LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-JRNL-580932

Determination of Performance Characteristics of Scientific Applications on Blue Gene/Q

C. Evangelinos, R. E. Walkup, V. Sachdeva, K. E. Jordan, H. Gahvari, I. H. Chung, M. P. Perrone, L. Lu, L. K. Lu, K. Magerlein

September 13, 2012

IBM Journal of Research and Development

Disclaimer

Determination of Performance Characteristics of Scientific Applications on Blue Gene/Q

C. Evangelinos, R.E. Walkup, V. Sachdeva, K.E. Jordan, H. Gahvari, I-H. Chung, M.P. Perrone, L. Lu, L-K. Liu, K. Magerlein

The IBM®Blue Gene®/Q (BG/Q) platform presents scientists and engineers with a rich set of hardware features: e.g. 16-cores per chip sharing an L2-cache, a wide SIMD unit, a 5-dimensional torus network, and hardware support for collective operations. An especially important feature is that the cores have 4 "hardware threads", which makes it possible to hide latencies and obtain a high fraction of the peak issue rate from each core. All of these hardware resources present some unique performance tuning opportunities on BG/Q. We provide an overview of several important applications and solvers and study them on BG/Q using performance counters and MPI profiles. We discuss how BG/Q tools help us understand the interaction of the application with the hardware and software layers, and provide guidance for optimization. Based on our analysis we discuss code improvement strategies targeting BG/Q. Information about how these algorithms map to the Blue Gene architecture is expected to impact future system design as we move to the Exascale era.

1. Introduction

The Blue Gene®/Q (BG/Q) platform is the next generation of the Blue Gene family of IBM supercomputers. It continues the tradition of low power, high density, reliability, and extreme scalability. Like its predecessors Blue Gene®/L (BG/L) and Blue Gene®/P (BG/P) its processor is based on a low power PowerPC core not intended for the enterprise market, extensively modified and enhanced by the addition of high performance computing (HPC)-targeted hardware features.

As part of the “bringup” and early testing for BG/Q, IBM Research was involved in porting and optimizing several scientific and engineering codes to BG/Q from its main US-based partners Lawrence Livermore (LLNL) and Argonne (ANL) National Labs as well as other customers. Traditional programming methods, message passing interface (MPI) and OpenMP, are supported on BG/Q, and our experience has been that porting codes to BG/Q is normally a simple process. Given the enhancements in the runtime environment for BG/Q, even codes that required modifications to run on BG/P could more easily be setup to run on BG/Q. Optimizing for performance, however, is a bit different.

The PowerPC A2 core in BG/Q is an in-order 64-bit core running at 1.6 GHz with Simultaneous Multi-Threading (SMT) capability using four hardware threads. The A2 core is a very simple core by current standards. It has two execution units: XU, the integer/load/store unit and AXU, the floating-point unit. When a single thread is running, the A2 core can issue at most one instruction per cycle. By using two or more hardware threads, the core can issue up to two instructions per cycle, one from each of the two execution units. With four hardware threads available on each core, there is ample opportunity for some thread to make progress if others are stalled. This design has the consequence that effective use of the four hardware threads is the key to getting the best overall application performance. On BG/Q, the performance per-thread is low by design, and it is necessary to make use of a large number of threads to obtain high

performance. In contrast, most current SMT-capable processors (IBM Power6/7 and Intel Core i3/5/7 families) have multiple execution units, and can issue more instructions per cycle, and operate at higher clock frequencies. The BG/Q core can provide excellent power efficiency, but that comes at a programming cost because a higher degree of parallelization is required to obtain comparable performance to systems that have faster, less power-efficient cores.

The floating-point unit is designed to handle 4-wide SIMD (QPX) instructions leading to a peak performance of 4 x 2 x 1.6=12.8 Gflop/s per core. To obtain a high fraction of peak, at least two threads must be active so that both execution units can be kept busy issuing loads/stores and address calculations, along with multiply-adds. Each core has a dedicated 16 KB L1 D-cache (64 byte lines) and an L1P prefetch buffer with a capacity of 32 lines (128 bytes long), with a ~24 cycle latency. The BG/Q node has one chip with 16 A2 cores (plus one for the OS and one spare), and a 32 MB shared L2 cache with a latency of ~84 cycles and up to ~300 GB/s effective bandwidth. There is 16GB of main memory per node, with ~350 cycle latency and ~28.5 GB/s memory bandwidth achievable in streaming mode. BG/Q nodes are interconnected by a 5D torus network with a 2GB/s bandwidth on each of 20 links to neighboring nodes (as opposed to the combination of a 3D torus and a "collective" tree network in BG/L and BG/P).

Compared to Blue Gene/P, the theoretical peak floating point speed is ~15 x more per node (4x cores per node, 2 x SIMD width and 1.88 x the frequency) while the sustained memory bandwidth is ~3 x more per node. The application performance ratio varies depending on resources used, but typical numbers for the LLNL and ANL set of production codes are ~8 x more performance per node on BG/Q relative to BG/P.

The paper is organized as follows: In section 2 we present our tools for performance analysis on BG/Q. We then proceed in section 3 to illustrate what was learned through the use of these tools about several HPC applications ported to BG/Q. In section 4 we go into a more detailed analysis illustrating the main challenges and remaining issues for some applications still being tuned for BG/Q. We summarize the performance information in section 5 and try to draw more general conclusions on the likely performance characteristics of existing scientific applications on BG/Q and the algorithmic and coding changes that may be beneficial going forward on this and future platforms with similar hardware attributes.

2. BG/Q performance analysis

2.1 MPI profiling on BG/Q

MPI (Message Passing Interface) provides a profiling interface which makes it possible to instrument MPI routines, and obtain detailed information about communication. Our internal IBM Research MPI profiling tool (libmpitrace/libmpihpm) [**1**] for BG/Q outputs performance summary data at the end of application execution, when `MPI_Finalize()` is called. In that sense it is quite similar to the mpiP [**2**], IPM [**3**], FPMPI [**4**] , MPInside [**5**], and other performance tools. "Mpitrace" allows the option of collecting a detailed trace for MPI events (usually provided by separate - higher overhead - tools). The choice of what events to monitor or not and how to display them is controlled via environment variables and the tool simply requires relinking. Additionally it provides the option of adding start and stop statements in the code for

fine tuning of what is being counted or traced. In our experience unless one is tracing or very latency sensitive, the overhead is usually small enough to allow using it in production mode. By default it produces an MPI summary for the processes with the least, the most and the median communication time as well as process 0.

The libmpitrace tool for BG/Q has been designed with extreme scalability in mind and tested at > 1 million MPI ranks. At that scale care is required to reduce or eliminate arrays that are dimensioned by the number of ranks, otherwise memory utilization can limit scalability; and of course the number of output files must be kept to a manageable level.

2.2 Hardware performance counters

BG/Q has a good set of hardware counters that can provide detailed information about work done by the processor cores, the memory sub-system, and the network. The BG/Q system software includes a BGPM layer for access to the performance monitor data. In practice it is useful to provide a thin layer, called HPM, on top of BGPM that can be easily called from Fortran, C, or C++. Portable interfaces like PAPI **[6]** are also supported but not used by our tools. The collection of hardware counter data was integrated with the MPI profiling software introduced above. That enables MPI to be used for data reduction, so that simple derived metrics can be provided directly, without post-processing more primitive counter output. On BG/Q there are separate counters for each hardware thread on every core, but the counters for the L2 cache and memory sub-system are shared across the node. Most applications use multiple MPI processes on each node, and so it is convenient to use MPI to collect node-wide aggregate counter data for analysis. Certain predefined groups of events (selectable via an environment variable) can be counted to provide more user-friendly derived information like instruction throughput, cache-hit rates and DDR traffic. The counting starts at `MPI_Init()` and ends at `MPI_Finalize()` if the transparent (re-linking only) approach is chosen. Alternatively the user can insert (nested named) start or stop statements in the source code to also investigate specific code regions.

