Beyond the CPU: Hardware Performance Counter Monitoring on Blue Gene/Q

Heike McCraw[1], Dan Terpstra[1], Jack Dongarra[1],
Kris Davis[2], and Roy Musselman[2]

[1] Innovative Computing Laboratory (ICL), University of Tennessee, Knoxville
1122 Volunteer Blvd, Knoxville TN, 37996
{mccraw,terpstra,dongarra}@icl.utk.edu
[2] Blue Gene System Performance
Dept. KOKA, Bldg. 30-2, IBM Rochester, MN 55901
{krisd,mussel}@us.ibm.com

Abstract. The Blue Gene/Q (BG/Q) system is the third generation in the IBM Blue Gene line of massively parallel, energy efficient supercomputers that increases not only in size but also in complexity compared to its Blue Gene predecessors. Consequently, gaining insight into the intricate ways in which software and hardware are interacting requires richer and more capable performance analysis methods in order to be able to improve efficiency and scalability of applications that utilize this advanced system.

The BG/Q predecessor, Blue Gene/P, suffered from incompletely implemented hardware performance monitoring tools. To address these limitations, an industry/academic collaboration was established early in BG/Q's development cycle to insure the delivery of effective performance tools at the machine's introduction. An extensive effort has been made to extend the Performance API (PAPI) to support hardware performance monitoring for the BG/Q platform. This paper provides detailed information about five recently added PAPI components that allow hardware performance counter monitoring of the 5D-Torus network, the I/O system and the Compute Node Kernel in addition to the processing cores on BG/Q.

Furthermore, we explore the impact of node mappings on the performance of a parallel 3D-FFT kernel and use the new PAPI network component to collect hardware performance counter data on the 5D-Torus network. As a result, the network counters detected a large amount of redundant inter-node communications, which we were able to completely eliminate with the use of a customized node mapping.

1 Introduction

With the increasing scale and complexity of large computing systems the effort of performance optimization and the responsibility of performance analysis tool developers grows more and more. To be of value to the High Performance Computing (HPC) community, performance analysis tools have to be customized

J.M. Kunkel, T. Ludwig, and H.W. Meuer (Eds.): ISC 2013, LNCS 7905, pp. 213–225, 2013.

as quickly as possible in order to support new processor generations as well as changes in system designs.

The Blue Gene/Q (BG/Q) system is the third generation in the IBM Blue Gene line of massively parallel, energy efficient supercomputers. BG/Q is capable of scaling to over a million processor cores while making the trade-off of lower power consumption over raw processor speed [4]. BG/Q increases not only in size but also in complexity compared to its Blue Gene predecessors. Consequently, gaining insight into the intricate ways in which software and hardware are interacting requires richer and more capable performance analysis methods in order to be able to improve efficiency and scalability of applications that utilize this advanced system.

Performance analysis tools for parallel applications running on large scale computing systems typically rely on hardware performance counters to gather performance relevant data from the system. The Performance API (PAPI) [3] has provided consistent platform and operating system independent access to CPU hardware performance counters for more than a decade. In order to provide the very same consistency for BG/Q to the HPC community - and thanks to a close collaboration with IBMs Performance Analysis team - an extensive effort has been made to extend PAPI to support hardware performance monitoring for the BG/Q platform. This customization of PAPI to support BG/Q also includes a growing number of PAPI components to provide valuable performance data that not only originates from the processing cores but also from compute nodes and the system as a whole. More precisely, the additional components allow hardware performance counter monitoring of the 5-dimensional (5D) Torus network, the I/O system and the Compute Node Kernel in addition to the CPU component.

This paper provides detailed information about the expansion of PAPI to support hardware performance monitoring for the BG/Q platform. It offers insight into supported monitoring features. Furthermore, it will discuss performance counter data of a parallel 3-dimensional Fast Fourier Transform (3D-FFT) computation. We explore the impact of a variety of node mappings on the performance of a 3D-FFT kernel and use the recently introduced PAPI network component for BG/Q to collect hardware performance counter data on the 5D-Torus network.

This paper is organized as follows. The next section provides a brief overview of the BG/Q hardware architecture with focus on the features that are particularly relevant for this project. Section 3 goes into detail on how PAPI has been expanded with five components to support hardware performance counter monitoring on the BG/Q platform. Our case study is discussed in Section 4 which includes a short description of the implementation of the parallel 3D-FFT algorithm with a two-dimensional data decomposition as well as results of the experimental study. We conclude and summarize our work in Section 5.

