in a logical 4-d process grid of $P = p_x \cdot p_y \cdot p_z \cdot p_t$ processes where p_x divides `nx`. In the parallel case, we assume $L_x = \texttt{nx}/p_x$ (similarly for the y, z, and t dimensions).

In the following analysis, we assume for simplicity that the domain is equally decomposed in all four dimensions and $L_x = L_y = L_z = L_t$. Recent work by He [8] et al. presents a study of the influence of the processor grid and sub-volume layout on the communication performance.

The application uses periodic or anti-periodic boundary conditions in each direction so all processes have exactly eight neighbors. Point-to-point messages are sent along the 4-d lattice and are triggered in so-called `gather` calls. Each gather call communicates in one direction and uses blocking communication, however, the conjugate gradient phase enables computation/communication overlap with nonblocking communication. The number of messages (gathers) $n(\mathcal{B})$ is specific to each kernel $\mathcal{B}$. The following table shows the message counts for the case where all dimensions are cut:

$\mathcal{B}$	$n(\mathcal{B})$
FF	$(\texttt{trajecs} + \texttt{warms}) \cdot \texttt{steps} \cdot 1616$
GF	$(\texttt{trajecs} + \texttt{warms}) \cdot \texttt{steps} \cdot 828$
LL	$(3 \cdot \texttt{steps} \cdot (\texttt{trajecs} + \texttt{warms}) + \left\lfloor \frac{\texttt{trajecs}}{\texttt{meas}} \right\rfloor) \cdot 8$
FL	$(3 \cdot \texttt{steps} \cdot (\texttt{trajecs} + \texttt{warms}) + \left\lfloor \frac{\texttt{trajecs}}{\texttt{meas}} \right\rfloor) \cdot 288$
CG	$16 \cdot \texttt{niters} + 16 \left\lfloor 2\texttt{steps} \cdot (\texttt{trajecs} + \texttt{warms}) + 2 \left\lfloor \frac{\texttt{trajecs}}{\texttt{meas}} \right\rfloor \right\rfloor$

FF, GF, LL, and FL perform a fixed number of gathers per invocation. Each CG iteration performs one gather for each of the eight directions of all even (and all odd) sites in a halo-zone of size three. In addition, at the first invocation, it needs one additional gather for each direction of all even (and odd) sites in a halo zone of size one. Each CG invocation sends 16 messages communicating the `su3` vectors one level deep and 16 messages communicating them three levels deep. CG is invoked twice every step (light and heavy quarks) and twice every measurement (which we neglect for simplicity). The measurements phase occurs after every trajectory and is when specific scalar quantities are computed.

a) Point-to-point Sizes: The code uses two major types of point to point operations. The first type is used in FF, GF, LL, and FL and communicates `su3` matrices with 3x3 complex values (18 floating point values) and a 1-element wide halo zone. Thus, a halo zone of one element needs to be communicated at the domain boundaries. As before, L_x, L_y, L_z, and L_t represent the lattice dimensions per process, A_x represents the message size sent along dimension x for the FF, GF, LL, and FL blocks, and f is the size of a single

floating point value:

$$A_x = 18 \cdot f \cdot L_y \cdot L_z \cdot L_t \qquad A_y = 18 \cdot f \cdot L_x \cdot L_z \cdot L_t$$
$$A_z = 18 \cdot f \cdot L_x \cdot L_y \cdot L_t \qquad A_t = 18 \cdot f \cdot L_x \cdot L_y \cdot L_z.$$

Like before, we assume $L_x = L_y = L_z = L_t$ and $V = L_x \cdot L_y \cdot L_z \cdot L_t$, we get $A(V) = 18 \cdot f \cdot \sqrt[4]{V^3}$.