2.3 Gprof and profil() on BG/Q

Traditional interrupt-driven profiling is provided on BG/Q – via the gprof mechanism (-pg) or the profil() routine. The overhead of -pg when used during compilation can be significant, but if used only at link-time, the overhead is normally small. The profil() routine is included in the system software stack, and can be used directly or with other profiling packages. However, for multi-threaded applications, the profil() routine provides data for only the master thread, while the –pg approach has been customized for BG/Q to support profiling for multiple threads. The mpitrace tool has integrated support for profil() and uses the same mechanisms employed for MPI and performance counter profiling to automatically present the user with profiles for the processes taking the most, the least and the median time as well as process 0.

2.4 Performance data repository

In order to characterize High Performance Computing (HPC) applications and to understand the system usage, we have implemented a performance data repository. The modified performance tools such as HPM and MPI profiler inject data into this performance data repository which is hosted using an SQL compatible database. This allows the merging of performance data from various source (systems and users). This consolidated performance data repository allows us to perform data mining more efficiently by using queries generated via a spreadsheet or a web page interface. As this repository is being populated it is helping architecture designers to realize how codes utilize the hardware (see **Figure 2**); thus it is expected to facilitate co-design for next generation HPC systems.

3. Application Performance

In this section we describe results of our performance characterization for several scientific codes. The results are partly still in flux due to the early state of compilers, the OpenMP runtime and the messaging libraries. Access to fast parallel I/O was generally very limited and we have tried to exclude I/O from our results and analysis.

3.1 LLNL Sequoia Benchmarks

The LLNL Sequoia benchmark codes [**7**] were used regularly from the earliest stages of hardware availability. The five "tier 1" benchmarks are all constructed as weak-scaling benchmarks that can be tested from a single process or thread out to many thousands of processes.

3.1.1 Radiation Transport: IRS

IRS [**8**], stands for Implicit Radiation Solver. It is a code written in C which solves the radiation transport equation with a flux-limited diffusion approximation with an implicit matrix solution for which it uses a preconditioned conjugate gradient method, and is a hybrid (MPI/OpenMP) code. It is run in a weak scaling mode as per the benchmark instructions. Our analysis of IRS showed the following main performance issues: (1) A memory access limitation in the rmatmult3() routine (multiplying many arrays concurrently) – the original loops had too many streams for the prefetch hardware to handle. The first optimization was to split the loop into a number of stages, to limit the number of streams. Extra stores and loads are required in that approach, but the overall performance improved significantly because the hardware prefetch mechanism could be used. (There is a limit of about 16 streams per core with default parameters). (2) Some additional performance was obtained by re-writing the code so that the compiler could recognize data that could be loaded once and re-used. (3) Part of the code as provided exhibited communication overhead scaling linearly with the number of MPI tasks; for large enough number of tasks this became important. This was identified using our tools and corrected a small constant value. The net speedup achieved by optimizing was about 2 x for 1-rack of BG/Q.

3.1.2 Radiation transport: UMT

UMT [**9**] is a deterministic multigroup 3D photon transport code for unstructured meshes also using mixed MPI and OpenMP. The code employs a mix of C++, C, Fortran90, and Python, and solves the first-order form of the steady-state Boltzmann transport equation. The benchmark rules required testing for both strong and weak scaling. UMT originally had just one OpenMP parallel region, which included the main computational routine, snswp3d. One might think that code generation in that routine would be the focus. However, our analysis shows that performance in that section is limited by bandwidth to memory, not code generation by the compiler (or by hand tuning). **Figure 1** shows the performance as a function of the number of threads per core, with and without QPX instructions generated by the compiler. Although the QPX improvement is significant at one thread per core, one hits the bandwidth limit at two threads per core. In contrast, code with scalar instructions continues to improve with additional threads, reaching nearly the same performance plateau with four threads per core.

The more successful tuning strategy for UMT involved improving the OpenMP coverage so that as much of the code as possible could fully utilize the available hardware threads.

3.1.3 Molecular Dynamics: LAMMPS

LAMMPS [**10**] (Large-scale Atomic/Molecular Massively Parallel Simulator) is a C++ molecular dynamics (MD) code from Sandia that has been widely used in academia and industry. The code as provided uses MPI as the parallelization strategy, without threading or OpenMP (later versions of LAMPPS now have some OpenMP acceleration as an option). The benchmark problem (classical MD EAM run in weak scaling mode) involved embedded-atom potentials, which have excellent locality, with nearest-neighbor communication in three dimensions. This code can use all of the hardware threads on BG/Q with 64 MPI ranks on each node. Excellent scaling of LAMMPS to over 1 million MPI ranks was demonstrated using this method. At that scale (16 racks of BG/Q), it is best to specify a three-dimensional process grid that fits naturally onto the 5D torus network. For example, the 16-rack BG/Q block has dimensions <A,B,C,D,E,T> of <16,8,8,8,2,64>, which can be grouped into an effective 3D topology of <128,128,64>.

For experimental purposes, OpenMP was introduced for force-evaluation, neighbor-list, and coordinate/velocity updates. This allows for a comparison between the pure MPI and hybrid MPI + OpenMP methods. In **Table 1** we compare the different programming methods (pure MPI vs. hybrid). In this particular example (512 nodes of BG/Q), the hybrid method was about 7% faster. As can be seen in Table **1,** there is somewhat less contention for L1 D-cache with the hybrid code, but overall the two approaches have very similar characteristics. Generally speaking, the main motivation for the hybrid programming method is to enable access to all of the hardware threads in applications where that would not be possible with an MPI-only approach (often due to memory limitations). In cases where both methods are workable, there is often a small advantage for the hybrid version.

3.2 ANL applications

A set of ten science applications were chosen by Argonne National Labs for performance evaluation on BG/Q. They are all covered in a companion paper in this issue – but we describe in a little more detail two of them that present extra interest – the overall performance of the applications compared on a per-node basis to BG/P (same number of cores used or same amount of total memory) can be seen in **Table 2** showing an improvement ranging from 6x to almost 17x.

3.2.1 Atmospheric Climate Simulations: HiRAM

HIRAM stands for High-Resolution Atmosphere Model and is based upon a widely used Climate code (in our case employing a Cubed-Sphere Non-Hydrostatic Dynamical Core) from the Geophysical Fluid Dynamic Laboratory (GFDL) [**11**]. The particular configuration studied is the classical Held-Suarez [**12**] benchmark run in strong scaling mode.

