# Analyses and Modeling of Applications Used to Demonstrate Sustained Petascale Performance on Blue Waters

Gregory Bauer, Torsten Hoefler, William Kramer
National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign

Robert Fiedler
Cray Inc.

*Abstract*—**The sustained petascale performance of the Blue Waters system will be demonstrated using a suite of several applications representing a wide variety of disciplines important to the scientific community of the US National Science Foundation. The geometric mean of the measured floating point rates for these applications running scientific problems of current interest at scale is used to compute the sustained petascale performance (SPP), which is a key acceptance metric for the system. In this report, we discuss the performance of these applications on Cray XE hardware. Our elemental modeling methodology splits each of the codes into a small set of performance-relevant kernels (typically around 5-7). We analyze those kernels in detail on the AMD Interlagos architecture to determine the achieved memory bandwidths and latencies, communication bandwidths and latencies, and floating point rates. This allows us to conclude, in a form similar to the Roofline model, how well each kernel performs with regards to each of the parameters. For example, if a kernel is mediocre in peak performance but utilizes 100% of the memory or network bandwidth, we can conclude that the kernel is utilizing the architecture well and is memory- (or communication-) bound. While a low floating point rate and a low bandwidth utilization suggests optimization opportunities. Our analyses should also provide us with insight into application performance on future systems.**

## I. Introduction

Assessing productivity and performance of large-scale supercomputers is a difficult task. Microbenchmarks are often used as a proxy to evaluate how a computing system performs. One problem with this approach is that application performance is a very complex problem and microbenchmarks may overlook important features of the system. For example, operating system noise was only recently discovered [1] and an analysis with microbenchmarks is very challenging [2].

Those problems can be avoided by using application benchmarks to evaluate system performance. One such benchmark is the solution of a large dense system of equations using gaussian elimination. For decades, the High Performance Linpack (HPL) benchmark has been used to evaluate and compare supercomputer performance. However, HPL has a very specific computational structure and requirements that make it a memory-bandwidth limited benchmark. However, deficiencies in the memory- or communication network can be hidden with growing matrix sizes because HPL supports cache blocking and requires $\Theta(N^3)$ floating point operations for $\Theta(N^2)$ data movement operations. It follows that the HPL performance strongly correlates with the system's peak performance. Thus, in general, simple application benchmarks such as HPL, or even HPCC, often do not represent the varying workload of an HPC system well.

The Blue Waters system is an Cray XE machine of unprecedented scale and performance. It combines the computational power of hundreds of thousands of AMD Interlagos cores and thousands of GPUs to exceed a peak performance of 10 petaflops. Most importantly, Blue Waters strives to deliver sustained petascale performance for a variety of applications from the National Science Foundation (NSF). It is explicitly not a goal of the system to achieve highest possible HPL or other microbenchmark performance. Instead, the Blue Waters design strives to deliver the best balance between CPU, memory, network, accelerator, and I/O performance as supported by the Cray XE and XK series.

The heterogeneous design of Blue Waters requires an explicit analysis of how the XE (x86) and XK (x86+GPU) parts contribute to the sustained petascale performance of the overall system. It is also necessary to find the right application mix to accurately evaluate this heterogeneous system. To demonstrate the capabilities and sustained performance of Blue Waters, the project team chose a set of twelve application benchmarks, eight to demonstrate x86 performance and four to demonstrate GPU performance. Each of these benchmarks represents a real science run from beginning to end, and thus represents typical use of the system. Those benchmarks form the Sustained Petascale Performance (SPP) benchmark suite that represents a typical NSF workload.

In the following sections, we will describe the SPP benchmark in detail. Then we discuss our performance modeling strategy to analyze and predict system performance, followed by a detailed discussion of all applications in the SPP suite and their performance characteristics. Due to the nature of the delivery schedule we have focused our efforts on the SPP x86 applications for this work.

## II. The Sustained Petascale Performance Benchmark

The Sustained Petascale Performance (SPP) benchmark stems from the Sustained System Performance (SSP) metric defined by Kramer as part of the PERCU project [3]. According to this definition, a benchmark should serve four purposes:

1) It enables one to compare two different computer systems.
2) It enables one to verify system performance and numerical solution correctness.
3) It enables one to monitor performance through the lifecycle of the system (regression testing).
4) It helps to guide the design of future systems.

To represent the average workload of the Blue Waters system, we chose a suite of applications from a variety of disciplines important to the NSF: Lattice Quantum Chromodynamics (MILC, Chroma), Materials Science (QMCPACK), Molecular Dynamics (NAMD), Geophysical Science (VPIC and SPECFEM3D), Atmospheric Science (WRF), Astrophysics (PPM), and Computational Chemistry (NWCHEM, GAMESS).

We refer to the combination of code and computational input problems as a ”benchmark configuration”. Eight benchmark configurations represent the performance of the x86 part of the SPP metric solving one input problem for each of the eight applications: MILC, QMCPACK, NAMD, VPIC, SPECFEM3D, WRF, PPM, and NWCHEM. Four configurations represent the GPU part solving one input problem for each of the four applications Chroma, NAMD, QMCPACK, and GAMESS.

### A. Reference FLOP Count

We define the unit for couting the floating point operations as FLOPS (plural of FLOP) and the according rate as FLOPS per second (FLOPS/s). The performance of the $i$-th benchmark configuration is derived from the total number of FLOPS required to solve the complete problem, the total number of processing elements used, and the total time. The required number of FLOPS for this specific problem, also called reference FLOPS is the minimal number of floating point operations required to solve the input problem on a single core. This avoids an artificial inflation of the FLOP count due to scaling (an inflated number of iterations or redundant computations).