2 Overview of the Blue Gene/Q Architecture

2.1 Hardware Architecture

The BG/Q processor is an 18-core CPU of which 16 cores are used to perform mathematical calculations. The 17th core is used for node control tasks such as offloading I/O operations which "talk" to Linux running on the I/O node. (Note, the I/O nodes are separate from the compute nodes; so Linux is not actually running on the 17th core.) The 18th core is a spare core which is used when there are corrupt cores on the chip. The corrupt core is swapped and software transparent. In the remainder of this paper we focus on the 17 usable cores only, since there are really only 17 logical units available on the CPU.

The processor uses PowerPC A2 cores, operating at a moderate clock frequency of 1.6 GHz and consuming a modest 55 watts at peak [6]. The Blue Gene line has always been known for throughput and energy efficiency, a trend which continues with the A2 architecture. Despite the low power consumption, the chip delivers a very respectable 204 Gflops [6]. This is due to a combination of features like the high core count, support for up to four threads per core, and a quad floating-point unit. Compared to its Blue Gene predecessors, BG/Q represents a big change in performance, thanks to a large rise in both core count and clock frequency. The BG/Q chip delivers 15 times as many peak FLOPS as its BG/P counterpart and 36 times as many as the original BG/L design (see Table 1 for comparison).

Table 1. Brief summary of the three Blue Gene versions

Version	Core Architecture	Instruction Set	Clock Speed	Core Count	Interconnect	Peak Performance
BG/L	PowerPC 440	32-bit	700 MHz	2	3D-Torus	5.6 Gigaflops
BG/P	PowerPC 450	32-bit	850 MHz	4	3D-Torus	13.6 Gigaflops
BG/Q	PowerPC A2	64-bit	1600 MHz	17	5D-Torus	204.8 Gigaflops

This PowerPC A2 core has a 64-bit instruction set compared to the 32-bit chips used in the prior BG/L and BG/P supercomputers. The A2 architecture has a 16 KB private L1 data cache and another 16 KB private L1 instruction cache per core, as well as 32 MB of embedded dynamic random access memory (eDRAM) acting as an L2 cache, and 8 GB (or 16 GB) of main memory [9]. The L2 cache as well as the main memory are shared between the cores on the chip.

Every BG/Q processor has two DDR3 memory controllers, each interfacing with eight slices of the L2 cache to handle their cache misses (one controller for each half of the 16 compute cores on the chip) [1,10]. This is an important feature that will be described in more detail in the discussion of the PAPI `L2Unit` component in Section 3.2.

BG/Q peer-to-peer communication between compute nodes is performed over a 5D-Torus network (note that BG/L and P feature a 3D-Torus). Each node has 11 links and each link can simultaneously transmit and receive data at 2 GB/s for a total bandwidth of 44 GB/s. While 10 links connect the compute

nodes, the 11th link provides connection to the I/O nodes. The I/O architecture is significantly different from previous BG generations since it is separated from the compute nodes and moved to independent I/O racks.

3 PAPI BG/Q Components

The general availability of PAPI for BG/Q is due to a cooperative effort of the PAPI team and the IBM performance team. This joint effort started with careful planning long before the BG/Q release, with the goal to design PAPI for Q as well as to design the Blue Gene Performance Monitoring API (BGPM) according to what is needed by PAPI and other HPC performance analysis tools, like e.g. HPCToolkit [2] that heavily use PAPI under the covers.

In general, hardware performance event monitoring for BG/Q requires user code instrumentation with either the native BGPM API or a tool like PAPI which relies on BGPM. The following five sections talk about the five different components that have been implemented in PAPI to allow users to monitor hardware performance counters on the BG/Q architecture through the standard Performance API interface.

3.1 Processor Unit Component

The PAPI `PUnit` component is handled as component 0 in PAPI - which is the default CPU component. Each of the 17 usable A2 CPU cores has a local Universal Performance Counting (UPC) module. Each of these modules provides 24 counters (14-bit) to sample A2 events, L1 cache related events, floating point operations, etc. Each local UPC module is broken down into five internal sub-modules: functional unit (FU), execution unit (XU), integer unit (IU), load/store unit (LSU) and memory management unit (MMU). These five internal sub-modules are easily identifiable through the event names. Table 2 shows an example selection of native `PUnit` events provided by the PAPI utility `papi_native_avail`.