This problem uses a grid with 2560 points in each of the two dimensions, on six faces of the cube, for a total of about 39 million grid points, each with 32 vertical levels. Such a high resolution model can make good use of a large number of processes and/or threads. The code is OpenMP enabled, with most of the OpenMP constructs distributing work over the vertical layers, or over one of the lateral dimensions. With one rack of BG/Q, excellent performance was obtained using eight MPI ranks per node, with 8 OpenMP threads per MPI rank. Due to the cubed-sphere geometry, the number of MPI ranks must be a multiple of 6, not a power of two. However, one can choose decompositions that use almost all of the processors in a power-of-two BG/Q partition. For the 1-rack job, we chose a decomposition of 6 x 44 x 31 = 8184 MPI ranks and 8 OpenMP threads per rank, for a total of 65472 threads. Scaling of the main computational loop remained linear, with ~3% of the time spent in communication when using the full rack. This application can scale out much further. A comparison with a similar job on BG/P (same code, same 6 x 44 x 31 MPI decomposition, but the maximum of 4 possible OpenMP threads) indicated a performance ratio of ~14 x for a node-to-node comparison; i.e. one rack of BG/Q (16K cores) was significantly faster than 8 racks of BG/P (32K cores).

The cubed-sphere model has a small load imbalance due to the non-uniform decomposition of the grid and the underlying science. This can be clearly seen in the MPI timing data: processes with the least work spend more time waiting (for the processes with the most work). There is a small additional amount of work for processes that have land-points vs. ocean points.

3.2.2 Computational Fluid Dynamics: DNS3D

DNS3D [**13**] is a computational fluid dynamics (CFD) code employing direct numerical simulation (DNS) of Navier-Stokes equations to solve turbulent viscous flow for the idealized case of a periodic 3D domain. The method employed is pseudo-spectral in space and 4^{th} order RK in time. The code is Fortran and MPI-only and relies on the P3DFFT [**14**] library for the parallel 3D FFTs and on FFTW or ESSL for the actual underlying 1D FFT. The 3D FFT approach of the P3DFTT handles the inherent scalability limitation of the usual "slab" approach of 3D FFT decompositions by converting to a "pencil" decomposition and relies on `MPI_Alltoallv()`. All-to-all represents a communication pattern that stresses networks and is challenging for most topologies. For the problem sizes (in weak scaling mode) that were

investigated the memory footprint per rank is small enough that one can run in a mode of using 64 MPI tasks per node.

The 5D torus network on BG/Q proved to be very successful in handling such a challenge. For a 2048^3 problem on 8192 cores (32768 MPI tasks) a speedup of 16.9 over BG/P was observed (more than the 15 ratio in peak flop/s). In fact one sees that while using the same number of physical cores (16 per node, 512 nodes) overall DNS solver time decreases from 3421.80 s to 2424.00 s to 1849.00 s (16, 32 and 64 MPI tasks per node) and the time per timestep drops from 16.61s to 12.12 s to 10.33 s – all this extra performance coming "for free" from SMT.

In our experience the 4-way SMT on BG/Q is very effective at hiding stalls due to data-dependencies (pipeline stalls) and/or stalls due to waiting for the memory subsystem. On the other hand, applications that have a very high memory-bandwidth requirement can hit that limit with one thread per core. The hardware counters on BG/Q can identify those situations.

3.3 European Applications

3.3.1 Meteorology: MESO-NH

MESO-NH [**15**] is a gridpoint limited area non-hydrostatic atmospheric meteorological model. It has been in development in France since the 1990s. The code has been parallelized for massive parallel processors (MPPs) and uses 3D fast Poisson (FFT-based) solvers for the (elliptic) pressure equation hence global array transposes (all-to-all based) are necessary. Also global reductions have to be used and are the other main collective call involved. Finally the code is fairly I/O dependent – this showed up in the profiles as an imbalance in MPI communications used to distribute input data (MESO-NH does not use parallel I/O and our I/O subsystem was still in its infancy and thus quite under-performing at the time). The benchmark problem was run in strong scaling mode.

We were unable to run more than 16 MPI tasks per BG/Q node for the benchmark problem size of 3456x3456x128 as the minimum memory footprint was too large to support 2 MPI tasks per core. Hence the only way to utilize the 4 hardware threads on the BG/Q cores was to add OpenMP or to attempt to use asynchronous communication threads (or both).

The benchmarking rules precluded code-changes but MESO-NH proved to be fairly amenable to automatic parallelization by the compiler. With the auto-parallelized code, the overall runtime dropped from 712s to 565s on 8192 MPI processes (1 process per physical core). Adding the use of asynchronous communication threads decreased the overall benchmark runtime to 538s. The profile of the code was fairly flat – the only "hot" spot at ~10% of runtime was a limiter of a variable: this evaluated an `if` statement for every gridpoint at every timestep and could not be refactored in a way that did not kill performance. Multithreading the loop however helped reduce the branch penalty. One can expect to do even better with manual OpenMP parallelization that avoids the extra overhead usually seen in autoparallelized codes.

4. A Deeper Look into Applications and Algorithms

Having seen the performance characteristics of a variety of scientific codes on BG/Q we proceed to look in more depth in this section in three different codes still being tuned for BG/Q that represent both important current and future application areas as well as different algorithmic approaches to the use of HPC. ROSETTA is an example of a class of highly parallel biology computations that employ machines as massive high-throughput compute engines but still require communication to distribute work and collect results. Efficient utilization of BG/Q motivates reducing the number of MPI processes and introducing threading. AMG2006 is indicative of a very popular family of solvers based on algebraic multigrid methods and lessons learned for AMG2006 are expected to be of wider use regarding efficiently using multigrid on multicore/manycore processors as well as optimizing it for hybrid (MPI-OpenMP) use. Finally the revere time migration (RTM) algorithm is representative of a large class of geophysical elasticity problems involving very large 3D stencils of great practical use in industry. RTM motivates use of the hardware threads as well as SIMDization of code.

4.1 Protein Modeling: ROSETTA

ROSETTA [**16**] is a library based object-oriented software suite which provides a system for predicting and designing protein structures, protein folding mechanisms, and protein-protein interactions. The library contains the various tools that ROSETTA uses, such as Atom, ResidueType, Residue, Conformation, Pose, ScoreFunction, ScoreType, and so forth. These components provide the data and services ROSETTA uses to carry out its computations. It is currently being ported and tuned for BG/Q as part of an ANL and IBM Research collaborative effort.

For our purposes, we have used the executable minbench part of the ROSETTA suite. The execution of minbench involves 2 basic steps: energy minimization and subsequent energy computation. The executable minbench works on a structure for 300 seconds, and tries to go through as many iterations as possible of these two steps. It reports the timings as T/iterations where T is the time taken and the iterations is the number of iterations completed.

The execution profile of minbench is thus consumed by the steps above in varying degrees based on the size of the input. For larger inputs, the energy minimization routine (dfpmin) takes the higher percentage of the time, while for smaller inputs the energy computation routines consume more time.

ROSETTA's calculations for the structure are embarrassingly parallel, and require no communication with other structures during computation. ROSETTA currently has support for 2 partitioning policies: a master-slave policy in which the master delegates tasks to slaves upon request, and a partitioning policy which provides subsets of the sequences to MPI processes based on their respective MPI ranks. A master-slave policy has proven to not be sustainable at large scale, as a single master can be overloaded with managing the slaves. To examine the bottleneck of the single master system, we ran minbench with 2048 slaves and 6144 jobs (64 nodes, 64 processes per node – the maximum allowed on BG/Q, utilizing all 4 hardware threads for MPI processes). In the MPI partitioning, the slave processes use `MPI_Recv()` for receiving

job ids; thus in this case, every slave process completes 3 `MPI_Recv()` in total. MPI timing data showed that the slaves were waiting to receive data from the master a large fraction of the total elapsed time. Thus, the policy of a single master is clearly a bottleneck with a high number of slave processes.