The CG kernel is more complex. It communicates `su3` vectors with 3-element complex valued vectors (six floating point values) for each even (or odd) lattice site in a 3-element wide halo zone in each iteration. This means the message size is $B(V) = \frac{18}{2} \cdot f \cdot \sqrt[4]{V^3} = \frac{A(V)}{2}$. Each CG invocation sends messages communicating the `su3` vectors with a halo-zone of size one (message size $B/3$) and messages with a halo zone of size three (message size B). For the case being modeled the CG kernel is invoked twice every step (once each for light and heavy quarks) and two times every measurement. We will write A (or B) instead of $A(V)$ (or $B(V)$) for brevity.

b) The full Analytic Point-to-point Model: We model the time to transmit a message of size x with $m(x)$. The full communication model for point-to-point operations using the message counts and the messages sizes A, $A/2$, and $A/6$ for the corresponding phases is shown in Equation (3):

$$\begin{aligned} T_{p2p} \quad = \quad & m(A) \cdot \left[3332(\texttt{trajecs} + \texttt{warms}) \cdot \texttt{steps} + 296 \left\lfloor \frac{\texttt{trajecs}}{\texttt{meas}} \right\rfloor\right] + \\ & 16m\left(\frac{A}{2}\right) \cdot \texttt{niters} + 16m(\frac{A}{6}) \cdot \\ & \left[2\texttt{steps} \cdot (\texttt{trajecs} + \texttt{warms}) + 2 \cdot \left\lfloor \frac{\texttt{trajecs}}{\texttt{meas}} \right\rfloor\right] \end{aligned} \tag{3}$$

2) Collective Communication: Only the CG kernel requires an allreduce of two floating point numbers during each iteration. An additional allreduce call is needed for initialization at each first call for heavy or light quarks (once per step and twice for each measurement). Thus, the total number of allreduce calls is:

$$\begin{aligned} n_{red} \quad = \quad & \texttt{niters} + 2 \cdot \\ & \left[\texttt{steps} \cdot (\texttt{trajecs} + \texttt{warms}) + 2 \cdot \left\lfloor \frac{\texttt{trajecs}}{\texttt{meas}} \right\rfloor\right] \end{aligned} \tag{4}$$

We model a small-message allreduce with the typical logarithmic implementation $T_{red}(P) = c + d\log(P)$. Where c is the startup overhead and d is the time per tree-level. The total collective communication time is simply $T_{coll}(P) = n_{red} \cdot T_{red}(P)$.

E. Combined Parallel Analytic Performance Model

In this simplified model, we do not consider the computation/communication overlap in the CG phase explicitly. Thus, Step A4 is omitted here. The total runtime of the application can be expressed as:

$$T_{par}(V, P) \quad = \quad T_{ser}(V) + T_{p2p}(V) + T_{coll}(P) \tag{5}$$

This analytic model has several open parameters, such as b_1, b_2, $s(\mathcal{B})$ for each of the five kernels, $m(x)$ for the message transmission and c and d for the allreduce communication. Without an explicit term for shared resource contention the parameters should be obtained in a manner in which the effect of such resource sharing is in the measured data. In practice the application code is run across a single node or across a few nodes and the compute times are obtained from timed regions. We now show how to parametrize this model for a specific parallel computer.

III. Parametrizing the Performance Model

Our first test system is a 120 node POWER5+ cluster. Each node had 16 1.9 GHz cores running AIX6 and all nodes were connected using an IBM Federation interconnect. The POWER5+ processor has a 32 KB L1 data cache, a 1.9 MB L2 cache and a shared 36 MB L3 cache (victim of L2).

To find the parameters, we generally have two different methods: *analytical*, where we derive the parameters from system parameters such as bandwidth or latency, and *empirical* where we measure different executions and fit the parameters to the analytical model.

Analytical modeling of single-core performance is infeasible as discussed in Section III-B. Thus, we measured the execution time on the target architecture of all kernels with multiple different lattice sizes (V) in order to get b_1, b_2, and s by fitting to Equation (1) as discussed in the following section. We remark here that the relative size of L_x, L_y, L_z, and L_t has only negligible impact on serial performance. Another, more time-consuming possibility to determine those parameters would be to run the kernels on a simulator.

The parameters $m(x)$, c, and d can either be collected with microbenchmarks in combination with analytic network models (e.g., LogGP [2], [10]) or empirically by fitting observed communication performance. We show in detail how a network model can be used in the following POWER5+ example. For the other systems we use the simpler fitting method.

A. Statistical Considerations for Fitting Parameters

While parametrization with empirical methods seems simpler than fully analytical modeling, we point out that great care is needed to guarantee validity of the results. Any measurement of properties of a complex physical system has variations and inaccuracies. It is very important to understand those variations to characterize them in the model. In the following, we describe how to use well-known statistical methods to determine the number of necessary measurements and to provide a the parametrized model that includes a measure of the variation in the measurements.

