Custom assignment of MPI ranks for parallel multi-dimensional FFTs: Evaluation of BG/P versus BG/L

Heike Jagode
The University of Tennessee - Knoxville
Oak Ridge National Laboratory (ORNL)
jagode@eecs.utk.edu

Joachim Hein
Edinburgh Parallel Computing Centre (EPCC)
The University of Edinburgh
joachim@epcc.ed.ac.uk

Abstract

For many scientific applications, the Fast Fourier Transformation (FFT) of multi-dimensional data is the kernel that limits scalability on a large number of processors. This paper investigates the extent of performance improvements for a parallel three-dimensional FFT (3D-FFT) implementation when using customized MPI task mappings. The MPI tasks are mapped in a customized fashion from the two-dimensional virtual processor grid of the algorithm to the physical hardware of a system with a mesh interconnect. We compare and analyze the outcomes on Blue Gene/P with those from previous investigations on Blue Gene/L. The performance analysis is based on bandwidth considerations. The results demonstrate that on Blue Gene/P, a carefully chosen MPI task mapping with regards to the network characteristics is more important compared to Blue Gene/L and yields significant improvement.

1. Introduction

For many scientific applications, parallel multi-dimensional Fast Fourier Transformation (FFT) routines form the key performance bottleneck that prevents the application from scaling to large numbers of processors. FFTs are often employed in applications requiring the numerical solution of a differential equation. In this case the differential equation is solved in Fourier space, but its coefficients are determined in position space. FFTs can also be efficient for the determination of the long-range forces, e.g. Particle-Mesh Ewald methods in molecular dynamics simulations. Most of these applications require the transformation between a three-dimensional position and a three-dimensional Fourier space.

Many previous parallel 3D-FFT implementations have used a one-dimensional virtual processor grid - 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 per dimension, this approach cannot offer scalability to several hundred or thousand processors as required for the modern HPC architectures. For this reason the developers of the IBM's Blue Matter application have been promoting the use of a two-dimensional virtual processor grid for FFTs in three dimensions [1, 2, 3]. This requires two all-to-all type communications. For lower processor counts, these two communication operations lead to inferior performance when compared to an implementation using a one-dimensional virtual grid. However this algorithm offers superior scalability to processor counts where a one-dimensional grid can no longer be employed [1, 4].

A current trend in supercomputer design is the return of the mesh type communication network. The systems on the Top500 list [5] utilizing more than 20000 processors arrange their compute nodes on a three-dimensional mesh communication network instead of a switched network. When using a mesh-type network, it is often possible to achieve substantial performance gains by taking the network characteristics into account. One example is to facilitate nearest neighbor communication by choosing a good MPI task mapping between the virtual processor grid of the application space and the physical processor mesh of the actual compute hardware.

In this article we investigate the scope for such performance improvements when mapping the MPI tasks of a parallel 3D-FFT implementation with a two-dimensional virtual processor grid onto a machine with a three-dimensional mesh as its communication network. In general, mapping is the process of assigning tasks to processors. In this paper, we define a mapping as an assignment of MPI ranks

onto processors. The investigations have been performed on the Jülich's Supercomputing Centre IBM Blue Gene/P. The results on Blue Gene/P (BG/P) have been compared with those from previous investigations on Blue Gene/L (BG/L).

Using FFTs to solve differential equations is a special case for what is commonly known as using spectral methods to solve a differential equation. We note that the optimization techniques investigated here can be easily applied to all spectral methods using a set of basis functions that factorize with respect to the spatial directions. That is, in case of three directions, each basis function $b(x, y, z)$ can be written as a product of three functions $f(x)$, $g(y)$ and $h(z)$. Besides the trigonometric functions used in FFTs, polynomials such as Chebyshev or Legendre polynomials or a combination of those are presently widely used for the functions f, g and h.

It would be very interesting to investigate the optimization techniques discussed in this paper also on a Cray XT architecture. However the PBS Pro job management system used by most installations to schedule batch jobs does not allow the user to request a partition of the machine with the required geometric shape. This reduces the constraints on placing large jobs and reduces the time spend on draining the machine prior to the job start up. The disadvantage of not offering such an option is a potential increase of communication time due to bottlenecks resulting from a fragmented machine partition. Reference [6] discusses possible performance gains from task placement on a Cray XT system. However the techniques discussed in this reference are more widely applicable and not specific to a mesh communication network.

This paper is organized as follows. Section 2 reviews the implementation of the parallel 3D-FFT algorithm with a two-dimensional data decomposition. A short overview of the BG/P system used for this study is provided in Section 3. The results of the experimental study on a 512-node partition as well as a comparison with earlier results from an IBM eServer BG/L are presented in Section 4. Results on a 4096-node partition are discussed in Section 5. The paper ends with conclusions and future work.

2 Review of parallel FFT algorithms

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

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

As shown by the underbraces, 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.

2.2 Parallelization

For the three-dimensional case, two different implementations - one-dimensional decomposition and two-dimensional decomposition of the data over the physical processor grid - have been recently investigated [1, 2, 4]. The parallel 3D-FFT algorithm using a two-dimensional decomposition is often referred to in the literature as the volumetric fast Fourier transform. In this paper we concentrate on the performance characteristics resulting from the MPI task placements of the two-dimensional decomposition method onto a mesh communication network. Reference [4] provides an initial investigation. Figure 1 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

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.

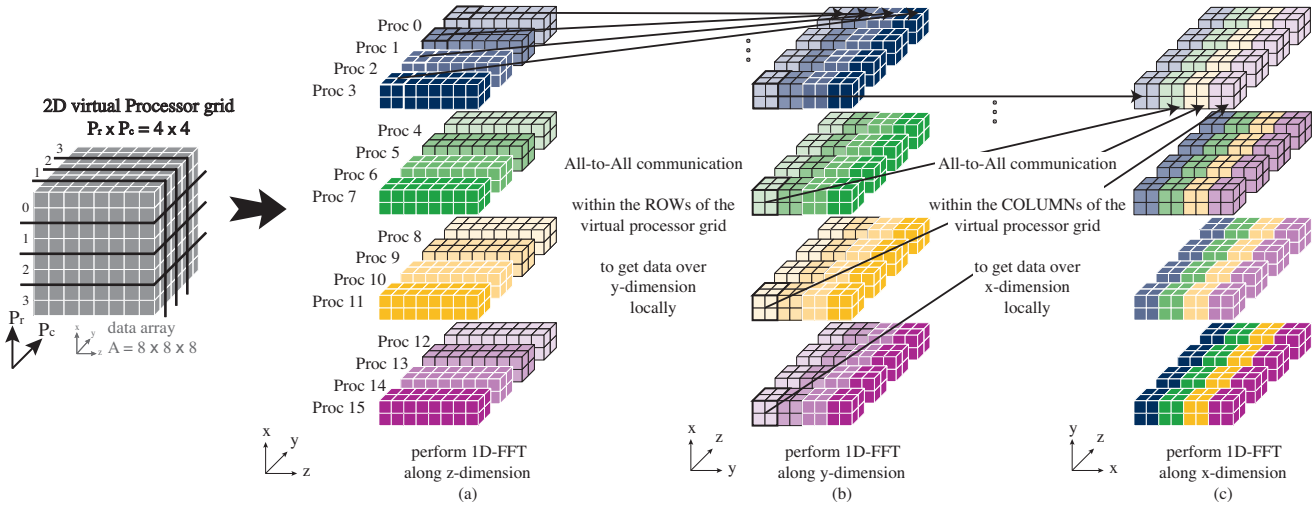

Figure 1: Computational steps of the 3D-FFT implementation using 2D-decomposition

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 [1, 4].