Since minbench has a very small memory footprint, we have been able to fit 64 MPI processes per node on BG/Q for most inputs, except for the largest input case, where 32 tasks can fit to perform work on every node. Essentially, despite this being a parallel MPI code, based on the execution characteristics of ROSETTA, we are using BG/Q as a high-throughput engine, using the maximum permissible number of processes on every node.

We have also collected machine-level characteristics for the inputs using the libhpm library on BG/Q. In **Table 3** are some of the key statistics for the smallest input as we scale the number of tasks from 4 to 64 on a single node. As can be seen, the memory traffic increases substantially as we increase the tasks. This leads to a significant slowdown in performance for a given number of MPI tasks, as can be seen in the wallclock time for a fixed number of iterations in minbench shown in Table **3**. However even given this slowdown, since it is less than a factor of 4 going from 16 to 64 processes, the highest throughput strategy is to use all 64 hardware threads as MPI processes. Moreover we see that at that mode of usage, we are seeing close to 50% of the maximum issue rate (mainly integer) from every core. Similar behavior is seen for the largest input as well.

Larger input cases cannot be expected to successfully employ all available hardware threads with MPI processes because the memory footprint grows. In fact given the 16GB per node constraint, large enough input data sizes might not be able to work with more than 8 or even 4 MPI tasks per node. It is when looking to this future set of problems that we expect threading to become important for ROSETTA on BG/Q. Thus the optimization strategy we are currently pursuing is based on using portable operating system interface (POSIX) threads instead of MPI processes within the node to do the energy calculation and minimization in ROSETTA. We expect that using threads and shared memory will allow us to employ all hardware threads in foreseeable cases. Given the form of parallelism ROSETTA exhibits, we also expect better performance by going to a 1 MPI process, 64 POSIX threads per node mode of use.

4.2 Iterative Solvers: AMG2006

Algebraic multigrid (AMG) is a popular solver for large, sparse linear systems that finds use in many science and engineering applications. It is also part of the LLNL Sequoia benchmark suite. It extends the coverage of multigrid methods, which are fast linear solvers for structured grid problems, to unstructured grids such as finite element meshes. What makes multigrid attractive on HPC platforms is its ideal algorithmic scalability. When it works well, multigrid solves a problem with *n* unknowns in *O(n)* time, making it highly suitable for solving the larger and larger problems for which scientists and engineers are making use of ever larger parallel machines. The key principle of multigrid is that instead of doing all the computational work on the original “fine-grid” problem, a sequence of coarser grids is used to accelerate the overall solution. There are a number of different cycling strategies that visit the fine-grid and coarse-grid problems a varying number of times. As our first attempt to port and tune AMG2006 on BG/Q

(as part of a research collaboration with LLNL and UIUC), we examined the simplest such strategy, a V-cycle, which entails a sequential progression from finest to coarsest and then back to finest. An overview of multigrid in general, along with more details about AMG, can be found in reference [**17**].

AMG has scaled very well on BG/L [**18**] and BG/P [**19**] in the past, but has run into problems on emerging multicore cluster architectures [**20**]. The cause was found to be a combination of several performance limiting factors, from switching delays to contention between tasks for resources on the interconnect [**21**]. This showed up the most on coarse grids, where there was a lot more interprocess communication and far less computation than what is seen on fine grids. In some cases, the work on coarse grid problems with only a few unknowns per core took longer than the work on fine grid problems with tens of thousands of unknowns per core.

This naturally raises the question of how well AMG performs on BG/Q, where the number of cores per node has increased to 16 from 2 on BG/L and 4 on BG/P. There is also the question of how AMG performs when using the multiple hardware threads per core that are available on the platform. A preliminary experiment, performed using AMG on a 3D 7-point Laplace model problem with $50 \times 50 \times 25$ points per core (the same one considered in [**21**]), shows us that AMG's scalability on BG/Q can be further enhanced. Running on 16, 128 and 1024 cores gives us 70, 75.5 and 80.4 ms per V-cycle on BG/P; the respective times on BG/Q are 32.6, 39.0 and 47.1ms. We varied core counts on both BG/P and BG/Q, using the best mix of MPI tasks and OpenMP threads per node for each case. The weak scaling results suggest room for improvement in the case of BG/Q with further platform-specific optimizations.

A deeper story is told by the number and ideal mix of threads and processes on BG/Q. BG/P supported hybrid MPI/OpenMP programming, but for AMG, the best performance always involved running one MPI process per core also known as Virtual Node (VN) mode. BG/Q allows for up to 64 parallel tasks per node, which can either be all MPI processes, all OpenMP threads, or some mix of the two, and the best mix of threads and processes varied in the experiments. When running on one node (16 cores), using 64 MPI tasks resulted in the best performance, but for larger core counts, the performance using 32 MPI tasks per node and no OpenMP was better. This would indicate that the interconnect has difficulty handling accesses from too many MPI tasks. The bigger issue, though, is that the application is not successfully making use of the available parallelism, as using 32 MPI tasks and no OpenMP means that only half of the four hardware threads per core are being utilized effectively.

Changes will thus be needed to ensure that AMG makes full use of BG/Q and scales effectively to it. First, since the number and ideal mix of threads and processes varies with total core count, running AMG will need to be coupled with some form of predictive model to ensure the best performance. Furthermore, since contention for internode communication reduces the ability of AMG to exploit the full parallelism of the machine, some form of data redistribution will also be necessary to reduce communication on coarse grids. Efforts are currently underway in both of these areas.

4.3 Seismic Imaging: Reverse Time Migration (RTM)

Seismic imaging is a critical element of oil and gas exploration, providing detailed three-dimensional images of the Earth's subsurface to locate and characterize hydrocarbon reserves. The generation of such images, even using approximations to the true physics, is extremely computationally intensive. In fact, many oil companies are now predicting that their computational requirements for seismic imaging will be in the exascale range by the end of the decade. To explore the computational requirements of seismic imaging, we have implemented on both BG/P and BG/Q one of the most advanced algorithms used in production exploration today, Reverse Time Migration. By taking advantage of BG/Q's new hardware features and resources, we have achieved a 13.4x performance speedup on node-to-node comparison over the previous generation BG/P.

The Reverse Time Migration (RTM) algorithm [**22**] is known in the industry for its superior imaging accuracy for difficult subsurface structures like salt domes which are poorly imaged by other algorithms but which are very effective at trapping oil and gas. Several variants of RTM exist with differing degrees of approximation to reality, all of which are based on solving the wave equation in time and all of which use single-precision arithmetic. We implemented an isotropic, acoustic RTM variation that makes the simplifying assumptions that the wave velocity is independent of wave direction and that no energy is absorbed by the medium. The volume in interest is represented by a rectangular grid that is partitioned among the available Blue Gene nodes; MPI communication is used between each neighboring subdomain for data exchange. The most computationally intensive part of the RTM algorithms is the simulation of the propagation of 3D seismic pressure waves. To simulate these waves, we used an explicit, finite difference approximation on a regular 3D grid applied to the wave equation. This approximation gives rise to a sparse stencil along the three spatial axes, which in practice is typically selected to be 8^{th} order in space (i.e., a 9 x 9 x 9 stencil) or larger. This stencil size can lead to instabilities that are handled by using small time steps, further increasing the computational demand of these methods. The stencil calculation consumes the majority of cycles in production seismic imaging, but fortunately is amenable to computation using SIMD. Other non-negligible operations required at each time step in our implementation include boundary condition handling, a 3D correlation calculation, and communication between nearest neighbor domain partitions.