If running the full problem on a single core is impossible or impractical, one can either run the full problem on multiple different core counts and show that the total number of FLOPS required is independent of the number of cores used, or one can extrapolate the FLOPS required to execute a large number of iterations (for an iterative application) from a smaller number of iterations. To demonstrate the former, one must run the full application on at least five different core counts such that the minimum and maximum differs by at least three orders of magnitude. For the latter method, an accurate model for the required number of iterations and FLOP counts per iteration must exist and the pre- and postprocessing of the algorithm must be taken into account separately.

### B. The SPP Metric

To define the SPP of the Blue Waters system, we use the following definitions:

- node type $\alpha$: the system consists of two types of nodes $\alpha \subset \{XE, XK\}$

- node counts $N$: the system has a specific number of nodes $N_\alpha$ of each type $\alpha$.
- benchmark configurations $C$: an application and an input set form a benchmark configuration. The ordered set $C_{XE}$ of eight configurations represents all XE benchmarks and the ordered set $C_{XK}$ of four configurations represents all XK benchmarks.
- per-node performance $P$: a configuration $i$ has a specific performance, $P^i_\alpha$, on each node of type $\alpha$.
- SPP contribution of configuration $i$: the SPP contribution of a configuration $i$ is the per-node performance $P^i_\alpha$ multiplied by the number of nodes $N_\alpha$.

The SPP performance of a homogeneous (single node type) system is defined as the gemoetric mean of the SPP contributions of all applications. The SPP performance of a heterogeneous system is defined as the sum of the SPP contributions for all node types. Each benchmark must be run on at least 20% of the total number of available nodes $N_\alpha$ of each type to measure $P^i_\alpha$, since this represents the anticipated system usage more closely than full-system jobs.

The total SPP performance for the Blue Waters system can be computed with

$$\sqrt[8]{\prod_{i=1}^{8} P^i_{XE}} \cdot N_{XE} + \sqrt[4]{\prod_{i=1}^{4} P^i_{XK}} \cdot N_{XK}$$

This number defines the sustained performance of the full system for a representative workload.

## III. Performance Modeling

Performance modeling can provide important insights into application performance. It can be used to predict performance at larger scale, or to understand scaling with different parameters (e.g., changing input problems and sizes and/or changing execution resources). In the SPP context, we use performance modeling during system bringup to evaluate how measured performance of the partial system compares to the expected performance of the full system. This makes performance modeling a valuable tool during bringup and enables us to check observed application performance against semi-analytic predictions.

We use our six-step method that we described in [4]. The six steps are divided into empirical and analytical steps. The first four steps are analytical, i.e., they are determined from the source code or specified by a domain expert:

1) Identify input parameters that influence runtime
2) Identify application kernels
3) Determine communication pattern
4) Determine communication / computation overlap

The last two steps are empirical, i.e., they require performing a series of benchmarks to substantiate the empirical or analytical performance models:

1) Determine sequential baseline
2) Determine communication parameters

The performance team at NCSA executed these steps to create very simple performance models for each of the SPP applications. Since single-core performance is essential to reach a high SPP performance, we also try to assess the resource consumption for each application by counting various events occurring on the Interlagos cores.

### *A. LibPGT - A simple performance modeling tool*

To instrument each kernel of the source code for performance modeling, we developed a very lightweight profiling library, LibPGT. LibPGT offers only a very simple interface to start and to end collection epochs and record execution time for up to two PAPI performance counters. An epoch covers a part of the execution time and cannot be nested.

The library collects a trace of the times and counters for each MPI process and invocation. We used the statistics package GNU R to interpret and display the output.

## IV. SPP Applications

The SPP Applications comprise a set of eight codes used by Blue Waters science teams running science problems of current interest. For each science area the applications utilize methods and algorithms common to other applications in that area, and differ mainly in the implementation details, so that each code is an appropriate representation of the expected science work in that area.

### *A. NAMD*

NAMD is an application for performing classical molecular dynamics simulations of biomolecules that is able to scale to 100 million atoms on hundreds of thousands of processors [5]. Interactions between atoms in the simulation are parameterized based on the species of each atom and its chemical role. The forces on all atoms are integrated by the explicit, reversible, and symplectic Verlet algorithm to simulate the dynamic evolution of the system with a timestep of 1 fs.

The force field includes bond forces among groups of 2-4 atoms, Lennard-Jones forces, and short-range and long-range electrostatics. All of these forces except the long-range electrostatics are localized and scale well on large distributed computers. The long-range electrostatics are computed using the FFT-based particle-mesh Ewald (PME) algorithm, which requires computing two 3-D FFTs every time step. Since NAMD is designed to overlap various force computations, with increasing numbers of compute nodes, the local computation time decreases to the point where the all-to-all communication for the FFT stage dominates execution time.

Reference FLOP counts for NAMD are determined by instrumenting the code using PAPI [6] to read hardware FLOP counters between global barriers inserted for benchmarking purposes. The benchmark simulation is run on as few processors as allowed by memory constraints to minimize redundant operations introduced by parallel execution. The same procedure may be followed on systems other than the target system to verify that similar operation counts are obtained regardless of architecture and compiler used. The counter data is compared to execution timings from the same simulation run on the intended number of cores with counters turned off to compute the floating-point performance.

### *B. MILC*

The MIMD Lattice Computation (MILC) collaboration [7] 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 US 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 (Blue Waters).

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 [8]. The essential data types in this work are three component complex vectors that represent quarks (matter fields), and $3 \times 3$ complex unitary matrices that represent 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 of the five typical kernels in production runs is the conjugate gradient (CG) solver that determines how the motion of the quarks is affected by the gluons. Computationally most of the phases (called FL, LL, FF and GF) are dominated by small, complex 3x3 matrix-matrix multiplication. The CG phase is composed of multiplication of complex 3x3 matrices with complex 3x1 vectors. The data for both cases are spread sparsely on the local lattice. As 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.