In addition to native events, a user can select predefined events (Presets) for the `PUnit` component on BG/Q. Out of 108 possible predefined events, there are currently 43 events available of which 15 are derived events made up of more than one native event.

Overflow: Only the local UPC module, L2 and I/O UPC hardware support performance monitor interrupts when a programmed counter overflows [1]. For that reason, PAPI offers overflow support for only the `PUnit`, `L2Unit`, and `IOUnit` components.

Fast versus Slow Overflow: `Punit` counters freeze on overflow until the overflow handling is complete. However, the `L2Unit` and `IOUnit` counters do not freeze on overflow. The L2 and I/O counts will be stopped when the interrupt is handled. The signal handler restarts L2 and I/O counting when done [1].

`PUnit` counters can detect a counter overflow and raise an interrupt within approx. 4 cycles of the overflowing event. However, according to the BGPM

Table 2. Small selection of PUnit events available on BG/Q

PUnit Event	Description
PEVT_AXU_INSTR_COMMIT	A valid AXU (non-load/store) instruction is in EX6, past the last flush point. - AXU uCode sub-operations are also counted by PEVT_XU_COMMIT instead.
PEVT_IU_IL1_MISS	A thread is waiting for a reload from the L2. - Not when CI=1. - Not when thread held off for a reload that another thread is waiting for. - Still counts even if flush has occurred.
PEVT_IU_IL1_MISS_CYC	Number of cycles a thread is waiting for a reload from the L2. - Not when CI=1. - Not when thread held off for a reload that another thread is waiting for. - Still counts even if flush has occurred.
PEVT_IU_IL1_RELOADS_DROPPED	Number of times a reload from the L2 is dropped, per thread - Not when CI=1 - Does not count when not loading cache due to a back invalidate to that address
PEVT_XU_BR_COMMIT_CORE	Number of Branches committed
PEVT_LSU_COMMIT_LD_MISSES	Number of completed load commands that missed the L1 Data Cache. - Microcoded instructions may be counted more than once. - Does not count dcbt[st][ls][ep]. - Include larx. - Does not includes cache-inhibited loads
PEVT_MMU_TLB_HIT_DIRECT_IERAT	TLB hit direct entry (instruction, ind=0 entry hit for fetch)
PEVT_MMU_TLB_MISS_DIRECT_IERAT	TLB miss direct entry (instruction, ind=0 entry missed for fetch)
...	...

documentation it takes up to approx. 800 cycles before the readable counter value is updated. This latency does not affect the overflow detection, and so we refer to a `PUnit` overflow as a "Fast Overflow".

The `IOUnit` and `L2Unit` take up to 800 processor cycles to accumulate an event and detect an overflow. Hence, we refer to this as a "Slow Overflow", and the program counters may alter up to 800 cycles or more after the event. This delay is due to the distributed nature of the performance counters. The counters are spread throughout the chip in multiple performance units. The hardware design consolidates the counters into one memory space continually, however it takes 800 cycles to visit all of the distributed units, hence the delay. The `IO` and `L2Units` are not thread specific, so there is no basis to stop counting for a single thread on overflow. However, the `PUnit` counters can be threaded, and the hardware has the ability to arm the distributed counts and freeze on overflow.

Multiplexing: PAPI supports multiplexing for the BG/Q platform. BGPM does not directly implement multiplexing of event sets. However, it does indirectly support multiplexing by supporting a multiplexed event set type. A multiplexed event set type will maintain sets of events which can be counted simultaneously, while pushing conflicting events to other internal sets [1].