All parameter fits were generated using a standard non-linear least-squares method with equal weighting[2]. Great effort was given to keeping the number of parameters to a minimum and to use fitting functions motivated from analytic models or physical reasoning. Fitting to a general function (such as a polynomial of an arbitrary degree) was not done as it does not necessarily provide any physical insight to the behavior being modeled.

We report the relative residual of the fit as the average of all relative residuals for all measurement values. The error

[2]We used the `nls` method of the R statistics package version 2.11.1.

of the fit is in practice small and is not described by the relative residual. The measured values reflect *natural* runtime variations on the systems that can be viewed as perturbative processes that move the performance of an application away from the optimal. In practice, it is the time for inter-node communication that exhibits this variation due to traffic on the fabric from other jobs.

B. The Parametrized Kernels

The following table provides the parameters for each critical block $\mathcal{B}$. It is important to notice that each block has different parameters which indicates that the working set and the time per site are different for the modeled operations. As discussed above, the CG phase is dominated by complex valued matrix-vector multiplication while the other phases do primarily complex valued matrix-matrix multiplication. To model the impact of shared resources the kernel data is obtained from runs using a full node or multiple nodes such that T_{ser} is affected by shared CPU and memory resources. The variation in all measurements and the fitted models have been well below 1%.

$\mathcal{B}$	$b_1(\mathcal{B})[\mu s]$	$b_2(\mathcal{B})[\mu s]$	$s(\mathcal{B})$
FF	255	326	2500
GF	88	157	1900
LL	1.3	2.2	2500
FL	30	56	2000
CG	0.425	0.483	1200

Figure 1 shows representative serial performance models for the GF phase (red line) and the actual benchmark results as boxplots (green boxes). We sampled multiple thousands of iterations and the measurement variation was very low for the serial performance measurements.

C. Communication Performance

We used the LogGP benchmark in Netgauge to determine the parameters for the intra- and internode communication performance. The LogGP model [2] is a model from the family of LogP [7] network models. It comprises the four parameters: L (maximum latency between two endpoints in the network), o (overhead per sent message), g (gap, i.e., time between consecutive messages), G (transmission time per byte), and P (number of processes).

The LogGP parameters can be benchmarked with a set of point-to-point measurements and parameters fits [10]. The following table lists the acquired parameter values:

link	range [byte]	L [μs]	o [μs]	g [μs]	G [μs/b]
intra	$0 < S \leq 32768$	2.7	3.5	3.0	0.00068
intra	$S > 32768$	2.7	33.5	3.0	0.00045
inter	$0 < S \leq 32768$	5.8	14	3.3	0.0013
inter	$S > 32768$	5.8	40	3.3	0.0011

We also benchmarked the allreduce performance for different P and fitted our model function $T_{red}(P)$ to $c = 0$ and $d = 3.65\mu s$.

1) Modeling Endpoint Network Congestion: Intra-node communication uses shared memory to exchange messages. Since two cores share one memory controller, we assume a congestion of two for this communication. The intra-node communication time for messages of size x is thus (for $o > g$):

$$m_{intra}(x) = L_{intra} + 2o_{intra} + (x-1) \cdot 2G_{intra}.$$

Each node has 2 network cards and 16 processes are utilizing them. Thus, we assume a congestion of eight for inter-node communication. We assume that the global Clos network is congestion-free for the sparse communication pattern that MILC uses. The inter-node communication time is thus:

$$m_{inter}(x) = L_{inter} + 2o_{inter} + (x-1) \cdot 8G_{inter}.$$

2) Communication Parameters: The 16 cores of a POWER5+ node are provided by 8 dual-core modules (DCM). The grid cells are packed optimally in 4x4 blocks to the available nodes. Thus, for $P \geq 256$ (a 2^4 grid of nodes), each node has neighbors in all four dimensions. In this configuration, exactly half of the communication volume is intra-node and half is inter-node. In addition, inter-node communication (DMA operations) and intra-node communication (memory copies) can overlap most of the communication time. Thus, we approximate a mixed mode communication model for $P > 256$ with

$$m(x) = \max\left\{m_{inter}\left(\frac{x}{2}\right), m_{intra}\left(\frac{x}{2}\right)\right\} = m_{inter}\left(\frac{x}{2}\right).$$