The MILC code has long-established 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 in identifying system problems. MILC has also used single node benchmarks of the 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 phase. Several studies analyze and compare the performance of MILC on different architectures [9], [10], [11]. For su3_rmd the amount of work is determined by several input parameters with the fundamental unit of work being a *step*. The flop counts per *step* depend on the lattice dimension and the number of CG iterations per step which depend on lattice size, physical parameters and convergence tolerances.

## C. PPM

PPM is used to study flash events in early generation stars which result in expulsion into the early interstellar medium of heavy elements formed in the so-called s-process. These flash events are not explosive, in that the star does not disintegrate as a result, but they are indeed violent. Simulations of these events with enormous grids, up to $4096^3$ cells, are used to compute in very accurate detail the global convection above the helium burning shell. These simulations provide values of coefficients needed in simplified statistical models of the turbulent convection, entrainment of gas from above the convection zone, and, at later times during the flash event, turbulent combustion of this entrained gas in the convection zone. These turbulence models are simulated as subgrid-scale models on grids up to $1024^3$.

The dominant phase in PPM is the advection/ppmmf routine. Within this routine the dominant loops involve data movement. The code employs methods for cache line flushing, volatile variables, and data prefetching to improve L1 data cache utilization. Empirical comparisons to trivial benchmarks (e.g. OpenMP parallelized matrix multiplication) show that artificially inflating the computational intensity relative to the amount of memory references results in obtaining estimates of theoretical upper bounds on FLOP rate performance. A recent performance analysis by the developers [12] provides the compute structure in detail.

CrayPAT and PGT library were used to gather hardware counter data and FLOP rates. These numbers were found to differ slightly from the hand-counted rates calculated in the code; most likely due to compiler optimizations that eliminate certain operations. Due to the shear size of the routine and number of times it is called in a typical small modest benchmark, we had to be skillful at gathering fine-grained data, in order to avoid introducitng significant overhead from the instrumentation.

## D. QMCPACK

QMCPACK uses quantum monte carlo methods (VMD and DMC) to solve the quantum many-body problem for particular atomic structure inputs. A number of random walkers (each a complete representation of the input state that randomly ”walks” around) is used to stochastically sample the energy domain in an attempt to find the lowest energy state within a statistical uncertainty that can be used to describe the system. Longer simulations can be run to reduce the uncertainty and refine the results.

A dominant kernel in QMCPACK is the einspline routine that involves 64 basis elements using piecewise cubic polynomials, matrix-vector products with 2 FLOPS per load and a Value-Gradient-Hessian at 20 FLOPS per load. The particle-by-particle update kernels are mostly BLAS-like operations using SIMD. QMCPACK implements a hybrid MPI+OpenMP programming model with communication being primarily point-to-point with some global allreduces. The frequency of communication is such that nearly ideal scaling is achieved on up to 200K cores [13]. The FLOP counts have been obtained empirically using CrayPat and libPGT.

## E. WRF

The Weather Research and Forecasting (WRF) Model [14] is a next-generation mesoscale numerical weather prediction system designed to serve both operational forecasting and atmospheric research needs. It features multiple dynamical cores, a 3-dimensional variational (3DVAR) data assimilation system, and a software architecture allowing for computational parallelism and system extensibility. WRF is suitable for a broad spectrum of applications across scales ranging from meters to thousands of kilometers.

The WRF-ARW core [15] is based on an Eulerian solver for the fully compressible nonhydrostatic equations, cast in flux (conservative) form, using a mass (hydrostatic pressure) vertical coordinate. Prognostic variables for this solver are column mass of dry air (mu), velocities u, v and w (vertical velocity), potential temperature, and geopotential. Non-conserved variables (e.g. temperature, pressure, density) are diagnosed from the conserved prognostic variables. The solver uses a third-order Runge-Kutta time-integration scheme coupled with a split-explicit 2nd-order time integration scheme for the acoustic and gravity-wave modes.

The main solver routine contains C preprocessing hooks for benchmark timers. This native functionality was expanded (and some erroneous logic corrected) in order to fully capture all relevant timing statistics. After this was tested, PGT library epoch markers were implemented to mirror the native microsecond timers. These were run and their agreement validated. This enabled FLOP rates to be calculated for each critical section, for any desired MPI rank. CrayPAT was earlier used to gather basic profiling and detailed FLOP counts, which were found to be in agreement with the finer grained instrumentation. The PGT markers were also used at loop level granularity internal to several (WSM5, WSM6, Morrison) microphysics schemes to compare the relative performance of these kernels.

## F. SPECFEM3D

The SPECFEM3D series of codes simulate the seismic wave propogation in the earth, representing the globe using a finite element mesh. While some of the codes model a region of the earth in isolation, SPECFEM3D_GLOBE is used to model propogation of waves from earthquakes through the entire earth, and is specifically designed to scale to systems with hundreds of thousands of processor threads. The code inputs seismagraphs and inverts them to synthesise what the displacement was at the epicenter of the earthquake. This is used, among other things, to model the displacement of a close-by earthquake to understand how buildings need to be constructed to withstand earthquakes. The SPECFEM3D_GLOBE application was a Gordon-Bell Finalist for the 2008 ACM/IEEE conference on Supercomputing [16].

We have run SPECFEM3D_GLOBE on the early Blue Waters hardware. SPECFEM3D_GLOBE has been run on up to 4107 XE nodes with over 130,000 processor threads. We have obtained counter data with both NCSA's PerfSuite [17] and CrayPAT.

The two primary phases of the code compute internal forces and related acceleration vectors [16] in the crust and mantle for each spectral element of the mesh: compute_element_tiso and compute_element_iso, called These two routines have a very similar structure, with small matrix-matrix products being the dominant operations.

