A Performance Counter Based Workload Characterization on Blue Gene/P

Karthik Ganesan Lizy John* Valentina Salapura† James Sexton†*
** Univ. of Texas at Austin † IBM T. J. Watson Research Labs*

***Abstract*—IBM's Blue Gene/P, the second generation of the Blue Gene supercomputer is designed with a Universal Performance Counter (UPC) Unit at each node capable of monitoring 256 events concurrently [1], unlike many microprocessors that provide only a few performance counters. In this paper we demonstrate the efficacy of the interface library that we have developed, taking advantage of the UPC unit, enabling users to effortlessly instrument applications and get a profound insight into its execution on the Blue Gene/P system which could scale in thousands of nodes. The interface library allows the user to monitor about 512 performance related events out of a total of 1024 possible events and aggregate the data collected at different nodes and compute meaningful metrics through data mining.**

Using the developed interface, we instrumented the NAS parallel benchmarks and collected the performance counter data. We studied the MFLOPS, L3-DDR Traffic and the dynamic instruction mix based on the counters in the FPU and the cache hierarchy for different compiler optimizations, modes of operations of the system and different L3, L2 configurations for the NAS benchmarks. Our analysis identifies that compiler optimization O5 along with "-qarch440d", which uses the architectural information of the chip in optimization, is very effective in incorporating a lot of SIMD instructions and results in the most efficient execution of the benchmarks. The experiments on the L3 size indicate that an L3 size of 4MB is optimal for the NAS benchmarks and they do not benefit by increasing it further. Also, the virtual node mode of operation of the Blue Gene/P system is very effective and yields superior performance for the selected benchmarks taking advantage of the chip multiprocessor architecture of the quad-core HPC chip.

I. Introduction

With petascale computing just over the horizon, the supercomputing community is becoming increasingly focused on developing strategies to take advantage of this new level of computing power. Petascale computing will eventually impact all scientific and engineering applications, but to reach its full potential, the problems of both hardware and software must be addressed. Among the different HPC providers, the IBM's Blue Gene systems occupy the #1 (LLNL Blue Gene/L) and a total of 4 of the top 10 positions in the TOP500 supercomputer list announced in November 2007. On 26th June, 2007, IBM Corporation unveiled Blue Gene/P [1], the second generation of the Blue Gene supercomputer, designed to run with a sustained performance of one petaflops and claims to be configured to reach an excess of three petaflops. Figure 1 gives an overview of the system architecture of the petaflop machine. Blue Gene/P is also claimed to be very energy efficient as in [2] when compared to all previous supercomputers, accomplished by using many small, low-power chips connected through five specialized networks.

To achieve petaflops performance at the cluster level, a performance estimate of 13.6 GFLOPS should be achieved at the node level. Performance at the node level can be increased substantially based on application optimization and tuning [3] [4] to the underlying architecture, which requires an insight into the execution of the corresponding application on the node.

Though the Blue Gene/P node is based on the traditional PowerPC architecture, there is enough innovation in the chip to demand new work in optimizing the application code run on it. An understanding of the memory hierarchy, the new floating point operations like the SIMD add-sub, SIMD floating point multiply-add and SIMD multiply is necessary to get a near peak performance at the node level.

One way of getting an insight into execution is through detailed software instrumentation, but it could perturb the application execution resulting in inaccurate measurements. But, modern processors include performance counters integrated into the hardware aiding in performing non-intrusive application monitoring in realtime. The hardware performance counters allow counting of performance related micro-architectural events in the processor [5] [6] [7] [8], enabling new ways to monitor and analyze performance. They fill the gap between detailed full system simulation and software instrumentation because they have lesser perturbation and can provide detailed information about the performance of the processor and the memory system.The novelty of this paper lies in the interface library that brings out the efficacy of a new performance monitoring paradigm (a PM Unit on the chip) for high performance computing, capable of monitoring over thousand concurrent events using the UPC unit on the Blue Gene/P.

The BlueGene/P performance monitoring unit provides more advanced features that are best taken advantage of with hardware-specific functions. Global accessibility of configuration and count values allowing a single monitoring thread executing as part of a system service, or as part of an application, and read the performance counters. This, along with a feature called thresholding (Raising an interrupt when specific counters reach corresponding thresholds) dynamically provides feedback to the various system optimization tasks like data placements, thread assignment and communication patterns.

0190-3918 2008
U.S. Government Work Not Protected by U.S. Copyright
DOI 10.1109/ICPP.2008.57