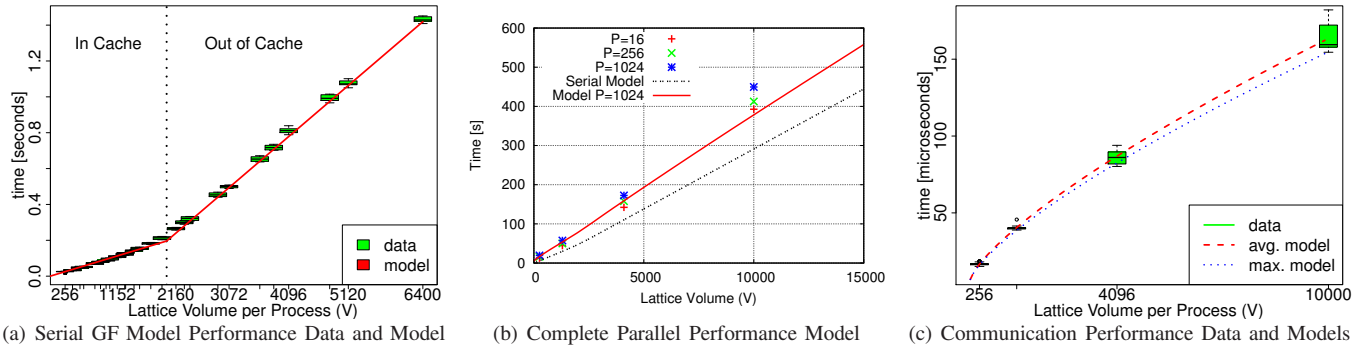
The inter- and intra-node communications happen in parallel as shown in the above equation, and thus the effective bandwidth for messages larger than 32 kiB is $\frac{2}{0.0011\mu s/B \cdot 8}$ MB/s $\approx$ 227 MiB/s.

We will now combine computation and communication into a full parallel performance model.

D. The Parallel Performance Model

Substituting b_1, b_2, s, $m(x)$, and c, d in the parallel model in Equation (5) results in the final parallel model. The model function and actual measurements are shown in Figure 1(b). The figure also includes the serial performance model to visualize the relative cost of computation and communication.

1) Comparing to a Semi-Analytic Network Model: In the previous sections we showed an analytic communication model on the POWER5+ system and have been able to mostly ignore the network congestion due to the full-bisection fat-tree. At this point, we want to remind the reader that the LogGP model does not account for network congestion. The (relatively low) effect of congestion on our POWER5+ system leads to only slightly underestimated runtimes for large process counts in Figure 1(b). We now compare the fully analytic network model with an semi-analytical model that we construct with curve fitting. For this, we will use a simplified network model that only considers the effective latency (cf. L) and the effective bandwidth (cf. 1/G) and sets all other parameters zero (cf. o=g=0). The models in Section III enable us to determine the effective bandwidth bw and latency lat for each application run (with varying lattice sizes) such that $m(x) = lat + \frac{x}{bw}$. Since the latency is known to be $5.8\mu s$, we fixed the latency parameter in the fitting. The following table



(a) Serial GF Model Performance Data and Model (b) Complete Parallel Performance Model (c) Communication Performance Data and Models

Fig. 1. Figures (a) and (c) show performance models and the measured performance as box-plots for the GF phase and the overall communication. Figure (b) shows the overall analytic parallel performance model for POWER5+ which shows the effects of network congestion for large lattice volumes. Figure (c) compares the data to models based on average data or selecting best data.

provides the communication parameters for the 256 and 1024 process cases for the GF phase (all other phases were similar).

P	lat $[\mu s]$	bw [MiB/s]	bw variation
256	5.8	250	5.6%
1024	5.8	209	5.0%

We conclude that the 256 process case is limited by the aggregate bandwidth ($\approx$ 227 MiB/s, see Section III-C2). The effective bandwidth is slightly higher due to communication-induced on-node process skew. At 1024 processes we observe a reduced bandwidth due to network congestion in the fat-tree. The variation in our model is a good indicator for the variance between different measurements.