3.2 L2 Unit Component

The shared L2 cache on the BG/Q system is split into 16 separate slices. Each of the 16 slices has a L2 UPC module that provides 16 counters with fixed events that can be gathered separately or aggregated into 16 counters (depending on the events chosen). Those 16 counters are node-wide, and cannot be isolated to a single core or thread. As mentioned earlier, every BG/Q processor has two DDR3 memory controllers, each interfacing with eight slices of the L2 cache to

handle their cache misses (one controller for each half of the 16 compute cores on the chip) [1,10]. The counting hardware can either keep the counts from each slice separate, or combine the counts from each slice into single values (which is the default). The combined counts are significantly important if a user wants to sample on overflows. Actually, the separate slice counts are not particularly interesting except for perhaps investigating cache imbalances because consecutive memory lines are mapped to separate slices. The node-wide "combined" or "sliced" operation is selected by creating an event set from the "combined" (default), or "sliced" group of events. Hence a user cannot assign events from both groups. Currently, there are 32 `L2Unit` events (16 events for the "combined" and "sliced" case, respectively) available on the BG/Q architecture.

Overflow: If `L2Unit` event overflow is desired, the overflow signal is "slow" (see the end of Section 3.1 for details that describe the difference between fast and slow overflow). As mentioned before, PAPI supports overflow for `PUnit` events as well as `L2Unit` and `IOUnit` events.

3.3 I/O Unit Component

The Message, PCIe, and DevBus modules - which are collectively referred to as I/O modules - together provide 43 counters. These counters are node-wide and cannot be isolated to any particular core or thread [1]. Note, the PCIe module is only enabled on the I/O nodes but disabled on the compute nodes. The counters for this specific I/O sub-module exist, however, there is currently no BGPM support for the I/O nodes. Currently, there are 44 `IOUnit` events available on the BG/Q architecture. The two I/O sub-modules - Message, and DevBus - are transparently identifiable from the `IOUnit` event names.

Overflow: If `IOUnit` event overflow is desired, the overflow signal is "slow" (see the end of Section 3.1 for details that describe the difference between fast and slow overflow).

3.4 Network Unit Component

The 5D-Torus network provides a local UPC network module with 66 counters - each of the 11 links has six 64-bit-counters. As of right now, a PAPI user cannot select which network link to attach to. Currently, all 11 network links are attached and this is hard-coded in the PAPI `NWUnit` component. We are considering options for supporting the other enumerations for network links as well. We can easily change to attaching the ten torus links only and leave the I/O link out. As for measuring the performance of an application's communication, both of the two configurations will work without limitations because the I/O links are not used for sending packets to another compute node. However, if users want to evaluate the I/O performance of an application, then they can do this via the current network component as well. This would not be the case when we use the torus links only. Currently, there are 31 `NWUnit` events available on the BG/Q architecture.

3.5 CNK Unit Component

By default a custom lightweight operating system called Compute Node Kernel (CNK) is loaded on the compute nodes while I/O nodes run Linux OS [4]. The CNK OS is the only kernel that runs on all the 16 compute cores. In general, on Linux kernels the "/proc" file system is the usual access method for kernel counts. Since CNK does not have a "/proc" filesystem, PAPI uses BGPM's "virtual unit" that has software counters collected by the kernel. The kernel counter values are read via a system call that requests the data from the lightweight compute node kernel. Also, there is a read operation to get the raw value since the system has been booted. Currently, there are 29 `CNKUnit` events available on the BG/Q architecture. Table 3 provides a small selection of `CNKUnit` events. The CNK functionality is heavily used by tools that support sample-based profiling like e.g. HPCToolkit [2]. Hence, with the `CNKUnit` Component, this is much easier handled on BG/Q than it was on BG/P.

Table 3. Small selection of CNKUnit events, available on the BG/Q architecture

`CNKUnit` Event	Description
`PEVT_CNKNODE_MUINT`	Number of Message Unit non-fatal interrupts
`PEVT_CNKNODE_NDINT`	Number of Network Device non-fatal interrupts
`PEVT_CNKHWT_SYSCALL`	System Calls
`PEVT_CNKHWT_FIT`	Fixed Interval Timer Interrupts
`PEVT_CNKHWT_WATCHDOG`	Watchdog Timer Interrupts
`PEVT_CNKHWT_PERFMON`	Performance Monitor interrupts
`PEVT_CNKHWT_PROGRAM`	Program Interrupts
`PEVT_CNKHWT_FPU`	FPU Unavailable Interrupts
...	...