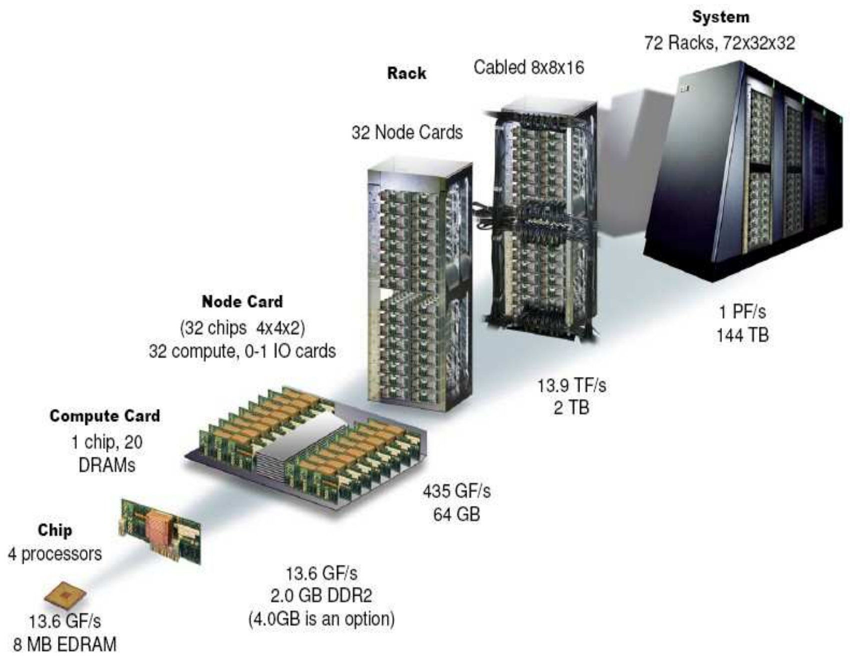


Fig. 1. Overview of Blue Gene/P, the first PetaFlop System

The developed interface library takes advantage of all the new features of the Performance Monitoring Unit on the chip and mainly helps in providing an isolated analysis of critical code regions through a set of start/stop function calls. The various uses of such an interface could be like (1) the system services can take advantage of these functions to disable monitoring system code perturbing the results of the application, (2) look at the change in counter values of workload specific events for defined program regions that have significant effect on the execution time (say, specific to data layout, inter-process communication etc.)

II. Related Work

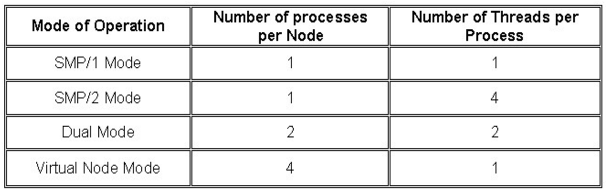
Considerable work has been done in the last few years using the performance counters [9] [10] to investigate the behavior of applications and identify performance bottlenecks [11] resulting from overly stressed micro-architecture components. Intel's Itanium processors [12] has monitoring of counters based on instruction opcodes, address range of the accessed data and even the privilege levels of the instructions.

In Blue Gene/L [13], the large number of available events in the CPU and the complex mapping of events onto possible physical counters is handled through a user-level API viz., BGLperfctr. This API contains a set of predefined mnemonics for each of the events and provides the user with an abstraction of 52 counters, unifying the UPC and FPU counters and extending them to 64-bit counters. But, BGLperfctr has the drawback of being system-specific. Application code that uses hardware counter information is normally nonportable. The Performance counter API (PAPI) [14] addresses the problem of high cost of maintaining such codes by providing a platform independent interface to control and read hardware counters on a large variety of platforms prevalent in HPC.

There have been a lot of analysis tools developed based on the performance counters [15] [16] [17] [18] for understanding and optimizing application performance. Most of these tools rely on software instrumentation of the application to obtain the values of the hardware performance counters through kernel services. But, most of the analysis techniques are for analyzing a single process data and are difficult to scale when the number of processors run into hundreds of thousands as explained by [19]. When users execute multiple experiments, this adds an extra dimension of complexity to this picture. These problems are addressed in this paper by the use of the dedicated UPC unit and the proposed interface library.

III. Blue Gene/P System Overview

The Blue Gene/P computer is a scalable, distributed-memory system consisting of up to 73,728 nodes. Each node is built around a single compute ASIC and 2 or 4 GB DDR2 DRAMs of main store. The Blue Gene/P compute ASIC is a highly integrated System-on-a-Chip (SoC) chip multiprocessor (CMP) based on PowerPC 450 processor cores. The chip contains four processor cores, each with private L1 instruction and data caches. Each core is coupled to a dual-pipeline SIMD floating point unit and to a private, prefetching L2 cache. In addition, the chip integrates a large, shared L3 cache, two memory controllers, five network controllers, and a performance monitor, as illustrated in Figure 2. The PowerPC 450 microprocessor is a high-performance, out-of-order industry-standard PowerPC processor originally targeted at high-end embedded systems. The processor



Mode of Operation	Number of processes per Node	Number of Threads per Process
SMP/1 Mode	1	1
SMP/2 Mode	1	4
Dual Mode	2	2
Virtual Node Mode	4	1

Fig. 3. Modes of Operations of a Blue Gene/P Node

supports 2-way superscalar instruction execution with a seven stage pipelined microarchitecture. The processor cores include highly associative first level instruction and data caches with a capacity of 32KB each.

A dual-pipeline SIMD floating point unit is attached to each processor core. The floating point unit pairs two floating-point register files and two execution pipes. Both primary and secondary register files are independently addressable, but they can be jointly accessed by SIMD instructions. SIMD execution exploits the data-level parallelism often present in high-performance computing workloads. SIMD reduces the number of instructions necessary to fetch, issue and complete, while increasing the number of operations completed.