2.3 Details of the Benchmark Application

The benchmark application is written in C. While the communication part of the algorithm is most important to this project, it is desirable to implement the full algorithm so as to allow the significance of potential improvements to the communication part of the algorithm to be evaluated against the total time of the algorithm. The benchmark application was run several times. The run measured on a hot L3 cache, which yielded the best performance with regard to the total amount of time taken for the entire three-dimensional forward FFT computation, is presented in this paper.

The application uses version 2.1.5 of the open-source "Fastest Fourier Transform in the West" (FFTW) library [7] to perform the required one-dimensional FFTs. The Vienna University of Technology offers a FFTW 2.1.5 version, known as FFTW-GEL [8], that is specifically optimized for the double floating point unit of the Blue Gene processors.

The MPI library provided by IBM as part of the system software is used for the required communication. In the application, a Cartesian communicator is used to create the virtual processor grid described in Section 2.2. This is divided into sub-communicators for the rows and the columns of the grid. The `MPI_Alltoall` routine is used for the communication kernel of our parallel three-dimensional FFT computation .

The `-mapfile` option of the job launcher `mpirun` on the Blue Gene architecture is used to implement the map between the virtual processor grid of the application and the physical processor mesh of the hardware.

3 Overview of the BG/P system

3.1 Processors and Operational Modes

The application described in Section 2 has been tested on the Jülich's Supercomputing Centre IBM Blue Gene/P, named JUGENE. This section gives a short overview of system features most relevant for this investigation. Further information can be found in [9]. The machine uses four IBM PowerPC 450 cores per node with a clock frequency of 850 MHz. The system in Jülich offers a total of 16,384 compute nodes. 2 GB of main memory are installed per node.

The BG/P architecture offers three main operational modes. In SMP mode, a single MPI task is placed on the node. Two MPI tasks are supported in DUAL and four in VN mode.

3.2 Partitions and Communication

The BG/P architecture offers five different networks that are dedicated and optimized for different tasks. For all-to-all communication, only the torus network is relevant. The internal logic of the torus network remains essentially unchanged from that of the BG/L system [9, 10]. It arranges nodes on a 3D torus, with communications taking place between nearest neighbors. The connecting links of this network offer a bandwidth of 4 bits per cycle per direction, which translates to 405 MB/s when using a clock frequency of $850 \cdot 10^6$ Hz.

The maximum length of the torus packets is 256 bytes, with the first 16 bytes being used for control information [9]. Additionally, 14 bytes of error control data are sent for each packet that is sent into the torus network. This results in a maximum utilization of the torus network of 89% and a limit of about 360 MB/s for the bandwidth. For a simple ping-pong benchmark using MPI, a sustained bandwidth of 358 MB/s, which is remarkably close to that limit, has been measured.

Each user application has to request a dedicated cuboidal partition of the machine. For small partitions the meshed network can only be configured as an open mesh, while for partitions of 512 nodes or multiples thereof, there is a choice of an open mesh or a full torus, with the latter being the default.

The BG/P system offers several ways to affect the runtime environment of parallel jobs. The most important one in our investigation of the MPI task placement is the mapfile. For each MPI task, the file contains the coordinates with respect to the physical 3D torus network of the machine on which this task is to be placed.

The BG/P system features another fundamental modification compared to the BG/L system. On BG/L the processor cores were responsible for injecting (or receiving) data packets to (or from) the network [9]. On BG/P an additional piece of hardware, a direct memory access engine (DMA), has been added that basically offloads most of this responsibility from the processor cores. Later we will see how this new piece of hardware affects performance. Table 1 summarizes the BG/L and BG/P architecture features that are relevant for this particular investigation.

Features	BG/L	BG/P
Node Processors	2 PowerPC 440	4 PowerPC 450
Proc. Frequency	700 MHz	850 MHz
Cache Coherency	Software managed	4-way SMP
Memory per Node	512 MB (256 MB/core) 1 GB (512 MB/core)	2 GB (512 MB/core)
Bandwidth per Link	Core injects/receives packets: 148 MB/s [11]	DMA injects/receives packets: 360 MB/s

Table 1: BG/L versus BG/P architecture features

4 Results of BG/P versus those of BG/L

Before investigating customized MPI task mappings on a large partition like the 4096-node partition, we will discuss the results on the smallest partition that offers torus connectivity on Blue Gene - the 512-node partition. All our investigations focus on the case of a full torus network. The results we compare within this section show what sort of mappings we should be aware of and even try to avoid. The choice of customized mappings discussed in this paper is based on earlier investigations and performance models presented in [4, 11]. These investigations have been performed on BG/L and the results show clearly that best performance can be achieved by using cube-shaped mapping patterns for either the rows or the columns of the two-dimensional virtual processor grid. For that reason we continue using these cubes as customized mappings on BG/P.

We start this section with drawing a comparison between the execution times for the entire 3D-FFT computation on BG/L and BG/P. Then we consider the times for the communication part only. Finally, we will go into detail for the communication part and individually compare the times for the communications between rows of the 2D virtual processor grid (first all-to-all type communication) and also for the communications between columns (second all-to-all type communication). To identify scaling bottlenecks of the different mappings we will consider the bandwidth utilization for each of them.

4.1 Analysis of Overall Performance

We compare the results for the default and customized MPI task mapping on BG/P with those on BG/L. Both mapping patterns - default and customized - using SMP mode on BG/P (which is CO mode on BG/L) are depicted in Figure 2 (a). In the remaining of this paper we call the mapping used for the customized mapping of the rows for case (a) an "8-cube". In addition to the mappings in SMP mode, we investigate mappings in DUAL and VN mode which show the same shape for the customized and default cases. Fig-

ure 2 (b) illustrates the default and customized mapping using DUAL mode on BG/P. We do not depict VN mode mapping on BG/P since it is similar to the other mappings except that it uses four MPI tasks per node (instead of one MPI task for SMP mode and two for DUAL mode).

For the purpose of clarity, the figure showing the SMP mode mapping (2 (a)) depicts only the basic image of one communicator. The full map for all rows of the virtual processor grid is constructed by using displacements of the basic image across the entire physical 3D torus network. More precisely, in Figure 2 (a), the 8-cube is displaced 64 times to fill the entire partition. The same applies to all other mapping patterns. The figure showing the DUAL mode mapping (2 (b)) even depicts the basic image of two communicators since here each node is divided between two communicators.