4 Case Study: Parallel 3D-FFT on BG/Q

As a case study, we implemented a parallel 3D-FFT kernel and want to explore how well the communication performs on the BG/Q network. The Fast Fourier Transforms (FFT) of multidimensional data are of particular importance in a number of different scientific applications but they are often among the most computationally expensive components. Parallel multidimensional FFTs are communication intensive, which is why they often prevent the application from scaling to a very large number of processors. A fundamental challenge of such numerical algorithms is a design and implementation that efficiently uses thousands of nodes. One important characteristics of BG/Q is the organization of the compute nodes in a 5D-Torus network. We will explore that in order to maintain application performance and scaling, the correct mapping of MPI tasks onto the torus network is a critical factor.

4.1 Definition of the Fourier Transformation

We start the discussion with the definition and the conventions used for the Fourier Transformation (FT) in this paper. Consider $A_{x,y,z}$ as a three-dimensional array of $L \times M \times N$ complex numbers with:

$$\begin{aligned} A_{x,y,z} \in \mathbb{C}\ x \in \mathbb{Z}\ \forall x,\ 0 \leq x < L \\ y \in \mathbb{Z}\ \forall y,\ 0 \leq y < M \\ z \in \mathbb{Z}\ \forall z,\ 0 \leq z < N \end{aligned}$$

The Fourier transformed array $\tilde{A}_{u,v,w}$ is computed using the following formula:

$$\tilde{A}_{u,v,w} := \underbrace{\sum_{x=0}^{L-1} \underbrace{\sum_{y=0}^{M-1} \underbrace{\sum_{z=0}^{N-1} A_{x,y,z} \exp(-2\pi \mathrm{i} \frac{wz}{N})}_{\text{1st 1D FT along } z}}_{\text{2nd 1D FT along } y} \exp(-2\pi \mathrm{i} \frac{vy}{M})}_{\text{3rd 1D FT along } x} \exp(-2\pi \mathrm{i} \frac{ux}{L}) \quad (1)$$

As shown by the under-braces, this computation can be performed in three single stages. This is crucial for understanding the parallelization in the next subsection. The first stage is the one-dimensional FT along the z dimension for all (x, y) pairs. The second stage is a FT along the y dimension for all (x, w) pairs, and the final stage is along the x dimension for all (v, w) pairs.

4.2 Parallelization

Many previous parallel 3D-FFT implementations have used a one-dimensional virtual processor grid - i.e. only one dimension is distributed among the processors and the remaining dimensions are kept locally. This has the advantage that one all-to-all communication is sufficient. However, for problem sizes of about one hundred points or more per dimension, this approach cannot offer scalability to several hundred or thousand processors as required for modern HPC architectures. For this reason the developers of the IBMs Blue Matter application have been promoting the use of a two-dimensional virtual processor grid for FFTs in three dimensions [5]. This requires two all-to-all type communications, as shown in Figure 1, which illustrates the parallelization of the 3D-FFT using a two-dimensional decomposition of the data array A of size $L \times M \times N$. The compute tasks have been organized in a two-dimensional virtual processor grid with P_c columns and P_r rows using the MPI Cartesian grid topology construct. Each individual physical processor holds an $L/P_r \times M/P_c \times N$ sized section of A in its local memory. The entire 3D-FFT is performed in five steps as follows:

