Task placement of parallel multi-dimensional FFTs on a mesh communication network

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

Joachim Hein, Arthur Trew
Edinburgh Parallel Computing Centre (EPCC)
The University of Edinburgh
{joachim | arthur}@epcc.ed.ac.uk

Abstract

For many scientific applications, the Fast Fourier Transformation (FFT) of multi-dimensional data is the kernel which limits scalability to large numbers of processors. This paper investigates an extension of a traditional parallel three-dimensional FFT (3D-FFT) implementation. The extension within a parallel 3D-FFT consists of customized MPI task mappings between the virtual processor grid of the algorithm and the physical hardware of a system with a mesh interconnect. Consequentially, we derived a simple model for the scope of performance of a large class of mappings on the basis of bandwidth considerations. This model enables us to identify scaling bottlenecks and hotspots of parallel, communication intensive 3D-FFT applications when MPI tasks are mapped in the default way onto the network. The predictions of the model are tested on an IBM eServer Blue Gene/L system. The results demonstrate that a carefully chosen mapping pattern with regards to the network characteristics yields significant improvement.

1. Introduction

The recent growth in performance of the fastest supercomputers in the world have been largely facilitated by an increasing number of processors utilized by the system. We expect this development together with the current trend of multiple processing cores on a single chip to continue over the next few years. This has important consequences for application developers: Efficiently utilizing a system with several hundred or thousand processors places high demands on the scalability of the application.

For many scientific applications, parallel multi-dimensional Fast Fourier Transformation (FFT) routines form the key performance bottleneck which 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.

Acknowledged 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 only 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 an inferior performance when compared to an implementation using a one-dimensional virtual grid. However this algorithm offers superior scalability, even to processor counts where a one-dimensional grid can no longer be employed [1, 13].

Another current trend in supercomputer design is the return of the mesh type communication network. The systems on the Top500 list [4] utilizing more than 20000 processors, arrange their processing chips 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. Out of it, a simple

model for the performance of a large class of mappings has been derived. This performance modeling is based on bandwidth considerations and enables us to identify scaling bottlenecks and hotspots of the parallel 3D-FFT application.

This paper is organized as follows. The next Section summarizes the relevant work on parallel FFTs. Section 3 reviews the implementation of the parallel 3D-FFT algorithm with a two-dimensional data decomposition. In Section 4 the expected performance of this algorithm on a large class of possible MPI task mappings is discussed from a bandwidth point of view. This identifies a number of promising mapping patterns with respect to performance improvements, which will be examined further in an experimental study. A short overview of the Blue Gene/L system used for this study is provided in Section 5, and Section 6 summarizes the details of the benchmark application. The results of the experimental study are presented and discussed in Section 7. The paper ends with the conclusions.

2 Related research

There is a broad literature on different aspects of parallel FFT implementations because of the tremendous importance of this kernel in many scientific applications. The parallelization of the one-dimensional FFT kernel has drawn the high attention of many library developers and programmers. This is a significant investigation since the recent generations of microprocessors have basically stopped due to physical limits. One consequence is that chip makers integrate multiple processor cores onto a single chip. On this account, Fast Fourier Transform algorithms suitable for shared memory processing (SMP) and multi-core architectures have been derived in [5]. The results presented in [5] show that the parallelization of one-dimensional FFTs for SMPs and multi-core systems is useful.

On this basis, a further investigation is presented in [6] where heuristics are developed for the parallelization of FFT schedules on SMP and multi-core systems. The FFT schedule computation is an empirical optimization technique that is successfully used by FFTW to generate a highly optimized library. The approach generates a large number of code variants with different parameter values. All those candidates run on the target machine and the one that gives the best performance is chosen.

Some other techniques have been investigated by several groups, for instance (1) to improve data locality of a one-dimensional FFT [7]; (2) development of high performance one-dimensional FFT kernels optimized for certain processor designs such as Blue Gene PowerPC 440 with its two floating point units that execute fused multipy-add instructions [8]; (3) a no-communication algorithm that is a parallel algorithm for a one-dimensional FFT without interprocessors communication which performs good for small problem sizes rather than mid-size or large problems [9].