Figure 3 presents the influence of the 8-cube mapping on the total performance of the entire 3D-FFT algorithm on BG/P. To obtain a more readable figure, the results have been normalized to the performance of the default mapping. More precisely, the performance result of the default mapping for the 8×64 virtual processor grid in SMP mode has been divided by the results of all other mappings. The measured times for the overall performance can be found in Appendix A. The same investigation has been carried out on BG/L and those results can be found in [11]. On BG/L (VN-mode) we obtained a substantial overall performance improvement due to the 8-cube mapping of 16% for mid-size problems and 10% for large problems. However, on BG/P we see an improvement only for the problem size of 512^3. This is due to the poor bandwidth utilization when using non-contiguous mappings which is discussed in more detail later in this paper.

The results in Figure 4 (a) show that in general, moving the 3D-FFT application from BG/L to BG/P obtains an overall improvement of about 30% for the entire 3D-FFT computation. It is interesting to note that assignment of a larger data sub-grid to each node, which is made possible by increased memory capacity per node as mentioned in Section 3, causes the performance on BG/P to continue to speed up linearly. If we compare the times for the communication part only as presented in Figure 4 (b), then the two all-to-all type communications are about twice as fast on BG/P compared to BG/L, independent of the problem size. This result was expected since the network on BG/P delivers more than twice as many bytes per cycle than that on BG/L, while the internal logic of the torus network remains essentially unchanged [9].

4.2 Analysis of Individual Communication Times

Now we compare the improvement we gained from customized mappings over default mappings on BG/L with the results on BG/P. To identify which of the two mappings (either for rows or columns) is the scaling bottleneck, the communication times for each of the two all-to-all type communications have been investigated individually. In Figure 5 we compare the results for the default versus customized mapping for the communication between rows (a) and columns (c) on BG/L in CO mode. Figure 5 (b) and (d) shows the results of the same comparison on BG/P in SMP mode.

On BG/L the 8-cube shows an improvement of 30% on the average when compared to the default mapping that fills one physical row of the torus network. Using cubes as the MPI task mapping pattern on BG/P, we gain an improvement of about 75% over the default mapping which is an amazing performance boost.

Although the research reported in this paper investigated bandwidth utilization, overall performance depends on both bandwidth utilization and latency. In general, a mesh network as arranged on Blue Gene can provide high communication performance between neighboring nodes as close as possible to each other. However, a drawback is that performance can suffer if the application topology does not map well to the physical network topology [12]. The 8-cube ensures communication between nearest neighbors which is even more important on BG/P since the data injection and reception (and with it the bandwidth) have been more than doubled. Considering the route, messages have to travel between nodes furthest away from each other, then this route is even shorter for the 8-cube mapping than for the communication pattern in line. As the problem size grows, this low latency together with the high bandwidth becomes more profitable.

The maps for the rows and the columns of the two-dimensional virtual processor grid are not independent. The entire 3D-FFT algorithm requires information to be exchanged through the entire (partition of the) machine. Therefore, by selecting a good mapping between the virtual processor grid and the physical mesh, we can only improve the times for communication between either rows or columns but not both. While we have 8-cubes for the communication between rows, the customized mapping pattern for the communication between columns is non-contiguous and fills the entire partition.

On BG/L we have seen a performance degradation of less than 5% on the average for the non-contiguous mapping

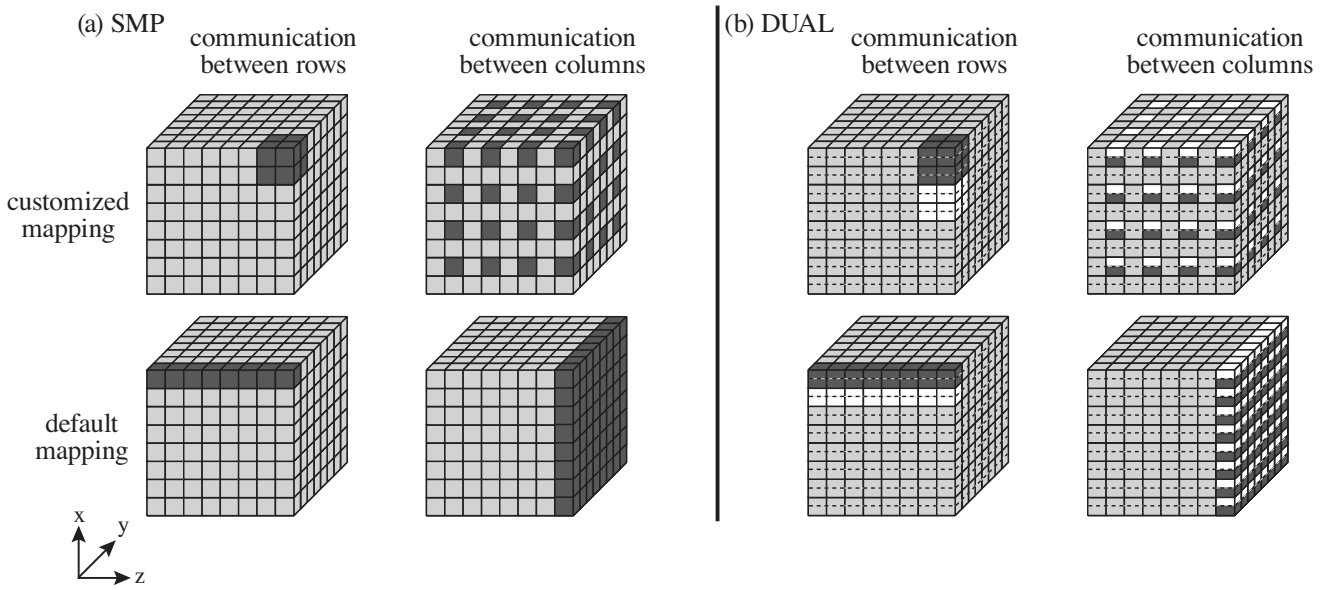

Figure 2: Customized vs default mapping on a 512-node partition with a division of processors in a Cartesian grid $P_r \times P_c$ on (a) 8×64 in SMP mode (CO mode on BG/L) and (b) 16×64 in DUAL mode

over the default mapping (see Figure 5 (c)). However, the gained improvement from the 8-cube mapping for the rows more than outweighs this performance loss so that for the overall result we were winning more than losing. We do not see the same on BG/P. When comparing the customized mapping with the default mapping for the communication between columns, we see a disconcerting performance degradation of more than 55% on average. This result illustrates that with the present system software on BG/P, the non-contiguous mappings badly damage performance.