In practice, production RTM 3D model sizes vary from 512^3 to 1024^3 or larger, with thousands or tens of thousands of time steps per simulation and tens or hundreds of thousands of simulations required for a single imaging task. However there is a constant pressure in the industry to grow these models to increase image resolution and to enable deeper imaging.

For performance evaluation, we used a model with 1024^3 grid points and only 100 time iterations, since runtime scales almost linearly in the number of time steps. The tests were run on a 64-node block on both BG/Q and BG/P systems. The tests on BG/P used 1, 2, and 4 cores (its maximum) per node while the tests on BG/Q used 1, 2, up to 64 ranks/node. The results show that RTM on BG/Q is 2.6 x faster on up to 4 ranks/node, 5 x faster for 8, 10.1 x for 16 and 12.6 x for 32. At 64 ranks/node it enjoys a 13.4 x node-to-node speedup over BG/P.

We have also collected machine-level characteristics for the inputs using libmpihpm on BG/Q. In **Table 4** are some of the key statistics for the case of strong scaling on RTM as we scale the number of tasks from 1 to 64 on a single node. We observe optimal performance at about 32

ranks/node. In the case of strong scaling, the size of each subdomain decreases and thus the data 3 locality increases as we increase the number of tasks. We can see the changes in L1, L1P , L2 and DDR memory traffic from 4 to 16 ranks/node and how these metrics then change in a different direction when the ranks/node go from 16 to 64. We also observed the significant improvement in overall IPC above 16 ranks/core. All these lead to performance improvements as well as excellent scaling properties. As we expected, being able to use multiple threads per node helps us a great deal in the RTM workload. We have not yet investigated the communication time carefully; we believe that there is scope for considerable improvement there, and as we increase the number of ranks per node (therefore the total number of MPI ranks since this experiment used a fixed 64 nodes) the communication becomes more costly. We are therefore optimistic that we can improve the scaling of this code further.

5. Analysis and Conclusions

The central observation from the analysis of the various applications described in section 3 (as well as others which we could not cover due to length limitations) is that the efficient use of the four hardware threads available for each BG/Q core is key to getting good performance. By using all four hardware threads for computation, one can often get a speed-up of ~2x (and more in some cases) relative to using a single thread per core. With four threads working, some thread can often make progress when another thread is stalled, resulting in good overall utilization of the core. We see instruction throughputs in the range of 40-60% of the theoretical limit for many applications. Although the instruction throughput per core is high, the performance per thread is low. One must use a very large number of threads (or processes) to obtain overall high performance on BG/Q. Speedup can be limited when there is contention among the threads for L1 D-cache, L1P buffer, or when a node-wide resource limit (such as memory bandwidth) is reached. Using at least two of the hardware threads per core is something that most codes running on BG/Q should attempt to do; three or four hardware threads is normally better.

In order to utilize the extra hardware threads one has the option of using more MPI processes or using thread-based parallelism (usually by coding in OpenMP – or even using the autoparallelizer – though POSIX threads can also be pursued in more task-based approaches – such as ROSETTA). The hybrid approach can help reduce contention for network resources and intranode memory/L2 bandwidth. It offers flexibility since one can adjust the mix of MPI process and threads to optimize performance and to make best use of the available memory. BG/Q has many features to support effective threading, with low overhead atomic operations and the ability for fast thread wake-up, etc.. Hence, we recommend a hybrid programming strategy for many applications.

SIMD utilization is complex issue. The best use of SIMD instructions has been in level-3 BLAS routines. Alignment constraints and the complexity of handling misaligned data often limit automatic SIMDization by the compiler. In addition, memory bandwidth can easily be saturated with scalar load/store instructions without the need for quad-word loads or stores. We suggest that end users focus on effective threading first, and then consider the potential for SIMD instructions, rather than focusing on single-thread performance first, and then adding additional threads.

On the communication front we have found that the 5D torus of BG/Q provides strong all-to-all performance for those codes that require it and very nice boundary ("halo") exchange. For regular Cartesian topologies, boundary exchange can benefit from a specific mapping of the ranks to the underlying hardware topology, something that is easier to accomplish on the 5D torus as it allows several embedded 3D mappings. The 5D interconnect helps keep the maximum number of hops in any dimension fairly small, even for rather large blocks (16 racks) of BG/Q.

In conclusion our study of scientific applications on BG/Q shows that they can attain a high percentage of their performance potential. As Figure **2** shows, most scientific applications on BG/Q (with the notable exception of those dominated by dense linear algebra operations on very large matrices such as Linpack) have an instruction mix that contains a very significant fraction of integer instructions (address calculations, load/stores, branches and other more involved book-keeping and logic) and are more likely to stress the integer pipeline – this is in line with independent findings [**23**]. In our experience so far the most useful tool for increasing performance is the use of the SMT capability of BG/Q, either by assigning multiple MPI tasks per core or by Hybrid (MPI/OpenMP or Pthreads) programming.

Acknowledgment

The BG/Q project has been supported and partially funded by Argonne National Laboratory and the Lawrence Livermore National Laboratory on behalf of the U.S. Department of Energy, under Lawrence Livermore National Laboratory subcontract no. B554331. Lawrence Livermore National Laboratory is operated by Lawrence Livermore National Security, LLC, for the U.S. Department of Energy, National Nuclear Security Administration under Contract DE-AC52-07NA27344.

References

[1] R.E. Walkup, "Using the mpitrace library," IBM Research Internal Report, Nov. 2011, see: http://www.hpcx.ac.uk/support/documentation/IBMdocuments/mpitrace .
[2] J. S. Vetter, and M. O. McCracken, "Statistical Scalability Analysis of Communication Operations in Distributed Applications," in *Proc. ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming (PPOPP)*, pp. 123-132, 2001.
[3] D. Skinner, "Performance monitoring of parallel scientific applications," LBNL Technical Report, LBNL-5503, 2005.
[4] W. Gropp, D. Gunter, and V. Taylor, "FPMPI: A Fine-tuning Performance Profiling Library For MPI," Poster presented at SC2001, November 2001, see: http://www.mcs.anl.gov/research/projects/fpmpi .
[5] D. Thomas, J.-P. Panziera, and J. Baron, "MPInside: a performance analysis and diagnostic tool for MPI applications," In *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering (WOSP/SIPEW '10)*, ACM, New York, NY, USA, pp. 79-86, 2001.
[6] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci, "A Portable Programming Interface for Performance Evaluation on Modern Processors," *The International Journal of High Performance Computing Applications*, Vol. 14, No. 3, pp. 189-204, 2000.
[7] Available: https://asc.llnl.gov/sequoia/benchmarks/