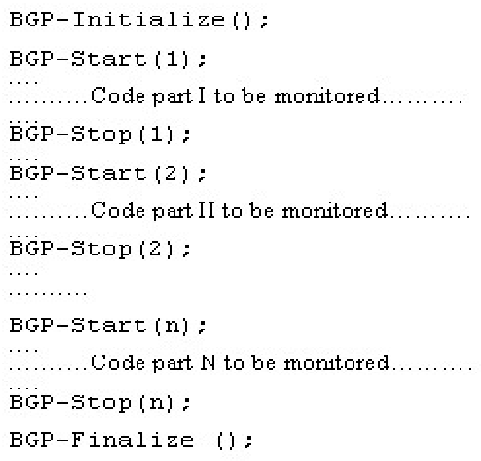
Blue Gene/P provides five dedicated communication networks: the torus network, the collective network, the barrier network, 10Gb/s Ethernet, and IEEE1149.1 (JTAG). The network interfaces are integrated on the same chip as the processing units. The main network is the torus, which provides high performance data communication to nearest neighbor nodes in a 3D mesh configuration (with ends wrapped around) with low latency and high throughput. The collective network supports efficient collective operations, such as broadcast and reduction.

A Blue Gene Node works in 4 operating modes viz., SMP/1 thread, SMP/4 threads, Dual Mode and the Virtual Node Mode. The table in figure 3 gives the details about the number of processes and threads per node in the different modes of operation.

A. Universal Performance Counter Unit

The Universal Performance Counters Unit (UPC) on the chip allows for monitoring a set of selected hardware events generated by a variety of on-chip event sources such as the processors, the floating point units, the snoop filters, the L2-caches, the L3-cache, and the network interfaces. For example, the events in the FPU include the number of single additions/subtractions, single multiplications, single divisions, SIMD addition/subtractions, SIMD multiplications, SIMD divisions, SIMD & single floating point multiply additions etc.

The UPC unit contains 256 64-bit counters. Each counter is configured with 4 configuration bits located in the configuration registers which determines the mode of operation of the counters. All counters and all configuration registers in the UPC module are mapped into the memory address space providing memory-mapped access to all counters and configuration register.



```
BGP-Initialize();
BGP-Start(1);
....
.........Code part I to be monitored.........
....
BGP-Stop(1);
....
BGP-Start(2);
....
.........Code part II to be monitored.........
....
BGP-Stop(2);
....
.........
BGP-Start(n);
....
.........Code part N to be monitored.........
....
BGP-Stop(n);
BGP-Finalize ();
```

Fig. 4. Using the Interface to instrument Code Snippets

The UPC unit can be programmed to count one of four set of events in four counter modes 0, 1, 2 and 3, each having 256 events. Usually, the whole UPC unit is set to a particular mode, which decides the purpose for which each of the counter is used. The counter event and interrupt setting is taken from the configuration register for each of the counter. Two bits for counter events determine what signaling on the selected counter input represents a count-event. The user has the choice between level-sensitive events (low-/or high-active) and edge-sensitive signaling (low-high-/or high-low transition).

The encoding of counter events bits is as follows:

- 00 - Use the high level sensitive mode (BGP_UPC_CFG_LEVEL_HIGH[1])
- 01 - Use the low-high edge sensitive mode (BGP_UPC_CFG_EDGE_RISE)
- 10 - Use the high-low edge sensitive mode (BGP_UPC_CFG_EDGE_FALL)
- 11 - Use the low level sensitive mode (BGP_UPC_CFG_LEVEL_LOW)

There is an interrupt enable bit in the configuration register that enables interrupt for this counter if it matches the threshold value. More details about the UPC unit can be found in [1]

IV. Performance Counter Interface Description

Although there are 1024 possible events that can be monitored in total, 256 of them can be measured in one run by a UPC unit. The interface that is developed to access the performance counters was designed in such a way that 512 events can be monitored in one single run