4.3 Analysis of Average Bandwidth

After discussing the impact of the customized mappings on the overall and communication performance, we now address the question of how well these mappings utilize the bandwidth. This study helps to identify scaling bottlenecks of the different mapping patterns used in this paper. For this investigation, we calculate the average bandwidth utilization per wire B_l from the measured communication times $t_{r|c}$ by solving equation (2) for B_l and inserting the number of communicators $C_{r|c}$ and the number of wires at the bi-section $L_{r|c}$ from Table (2). A comparison to the hardware limit of 358 MB/s for B_l shows how well the bandwidth is utilized by the different mapping patterns.

$$B_l = \frac{D_T}{4 \cdot t_{r|c} \cdot C_{r|c} \cdot L_{r|c}} \quad (2)$$

	Communicators		Number of Links	
	$rows$	$cols$	$rows$	$cols$
8×64 default	64	8	2	16
8×64 customized	64	8	4	16
16×64 default	64	16	2	8
16×64 customized	64	16	4	8
32×64 default	64	32	2	4
32×64 customized	64	32	4	4

Table 2: Number of communicators $C_{r|c}$ and links at the bi-section $L_{r|c}$ for each of the different mappings

We denote the total amount of data involved in the 3D-FFT by D_T. The factor $\frac{1}{4}$ in equation (2) results from the fact that each part of the mesh holds $\frac{1}{2}$ of the data. In all-to-all communication each part has to keep half of its data and send half of its data to the other part. Hence $\frac{1}{4}$ data have to move from the first part of the mesh to the second and $\frac{1}{4}$ data have to move the other way.

Furthermore, the time for the communication between rows of the two-dimensional virtual processor grid is referred to as t_r while we use t_c for the communication time between columns.

Figures 6 (a) and (b) present the bandwidth utilization for the communication between rows of the two-dimensional virtual processor grid on BG/L and BG/P, respectively. The grid rows are mapped onto small and dense 8-cube patterns (as shown in Figure 2). The results for the communica-

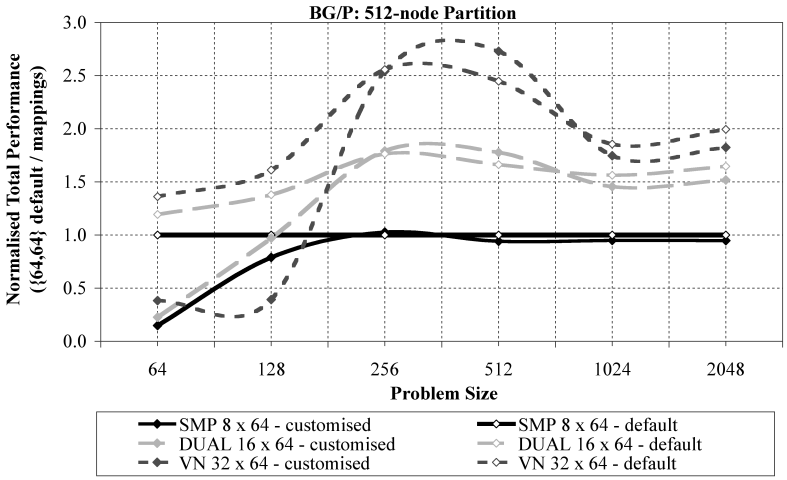

Figure 3: Comparison of the relative performance of the entire 3D-FFT algorithm on a 512-node partition when deploying 8-cube mappings

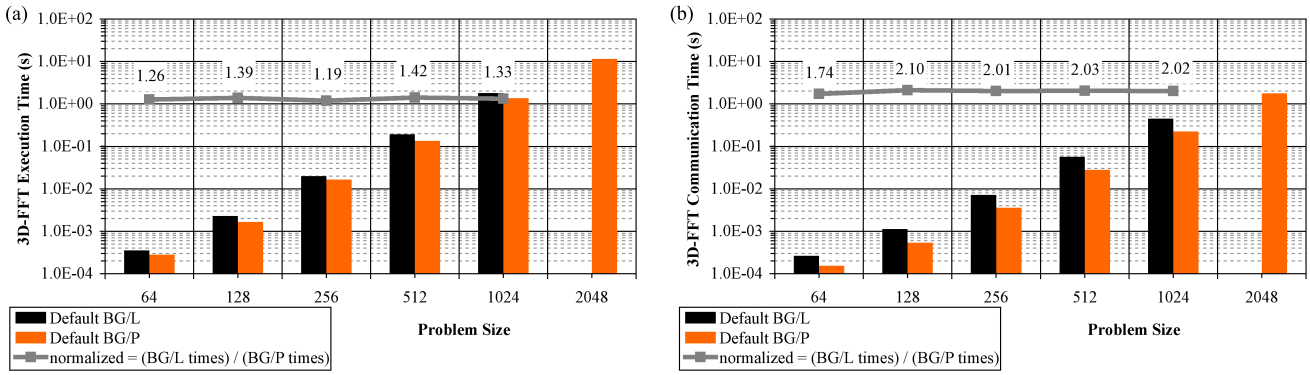

Figure 4: (a) Execution time of entire 3D-FFT computation – (b) Communication Time (two all-to-all type communications) on a 512-node partition with a division of processors in a Cartesian grid $P_r \times P_c$ of 8×64 in SMP mode (CO mode on BG/L)

tion between columns of the virtual processor grid are presented in Figure 6 (c) for BG/L and (d) for BG/P. These are mapped with a non-contiguous pattern, complementary to the 8-cube, with small gaps equally distributed over the entire 512-node partition. The two figures presenting the bandwidth utilization on BG/L contain the results for CO and VN mode. The results for BG/P are presented in the same way, but here we have three modes, SMP, DUAL, and VN mode as explained in Section 3.

We do not go into detail for the results of BG/L since those have been extensively discussed in [11]. However, those results provide a basis for understanding what impact the modernization of BG/P has on the 3D-FFT application.

Figure 6 (a), (b), and (c) demonstrate clearly the efficiency of the BG/L and BG/P systems with respect to the utilization of the bandwidth through the links of the bi-section. Given a large enough problem, for most of the investigated maps, the average bandwidth utilization is amazingly close to the hardware limit. The most important exceptions are 1) the performance for the 8-cube in CO mode on BG/L and 2) the non-contiguous mapping results of BG/P presented in Figure 6 (d). We will discuss these two exceptions in more detail.

8-cube BG/L: For the 8-cube in CO mode on BG/L the performance saturates at an average bandwidth per wire of around 90 MB/s, which is substantially below the hardware limit. By contrast, in VN mode the performance is close to the hardware limit and the 8-cube delivers a sizable boost to the performance. In contrast, BG/P achieves a performance close to the hardware limit, independent of the three modes.