Those above mentioned investigations out of a vast literature on FFTs are all valuable with respect to the one-dimensional FFT algorithm. Another matter of fact for the efficient utilization of supercomputers is a neat mapping of MPI tasks onto the physical network to achieve optimal load balance of the data and to minimize communication time [10]. Different MPI task mappings for the Qbox application have briefly been explored in [11]. The Qbox application implements First-Principle Molecular Dynamics, an accurate atomic simulation approach. The results presented in [11] show that the task layout choice can significantly impact the performance.

Another important area of application for multi-dimensional FFTs is three-dimensional turbulence. Turbulent flows can be found amongst others in stellar physics or atmospheric and oceanographic science [12]. The crossover from three- to two-dimensional turbulence is based on cascade models which are derived from the Fourier space formulation of the Navier-Stokes equations of motion. In [12] different FFT packages have been compared together with replacing MPI tasks by using the environment variable `BGLMPI_MAPPING`. The results show that for more than 256 cores, using a customized MPI task layout brought the shortest execution time.

All these examples reveal that an MPI task layout choice depends heavily on the application and also on the size of the application as clearly shown in [10]. The purpose of this paper is to investigate different MPI task mappings between the virtual processor grip of the implemented three-dimensional FFT algorithm and the physical hardware of the system; and from this outcome to derive a theoretical model for the performance of a large class of mappings. Since many different scientific applications rely on a large number of FFTs, the optimization for this computationally expensive kernel can be invoked from those applications.

3 Review of parallel FFT algorithms

3.1 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:

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

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

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

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.

3.2 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, 13]. 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 of the MPI task placements of the two-dimensional decomposition method onto a mesh communication network. Reference [13] 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 [14]. 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 now performed in 5 steps

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.
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 communication 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, 13].

4 All-to-All transformations on meshed network

4.1 Virtual processor grid and physical processor mesh

The key point of this paper is to investigate how the performance of the 3D-FFT can be influenced by the choice of MPI task mapping between the virtual processor grid and the physical processor mesh or torus of the machine. We assume a (partition of the) system of cuboidal shape in 3 dimensions with a physical mesh of processors sized $n_x \times n_y \times n_z$. It is absolutely crucial not to confuse this physical mesh with the virtual processor grid of dimension $P_r \times P_c$. We denote the total number of processors with P and if we use all processors available for the 3D-FFT, we get

$$P = n_x\, n_y\, n_z = P_r\, P_c\,. \tag{2}$$

If we denote the total amount of data involved in the 3D-FFT by D_T and assume this data can be evenly divided onto the processors, the amount of data D_r held by each row of the virtual processor grid becomes

$$D_r = \frac{D_T}{P_r} = P_c\frac{D_T}{P}\,. \tag{3}$$

Each of the P_r All-to-All transformations in the second step of the algorithm needs to redistribute this amount of data. A similar equation holds for the second All-to-All transformation in the fourth step.

In the remainder of this section, we discuss how the mapping of the rows of the virtual processor grid onto the physical processor mesh impacts the interconnect bandwidth available to each individual row. Since the map between the virtual grid and the physical mesh has to place all the grid rows simultaneously and the performance of the worst performing row will determine the overall performance, we restrict ourselves to maps which obey certain symmetries. The symmetries protect against unequal performance across different rows and make it easier to fill the entire physical mesh of the machine by applying a displaced version of the same basic map for each of the rows.