IV. Performance on Various Architectures

In the following, we demonstrate how the developed performance model can be used to concisely compare the performance of MILC on various architectures. We will present parameters for the semi-analytical serial computation model and the parallel communication model. We will consider multiple architectures with a low bisection bandwidth (low-dimensional Torus topologies) and will thus use empirical methods to fit the data to the communication models presented in Section III.

The effects of varying process counts (contention, interference) are hard to capture analytically because a model would need to consider the application communication topology, the network topology and routing, and the process-to-node mapping. Thus, we provide separate model parameters for each process count P. Finding semi-analytical models with P as open parameter is an interesting topic for future work.

We observed a huge variation in measured network performance among iterations (and identical runs) on some of the investigated production systems (e.g., Cray) while others (e.g., POWER5+ and BG/P) showed very little variation. We attribute this variation to interference with other applications (cf. *network noise* in [12]). However, using the statistical methods explained in Section III-A, we are able to provide the relative variance for each of the parameters in brackets.

We also use our performance models to demonstrate the performance loss due to network congestion and variation. We derive three parametrized models for each process count: (1) an *ideal* network model that assumes an idealized full-bisection bandwidth network where the communication is only limited by the injection parameters (cf. the LogGP model), (2) the best observed performance without application interference, i.e., a *noiseless* network, and (3) the mean (*expected*) performance for the executed runs.

The performance on an ideal network can easily be constructed with analytical modeling of the serial base-case and LogGP parameters as discussed in Section III-C. We provide latency numbers and theoretical per-core bandwidth numbers for all investigated architectures. In order to parametrize the noiseless model, we run a long series of experiments and model the iteration with the lowest observed congestion (i.e., the highest performance). The expected network performance model is derived from all observed measurements (across multiple iterations and runs) with statistical methods (cf. Section III-A).

All reported numbers in this section were run without special tuning and with default parameters on the architectures. We used existing MILC optimizations (e.g., fast compiler flags, SSE optimizations on x86, and derived datatypes) to provide *best effort* performance numbers. However, the focus of this work is on modeling the observed performance on the given systems not on finding the combination of flags and code variations that give the best performance.

A. Performance on BlueGene/P

We measured the performance of the different phases on Surveyor at Argonne National Laboratory. Each compute node contains a quad-core 850MHz PowerPC 450 processor and all jobs were run using all cores (VN mode). The L1 and L2 caches are 32 KB and a 2 KB prefetch buffer, respectively. There is a shared 8 MB L3 cache. The performance measurements of the serial kernels had a very low variation (less than 0.3%), leading to the following serial model parameters:

$\mathcal{B}$	$b_1(\mathcal{B})[\mu s]$	$b_2(\mathcal{B})[\mu s]$	$s(\mathcal{B})$
FF	1410 (0.02%)	1479 (0.05%)	1500
GF	483 (0.09%)	567 (0.20%)	2000
LL	8.33 (0.10%)	9.4 (0.20%)	2500
FL	178 (0.04%)	200 (0.05%)	2500
CG	3.04 (0.06%)	3.05 (0.20%)	4000

The percentages in brackets represent the variance of the measurements compare to the fit. Like on the POWER5+ system, we did not observe significant network noise, thus, the effective noiseless and expected network performance is:

P	lat $[\mu s]$	bw$_{mean}$ [MiB/s]	bw$_{max}$ [MiB/s]
64	19.2 (3.1%)	303 (0.9%)	304
2048	16.4 (1.1%)	205 (0.1%)	206
4096	16.1 (2.0%)	118 (0.2%)	119

The latency and bandwidth values are reported as observed by the application. This means that the effective latency is higher than the observed nearest-neighbor latency ($\approx 9.8\mu s$) because it includes multiple hops in the torus network and congestion effects (depending on the process-to-core mapping). Similarly, the effective bandwidth is, due to network contention, lower than the nearest neighbor bandwidth per core, which is $\left(\approx \frac{375 \text{ MiB/s} \cdot 6 \text{ directions}}{4 \text{ cores}} \approx 562 \text{ MiB/s}\right)$.

B. Performance on Cray XT5

Kraken is a Cray XT5 installed at the National Institute for Computational Sciences. Each compute node contains two 6-core AMD Istanbul 2.6 GHz processors. The L1 and L2 caches are 128 KB and 512 KB, respectively. There is a shared 6 MB L3 cache on each processor. The nodes are networked via the Cray SeaStar2+ chips connected in a $25 \times 16 \times 24$ 3D Torus.