[1]BGP - Blue Gene/P, UPC - Universal Performance Counter Unit

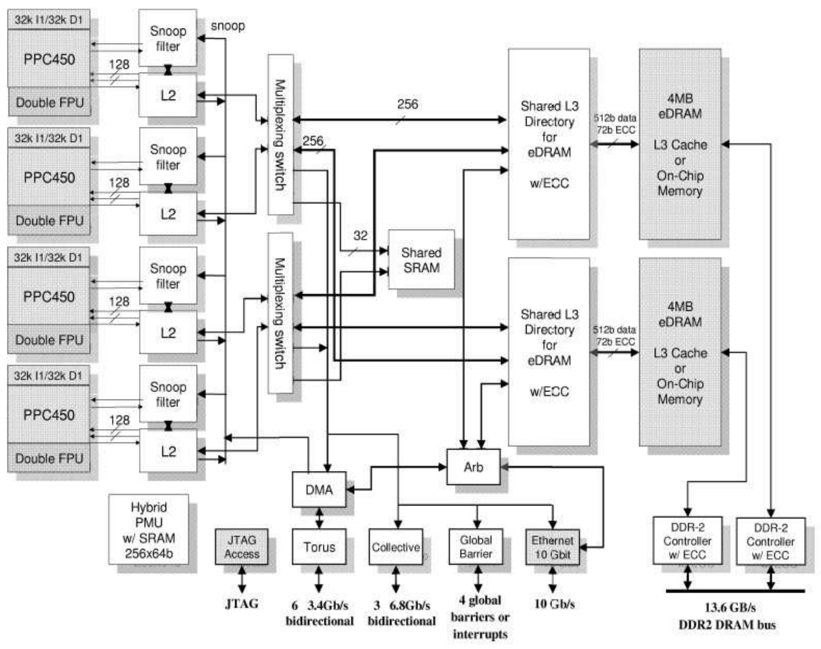


Fig. 2. Blue Gene/P Node Architecture

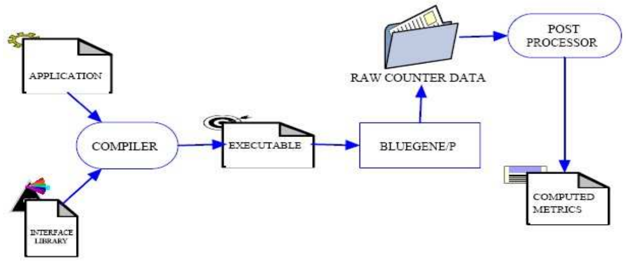


Fig. 5. Collecting performance information of an application using the Library

by monitoring the first 256 events in the even numbered node cards and the second 256 events in the odd numbered node cards. The developed interface library provides 4 basic function calls: BGP_Initialize(), BGP_Start(), BGP_Stop() and BGP_Finalize().

The BGP_Initialize() function of the interface selects one of the 4 modes in which the UPC unit should function, selecting the set of 256 events of the total 1024 possible events and initializes all the counters of the UPC unit. The code parts that needs to be monitored are preceded and succeeded by the BGP_Start() and the BGP_Stop() functions and each such pair of Start and Stop functions constitute a set. The functions BGP_Start(set #) and BGP_Stop(set #) start and stop monitoring all the 256 counters for that particular part of the code and stores it associated with the set number that is specified. The function BGP_Finalize() dumps the difference in counter data between the corresponding pairs of BGP_Start() and the BGP_Stop() functions of all the sets into a binary file at each node. A post processing tool was developed for data mining the dumped binary files. A few sanity checks were performed to find the overhead associated with the interface compared to the normal application execution.

Though the application suffers a longer execution time when instrumented, the overhead incurred is mainly in printing the data into the binary files. This in no way affects the accuracy of the counter data because the monitoring of counters is stopped after the BGP_Stop() function and the time taken after that in printing just increases the execution time. To know the exact execution time of the run, a counter to count the clock cycles was used and it's data is recorded in the stop() function. It was checked against the readings of the Time_Base_Register of the chip to find the overhead associated with the interface. The total overhead encountered in initializing the UPC unit, the start() and the stop() functions were measured to be 196 machine cycles. This way, the interface is used to instrument simple code snippets as illustrated in figure 4. It is to be noted that the overhead mentioned above also includes the initialization process. To monitor different parts of the application, the initialize is called once and multiple start() and stop() calls are added. Thus the overhead incurred should be far less than 196 cycles for each of the start, stop call pairs. But, when compared to the number of machine cycles taken by the scientific applications of today, this number is negligible. Thus, the application performance is affected to the least due to the presence of the dedicated UPC unit in the chip.

The interface is also integrated with the MPI library for easy instrumentation of MPI based applications and is provided as one library. The functions BGP_Initialize() & BGP_Start() are added to MPI_Init() and the functions BGP_Stop() & BGP_Finalize() functions are added to MPI_Finalize() and these new definitions of the functions in the new MPI library overrides any previous versions. Linking this library with any MPI based application during compile time gets the application instrumented. To instrument sequential applications, the (Initialize, Start) and (Stop, Finalize) function calls are added to the start and the end of the main function respectively and the application is compiled by linking it with the interface library. After getting the raw counter data dumped into binary files, the post processing tools provided are used to compute useful metrics. Figure 5 shows the process of collecting the performance data of an application using the library.

The post processing tools that are provided read all the files dumped by each node and computes the statistics viz., minimum, maximum and the arithmetic mean of each of the 512 counters. The data is checked based on the number of records and the length of each record and also for the range of values in the different counter readings to eliminate possible errors in the data. Then, based on the counter data from the 512 counters, different user defined metrics can be computed. For example, the performance of the application is computed in terms of MFLOPS based on the data of all the floating point counters like the counter for FPAdd-Sub, FPMult, FPDiv, FPFMA, FPSIMDAdd-Sub, and FPSIMDFMA. Similarly, a metric for the traffic between the L3 and the DDR (DDR Bandwidth) is computed based on the different counters associated with L3 and DDR.

The relevant metrics selected by the user are printed as a record for each application into .csv files, which can be used with Microsoft Excel or Open office calc for further investigations. Options can also be specified to print the statistics of all the 512 counters or print every counter value read in every node into one massive .csv file along with the statistics.