1. Each processor performs $L/P_r \times M/P_c$ one-dimensional FFTs of size N

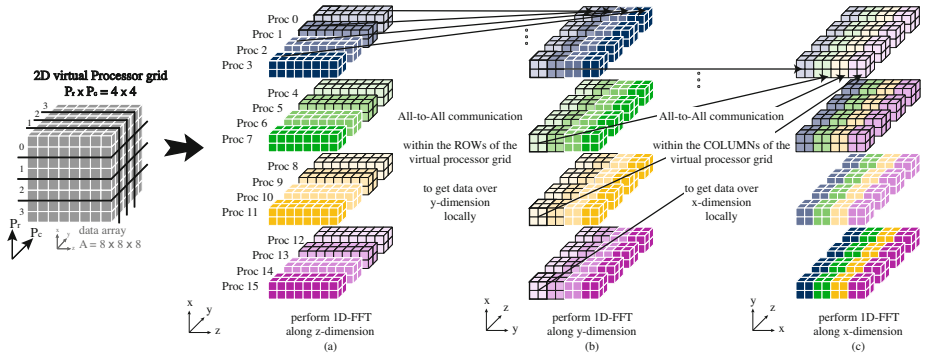


Fig. 1. Computational steps of the 3D-FFT implementation using 2D-decomposition

2. An all-to-all communication is performed within each of the rows - marked in the four main colors - of the virtual processor grid to redistribute the data. At the end of the step, each processor holds an $L/P_r \times M \times N/P_c$ sized section of A. These are P_r independent all-to-all communications.
3. Each processor performs $L/P_r \times N/P_c$ one-dimensional FFTs of size M.
4. A second set of P_c independent all-to-all communications is performed, this time within the columns of the virtual processor grid. At the end of this step, each processor holds a $L \times M/P_c \times N/P_r$ size section of A.
5. Each processor performs $M/P_c \times N/P_r$ one-dimensional FFTs of size L

For more information on the parallelization, the reader is referred to [5,8].

4.3 Communication Network Topology

As mentioned before, the network topology for BG/Q is a 5D-Torus. Every node is connected to its ten neighbor nodes through bidirectional links in the $\pm A$, $\pm B$, $\pm C$, $\pm D$, and $\pm E$ directions. This appears to be a significant change compared to BG/Q predecessors, both of which feature a 3D-Torus. Here every node is connected to its six neighbor nodes through bidirectional links in the $\pm A$, $\pm B$, and $\pm C$ directions. To maintain application performance, an efficient mapping of MPI tasks onto the torus network is a critical factor.

The default mapping is to place MPI ranks on the BG/Q system in $ABCDET$ order where the rightmost letter increments first, and where $< A, B, C, D, E >$ are the five torus coordinates and $< T >$ ranges from 0 to $N - 1$, with N being the number of ranks per node [7]. If the job uses the default mapping and specifies one process per node, the following assignment results:
MPI rank 0 is assigned to coordinates $< 0, 0, 0, 0, 0, 0 >$
MPI rank 1 is assigned to coordinates $< 0, 0, 0, 0, 1, 0 >$
MPI rank 2 is assigned to coordinates $< 0, 0, 0, 1, 0, 0 >$
The mapping continues like this, first incrementing the E coordinate, then the D coordinate, and so on, until all the processes are mapped. The user can choose

a different mapping by specifying a different permutation of $ABCDET$ or by creating a customized map file.

4.4 Network Performance Counter Analysis

We ran our 3D-FFT kernel on a 512 node partition, utilizing half a rack on the BG/Q system at Argonne National Laboratory, using all 16 compute cores per node for each run. Table 4 summarizes the number of nodes that are available in each of the five dimensions. Also the torus connectivity is shown for each dimension, while 1 indicates torus connectivity for a particular dimension, 0 indicates none.

Table 4. Torus connectivity and number of nodes in each of the five dimensions for a 512 node partition

	A	B	C	D	E	T
nodes	4	4	4	4	2	16
torus	1	1	1	1	1	-

For the 512 node partition, we have a total of 8,192 MPI tasks, and for the virtual two-dimensional process grid, we chose 16×512, meaning that each subgroup has 16 MPI tasks and we have 512 of those subgroups. Since we want to know how well the communication performs on the 5D-Torus network, we use the new PAPI network component to sample various network related events. The number of packets sent from each node is shown in Figure 2 for a problem size of 512^3. This includes packets that originate as well as pass through the current node. It is important to note, these are the numbers for only the all-to-all communication within each subgroup (only first all-to-all), not including the second all-to-all communication between the subgroups.

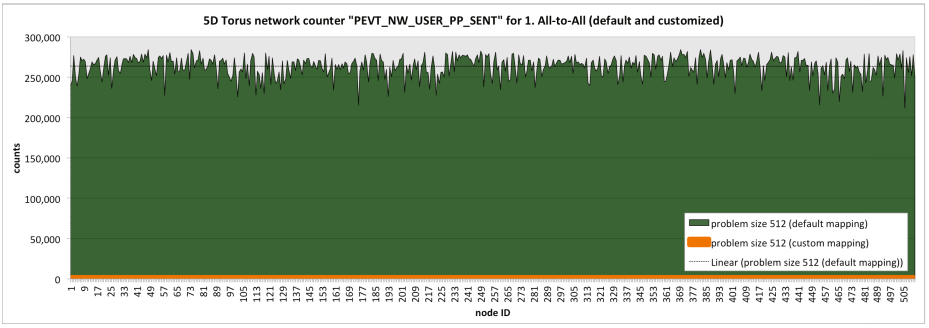


Fig. 2. Network counter data collected with PAPI. This event counts the number of packets originating and passing through the current node for the first all-to-all communication.