## G. NWCHEM

The purpose of electronic structure calculations is to predict physical properties of chemical systems by solving fundamental quantum mechanical Schroedinger equation when experimental data are not available or difficult to obtain, and when highly reliable theoretical predictions are of paramount importance.

The SPP benchmark in NWChem uses coupled cluster theory for numerical accounting of electron correlation effects in molecular simulations. As a real-science application the SSP tests will be executed on DNA fragments and water clusters.

DNA is the carrier of genetic information of all living organisms. The DNA structure incorporates replication, genetic data storage and retrieval, mutation and repair mechanisms, structure variability, strength, and flexibility, among the long list of other unique properties specific to this molecule. Unraveling the mechanism of functioning of DNA is amongst the top pharmacological and bioengineering priorities. Gaining atomic-level understanding of structure and function of DNA requires the availability of accurate electronic structure methods such as coupled cluster, which is implemented in the NWChem package.

Water clusters are another example where the progress in understanding physics of water in its different forms is impossible without the access to highly accurate electronic structure calculations. Such simulations are necessary in order to explain anomalous properties of water and to calibrate approximate computational methods, which are of great importance for biophysics, atmospheric and environmental sciences. Coupled cluster methods are the only theoretical tools that can reliably predict binding energy in water clusters.

For the coupled-cluster calucations in NWChem with the inputs discussed above, the main computational kernels in the coupled cluster method are 1) self-consistent field (SCF) 2) integral transformation from atomic orbital (AO) to molecular orbital (MO) basis 3) single excitations 4) double excitations 5) triple excitations

These kernels were profiled by calling built-in performance measurement subroutines in NWChem using the PAPI_flips() function to measure the number of FLOPS. CrayPAT is unable to perform sampling experiments due to the very complicated internal structure of NWChem. However, CrayPAT tracing results fully agree with the data obtained from manually coded performance measurements.

The main computational cost in the coupled cluster method comes from the huge number of matrix multiplication operations performed in the code (kernels 3-5). The matrix multiplications are implemented via calls to the BLAS DGEMM routine. The amount of floating point operations are proportional to $N^4$, $N^5$, $N^6$, and $N^7$ for kernels 1, 2, 3+4, and 5, respectively, where $N$ is number of basis functions. The kernels are computed one after another. Kernels 1, 3 and 4 are iterative, which leads to frequent and expensive gather operations. Kernel 2 needs all AO integrals on every compute node, requiring substantial communication. Kernels 1, 2, and 3 have relatively little work to do in comparison to the amount of work for kernels 4 and 5, leading to load imbalance. The computational cost of kernel 5 increases as $N^2$ relative to the cost of kernel 4. Thus as the systems become larger, the load imbalance between kernels 4 and 5 increases.

The theoretical model for the amount of FLOPS for kernels 3+4 only is given in Stanton et al[18] as $0.5n^4N^2 + 0.25n^2N^4 + 4n^3N^3 + nN^4 + 6n^2N^3 + 10n^3N^2 + n^4N$ where $n$ is number of occupied molecular orbitals and $N$ is number of virtual molecular orbitals. The actual implementation of the coupled clusted method in NWChem is described in R. Kobayashi et al [19], so the above formula can be validated and corrected, if necessary.

### H. VPIC

We are currently evaluating the VPIC particle-in-cell plasma physics application as a member of the SPP application suite. It is a well-performing application [20] that has been run at many HPC centers.

TABLE I
OBSERVED PERFORMANCE COUNTERS

| Meaning | Counter |
|---|---|
| Floating point instructions completed ($f$) | PAPI_FP_OPS |
| Total instructions completed ($i$) | PAPI_TOT_INS |
| Memory accesses (loaded cache lines, ($c$) | perf::PERF_COUNT_HW_ CACHE_LL:MISS+ DATA_PREFETCHER:ALL |
| Total loads/stores ($ls$) | PAPI_L1_DCA |
| Stall cycles ($st$) | INSTRUCTION_FETCH_STALL |

## V. APPLICATION CHARACTERIZATION

We will now show some representative examples for our application classification and the identification of kernels. In the early stages of this project, we focus on the single-core performance of each kernel and how effectively it uses the available system. The first step is to identify the kernels based on hardware counter data measuring the Instructions Per Cycle (IPC) while the code executes.

Our method of choice for this is to generate *IPC traces* and identify regions of similar behavior. We use LibPGT to generate such traces. LibPGT simply registers an alarm to be woken up at a given interval and collects the number of retired instructions and the number of cycles for this interval. It then records the IPC at the end of each interval and optionally also records a stack trace to identify the functions involved in a certain kernel.

For each kernel, we derive performance characteristics, i.e., how well each kernel is using the hardware subsystems. To do this, we collect several PAPI present event and native event counters for each phase (see Table I).

With another parameter time ($t$), we can compute the following performance characteristics for each kernel from those counters: (1) instruction rate (MIPS): $\frac{i}{t}$, (2) floating point rate $\frac{f}{t}$ (MFLOPS/s), (3) memory bandwidth $\frac{c \cdot cl\_size}{t}$ (MiBPS), (4) computational intensity (CI) $\frac{f}{ls}$, (5) arithmetic intensity (AI) $\frac{f}{c \cdot cl\_size}$. Additional derived parameters are instructions per cycle (IPC) and the effective clock frequency in GHz (effGHz).