The number of events being as large as 1024, the interface can be used to perform a lot of useful performance related research. A few example usage of the Developed Interface:

- Analyze the dynamic instruction profile of the applications for different compiler optimizations and infer their effectiveness.
- Getting insights into the execution of applications and tune applications for better performance based on the counter data.
- Monitor the counters for L3 Cache & DDR by varying the L3 cache parameters to see their effect on the L3-DDR traffic.
- Vary the prefetching amount at L2 level and identify the most effective operation mode for modern workloads based on the counters for L2.
- Getting a feedback about the effectiveness of various architectural enhancements and help in improving future designs.

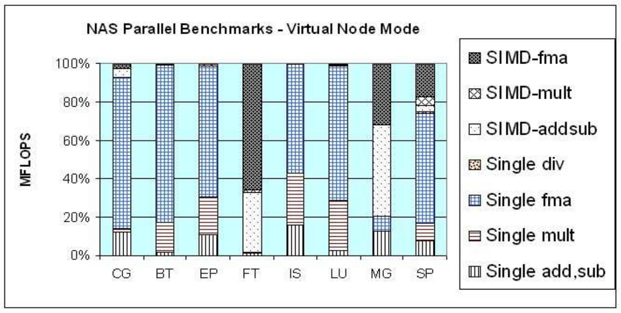


Fig. 6. Dynamic FP Instruction Profile of the NAS Parallel Benchmarks

It is to be noted that this interface library, along with the UPC unit on the chip, addresses the scalability problems of the single process performance monitoring tools of today. For systems like the Blue Gene/P, the number of nodes will scale into thousands and these problems have to be addressed.

V. Dynamic Instruction Profile of the NAS parallel benchmarks

The class C NAS parallel benchmarks [20] were instrumented and the counter data for the different floating point operations were observed by running it with 128 processes for MultiGrid (MG), 3-D FFT PDE (FT), Embarrassingly parallel (EP), Conjugate Gradient (CG), Integer Sort (IS), LU Solver (LU) on 32 nodes in the Virtual Node Operation mode of the BG/P node. 121 processes were used for Penta-diagonal Solver (SP) and Block Tri-diagonal Solver (BT) because the number of processes are supposed to be a square number for the two benchmarks. In the process of calculating the MFLOP rate, the floating point dynamic instruction mix of the benchmarks on the node was studied. Figure 6 shows the distribution of the Floating Point operations into single add-sub, single mult, single FMA, single div, SIMD add-sub, SIMD FMA and SIMD mult. For the NAS benchmarks viz., Multigrid (MG) and 3-D FFT PDE (FT), it is evident that they exploit the SIMD add-sub and SIMD FMA instructions of the chip extensively. And for the remaining benchmarks, viz., Embarrassingly Parallel (EP), Conjugate Gradient (CG), Integer Sort (IS), LU Solver (LU), Penta-diagonal Solver (SP) and Block Tri-diagonal Solver (BT), it can be observed that the Floating Point single multiply-add instruction has been used largely, bringing out the significance of having the single and the SIMD FMA units on a HPC chip.

VI. Performance for various levels of Compiler Optimizations

The counters of the floating point SIMD units were recorded for the usage of different compiler optimization flags viz., -qarch, -O with -qstrict, -O3, -O4 and -O5. The -O flag is the default optimization level, in which the common subexpression elimination, code motion, dead code elimination, instruction reordering and branch straightening are performed on the code. The -qstrict option makes sure that

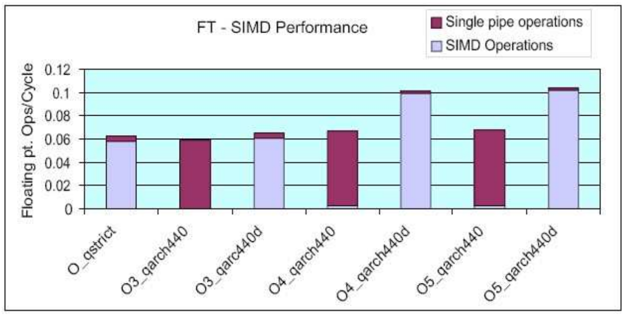


Fig. 7. FT - SIMD instructions for different compiler optimizations

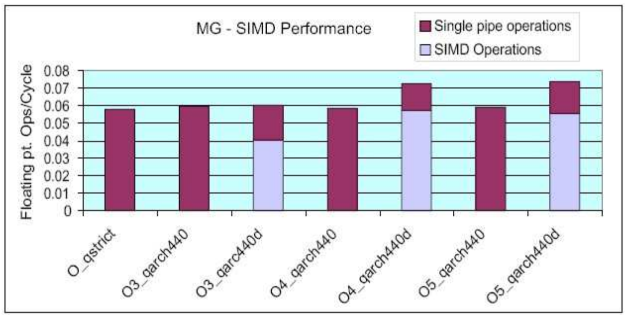


Fig. 8. MG - SIMD instructions for different compiler optimizations

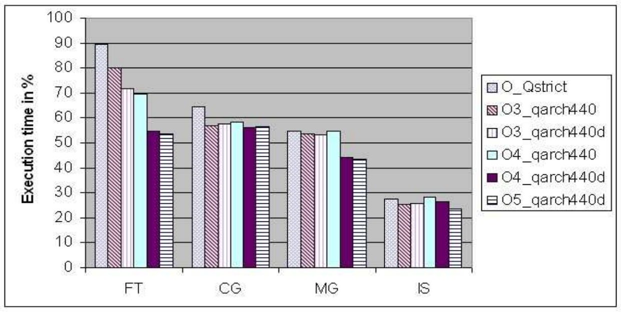


Fig. 9. Variation in Execution time with different compiler optimizations on NAS Parallel Benchmarks