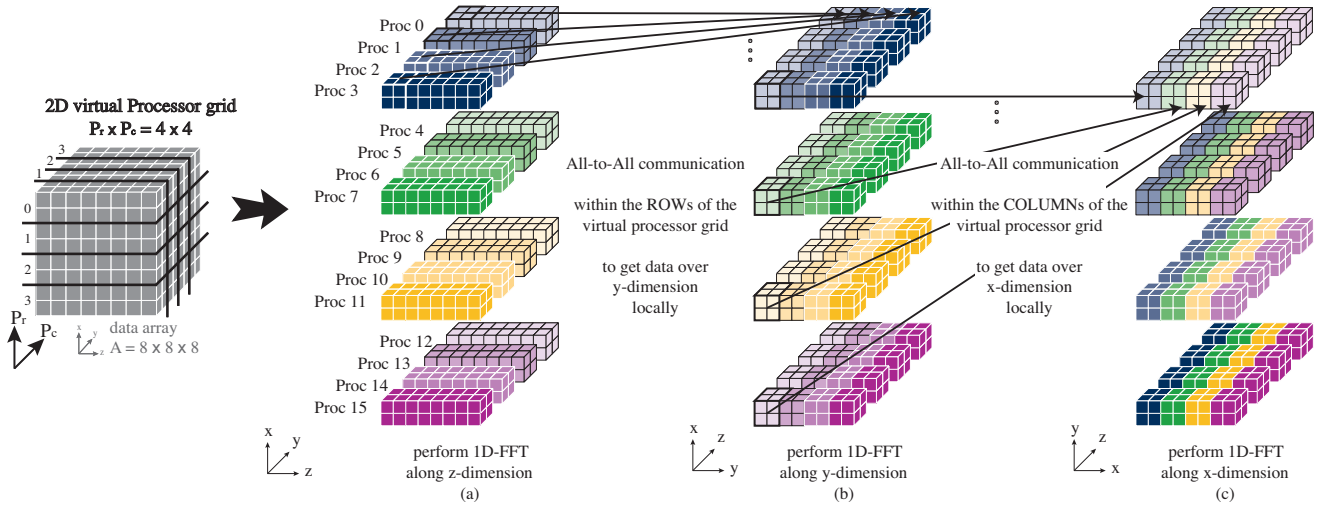

Figure 1: Computational steps of the 3D-FFT implementation using 2D-decomposition

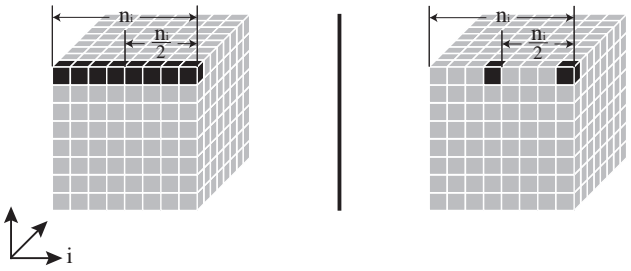

Figure 2: Example mappings of virtual grid rows with a $\frac{n_i}{2}$ translation symmetry

4.2 Images with an internal translation symmetry

The following discussion applies to all three directions x, y and z of the physical mesh in a similar fashion. Hence, we can restrict ourselves to a single direction which we call the i-direction. The extent of this direction is n_i. Let us consider maps whose images of the virtual grid rows have an internal translation symmetry of $\frac{n_i}{2}$ or are a union of such images. The translation symmetry (also called sliding symmetry) associates the same shapes facing in the same direction. This translation symmetry assumes periodic boundary conditions with a period of n_i, irrespective of properties of the physical network. More precisely, the shapes are replicated with a period of n_i to infinity by rigid translation in the i-direction. The obvious example for such an image is a row of the virtual grid being mapped onto a number of full rows of the physical mesh, which are parallel to the i-direction. Another example would be two processors, one placed at position one of the row, the other at position $\frac{n_i}{2}+1$ of the row. These examples are illustrated in Figure 2. A full map for all the rows of the virtual processor grid can be constructed by using displaced versions of this basic image.

The first All-to-All communications of the 3D-FFT has to take place internally to each individual image of the virtual rows, with no communication between images of different virtual rows.

Let us now consider the bi-section. In case of an open mesh, we insert a plane orthogonal to the i-direction in the middle of the i-direction. In case of a toroidal communication network we need to insert two such planes, separated by $\frac{n_i}{2}$. In either case, we will have half of the processors of the virtual row in each part. This will hold for all virtual rows, if their images are displaced versions of the same basic image. Since we assume an even distribution of the data D_r onto all the processors of the virtual row, each part of the mesh will hold $D_r/2$. In the All-to-All communication each part has to keep half of its data and send half of its data to the other part. Hence $D_r/4$ data have to move from the first part of the mesh to the second and $D_r/4$ data have to move the other way. Considering the bi-sectional bandwidth available for each individual image will give us a lower bound on the communication time.