It is useful to compare the application kernel performance characteristics to those of well-known kernels. For the simple kernels the derived param-

TABLE II
PERFORMANCE CHARACTERISTICS FOR SIMPLE KERNELS

| kernel | MIPS | MFLOPS/s | MiBPS | CI | AI | IPC | effGHz |
|---|---|---|---|---|---|---|---|
| triad s | 300 | 407 | 3958 | 1.1 | 0.1 | 0.1 | 2.3 |
| triad l | 241 | 156 | 1574 | 1.0 | 0.1 | 0.1 | 2.6 |
| stencil s | 1089 | 2508 | 9172 | 1.4 | 0.3 | 0.5 | 2.3 |
| stencil l | 181 | 458 | 1684 | 1.4 | 0.3 | 0.1 | 2.6 |
| dgemm l | 3690 | 7940 | 3297 | 5.0 | 2.4 | 1.6 | 2.3 |
| reg int | 2000 | 0 | 0 | 0.0 | 0.0 | 0.8 | 2.6 |

eters are presented in Table II. For the kernels the problem size is specified with s indicating small or in-cache and l indicating large or out-of-cache. All integer cores in an AMD Interlagos die (there are two die per processor) were used when collecting the data for the table. The stencil kernel is a standard 5-point stencil, and the register int kernel refers to a problem that uses only registers with integer operations. In general the effective clock frequency follows the problem size, except for the register case. For the case of the small stencil the effective clock frequency changed from 2.3 GHz to 2.45 GHz to 2.58 GHz as the number of integer cores used on an Interlagos die went from 8 (all cores) to 4 (1 core per Bulldozer module) to 1. This is an example of AMD's Turbo CORE frequency scaling, which (although not demonstrated here) is able to increase the clock frequency even when all integer cores are loaded.

We now show IPC traces and how we derive the kernels and their performance characteristics. Unless noted otherwise, the sample interval is 4 ms and applications were run with one MPI task or one OpenMP thread per AMD Interlagos integer core.

## A. NAMD

Currently we have only a cursory IPC trace and aggregate counter data for NAMD. The goal is to provide performance data on the bonded, non-bonded and particle-mesh Ewald (PME) phases of the code. The counter data is in the process of being collected using the internal PAPI calls already in the code. Some changes were needed to allow for the use of native events. As NAMD employs the Charm++ framework and as such is inherently asynchronous and adaptive, it is difficult to view the execution of the phases in a sequential manner. The traditional rates and parameters in Table III are in line with the current understanding of NAMD's performance. The asynchronous nature of execution appears to skew the effective clock rate measurement.

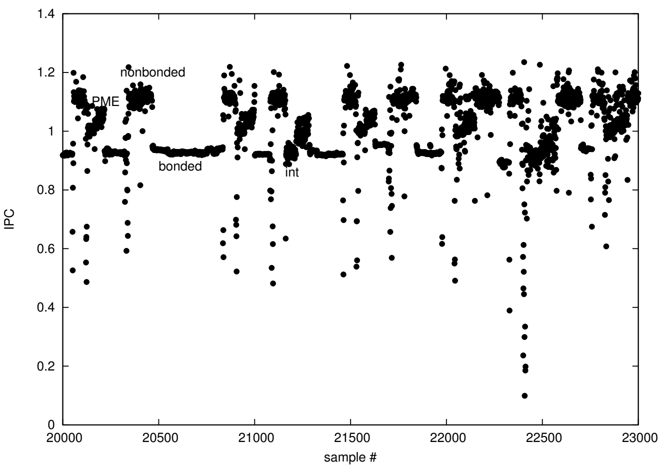

Fig. 1. **An IPC trace for NAMD.**

TABLE III
KERNEL PERFORMANCE CHARACTERISTICS FOR NAMD

| phase | MIPS | MFLOPS/s | MiBPS | CI | AI | IPC | effGHz |
|---|---|---|---|---|---|---|---|
| nonbonded | 2460 | 1377 | 7506 | 1.1 | 0.2 | 1.1 | 2.3 |
| PME | 1772 | 1408 | 3299 | 1.7 | 0.4 | 0.8 | 2.3 |
| bonded | 1617 | 723 | 1821 | 0.8 | 0.4 | 0.7 | 2.3 |
| integrate | 1394 | 581 | 4573 | 0.8 | 0.1 | 0.6 | 2.3 |

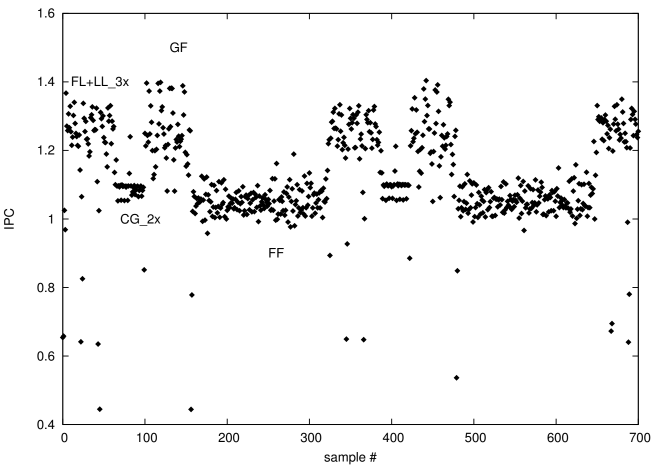

Fig. 2. **An IPC trace for the su3_rmd application is show for 2 steps. The 5 phases are labeled.**

### B. MILC

We ran MILC in the optimal configuration with one MPI task per integer core (16 tasks per Interlagos CPU) with a local lattice of 6x6x6x6. Figure 2 shows the IPC trace and annotated phases for 2 steps of the su3_rmd application. The number of iterations for the CG phases are small due to the overall small total lattice size.