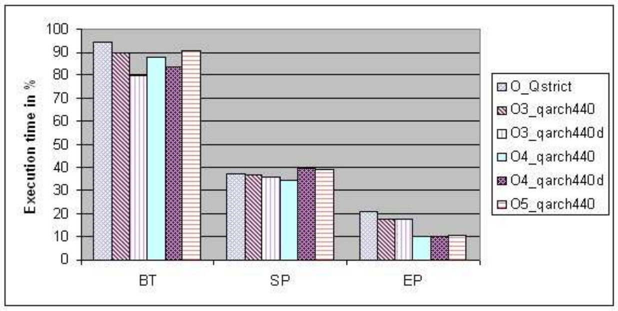


Fig. 10. Variation in Execution time with different compiler optimizations on NAS Parallel Benchmarks

the optimizations do not change the semantics of the program.

The -O3 optimization level does all the optimizations done at the O2 level along with strength reduction, more aggressive code motion and scheduling. The -O4 optimization level includes the -qarch, -qtune, -qcache and -qhot along with O3 optimization. The -qtune flag uses processor specific information like the cache size, pipelining details etc to tune the application to give the best performance on the processor. The -qarch introduces processor specific instructions that can improve performance, but at the cost of producing an executable which will run only on that specific processor. The -qcache option is used to specify the cache configuration and the -qhot option does a lot of expensive optimizations on the loops. The -O5 level of optimization does a lot of inter-procedural analysis in optimization.

The -qarch440d which is used along with O3, O4 and O5, which uses more specific details about the processor used in the Blue Gene/P node for optimizing. The counters for the floating point SIMD units were used along with the other floating Point unit counters to find the proportion of the SIMD instructions incorporated into the code due to the usage of -qarch440 compiler flag. This flag instructs the compiler to identify and extract the portions of code with data parallelism, which can be executed on the SIMD floating point unit operating on two sets of data in parallel.

Figures 7 & 8 illustrate this phenomenon for the FFT (FT) and the MultiGrid (MG) benchmarks, respectively, when -qarch440d flag is used over the normal optimizations. We observe that these applications can significantly benefit from extracting data parallelism, and deploying SIMD FPU. In addition to SIMD FPU operations, the SIMD compiler option introduced a lot of quadloads and quadstores in the instruction mix, further reducing the number of required double and single store operations.

As a result of extracting data parallelism and more aggressive compiler optimizations, the overall execution time is reduced. The execution time in cycles for the different compiler optimizations were also recorded using the CYCLE_COUNT counter to know the effectiveness of the different compiler optimizations. We list the execution times for NAS benchmark applications in Figures 9 and 10. Some applications, like FT and EP, show data parallelism which can be extracted by compiler efficiently, and the most effective compiler optimization reduces execution time up to 60%, compared to the baseline execution. Other applications benefit lesser from compiler optimizations. Please note that the results reported are preliminary data, and that we expect the results to improve with ongoing compiler development. We will report the updated results for the final paper version.

VII. Varying L3 Cache Parameters

By using the performance counters that give insights about the L3-DDR Traffic, a metric for the L3-DDR traffic was formulated. This aggregate metric was recorded for various L3 cache sizes. The L3 cache size was varied from 0MB to 8MB

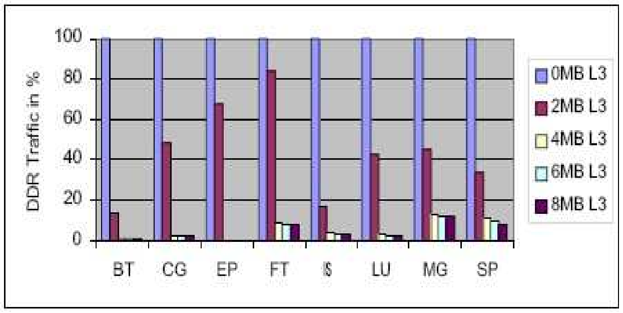


Fig. 11. Varying L3 size for class C NAS Parallel Benchmarks

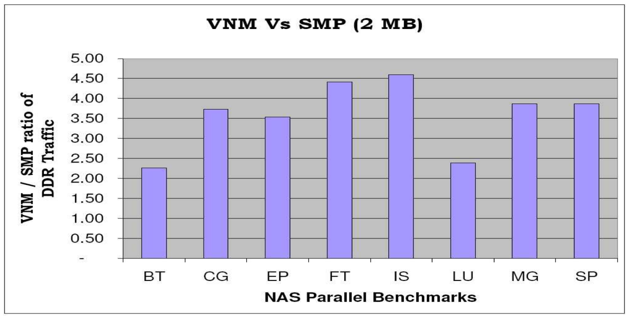


Fig. 12. Ratio of DDR traffic when using 4 processors on a chip to a single processor on a chip. VNM - Virtual Node Mode, uses 4 processors. SMP - Uses only 1 processor on a node