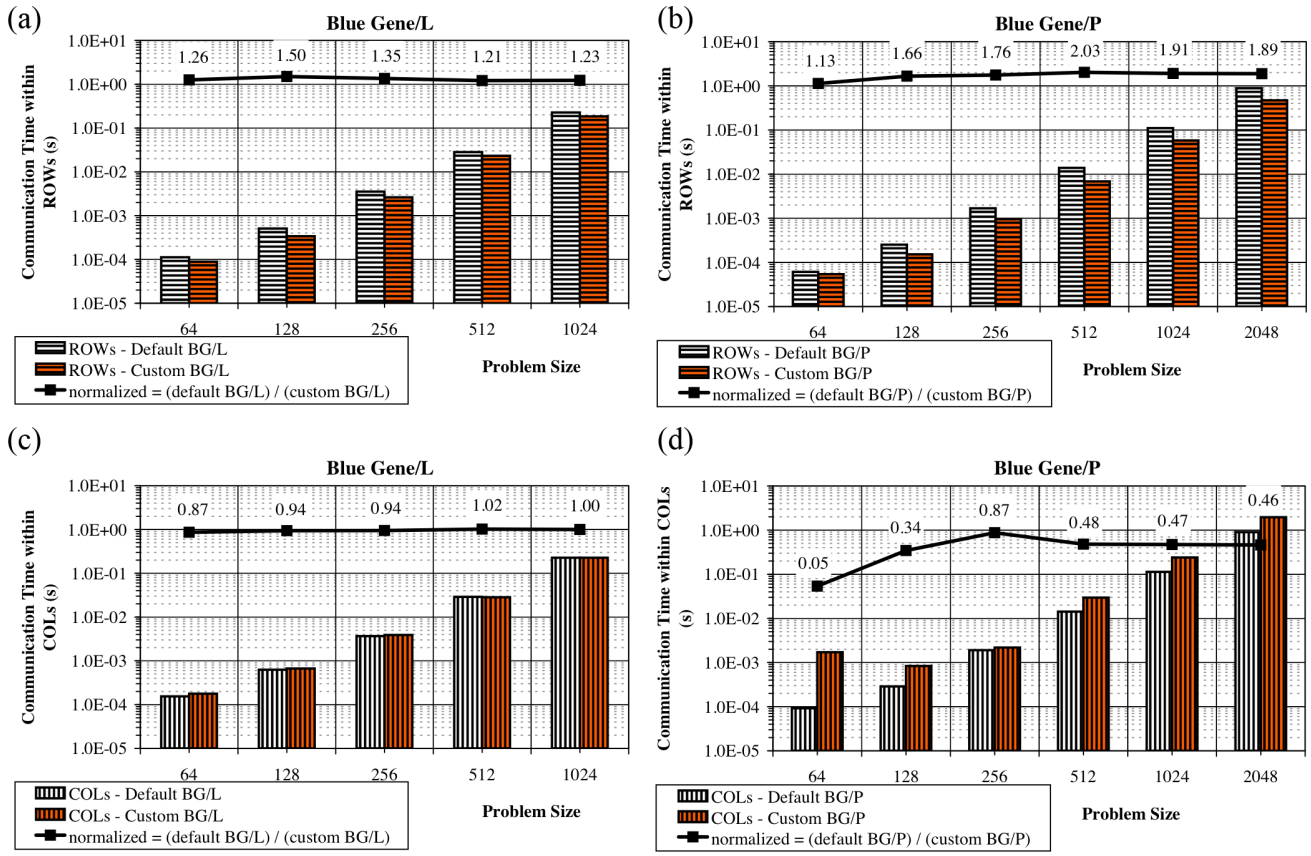

Figure 5: Individual Communication time between rows and columns of the 2D virtual processor grid (BG/L vs BG/P)

We can only speculate about the reasons for this behavior on BG/L in CO mode. For the 8-cube the ratio of links at the bi-section to processors is very large. The compute power of a single core in CO mode might be insufficient to simultaneously manage the overheads of the MPI call and the insertion of the data into the network. Outsourcing some of this work to the second core, which is supposed to act as a communication co-processor, is known to be difficult due to the lack of cache coherency between L1-caches on the node. In VN mode, when each core manipulates its own private data per node, the problems associated with the lack of cache coherency goes away. The same hardware as in the case of the CO mode is now capable of saturating the links of the bi-section. On BG/P cache coherency is no longer managed by software but is handled in hardware (symmetrical multiprocessing) [9, 13]. Furthermore, the compute power on a single core has increased compared to BG/L, so that in fact, we do not experience this behavior for the 8-cube in SMP mode on BG/P.

non-contiguous mapping on BG/P: The second exception that does not show a bandwidth utilization close to the hardware limit is the non-contiguous mapping on BG/P. We have seen from the results presented in Figure 5 (d) that non-contiguous mappings are more damaging on BG/P than was the case on BG/L. Investigations of the bandwidth utilization are expected to clarify this behavior.

Increasing the problem size causes the bandwidth utilization to increase and to reach a maximum at a certain point. After this, a heavy drop in bandwidth utilization and performance occurs. We would like to point out once more that the non-contiguous mapping pattern for one communicator as shown in Figure 2 is displaced seven times to fill the entire 512-node partition. So in total, there are eight different communicators communicating concurrently and they are intermixed.

The reason for this might be that the different communicators - mapped in a non-contiguous fashion - disturb each other. To investigate whether the performance drop is due to mutual disturbance of the communicators, we ran a 2D-FFT computation on the 512-node partition with only one communicator. This communicator is mapped in a non-