TABLE IV
KERNEL PERFORMANCE CHARACTERISTICS FOR MILC

| phase | MIPS | MFLOPS/s | MiBPS | CI | AI | IPC | effGHz |
|---|---|---|---|---|---|---|---|
| LL | 1123 | 707 | 3179 | 1.1 | 0.2 | 0.5 | 2.2 |
| FL | 1475 | 1425 | 3233 | 1.9 | 0.4 | 0.6 | 2.4 |
| FF | 1305 | 1057 | 2055 | 1.2 | 0.5 | 0.5 | 2.4 |
| GF | 1414 | 1087 | 3719 | 1.4 | 0.3 | 0.6 | 2.4 |
| CG | 1353 | 1082 | 3051 | 1.7 | 0.4 | 0.6 | 2.5 |

The MILC code consists of five different phases with different performance characteristics/signatures: LL, FL, FF, GF, CG. Table IV shows the performance characteristics for each kernel. The FL phase has the highest floating point rate and also the highest computational intensity. Its arithmetic intensity is average, pointing towards very high locality. LL is lower in all regards but not a key routine overall. The most important routine, CG, performs two floating point operations per loaded word from main memory but loads nearly three words for each floating point operation from caches. Its overall performance is 1 GFLOPS/s and with an effective clock frequency of 2.4 GHz.

### C. PPM

The IPC trace for PPM is shown in Figure 3, where a sawtooth structure is seen for each evaluation of the ppmmf kernel.

TABLE V
KERNEL PERFORMANCE CHARACTERISTICS FOR PPM

| phase | MIPS | MFLOPS/s | MiBPS | CI | AI | IPC | effGHz |
|---|---|---|---|---|---|---|---|
| ppmmf | 1784 | 3351 | 2839 | 3.0 | 1.1 | 0.7 | 2.4 |

The overall characteristics for PPM are shown in Table V. The high FLOP rate is a result of the cache-blocked and nearly cache-contained design of the data structures, and a high computational intensity.

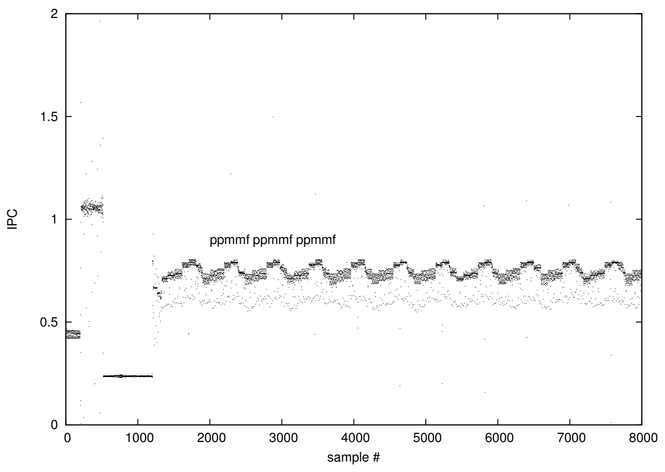

Fig. 3. **An IPC trace for PPM. The repeated ppmmf phase is labeled.**

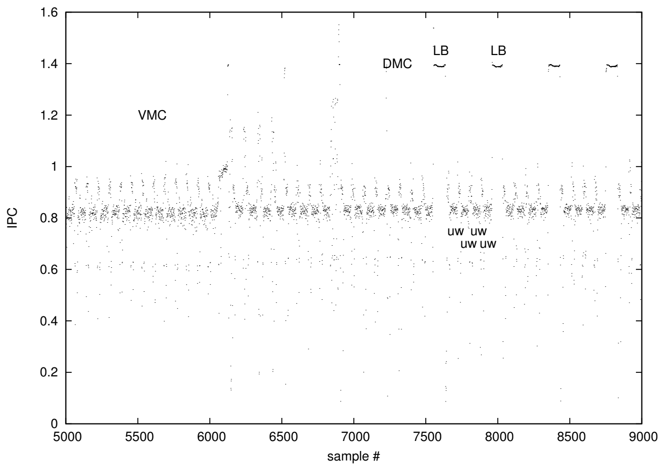

Fig. 4. **An IPC trace for the qmcpack application is show for some VMC steps and DMC steps. The DMC phases LB and uw are labeled.**

### D. QMCPACK

The QMCPACK application was run with 2 MPI tasks per Bulldozer module, i.e. 16 MPI tasks per Interlagos processor. The IPC trace in Figure 4 shows steps near the end of the Variational Monte Carlo (VMC) phase and steps at the beginning of the Diffusion Monte Carlo (DMC) phase. The dominant DMC and VMC phases show similar structure with an average IPC of 0.8, but the DMC has regions where the IPC jumps to 1.2, moving the average IPC to 0.9 as a whole. The two phases of the DMC phase are labeled as load-balance (LB) and update-walker (uw). Table VI provides the overall and phase-specific performance.

TABLE VI
KERNEL PERFORMANCE CHARACTERISTICS FOR QMCPACK

| phase | MIPS | MFLOPS/s | MiBPS | CI | AI | IPC | effGHz |
|---|---|---|---|---|---|---|---|
| ALL | 2083 | 943 | 1933 | 1.1 | 0.5 | 0.9 | 2.3 |
| uw | 1902 | 1177 | 2433 | 1.5 | 0.5 | 0.8 | 2.3 |
| LB | 3155 | 0 | 18 | 0.0 | 0.0 | 1.4 | 2.3 |

## *E. WRF*

The top 5 compute phases for WRF are shown in Figure 5 and detailed in Table VII. The microphysics phase (MP) phase dominates a typical step, followed by the planet boundary layer (PBL) and two sections of the Runge-Kutta step: scalar update (RKs) and time averaging (RKt).