$\mathcal{B}$	$b_1(\mathcal{B})[\mu s]$	$b_2(\mathcal{B})[\mu s]$	$s(\mathcal{B})$
FF	195 (0.4%)	610 (0.2%)	1000
GF	74 (0.4%)	387 (0.3%)	1500
LL	1.96 (1.5%)	7.9 (0.2%)	1500
FL	23 (0.2%)	131 (0.2%)	1500
CG	0.66 (4.7%)	1.37 (0.7%)	1000

The effective network performance is:

P	lat $[\mu s]$	bw$_{mean}$ [MiB/s]	bw$_{max}$ [MiB/s]
1024	12.1 (10%)	167 (7%)	231
2048	10.3 (5%)	211 (3.8%)	264

We measured a nearest-neighbor latency of 8.4 μs and a bandwidth of 1.88 GiB/s. The theoretical maximum injection bandwidth per core is $\approx \frac{3.2 \text{ GiB/s}}{12 \text{ cores}} \approx 266$ MiB/s while the network bandwidth in the Torus is much higher [15]. The effective allreduce parameters (cf. Section III-C) are $a = 238\mu s$ and $b = 51\mu s$. The low performance of allreduce may be due to communication imbalance in the system because the allreduce time, as observed by the application, includes synchronization overheads.

C. Performance on Cray XE6

We measured the performance of the different phases on the Hopper supercomputer. Hopper is a Cray XE6 installed at the National Energy Research Scientific Computing Center (NERSC). Each compute node contains two 12-core AMD 6172 (MagnyCours) 2.1 GHz processors. The L1 and L2 caches are 128 KB and 512 KB, respectively. There are two 6 MB L3 caches per processor. Each cache is shared among six cores. The nodes of Hopper are connected by the Cray Gemini Network and connected in a $17 \times 8 \times 24$ 3D Torus.

$\mathcal{B}$	$b_1(\mathcal{B})[\mu s]$	$b_2(\mathcal{B})[\mu s]$	$s(\mathcal{B})$
FF	232 (0.5%)	483 (0.3%)	1000
GF	81 (0.4%)	261 (0.4%)	1500
LL	1.56 (0.2%)	4.8 (0.1%)	1500
FL	27 (0.5%)	90.7 (0.4%)	1500
CG	0.57 (0.7%)	1.07 (0.2%)	600

The effective network performance is:

P	lat $[\mu s]$	bw$_{mean}$ [MiB/s]	bw$_{max}$ [MiB/s]
1024	13.8 (7.0%)	595 (2.0%)	718
2048	6.6 (7.2%)	459 (0.8%)	486
8192	41.5 (4.8%)	232 (1.7%)	253

We measured a latency of 2.08 μs and a bandwidth of 2.5 GiB/s for the slowest link between two neighboring nodes (not sharing the same Gemini). The theoretical available injection bandwidth per core is $\approx \frac{7 \text{ GiB/s}}{24 \text{ cores}} \approx 292$ MiB/s. See Alverson, Roweth, and Kaplan [3] for details on the Cray Gemini interconnect. The observed bandwidth for small P is higher than the per-core bandwidth because parts of the communication are done through shared memory in the large SMPs (cf. Section III-C2). The effective allreduce parameters (cf. Section III-C) are $a = 21.5\mu s$ and $b = 8.5\mu s$.

D. Performance on an InfiniBand Cluster

The ds1 cluster at Fermilab was designed for lattice QCD applications. Each node contains four 8-core 2.0 GHz AMD Opteron 6128 processors. The L1 and L2 caches are 128 KB and 512 KB, respectively. There is a shared 12 MB L3 cache. The nodes are connected via a QDR InfiniBand network. The leaf switches have 36 ports and there is a 3:1 oversubscription between the leaves and the spine switch.