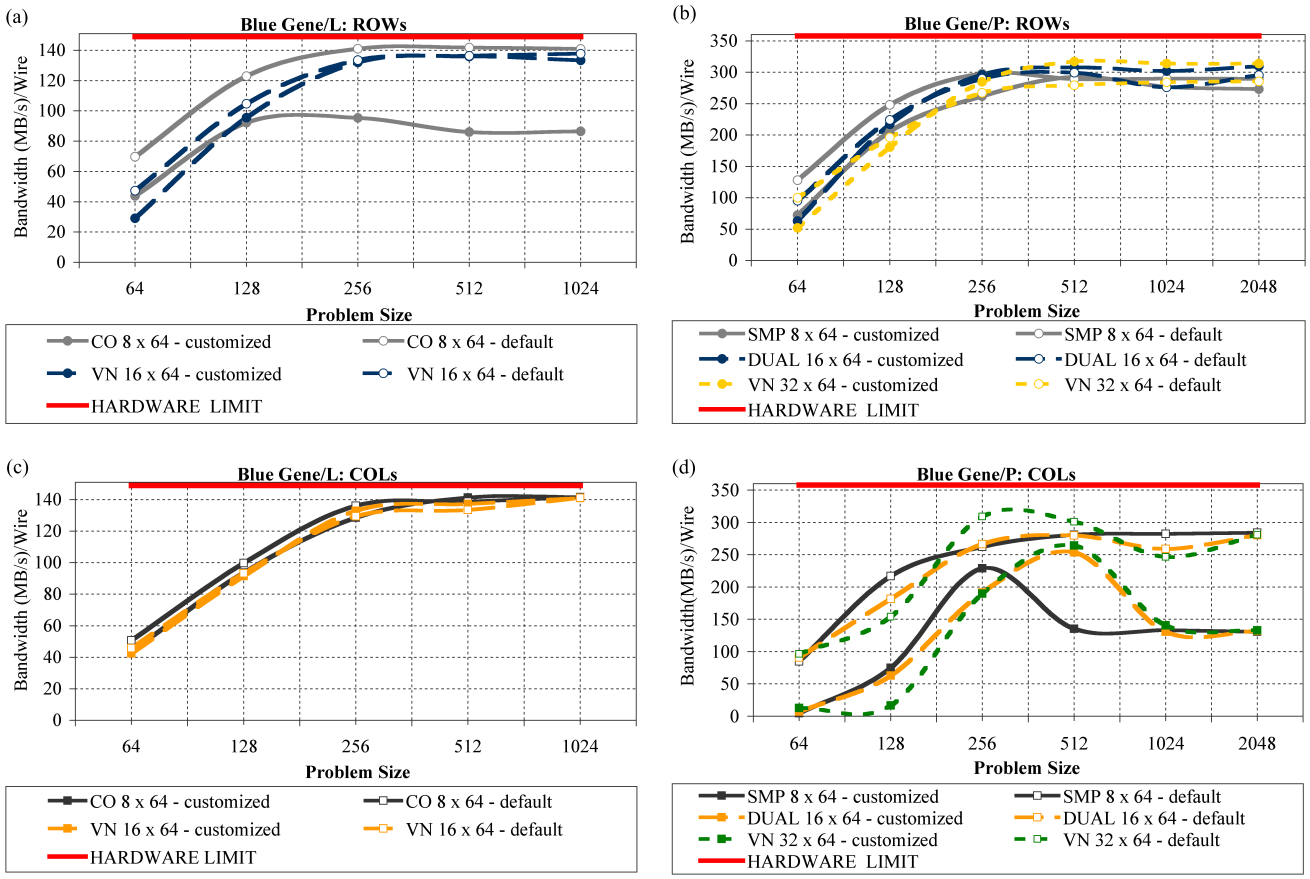

Figure 6: Average bandwidth utilization per link of the bi-section for the communication between rows (a, b) and columns (c, d) (BG/L vs BG/P)

contiguous way as shown in Figure 2 (a). More precisely, we used only 64 nodes of the entire 512-node partition, so that this time no other communicators disturb communication. Again, this investigation has been carried out on BG/P as well as BG/L. Figure 7 depicts the average bandwidth utilization per link on (a) BG/L and (b) BG/P.

On BG/L we see a poor performance when using only one communicator. On the other hand, using eight concurrent communicators performs similar to the default mappings and comes reasonably close to the hardware limit. The results for the customized mappings on BG/P are similarly poor, independent of the use of one communicator or eight concurrent communicators. From this it can be concluded that the poor performance has a geometry reason rather than mutual disturbance issues. As an example, the BG/P performance for mid-size problems such as 128^3 and 256^3 is inferior to what we see on BG/L. BG/P with the present system software - as the latest release of the parallel Blue Gene supercomputer line - does not manage to achieve the same performance as its forerunner BG/L.

As mentioned in section 3.2, a DMA engine has been added to offload data packets injection (or reception) from the processor cores. It seems that this new piece of hardware is responsible for the performance difference we experience between BG/L and BG/P results. It is very likely that the system software - in particular the all-to-all call inside the MPI library - is not fully tuned yet to the new hardware. We did not see this sort of problem on BG/L.

The most important lesson learned from this section is that mapping MPI tasks onto the physical mesh in a non-contiguous way is extremely damaging for the performance. With this result in mind, we know that on large partitions, such as a $16 \times 16 \times 16$ partition, even the default mapping is no longer contiguous. In the next section we will discuss how a carefully chosen customized mapping pattern can yield significant improvements.

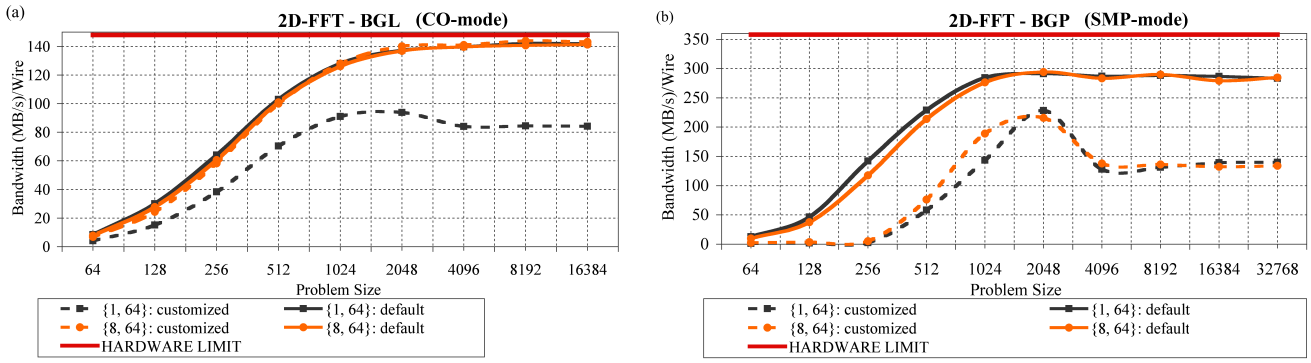

Figure 7: Average bandwidth utilization per link of the bi-section for the communication of a 2D-FFT computation on (a) BG/L and (b) BG/P in SMP-mode

5 Experimental Investigations on the 16^3-node partition on BG/P

In this section, we investigate a $4 \times 4 \times 4$ cube mapping pattern for a $16 \times 16 \times 16$ partition on BG/P which is in fact the small cube mapping on this large partition proportional to the $2 \times 2 \times 2$ cube on the 512-node partition. In the remainder of this paper, we call the customized $4 \times 4 \times 4$ cube mapping for the communication between rows a "64-cube". Figure 8 (a) depicts the 64-cube mapping and its corresponding mapping pattern for the communication between columns using SMP mode on BG/P. The default mapping is presented in part (b) of this Figure. In addition to the mappings in SMP mode, we investigate mappings in DUAL and VN mode which show the same shape for the customized and default cases. We do not depict DUAL and VN mode mappings since they are similar to the other mappings except that they use two MPI tasks per node in DUAL mode and four MPI tasks per node in VN mode (instead of one MPI task in SMP mode).

Similar to Figure 2, these figures here depict only the basic image of one communicator. The full map for all rows and columns of the virtual processor grid is constructed by using displacements of the basic image across the entire physical 3D torus network. More precisely, in Figure 8 (a) the 64-cube is displaced 64 times to fill the entire partition. The same applies to all other mapping patterns.

As mentioned earlier, when a certain partition size is reached, the default case is no longer able to map MPI tasks in a contiguous way onto the network. We will consider how much we can profit by using customized mappings instead. First, we will have a look at the results for the overall performance of the entire 3D-FFT computation using customized mapping instead of the default choice. Secondly, we look deeper into detail at the communication part to identify from where performance variances accrue.

5.1 Analysis of Overall Performance

Figure 9 presents the influence of the 64-cube mapping on the total performance of the entire 3D-FFT algorithm. To obtain a more readable figure, the results have been normalized to the performance of the default mapping. More precisely, the performance result of the default mapping for the 64×64 virtual processor grid in SMP mode has been divided by the results of all other mappings. The measured times for the overall performance can be found in Appendix A. For small problems such as 64^3 and 128^3 about 93% of the total execution time is spent in communication. For all the other investigated problem sizes this percentage is about 35% on a sustained basis. For those small problems and hence small messages, the VN mode does not show any benefits. The SMP mode is more efficient for small problems since the shorter time spent in communications cancels out the advantage of using more processors for the computations. Aside from that, in some cases, for DUAL and VN mode the problem size is too small to divide the problem between the number of processors. For large problem sizes, DUAL mode and, even more, VN mode become attractive, independent of whether the customized or the default mapping is used.