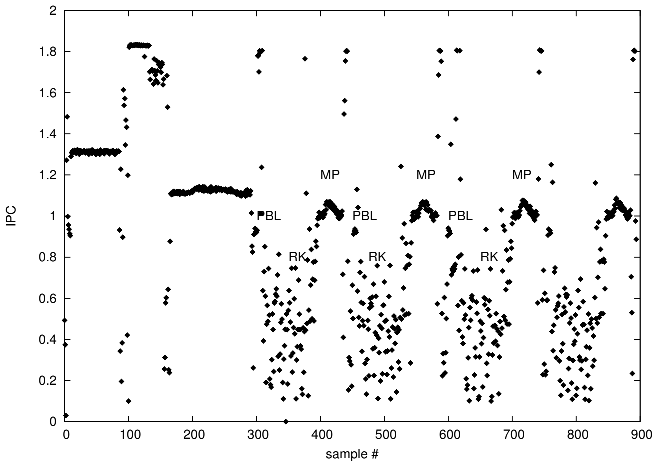

Fig. 5. **An IPC trace for the WRF application is show for 3 steps. The dominant phases are labeled.**

TABLE VII
KERNEL PERFORMANCE CHARACTERISTICS FOR WRF

| phase | MIPS | MFLOPS/s | MiBPS | CI | AI | IPC | effGHz |
|---|---|---|---|---|---|---|---|
| MP | 2647 | 590 | 1288 | 0.5 | 0.5 | 1.0 | 2.6 |
| PBL | 2197 | 566 | 4511 | 0.5 | 0.1 | 0.9 | 2.6 |
| RKt | 1328 | 2695 | 11842 | 2.0 | 0.2 | 0.6 | 2.3 |
| RKs | 1764 | 1120 | 4967 | 0.8 | 0.2 | 0.7 | 2.5 |

## *F. SPECFEM3D_GLOBE*

Two IPC traces for SPECFEM3D_GLOBE are shown in Figure 6 and Figure 7 with the later providing a close up detail of several time steps where the different phases of a step are labeled. As mentioned above the two phases are repsonsible for isotropic and transverse isotropic force calulations of the mantle and crust and thus are similar in structure and instruction mix, with the difference being in the data layout. Table VIII shows reasonable floating point performance, and that the power demands on the processor cause the effective frequency to be at the lower setting.

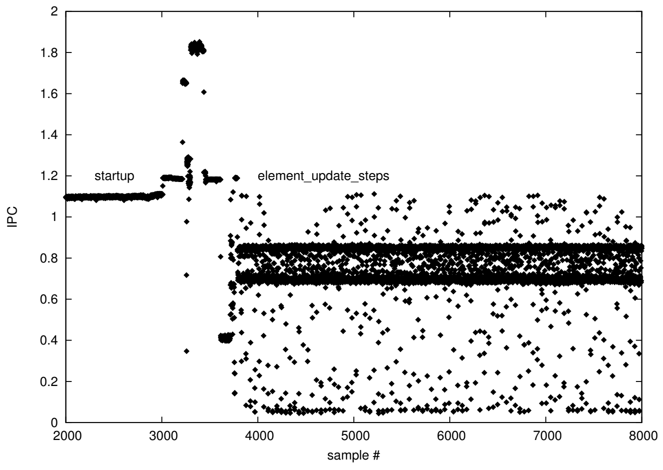

Fig. 6. **An IPC trace for the SPECFEM3D_GLOBE application showing startup and compute steps.**

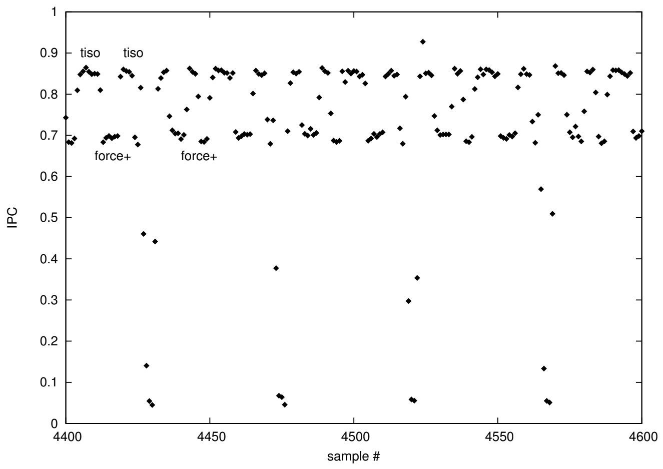

Fig. 7. **A closeup of the IPC trace for the SPECFEM3D_GLOBE application is shown with the phases within a step labeled.**

TABLE VIII
KERNEL PERFORMANCE CHARACTERISTICS FOR SPECFEM3D_GLOBE

| phase | MIPS | MFLOPS/s | MiBPS | CI | AI | IPC | effGHz |
|---|---|---|---|---|---|---|---|
| tiso | 1973 | 2010 | 1197 | 1.9 | 1.8 | 0.8 | 2.3 |
| forces | 1602 | 1736 | 4577 | 1.5 | 0.4 | 0.7 | 2.3 |
| iso | 1474 | 1396 | 1617 | 1.6 | 0.9 | 0.6 | 2.3 |

## G. NWCHEM

NWCHEM was run with 1 process per Bulldozer module, i.e., 8 processes per Interlagos processor. The IPC trace clearly shows three different phases (the first two phases are too short to be visible in the plot).

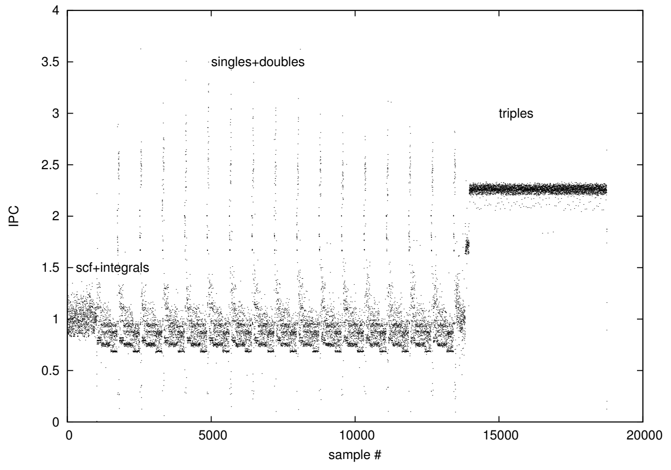

Fig. 8. **An IPC trace for NWCHEM. The phases are labeled.**

In Table IX detailed counter collection for phases 1 and 2 exhibit a rather low computational intensity and locality, and thus have low floating point performance. The phases 3+4 (singles and doubles) improve on phases 1 and 2, running at about 1.3 GFLOPS/s. The triples phase 5 entails large numbers of dense matrix multiplications and thus achieves a very high performance of nearly 7 GFLOPS/s with an effective clock frequency of 2.6 GHz.