$\mathcal{B}$	$b_1(\mathcal{B})[\mu s]$	$b_2(\mathcal{B})[\mu s]$	$s(\mathcal{B})$
FF	247 (0.3%)	435 (0.1%)	2000
GF	83 (0.2%)	225 (0.2%)	2000
LL	1.6 (0.8%)	4.3 (0.6%)	2000
FL	27 (0.2%)	79 (0.2%)	2000
CG	0.71 (2.1%)	0.97 (0.6%)	1000

The effective network performance is:

P	lat $[\mu s]$	bw$_{mean}$ [MiB/s]	bw$_{max}$ [MiB/s]
1024	26.7 (10%)	269 (2.7%)	315
2048	33.6 (16%)	164 (3.0%)	209

We measured a latency of 1.5 μs and a bandwidth of 2.46 GiB/s. The theoretical available injection bandwidth per core is thus ≈ 77 MiB/s. Again, parts of the communications are done through shared memory, leading to the high effective bandwidth. The network performance on InfiniBand showed a high variation between runs and iterations and our statistical model captures the variation in the latency and bandwidth parameters.

V. Summary and Discussion

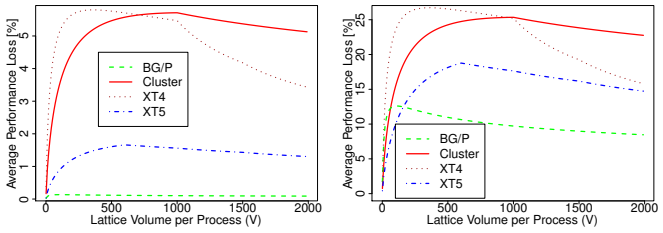
In this work, we presented a semi-analytical model for the MILC su3_rmd code. We showed how performance of a full application can be combined from the serial performance of application kernels and communication overhead by combining analytic methods (counting message numbers, sizes, and kernel invocations based on the code) and empirical methods (parameter fitting actual performance to serial and parallel performance models).

We show that the performance of MILC su3_rmd can be expressed with a small set of parameters that enable prediction of execution times, observation of performance bottlenecks, and effective comparison of architectures (also across different publications). We show that our model can be applied to multiple different architectures with relatively low error rates.

The performance models can be used for several tasks and can be extended in multiple directions. We now discuss possible use-cases of our parametrized performance model. It is now clear that one can easily compare the in-cache serial performance (b_1) that is mainly influenced by the on-chip design and the out-of-cache serial performance (b_2) that is mainly influenced by the memory subsystem. One interesting observation is that the older AMD Istanbul CPUs in the XT5 perform better than the newer MagnyCours in XE6 due to the higher CPU frequency for small lattice volumes. However, the slower memory subsystem on XT5 shows lower performance than XE6 for larger lattice volumes. We also see that a BlueGene core is about a factor of 4–7 slower than an x86 core, however, for out-of-cache computations, the difference is only a factor of 2–3. The developed serial models can be used to design a system with the right relation between CPU and memory performance for MILC.

The network is important for large-scale computations. Our semi-analytical network models show the ideal, the noise-less, and the expected performance of the networks. The effective bandwidth depends on the network topology, the routing strategy, the application communication topology, and the process-to-node mapping. This complex relation makes it nearly impossible to derive analytic models for the effective bandwidth of realistic applications running on a nontrivial network topology and routing. For example, we show that congestion in the 3D torus topologies reduces the effective bandwidth significantly (up to a factor of four for large runs). We also model how network noise (background traffic) affects the effective bandwidth and show a nearly 50% degradation due on the InfiniBand system. We conjecture that this is due to the static routing in InfiniBand [11].

This enables us to quantify the expected loss in application performance due to network noise for different lattice sizes (the difference between the best seen and the statistically expected performance). Figure 2(a) plots the performance loss for the whole application run. Small lattice sizes are less effected because the relative variance for small messages is lower (cf. Figure 1(c)) and very large lattice sizes are dominated by computation. BlueGene/P has no significant performance loss due to network noise while all other systems are affected.



(a) Network Bandwidth Variation (b) Network Congestion

Fig. 2. Lost Application Performance due to network parameters for `steps=meas=1` and `niters=2000`.

We can also investigate the performance loss due to the network topology and routing, compared to an ideal congestion-less network (full effective bisection bandwidth). Figure 2(b) shows the relative loss for a full application run showing up to 25% lower performance.

The two examples show overheads of the certain properties of the network that depend on the communication/computation ratio and other application and system parameters. A performance model helps to design a computer system for the application requirements (e.g., optimize the money-bandwidth trade-off) and also offers multiple other avenues for analyzing the effect of optimizations and architectural parameters, such as communication optimizations.