We observe even for SMP mode an overall performance improvement for the dense 64-cube over the non-contiguous default mapping of about 10% for mid-size problems such as 512^3 and 1024^3. A much superior improvement can be achieved in DUAL and VN mode. Here for mid-size problems the entire 3D-FFT application runs about 70% faster in DUAL mode, while in VN mode the computation is more than twice as fast. For the largest problem investigated in

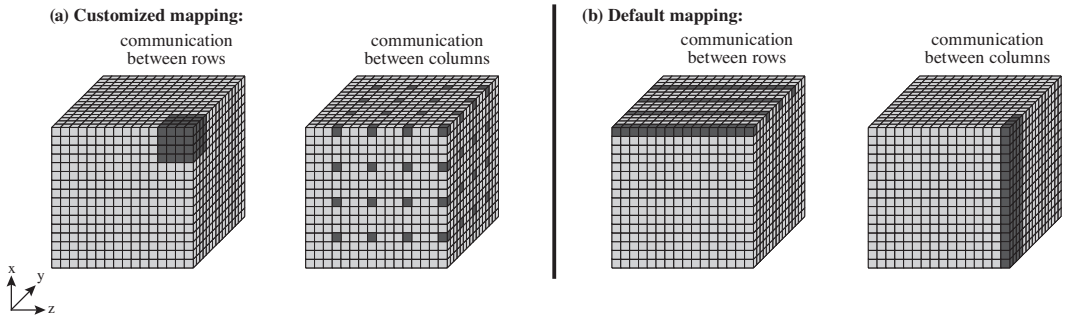

Figure 8: (a) Customized vs (b) default mapping on a 4096-node partition in SMP mode with a division of processors in a Cartesian grid $P_r \times P_c$ of 64×64

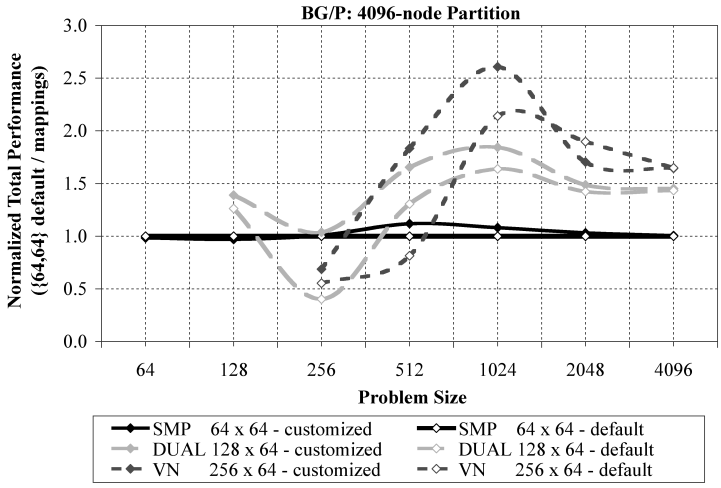

Figure 9: Comparison of the relative performance of the entire 3D-FFT algorithm on a 4096-node partition when deploying 64-cube mappings

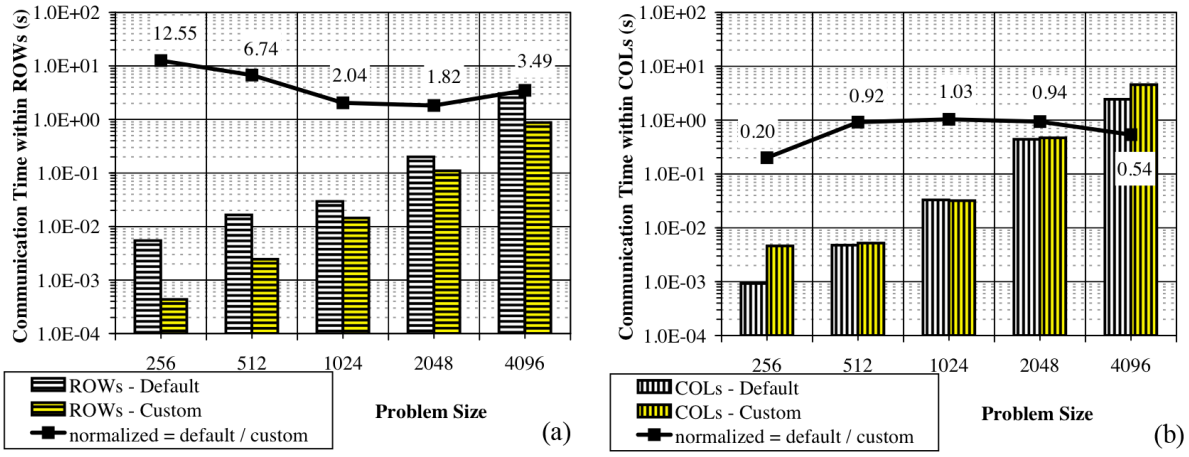

Figure 10: Individual Communication time between rows (a) and columns (b) with a division of processors in a Cartesian grid $P_r \times P_c$ of 256×64 (in VN mode)

this paper, with a problem size of 4096^3, the performance for default and customized mappings is basically the same.

5.2 Analysis of Individual Communication Times

The default mapping for the communication between rows is non-contiguous. However, it is interesting to note that the corresponding mapping pattern for the customized 64-cube is also non-contiguous. By looking at the performance for each of the two all-to-all type communications individually, we can determine from where the improvement for the entire 3D-FFT computation as discussed above accrues. As shown in Section 5.1, the best performance can be achieved by using a customized mapping in VN mode on BG/P. On this account, we consider the times for communication between rows and columns individually for the VN mode example. More precisely, in VN mode we have a division of processors in a Cartesian grid $P_r \times P_c$ of 256×64.

64-cube for ROWs: In Figure 10 (a) we compare the communication times between rows of the default mapping with those of the customized mapping. We learned in Section 4.2 that the 8-cube offers an amazing performance boost of about 75% over the default mapping in a line. We see the same behavior for the 64-cube on the large partition. Here we are even four times faster on average which is a terrific enhancement. This extreme improvement is because of ensuring communication between processors as close as possible to each other compared to the non-contiguous mapping we get for the default case.

However, the excellent performance we experience for mid-size problems decreases by further increasing the problem size. It seems likely that the bandwidth utilization for the communication within the 64-cube is affected by hot spots in the middle of the bi-section. Assuming message routing along the shortest path, there are more sender-receiver pairs for which message routing through the center of the bi-section is among the shortest paths than there are pairs for the links at the corners of the bi-section. Nevertheless, the cube-shaped mapping is still more than twice as fast on average for large problem sizes than the default case.

64-cube complementary mapping for COLUMNs: In Figure 10 (b) we compare the communication times between columns for the default mapping with those for the customized mapping. The customized maps, complementary to the 64-cube, are non-contiguous with gaps equally distributed over the entire 4096-node partition, enabling torus connectivity in all three dimensions. The default case maps the MPI task onto four lines next to each other, which means that there is torus connectivity in only one direction. Both mappings are depicted in Figure 8 for SMP mode. However, since we investigate VN mode, it is important to note that the one basic image for the communication between columns is now shared by four communicators.