The PAPI network counter data greatly helped to evaluate the communication of the parallel 3D-FFT kernel as it clearly shows an unnecessary large number of packets that cross a node if the default MPI task mapping is used. The collected network data for a medium problem size of 512^3 counts approx. 270,000 packets which originate from and pass through each single node. Note, the count variation from node to node is due to the routing algorithm which may pass more packets through some nodes but not others based on how congested alternative routes are in the network. On the whole, we consider this number of packets for such a fairly small problem size extremely high, which is also the cause of the discovered network congestions. Without the network counter data, a user may merely pursue with speculations about various reasons of the poor performance. However, the data allows a much more concentrated analysis that assists with taking more settled instead of speculative actions.

In order to resolve this type of network congestion, we examined a variety of customized MPI task mappings, which heavily depend on the chosen 2D processor grid of the parallel 3D-FFT implementation. For each experiment, the network counter data distinctly indicated either a stationary or improved congestion of the network. The analysis shows that the reason for the high numbers is the placement of MPI tasks onto the network using the default mapping which results in a lot of inter-node communications. It appears that even when using a total of five dimensions for a torus network, the default mapping can still result in severe performance degradations due to congestions. This stresses all the more how critical the correct mapping of MPI tasks onto the torus network is, even when we utilize a five-dimensional torus. The default mapping places each task of a subgroup on a different node, as can be seen from Table 5(a) that summarizes the default MPI task mapping on the 5D-Torus for one communicator.

Table 5. MPI task mappings on the 5D-Torus for the 512 node partition run, using a 2D virtual processor grid 16×512. For simplicity, each table presents the mapping of only one out of a total of 512 communicators.

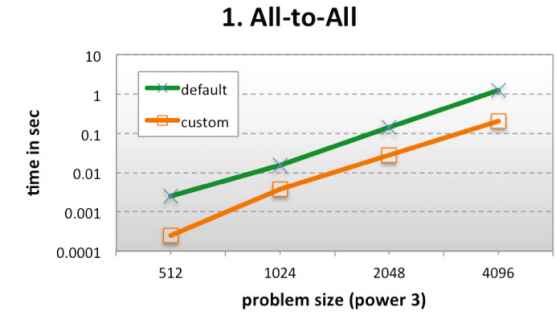
(a) Default MPI-task mapping

rank	A	B	C	D	E	T
0	0	0	0	0	0	0
512	0	1	0	0	0	0
1,024	0	2	0	0	0	0
1,536	0	3	0	0	0	0
2,048	1	0	0	0	0	0
2,560	1	1	0	0	0	0
3,072	1	2	0	0	0	0
3,584	1	3	0	0	0	0
4,096	2	0	0	0	0	0
4,608	2	1	0	0	0	0
5,120	2	2	0	0	0	0
5,632	2	3	0	0	0	0
6,144	3	0	0	0	0	0
6,656	3	1	0	0	0	0
7,168	3	2	0	0	0	0
7,680	3	3	0	0	0	0

(b) Customized MPI-task mapping

rank	A	B	C	D	E	T
0	0	0	0	0	0	0
512	0	0	0	0	0	1
1,024	0	0	0	0	0	2
1,536	0	0	0	0	0	3
2,048	0	0	0	0	0	4
2,560	0	0	0	0	0	5
3,072	0	0	0	0	0	6
3,584	0	0	0	0	0	7
4,096	0	0	0	0	0	8
4,608	0	0	0	0	0	9
5,120	0	0	0	0	0	10
5,632	0	0	0	0	0	11
6,144	0	0	0	0	0	12
6,656	0	0	0	0	0	13
7,168	0	0	0	0	0	14
7,680	0	0	0	0	0	15

The above mentioned network counter data analysis for various customized mappings promotes a mapping that places all the tasks from one subgroup onto the same node which significantly reduced the amount of communication. Table 5(b) presents the optimum customized MPI task mapping on the 5D-Torus for the same communicator as was used in Table 5(a). Since each subgroup has 16 MPI tasks, and since we have 16 compute cores per node, we can place one entire subgroup on each node. By doing so, all the high numbers reported for the network counter were reduced to zeroes, resulting in no inter-node communication at all. The results presented in Figure 3 show that the customized mapping gives us a performance improvement of up to a factor of approx. 10 (depending on the problem size) for the first all-to-all. Note, there was no degradation in performance for the second all-to-all with the customized mapping. For the entire 3D-FFT kernel - which consists of three 1D-FFT computations and two all-to-all communications - we see an improvement ranging from 10 to 18% for various mid-size problems.