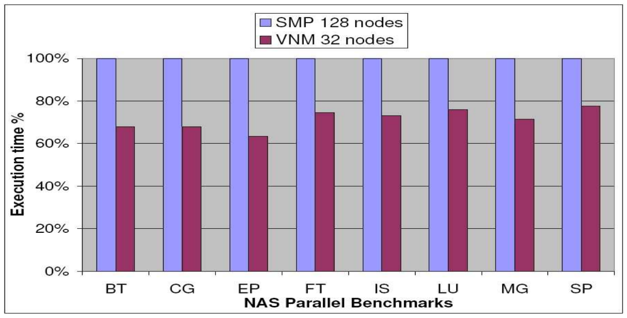


Fig. 13. Increase in execution time per node when using 4 processors on a chip to a single processor on a chip. VNM - Virtual Node Mode, uses 4 processors. SMP - Uses only 1 processor on a node

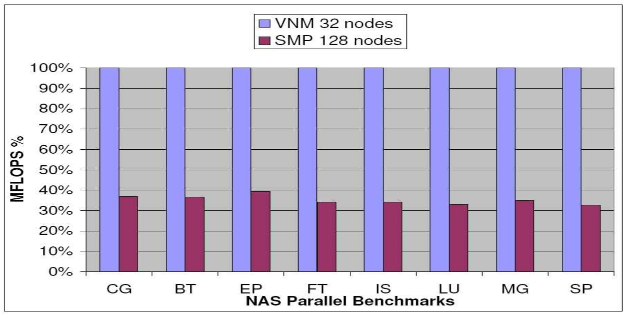


Fig. 14. Increase in MFLOPS per chip when using all four processors on a chip to using a single processor.

in steps of 2MB. The L3 cache size of 0MB is equivalent to a system without any on-chip L3 cache, but only with private L1 caches. In this configuration, all memory requests have to be satisfied from the off-chip DDR main memory. The traffic to DDR for various L3 cache sizes is plotted in fig.11. It is clearly evident that the Class-C NAS parallel benchmarks benefit a lot when the cache is increased from 0MB to 2MB and then from 2MB to 4MB. Even adding a small L3 cache of 2MB reduces significantly the number of requests to the DDR memory. The misses are reduced to nearly 10% of the total accesses when the cache size is set to be 4MB, where the application and data footprint fits in the on-chip L3 cache. But, on the other hand, the benefit is very less when the cache size is increased beyond 4MB. But, it is to be noted that today's real world scientific applications span a bigger footprint than most of the NAS benchmarks.

VIII. Using four processors on a chip

To determine the impact of using four processors on a chip, and sharing resources between the four processors, we compare the execution of the same applications on all four processor cores and on a single processor core, and measure execution time, number of requests to the off-chip DDR memory, and achieved performance for the two configurations. We observed that even while per-processor performance is reduced when using all four processors available on a chip, using all available processors on a chip reduces overall execution time of applications with a sharp increase in the resource utilization.

More specifically, we compare the execution of the class C NAS parallel benchmarks with 128 processes on 32 nodes in Virtual node mode to same 128 processes on 128 nodes in SMP/1 mode. As listed in Figure 3, Virtual node mode is the mode in which there are 4 MPI processes that run in each node, while SMP/1 mode is the mode where one process with only one thread is run in each node. To perform a fair comparison, in SMP/1 mode, where one process uses the whole compute node, we reduced the L3 cache size to 2 MB per node using the svchost options while booting a node.

From a memory perspective, we observed the increase in traffic to DDR memory when using all four processors on a chip instead of only one processor. It can be observed from Figure 12, that there is a 3 times increase in the traffic to DDR RAM in average. While for most applications increase in traffic to DDR RAM when using four processors instead of one is less than 4 times, only for FT and IS applications the number of requests increased more than four times due to memory port contention and cache interference.

While using multiple processors on a chip increases traffic to off-chip memory, the execution time on a single processor will also increase due to various on-chip resource sharing, and inter-processor interference. The increase in the execution time of applications due to sharing on-chip resources when using all four processors (results for 128 processes on 32 nodes VNM) to using only one processor (results for 128 processes on 128 nodes SMP/1) was found to be just 30%, as shown in Figure 13. This confirms the effectiveness of the chip multiprocessor architecture of Blue Gene/P. If the processes that were executed concurrently were from different types of applications with heterogenous characteristics like memory bound, I/O bound, processor bound etc, the performance of the Virtual Node Mode can be expected to be far better than 4 copies of the same process conflicting for the same resources at any point of execution.

There is a a significant increase in the utilization of the resources in the chip when all the four processors are used. This phenomenon is illustrated in Figure 14, where delivered MFLOPS per chip is about 2.5 times higher when using all available processors on a chip instead only a single processor.

IX. Conclusion

Usage of Performance counters has become widespread in evaluating the performance of applications and in unveiling the opportunities to improve performance. In this paper, we have brought out the utility of the developed performance counter interface library and the presence of a dedicated UPC unit on a High Performance Computing chip. We aimed at providing the performance analysis tools as early as possible, so that applications can be tuned for Blue Gene/P machine, which is in its final stages of design and assembly. As an example, we have profiled the NAS parallel benchmarks using the Interface library and analyzed the results.