[8] Available: https://asc.llnl.gov/sequoia/benchmarks/IRS_summary_v1.0.pdf
[9] Available: https://asc.llnl.gov/sequoia/benchmarks/UMT_summary_v1.0.pdf
[10] S. Plimpton, "Fast Parallel Algorithms for Short-Range Molecular Dynamics", *J. Comp. Phys.*, Vol. 117, No. 1, pp. 1-19, 1995.
[11] M. Zhao, I. M. Held, S-J. Lin, and G.A. Vecchi, "iSimulations of Global Hurricane Climatology, Interannual Variability, and Response to Global Warming Using a 50km Resolution GCM," *J. Climate*, vol. 33, pp. 6653-6678, 2009.
[12] I. M. Held, and M. J. Suarez, "A proposal for the intercomparison of the dynamical cores of atmospheric general circulation models," *Bull. Amer. Meteor. Soc.*, Vol. 75, pp. 1825–1830, 1994.
[13] D. J. Kerbyson, and K. J. Barker, "A Performance Model of Direct Numerical Simulation for Analyzing Large-Scale Systems," In *Proceedings of the 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum (IPDPSW '11)*. IEEE Computer Society, Washington, DC, USA, pp. 1824-1830, 2011.
[14] D. A. Donzis, P. K. Yeung, and D. Pekurovsky, "Turbulence Simulations on O(10^4) Processors," In *TeraGrid 08,* Jun. 2008, see: http://computing.ornl.gov/workshops/scidac2010/presentations/d_donzis.pdf .
[15] L. Giraud, R. Guivarch, and J. Stein, "Parallel distributed FFT-based solvers for 3-D Poisson problems in Meso-scale atmospheric simulations," *Int. J. High Perform. Comput. Appl.*, vol. 15, pp. 36-46, 2001.
[16] C. A. Rohl, C. E. M. Strauss, K. M. S. Misura, and D. Baker, "Protein structure prediction using Rosetta," *Methods in Enzymology*, Vol. 383, pp. 66-93, 2004.
[17] U. Trottenberg, C. Oosterlee, and A. Schüller, *Multigrid*. San Diego, CA: Academic Press, 2001.
[18] R. D. Falgout. "An Introduction to Algebraic Multigrid," *Computing in Science and Engineering*, Vol. 8, no. 6, pp. 24-33, 2006.
[19] A. H. Baker, R. D. Falgout, T. V. Kolev, and U. M. Yang. "Scaling hypre's multigrid solvers to 100,000 cores." In M. W. Berry, K. A. Gallivan, E. Gallopoulos, A. Grama, B. Philippe, Y. Saad, and F. Saied, eds. *High-Performance Scientific Computing*. New York, NY: Springer, pp. 261-280, 2012.
[20] A. H. Baker, T. Gamblin, M. Schulz, and U. M. Yang. "Challenges of Scaling Algebraic Multigrid Across Modern Multicore Architectures," in *Proc. 25th IEEE International Parallel and Distributed Processing Symposium*, Anchorage, AK, pp. 275-286, 2011.
[21] H. Gahvari, A. H. Baker, M. Schulz, U. M. Yang, K. E. Jordan, and W. Gropp. "Modeling the Performance of an Algebraic Multigrid Cycle on HPC Platforms," in *Proc. 25th ACM International Conference on Supercomputing*, Tucson, AZ, pp. 172-181, 2011.
[22] E. Baysal, D. D. Kosloff, and J. W. C. Sherwood, "Reverse-time migration," *Geophysics*, Vol. 48, pp. 1514–1524, 1983.
[23] K. Rupnow, A. Rodrigues, K. Underwood, and K. Compton, "Scientific applications vs. SPEC-FP: a comparison of program behavior," In *Proceedings of the 20th annual international conference on Supercomputing* (ICS '06). ACM, New York, NY, USA, pp. 266-274, 2006.

Received March 26, 2012; accepted for publication September 14, 2012

Figures and tables

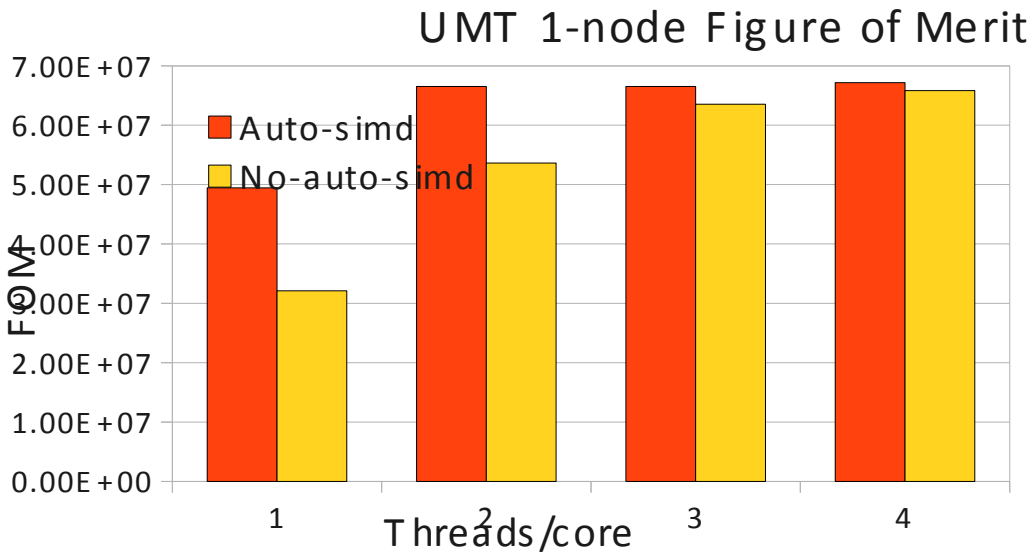

Figure 1: UMT Figure Of Merit (FOM) values for the Sequoia benchmark as a function of the number of hardware threads used and whether QPX was employed or not.

Metrics	1 thread /core MPI	2 threads /core MPI	4 threads /core MPI	1 thread /core Hybrid	2 threads /core Hybrid	4 threads /core Hybrid only
Weighted Gflop/s per node	2.543	3.922	5.729	2.456	4.015	6.077
Relative speedup	1.000	1.541	2.229	1.000	1.630	2.458
IPC per core	0.230	0.356	0.554	0.234	0.372	0.567
% Max Issue Rate	22.99	25.38	40.41	23.40	26.56	40.68
Relative total instruction count	1.000	1.006	1.071	1.000	0.973	0.986
Relative FPU instruction count	0.290	0.290	0.290	0.277	0.277	0.279
L1 D-cache hit %	93.10	88.77	84.22	93.09	90.40	87.95
L1P buffer hit %	0.84	0.85	0.59	0.84	1.11	1.22
L2 cache hit %	5.87	10.07	14.77	5.57	7.98	10.29
DDR traffic %	0.19	0.31	0.42	0.51	0.52	0.54
DDR loads (bytes/cycle/node)	0.61	1.16	2.17	1.05	1.57	2.68
DDR stores (bytes/cycle/node)	0.17	0.33	0.61	0.44	0.77	1.32
DDR total (bytes/cycle/node)	0.78	1.49	2.78	1.49	2.35	4.00

Table 1: Performance characteristics for LAMMPS

Application	Q/P ratio	comments
DNS3D	16.9	2048^3 problem, 8K cores, 64 ranks/node
FLASH	5.9	rtflame, 2K cores, 64 ranks/node, no MPI-IO
GFMC	10.5	c12-test, 2K cores, 8 ranks/node, some QPX
GPAW	8.5	Using early ESSL libraries, 32 ranks/node
GTC	11.2	M0180 problem, 2K cores, 16 ranks/node, 4thds
GFDL	14.2	16K cores, 8 ranks/node, 8 thds
LS3DF	8.1	Using GPFS ; performance sensitive to I/O
MILC	6.1	32^3 x 64 problem, 2K cores, 64 ranks/node, no QPX
NAMD	8.5	atpase, 2K cores, 32 ranks/node, no code tuning
NEK	7.4	medium case, 2K cores, 32 ranks/node, some QPX

Table 2: Relative speedup per node (peak scaling is ~15x per node) for Mira SoW applications.