TABLE IX
KERNEL PERFORMANCE CHARACTERISTICS FOR NWCHEM

| phase | MIPS | MFLOPS/s | MiBPS | CI | AI | IPC | effGHz |
|---|---|---|---|---|---|---|---|
| 1 | 2616 | 431 | 5464 | 0.3 | 0.1 | 1.0 | 2.6 |
| 2 | 2660 | 398 | 4818 | 0.3 | 0.1 | 1.0 | 2.6 |
| 3+4 | 2463 | 1246 | 6030 | 0.9 | 0.2 | 1.0 | 2.6 |
| 5 | 4156 | 6876 | 15583 | 3.5 | 0.4 | 1.6 | 2.6 |

## VI. Summary and Conclusions

We described a new, more realistic metric to assess the performance of a supercomputer for a typical computing center workload. We gave a specific example of this metric with eight applications, and discussed in detail both the codes and the science performed by each science team, as well as the most critical application performance characteristics.

We then demonstrated our method to identify kernels in an application and to assess the performance characteristics of each of the kernels using hardware performance counter data.

Our analysis is an important first step towards a detailed understanding of each application's performance and a good classification of the representative workload.

## Acknowledgments

We thank the Blue Waters Applications and User Services team, especially the SPP subgroup (alphabetically): Victor Anisimov, Eric Bohm, Robert Brunner, Ryan Mokos, Craig Steffen, and Mark Straka for their contributions to this work.

## References

[1] F. Petrini, D. J. Kerbyson, and S. Pakin, "The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q.," in *Proceedings of the ACM/IEEE SC2003 Conference on High Performance Networking and Computing*, p. 55, ACM, November 2003.

[2] T. Hoefler, T. Schneider, and A. Lumsdaine, "Characterizing the Influence of System Noise on Large-Scale Applications by Simulation," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC'10)*, Nov. 2010.

[3] W. T. Kramer, *PERCU: A Holistic Method for Evaluating High Performance Computing Systems*. PhD thesis, EECS Department, University of California, Berkeley, Nov 2008.

[4] T. Hoefler, W. Gropp, M. Snir, and W. Kramer, "Performance Modeling for Systematic Performance Tuning," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11), SotP Session*, Nov. 2011.

[5] C. Mei, Y. Sun, G. Zheng, E. J. Bohm, L. V. Kale, J. C. Phillips, and C. Harrison, "Enabling and scaling biomolecular simulations of 100 million atoms on petascale machines with a multicore-optimized message-driven runtime," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, (New York, NY, USA), pp. 61:1–61:11, ACM, 2011.

[6] D. Terpstra, H. Jagode, H. You, and J. Dongarra, "Collecting performance data with papi-c," *Tools for High Performance Computing 2009*, pp. pp. 157–173.

[7] C. Bernard *et al.*, "Studying Quarks and Gluons On Mimd Parallel Computers," *International Journal of High Performance Computing Applications*, vol. 5, no. 4, pp. 61–70, 1991.

[8] M. A. Clark and A. D. Kennedy, "Accelerating fermionic molecular dynamics," *NUCL.PHYS.PROC.*, vol. 140, p. 838, 2005.

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

[10] 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.

[11] G. Bauer, S. Gottlieb, and T. Hoefler, "Performance Modeling and Comparative Analysis of the MILC Lattice QCD Application su3 rmd," in *Accepted at CCGrid 2012*, May 2012.

[12] P.-H. Lin, J. Jayaraj, and P. Woodward, "A study of the performance of multifluid ppm gas dynamics on cpus and gpus," in *Application Accelerators in High-Performance Computing (SAAHPC), 2011 Symposium on*, pp. 42 –51, july 2011.

[13] J. Kim and K. E. anf David Ceperley, "Quantum monte carlo keeping up with the hpc evolution."

[14] "Weather Research and Forecasting Model." http://www.wrf-model.org/.

[15] UCAR, "Advanced-Research WRF Dynamics and Numerics." http://www.mmm.ucar.edu/wrf/users/docs/wrf-dyn.html.

[16] L. Carrington, D. Komatitsch, M. Laurenzano, M. M. Tikir, D. Michea, N. Le Goff, A. Snavely, and J. Tromp, "High-frequency simulations of global seismic wave propagation using specfem3d_globe on 62k processors," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, (Piscataway, NJ, USA), pp. –11, IEEE Press, 2008.

[17] R. Kufrin, "Measuring and improving application performance with perfsuite," *Linux J.*, vol. 2005, pp. 4–, July 2005.

[18] J. Stanton, J. Gauss, J. Watts, and R. Bartlett *J. Chem. Phys.*, vol. 94, pp. 4334–, 1991.

[19] R. Kobyashi and A. Rendell *Chem. Phys. Letters*, vol. 265, pp. 1–, 1997.

[20] K. Bowers, B. Albright, B. Bergen, L. Yin, K. Barker, and D. Kerbyson, "0.374 pflop/s trillion-particle kinetic modeling of laser plasma interaction on roadrunner," *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, p. 1, 2008.