The dynamic instruction profile of the benchmarks were observed with various levels of compiler optimizations on Blue Gene/P, to provide a feed back to the ongoing compiler development. The Virtual Node operating Mode and the SMP modes were studied, bringing out the effectiveness of the Virtual Node Mode of Operation exploiting Blue Gene's chip multiprocessor architecture. It was observed that a L3 size of 4 MB looks optimal for the NAS benchmarks.

We look forward to use the developed interface to evaluate the performance of diverse applications by varying the hardware parameters like prefetch amount in L3, prefetch amount in L2 etc and conclude on the optimal values for the modern workloads. The Blue Gene/P chip has a lot of architectural enhancements over Blue Gene/L and new features that needs to be evaluated with the help of the Interface. We are also curious to see the performance of using OpenMP with MPI on the multicore nodes of this parallel machine. Since the number of performance counter events being 1024, there is a lot of scope for research based on these counters.

It is to be noted that the applications can be instrumented without any need for changing the code with the help of the new MPI library that is provided. Also, the presence of the dedicated UPC unit along with this interface library makes performance monitoring possible when the number of nodes scale into thousands.

X. Acknowledgements

This work has been supported and partially funded by Argonne National Laboratory and Lawrence Livermore National Laboratory on behalf of the United States Department of Energy under Subcontract No. B554331. I would also like to acknowledge the University of Texas at Austin for the facilities provided through the laboratory for computer architecture.

References

[1] Valentina Salapura Karthik Ganesan Alan Gara Michael Gschwind James C. Sexton and Robert E. Walkup. Next-generation performance counters: Monitoring over thousand concurrent events.

[2] http://en.wikipedia.org/wiki/blue_gene. *Wikipedia.*

[3] N. Garner K. London S. Browne, J. Dongarra and P. Mucci. A scalable cross-platform infrastructure for application performance tuning using hardware counters. *Proc. SC2000: High Performance Networking and Computing Conf*, 2000.

[4] S. Moore P. Mucci K. Seymour K. London, J. Dongarra and T. Spencer. Enduser tools for application performance analysis using hardware counters. *Proc. International Conference on Parallel and Distributed Computing Systems,*, 2001.

[5] IBM Corporation. The power4 processor introduction and tuning guide. *http://www.redbooks.ibm.com/pubs/pdfs/redbooks/sg247041.pdf.*

[6] AMD. Athlon processor x86 code optimization guide. *http://www.freewebs.com/gkofwarf/amdperfmon.htm.*

[7] Intel Corporation. Intel itanium 2 reference manual for software development and optimization. *http://www.intel.com/design/itanium2/manuals/251110.htm.*

[8] B. Sprunt. Pentium 4 performance monitoring features. *IEEE Micro, 22(4):72-82,*, August 2002.

[9] L. DeRose. The hardware performance monitor toolkit. *In Proceedings of Euro-Par, pages 122131*, August 2001.

[10] Robert W. Wisniewski Reza Azimi, Michael Stumm. Online performance analysis by statistical sampling of microprocessor performance counters. *Proceedings of the 19th annual international conference on Supercomputing*, 2005.

[11] B. Buck and J. K. Hollingsworth. Using hardware performance monitors to isolate memory bottlenecks. *Proceedings of Supercomputing02*, November 2002.

[12] Intel Corporation. Intel ia-64 architecture software developer's manual, volume 4: Itanium processor programmer's guide,. 2000.

[13] Luiz DeRose Pedro Mindlin, Jose R. Brunheroto and Jose E. Moreira. Obtaining hardware performance metrics for the bluegene/l supercomputer. *Lecture Notes in Computer Science - Springer Berlin / Heidelberg, Volume 2790/2004*, 2003.

[14] S. Moore P. Mucci D. Terpstra H. You J. Dongarra, K. London and M. Zhou. Experiences and lessons learned with a portable interface to hardware performance counters. *Proceedings of PADTAD Workshop, IPDPS Meeting*.

[15] Research Centre Juelich GmbH. The performance counter library: A common interface to access hardware performance counters on microprocessors. *http://www.fzjuelich.de/zam/PCL/.*

[16] J. M. May. Software for multiplexing hardware performance counters in multithreaded programs. *Proceedings of 2001 International Parallel and Distributed Processing Symposium,*, April 2001.

[17] M. Pettersson. Linux x86 performance-monitoring counters. *http://www.freewebs.com/gkofwarf/x86perftools.htm Uppsala University*, 2002.

[18] L. DeRose J. Caubet, J. G. J. Labarta and J. Vetter. A dynamic tracing mechanism for performance analysis of openmp applications. *Proceedings of the Workshop on OpenMP Applications and Tools - WOMPAT 2001,*, July 2001.

[19] D. H. Ahn and J. S. Vetter. Scalable analysis techniques for microprocessor performance counter metrics. *Proceedings of SC2002, Baltimore, Maryland,*, November 2002.

[20] W. Saphir R. van der Wijngaart A.Woo D. Bailey, T. Harris and M.Yarrow. The nas parallel benchmarks 2.0. *Technical Report NAS-95-929, NASA Ames Research Center,*, December 1995.