Acknowledgments: This work was supported by NCSA's Blue Waters project (NSF OCI 07-25070 and the State of Illinois); and DOE grants FG02-91ER 40661 and DE-FC02-06ER41443. One of us (S.G.) gratefully acknowledges sabbatical support from NCSA. We thank the ALCF (Surveyor), NICS (Kraken), NERSC (Franklin and Hopper), and the LQCD Computing Project at FNAL (ds1) for access to computers used for benchmarking.

REFERENCES

[1] A. Agarwal, J. Hennessy, and M. Horowitz. An analytical cache model. *ACM Trans. Comput. Syst.*, 7:184–215, May 1989.

[2] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman. LogGP: Incorporating Long Messages into the LogP Model. *Journal of Parallel and Distributed Computing*, 44(1):71–79, 1995.

[3] R. Alverson, D. Roweth, and L. Kaplan. The gemini system interconnect. In *High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on*, pages 83 –87, aug. 2010.

[4] B. J. Barnes, B. Rountree, D. K. Lowenthal, J. Reeves, B. de Supinski, and M. Schulz. A regression-based approach to scalability prediction. In *International conference on Supercomputing*, ICS '08, pages 368–377, New York, NY, USA, 2008. ACM.

[5] C. Bernard et al. Studying Quarks and Gluons On Mimd Parallel Computers. *International Journal of High Performance Computing Applications*, 5(4):61–70, 1991.

[6] M. A. Clark and A. D. Kennedy. Accelerating fermionic molecular dynamics. *NUCL.PHYS.PROC.*, 140:838, 2005.

[7] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: towards a realistic model of parallel computation. In *Princ. Pract. of Par. Progr.*, pages 1–12, 1993.

[8] J. He et al. Layout-Aware Scientific Computing: A Case Study with MILC. In *Scala Workshop, SC'11*, 2012. To appear.

[9] T. Hoefler, W. Gropp, W. Kramer, and M. Snir. Performance modeling for systematic performance tuning. In *State of the Practice Reports*, SC '11, pages 6:1–6:12, New York, NY, USA, 2011. ACM.

[10] T. Hoefler, A. Lichei, and W. Rehm. Low-Overhead LogGP Parameter Assessment for Modern Interconnection Networks. In *Proc. of the 21st IEEE Intl. Par. & Distr. Proc. Symp.* IEEE Comp. Soc., March 2007.

[11] T. Hoefler, T. Schneider, and A. Lumsdaine. Multistage Switches are not Crossbars: Effects of Static Routing in High-Performance Networks. In *Proc. of the 2008 IEEE Intl. Conf. on Cluster Comp.*, Oct. 2008.

[12] T. Hoefler, T. Schneider, and A. Lumsdaine. The Impact of Network Noise at Large-Scale Communication Performance. In *23rd IEEE Int. Par. & Distr. Proc. Symp. (IPDPS), LSPP Workshop*, May 2009.

[13] D. J. Kerbyson et al. Predictive performance and scalability modeling of a large-scale application. In *SC'01: 2001 ACM/IEEE Conf. on Supercomputing*, pages 37–37, New York, NY, USA, 2001. ACM.

[14] M. M. Mathis, N. M. Amato, and M. L. Adams. A general performance model for parallel sweeps on orthogonal grids for particle transport calculations. In *ICS*, pages 255–263, 2000.

[15] K. T. Pedretti, C. Vaughan, K. S. Hemmert, and B. Barrett. Application sensitivity to link and injection bandwidth on a cray xt4 system. In *Proc. of the 2008 Cray Users' Group Annual Conference*, 2008.

[16] S. Pllana, S. Benkner, F. Xhafa, and L. Barolli. Hybrid performance modeling and prediction of large-scale computing systems. In *Proc. of CISIS 2008*, pages 132–138, 2008.

[17] N. J. Wright, W. Pfeiffer, and A. Snavely. Characterizing Parallel Scaling of Scientific Applications using IPM. In *proc. of the 10th LCI Conference*, Mar. 2009.

[18] W. Xingfu and V. Taylor. Performance analysis and modeling of the scidac milc code on four large-scale clusters. Technical report, College Station, TX, USA, 2007.