Processes /Node	FPU Instr %	FXU Instr %	Weighted Gflops per node	% Max Issue Rate	L1 Cache hit rate	L2 Cache Hit Rate	DDR %	Ld (bytes /cycle)	St (bytes /cycle)	Total (bytes /cycle)	Time for 200 iterations
4	16.17	83.83	0.293	24.41	92.29	7.71	0	.007	.001	.009	1.92
8	15.23	84.77	0.544	24.08	92.73	7.27	0	.193	.012	.205	1.99
16	14.43	85.57	.885	20.69	93.17	5.98	0.85	2.28	.19	2.48	2.42
32	12.38	87.62	1.32	31.48	93.04	5.72	1.25	5.28	.38	5.66	3.01
64	11.41	88.59	1.83	47.62	91.72	7.14	1.13	8.97	.59	9.57	4.18

Table 3: Performance characteristics for ROSETTA (20.pdb, small input case).

Processes/Node	1	2	4	8	16	32	64
FPU Instr %	51.77	51.45	50.60	41.99	40.98	38.14	30.02
FXU Instr %	48.23	48.55	49.40	58.01	59.02	61.86	69.98
Intr/Cycle per Core	0.3272	0.3220	0.3296	0.3929	0.3843	0.5047	0.6325
%Max Issue Rate	32.72	32.20	32.96	39.29	38.43	31.22	44.26
Total Weighted Gflops/Node	1.383	2.702	5.427	10.735	20.949	25.056	24.648
L1 Cache Hit Rate	82.84	82.71	83.71	88.00	88.02	84.94	83.36
L1 P Buffer Hit Rate	9.77	9.82	9.92	10.79	10.70	6.30	2.83
L2 Cache Hit Rate	6.87	6.89	5.72	0.63	0.66	7.54	11.94
DDR %	0.53	0.58	0.65	0.58	0.62	1.22	1.87
LD (Byte/Cycle)	0.263	0.529	1.132	2.529	5.053	6.517	7.435
ST (Byte/Cycle)	0.100	0.196	0.407	0.957	1.956	2.535	3.085
Total (Byte/Cycle)	0.362	0.725	1.539	3.486	7.009	9.052	10.52
Total Run Time (Less Init and Final I/O)	111.33	56.77	28.4	14.38	7.52	6.02	5.68
Node-to-Node Speedup over BGP	2.6	2.6	2.7	5.3	10.1	12.6	13.4

Table 4: Performance characteristics for RTM.

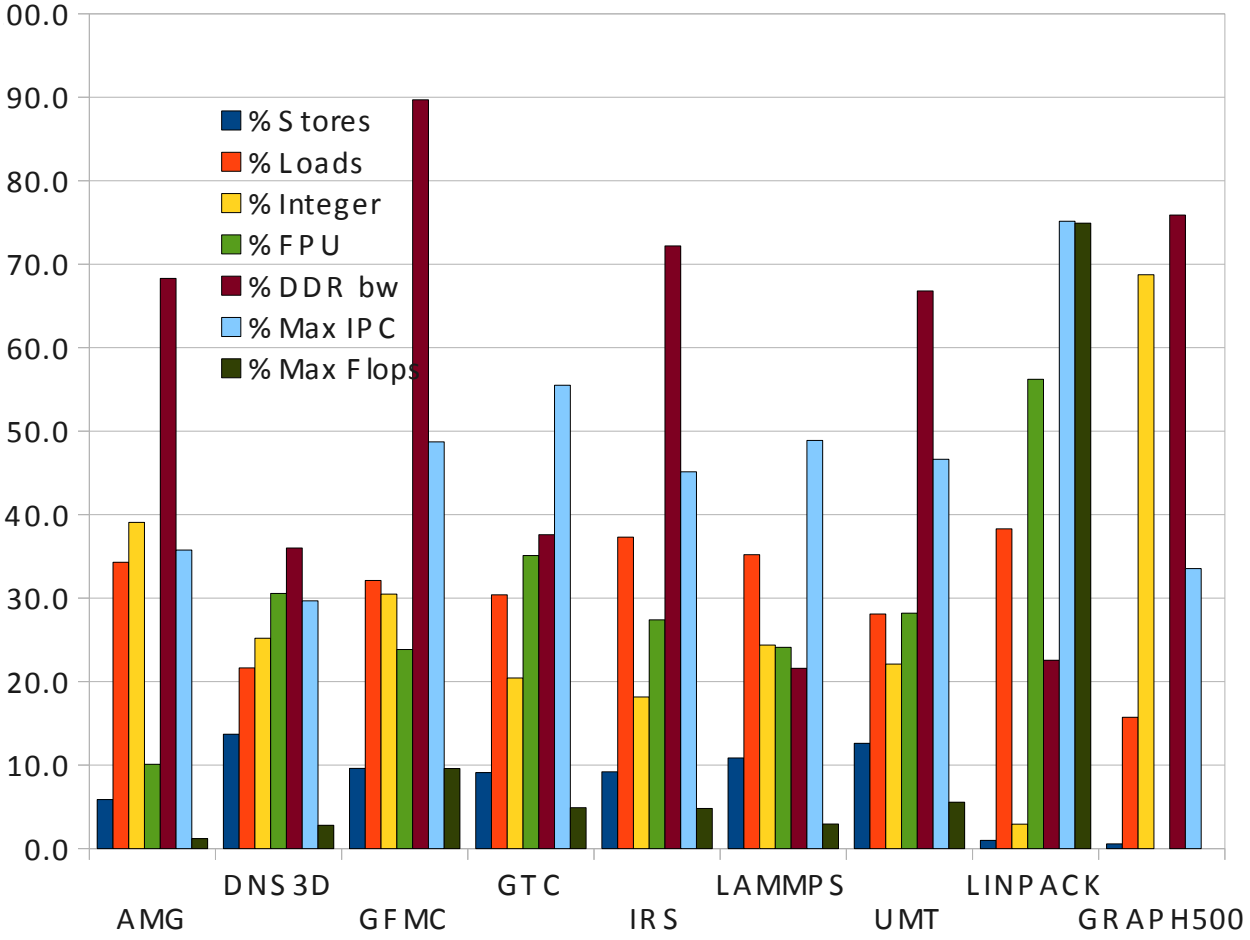

Figure 2: Applications can get a good fraction of the maximum issue rate out of every core, as long as multiple hardware threads are used effectively : average is ~45% max IPC.

Constantinos Evangelinos *IBM Research Division, One Rogers St., Cambridge, MA 02142-1203 (cevange@us.ibm.com)* Dr. Evangelinos is a Research Staff Member in the High Performance Computing Applications and Tools group of the Computational Science Center at the IBM T.J. Watson Research Center. He received a B.A. Degree in mathematics from Cambridge University and Sc.M. and Ph.D. degrees in applied mathematics from Brown University. He then worked as a Postdoc and a Research Scientist at the Massachusetts Institute of Technology until the end of 2010 before joining IBM. His current research interests are in parallel computing and performance modeling applied to computational methods for fluid flow (both CFD and GFD), data assimilation, adjoint techniques and automatic differentiation. He is a member of the IEEE Computer Society and the Association for Computing Machinery (ACM).