Comm	10.2x	4x	5.2x	6x
3D FFT	18%	16%	15%	10%

Fig. 3. Performance comparison for the first all-to-all communication using default and customized mapping. The table at the bottom presents the performance improvement of the customized mapping for each problem size and for the communication as well as the entire 3D-FFT kernel respectively.

5 Conclusion

Performance analysis tools for parallel applications running on large scale computing systems typically rely on hardware performance counters to gather performance relevant data from the system. In order to allow the HPC community to collect hardware performance counter data on IBM's latest Blue Gene system BG/Q, PAPI has been extended with five new components.

The PAPI customization for BG/Q accesses the BGPM interface under the covers, allowing users and third-party programs to monitor and sample hardware

performance counters in a traditional way using the default PAPI interface. The recently added PAPI components allow hardware performance counter monitoring not only for the processing units but also for the 5D-Torus network, the I/O system, and the Compute Node Kernel.

As a case study for using hardware performance monitoring beyond the CPU we implemented a parallel 3D-FFT kernel and instrumented it with PAPI for communication evaluation on the BG/Q system at Argonne National Laboratory. The collected network counter data considerably helped evaluating the communication for the 5D-torus partition as well as made us look deeper into where tasks are located by default on the 5D network, and how to improve the task location based on the algorithm's features. With the default mapping of MPI tasks onto the torus network, the network counters detected a large amount of redundant inter-node communications. By employing a custom mapping, we were able to eliminate the unnecessary communication and achieve more than a ten-fold bettering for the all-to-all communication which consequently leads to up to 18% performance improvement for the entire 3D-FFT kernel on 8,192 cores.

Acknowledgments. This material is based upon work supported by the U.S. Department of Energy Office of Science under contract DE-FC02-06ER25761. Access to the early access BG/Q system at Argonne National Laboratory was provided through the ALCF Early Science Program.

References

1. BGPM Documentation (2012)
2. Adhianto, L., Banerjee, S., Fagan, M., Krentel, M., Marin, G., Mellor-Crummey, J., Tallent, N.R.: HPCToolkit: Tools for performance analysis of optimized parallel programs. Concurrency and Computation: Practice and Experience 22(6), 685–701 (2010)
3. Browne, S., Dongarra, J., Garner, N., Ho, G., Mucci, P.: A portable programming interface for performance evaluation on modern processors. International Journal of High Performance Computing Applications 14(3), 189–204 (2000)
4. Budnik, T., Knudson, B., Megerian, M., Miller, S., Mundy, M., Stockdell, W.: Blue Gene/Q Resource Management Architecture (2010)
5. Eleftheriou, M., Moreira, J.E., Fitch, B.G., Germain, R.S.: A Volumetric FFT for BlueGene/L. In: Pinkston, T.M., Prasanna, V.K. (eds.) HiPC 2003. LNCS (LNAI), vol. 2913, pp. 194–203. Springer, Heidelberg (2003)
6. Feldman, M.: IBM Specs Out Blue Gene/Q Chip (2011), `http://www.hpcwire.com/hpcwire/2011-08-22/ibm_specs_out_blue_gene_q_chip.html`
7. Gilge, M.: IBM system Blue Gene solution: Blue Gene/Q application development. IBM Redbook Draft SG24-7948-00 (2012)
8. Jagode, H.: Fourier Transforms for the BlueGene/L Communication Network. Master's thesis, EPCC, The University of Edinburgh (2006), `http://www.epcc.ed.ac.uk/msc/dissertations/2005-2006/`
9. Morgan, T.P.: IBM Blue Gene/Q details (2011), `http://www.multicoreinfo.com/2011/02/bluegeneq`
10. Morgan, T.P.: IBM's Blue Gene/Q super chip grows 18th core (2011), `http://www.theregister.co.uk/2011/08/22/ibm_bluegene_q_chip`