For the non-contiguous customized mapping we see a performance degradation of less than 30% on average. This was expected from the lesson learned on the 512-node partition. However, we can draw conclusions that the 64-cube for the communication between rows is good enough to cancel out the performance degradation we get from the non-contiguous mapping for the communication between columns. For large problem sizes such as 4096^3, the performance for the non-contigous mappings spirals downward. This is the reason that for such a large problem the overall performance for default and customized mappings is basically the same as mentioned above.

6 Conclusions and Future Work

This paper investigates the potential performance benefit from MPI task placement for the volumetric Fast Fourier Transformation on a modern massively parallel machine with a meshed or toroidal communication network. Earlier investigations performed on BG/L and performance models have clearly shown that the best performance can be achieved by using cube-shaped mapping patterns for either rows or columns of the two-dimensional virtual processor grid. Our experimental results show that performance benefits of more than 30% on average for the entire 3D-FFT algorithm are possible when using cube-shaped mappings on a 4096-node partition in DUAL or VN mode on BG/P. For mid-size problems such as 256^3 or 512^3 the entire 3D-FFT application runs more than twice as fast. As an example, the observed performance increase of the communication part due to customized MPI task placements is as large as 178% for a 512^3 problem running in VN mode on BG/P. The reason for this performance boost is that on such a large partition, there is a high potential that the default mapping is also non-contiguous. Non-contiguous mapping patterns badly damage performance on BG/P compared to BG/L. There could be several factors for this, however the newly added DMA engine on BG/P or the system software are probably the major contributing factors. Particularly, $MPI\_Alltoall$ operations do not appear to be well-tuned for the new system configurations. Future work will entail more detailed investigations of the DMA engine as well as the algorithm used for data routing.

Acknowledgements

The authors would like to thank the Jülich Supercomputing Centre for access to Blue Gene/P and sustained support. Furthermore, Shirley Moore (UTK) and Sadaf Alam (ORNL) are greatly acknowledged for reading and commenting on draft versions of this paper.

A Performance results for 3D-FFT computation

The following table gives the total times used by our benchmark for a forward transformation on a 512-node partition on the Jülich Supercomputing Centre Blue Gene/P.

Problem size:	64^3	128^3	256^3	512^3
SMP 8×64 cust.:	1.897 ms	2.108 ms	16.162 ms	144.370 ms
SMP 8×64 def.:	0.283 ms	1.664 ms	16.581 ms	136.015 ms
DUAL 16×64 cust.:	1.244 ms	1.713 ms	9.274 ms	76.504 ms
DUAL 16×64 def.:	0.237 ms	1.205 ms	9.419 ms	81.794 ms
VN 32×64 cust.:	0.740 ms	4.243 ms	6.532 ms	49.86 ms
VN 32×64 def.:	0.208 ms	1.032 ms	6.496 ms	55.56 ms

Problem size:	1024^3	2048^3
SMP 8×64 cust.:	1.44 s	12.08 s
SMP 8×64 def.:	1.37 s	11.45 s
DUAL 16×64 cust.:	942.60 ms	7.54 s
DUAL 16×64 def.:	877.88 ms	6.95 s
VN 32×64 cust.:	785.22 ms	6.27 s
VN 32×64 def.:	738.55 ms	5.74 s

The next table gives the total times used by our benchmark for a forward transformation on a 4096-node partition on the Jülich Supercomputing Centre Blue Gene/P.

Problem size:	64^3	128^3	256^3	512^3
SMP 64×64 cust.:	0.526 ms	2.096 ms	3.667 ms	17.69 ms
SMP 64×64 def.:	0.517 ms	2.034 ms	3.692 ms	19.79 ms
DUAL 128×64 cust.:	—	1.464 ms	3.557 ms	11.97 ms
DUAL 128×64 def.:	—	1.612 ms	9.159 ms	15.15 ms
VN 256×64 cust.:	—	—	5.368 ms	10.80 ms
VN 256×64 def.:	—	—	6.680 ms	24.31 ms

Problem size:	1024^3	2048^3	4096^3
SMP 64×64 cust.:	174.29 ms	1.82 s	15.56 s
SMP 64×64 def.:	188.51 ms	1.88 s	15.66 s
DUAL 128×64 cust.:	102.28 ms	1.26 s	10.79 s
DUAL 128×64 def.:	115.01 ms	1.32 s	10.93 s
VN 256×64 cust.:	72.28 ms	1.10 s	9.52 s
VN 256×64 def.:	88.08 ms	0.99 s	9.50 s

References

[1] M. Eleftheriou, et al., "A Volumetric FFT for BlueGene/L", in G. Goos, J. Hartmanis, J. van Leeuwen, editors, volume 2913 of Lecture Notes in Computer Science, page 194, Springer-Verlag, 2003.

[2] M. Eleftheriou, et al., "Performance Measurements of the 3D FFT on the Blue Gene/L Supercomputer", J.C. Cunha and P.D. Medeiros (Eds.): Euro-Par 2005, LNCS 3648, page 795, 2005.

[3] S. Alam, et al., "Performance Characterization of Molecular Dynamics Techniques for Biomolecular Simulations", PPOPP'06, New York City, New York, USA, 2006.

[4] H. Jagode, "Fourier Transforms for the BlueGene/L Communication Network", MSc thesis, The University of Edinburgh, 2006, http://www.epcc.ed.ac.uk/msc/dissertations/2005-2006/

[5] `http://www.top500.org`

[6] Joachim Hein, Heike Jagode, Ulrich Sigrist, Alan Simpson, Arthur Trew, "Parallel 3D-FFTs for multicore nodes on a mesh communication network", Paper presented at CUG 2008, Helsinki, Finland, May 5-8, 2008

[7] FFTW Homepage, http://www.fftw.org

[8] J. Lorenz, S. Kral, F. Franchetti, C.W. Ueberhuber., "Vectorization techniques for the Blue Gene/L double FPU", IBM Journal of Research and Development, Vol. 49, page 437, 2005

[9] IBM BG team, "Overview of the IBM Blue Gene/P project", IBM Journal of Research and Development, Vol. 52, page 199, 2008.

[10] N. R. Adiga, et al., "Blue Gene/L torus interconnection network", IBM Journal of Research and Development, Volume 49, page 265, 2005.

[11] H. Jagode, et al., "Task placement of parallel multidimensional FFTs on a mesh communication network", University of Tennessee Knoxville, Technical Report No. ut-cs-08-613, 2008, http://www.cs.utk.edu/ library/2008.html

[12] D. Kerbyson, et al., "Performance Modeling of the Blue Gene Architecture", IEEE International Symposium on Modern Computing (JVA'06), 2006.

[13] C. P. Sosa, "IBM System Blue Gene Solution: Blue Gene/P Application Development", IBM Redbook, 2007, http://www.redbooks.ibm.com/abstracts/sg247287.html