Robert E. Walkup *IBM T.J. Watson Research Center, 1101 Kitchawan Rd. Yorktown Heights, NY, 10598 (walkup@us.ibm.com)* Dr. Walkup is a Research Staff Member at the IBM T.J. Watson Research Center. He has extensive experience in both benchmarking and porting scientific and technical applications for the IBM pSeries and Blue Gene. He has been instrumental in tuning DOE codes for IBM platforms over the past few decades. His interests include code optimization, MPI and OpenMP performance issues, and performance bottlenecks. Bob obtained a Ph.D. degree in physics from Massachusetts Institute of Technology and has published papers involving both experimental and computational physics.

Vipin Sachdeva *IBM Research Division, One Rogers St., Cambridge, MA 02142-1203 (*vsachde@us.ibm.com*)* Vipin Sachdeva is a Senior Engineer in IBM Systems and Technology Group specializing in High-Performance Computing working as part of an extended team in the Computational Science Center at IBM T.J. Watson Research Center. He received his Master of Sciences from the University of New Mexico in 2005 with distinction. He is currently working on a Ph.D. in the area of computational science at Georgia Institute of Technology. He is a co-author of several papers in the area of applying high-performance computing to computational science areas including bioinformatics, computational fluid dynamics, reservoir simulation and seismic processing. His work has been recognized with awards including HPC Challenge Class 2 awards at Supercomputing 2007, IEEE Scale2011 Challenge and the IBM Outstanding Technology Achievement Award in 2011. His research interests include applying high-performance computing in analytics, and running commodity software efficiently on advanced supercomputers.

Kirk E. Jordan *IBM Research Division, One Rogers St., Cambridge, MA 02142-1203 (*kjordan@us.ibm.com*)* Dr. Jordan is Emerging Solutions Executive and Associate Program Director in the Computational Science Center at IBM T.J. Watson Research Center. He oversees development of applications for IBM's advanced computing architectures, investigates and develops concepts for new areas of growth involving high performance computing (HPC), and provides leadership in high-end computing and simulation in such areas as computational fluid dynamics, systems biology and high-end visualization. Dr. Jordan is a member of the IBM Academy of Technology. With a Ph.D. in Applied Mathematics, he held computational science positions at Exxon R&E, Argonne National Lab, Thinking Machines and Kendall Square Research before joining IBM in 1994. He holds leadership positions in the Society for Industrial

and Applied Mathematics (SIAM), including Chair of Computational Science and Engineering SIAG, member of the Committee on Science Policy and is a SIAM Fellow. He is on several boards including Chair of the Board of Trustees for the Mathematical Biosciences Institutes at The Ohio State University. He is associate editor of several international journals and Guest Editor for two recent issues of IBM's Journal for Research and Development. His research interests include developing techniques for the efficient use of advanced architecture computers for modeling and simulation physical and biological phenomena.

Hormozd Gahvari *Computer Science Department, University of Illinois at Urbana-Champaign, Urbana, IL, 61801 (*gahvari@illinois.edu*)* Hormozd Gahvari is a Ph.D. Candidate in Computer Science at the University of Illinois at Urbana-Champaign. His research focuses on high-performance computing applications and how they would have to change to adapt to future machines, especially exascale systems. For his dissertation, he is working on adjustments to algebraic multigrid to enable its scalability to next-generation parallel machines. He has spent three summers with the hypre team at Lawrence Livermore National Laboratory and with researchers at IBM working on these techniques.

I-Hsin Chung *IBM T.J. Watson Research Center, 1101 Kitchawan Rd. Yorktown Heights, NY, 10598 (ihchung@us.ibm.com)* Dr. Chung is a Research Staff Member at the IBM Thomas J. Watson Research Center. His research interests include performance tuning, performance analysis, and performance tools. His experience includes designing and developing performance tools on IBM platforms such as IBM Power Systems on AIX and Linux, and the Blue Gene systems. Prior to joining IBM Research, he received his Ph.D. in Computer Science from the University of Maryland, College Park in 2004.

Michael P. Perrone *IBM T.J. Watson Research Center, 1101 Kitchawan Rd. Yorktown Heights, NY, 10598 (mpp@us.ibm.com)* Dr. Perrone is an IBM Master Inventor with over 20 years of experience in computer science, including 8 years in high performance computing and 6 years in seismic imaging. His mission, as manager of IBM Research Multicore Computing, is to drive exascale architectural co-design by performing detailed exploration of a wide variety of customer computational workloads, identifying specific actionable hardware requirements and to drive these requirements into next generation hardwired system designs. This work has lead to deep insight into algorithmic optimization techniques, including the analysis of multicore strengths and weaknesses, and has led to the design of novel supercomputing algorithms. His team has won the Graph500 competition three times in a row, and has developed ground breaking algorithms for stream processing, including OPRA. His recent projects cover a variety of HPC workloads, including business analytics, graph algorithms, network intrusion detection, financial data stream processing, high-speed text indexing, seismic imaging, reservoir modeling, computational fluid dynamics, image processing, carbon sequestration and bioinformatics. His research includes algorithmic optimization for a variety of multicore processors, parallel computing and statistical machine learning. He received his PhD in Physics from Brown University.

Ligang Lu *IBM T.J. Watson Research Center, 1101 Kitchawan Rd. Yorktown Heights, NY, 10598 (*lul@us.ibm.com*)* Dr. Lu is a Research Staff Member in the Computational Science Center at IBM T.J. Watson Research Center, Yorktown Heights, NY. His research interests

include high performance computing in seismic imaging, data analytics, and video/imaging processing. He received a Ph.D. degree in Electrical Engineering from Rensselaer Polytechnic Institute (RPI), Troy, NY, where he received the Allan Dumont Prize for outstanding achievement. He was a past Chair of the IBM Research Signal Processing PIC and has served as a co-chair of VCIP 2008 as well as Technical/Program Committees for ICIP, VCIP, ICASSP, ICME, etc. He received an Outstanding Technical Innovation Award, a Patent Award, and numerous Invention Awards.

Lurng-Kuo *IBM T.J. Watson Research Center, 1101 Kitchawan Rd. Yorktown Heights, NY, 10598* Dr. Lurng-Kuo received an M.S. degree from National Chiao Tung University, Hsin-Chu, Taiwan, in 1987, and a Ph.D. degree in computer engineering and communications from the University of Maryland, College Park in 1993. He worked as a solutions architect at the IBM T.J. Watson Research Center, where he specialized in high performance computing and multicore computing solutions, covering multiple industrial workloads including Financial Services Modeling and High Frequency Trading, Digital Media, Seismic, Stream Processing, and Low Latency Messaging. He previously worked as a program manager for IBM's Blue Gene (BG/L) System. In addition, he served as an adjunct professor at Fordham University, Columbia University, and the Polytechnic Institute of New York University. He also served as visiting professor at King Abdullah University of Science and Technology (KAUST) in Saudi Arabia, where he taught high performance computing and parallel programming paradigms. Kurng-Kuo passed away in 2012.

Karen Magerlein *IBM T.J. Watson Research Center, 1101 Kitchawan Rd. Yorktown Heights, NY, 10598 (*kmager@us.ibm.com*)* Karen Magerlein is an advisory engineer with the Multicore Computing Department at the Thomas J. Watson Research Center. She received a B.S. degree in computer science from Duke University in 1980 and joined IBM's Research Division the same year. She worked on programming tools, binary image manipulation, the JPEG compression standard, and image processing software for digital library applications before adopting her current focus on application programming for multicore computers. She completed her M.S. degree in computer science at Columbia University in 1995, and is a co-author on over 30 articles and patents.