The total bi-sectional bandwidth through the above plane(s) is the bandwidth B_l of an individual link multiplied with number of links which is the area A_i, which is the number of processors in the plane(s). In case of a toroidal network this has to be multiplied by a factor of two. We introduce a torus-factor g_t with

$$g_t = \begin{cases} 1 \text{ mesh network} \\ 2 \text{ toroidal network} \end{cases} \tag{4}$$

We get a total bi-sectional bandwidth $B_{T,i}$ associated with the direction i of

$$B_{T,i} = A_i \, g_t \, B_l \, . \tag{5}$$

Since the bi-section intersects all the images of the virtual grid-rows, $B_{T,i}$ gets divided evenly between the rows of the

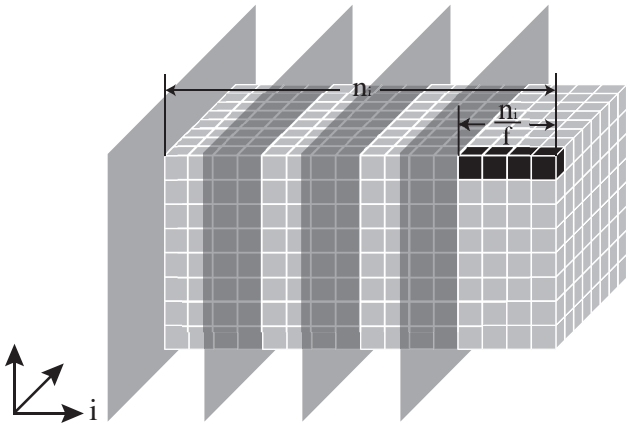

Figure 3: Example mapping of a virtual grid row, with an extent of n_i/f_i and a translation symmetry of $n_i/(2f_i)$

virtual processor grid. For the bandwidth $B_{r,i}$ available for each individual virtual grid row we get

$$B_{r,i} = \frac{B_{T,i}}{P_r} = \frac{A_i g_t B_l}{P} P_c \tag{6}$$

From this we get a time $t_{r,i}^{(f)}$, which is the minimum time required for the data transfer in the i-direction through the bi-section of

$$t_{r,i}^{(f)} = \frac{\frac{1}{4} D_r}{B_{r,i}} = \frac{D_T}{4 A_i g_t B_l} = \frac{n_i D_T}{4 P g_t B_l} \,. \tag{7}$$

The last step uses $A_i n_i = P$.

4.3 Images spanning fractions of directions

We now consider the situation that f_i planes orthogonal to the i-direction are placed inside the physical processor mesh, with f_i being an integer larger than one. These planes are spaced at regular intervals n_i/f_i. We assume the basic image of an individual row of the virtual processor grid can be placed inside the space between two neighboring planes and has an internal translation symmetry of $n_i/(2f_i)$ with respect to a period of n_i/f_i. This is illustrated in Figure 3. Also here the translation symmetry (or sliding symmetry) involves the same shapes facing in the same direction. The full map for all rows is now constructed by using displacements of the above basic image across the entire physical mesh of the hardware.

Assuming efficient message routing along the shortest path, no message will cross any of the f_i planes, since this would result in a longer than necessary route. We can therefore regard the first set of All-to-All transformations within the rows of the virtual grid as being executed independently on f_i independent machines. Each of these f_i machines has an extent of its physical mesh of

$$n_i' = \frac{n_i}{f_i} \,. \tag{8}$$

The crucial observation is that each of these machines now holds

$$D_T' = \frac{D_T}{f_i} \tag{9}$$

data. Since no message crosses any of the f_i planes, we always have an open mesh and

$$g_t' = 1 \,. \tag{10}$$

So far we have not split any of the directions orthogonal to the i-direction, which keeps the area of the bi-section unchanged

$$A_i' = A_i \,. \tag{11}$$

Inserting everything into a formula similar to equation (7), we obtain

$$t_{r,i}^{(p)} = \frac{D_T'}{4 A_i' g_t' B_l} = \frac{D_T}{4 f_i A_i B_l} = \frac{n_i D_T}{4 f_i P B_l} \,. \tag{12}$$

Again, for the last step, we have used $n_i A_i = P$. When compared to equation (7), this is an improvement of f_i/g_t.

Obviously this formula will not hold for $n_i = f_i$, since this case does not have an internal translation symmetry of $n_i/(2f_i) = 1/2$. For $n_i = f_i$ all processors belonging to the image of a single row of the virtual grid are within a single layer of the physical mesh. This layer is orthogonal to the i-direction. There are no communication requirements in the i-direction in this case and the associated time is zero

$$t_{r,i}^{(1)} = 0 \,. \tag{13}$$

4.4 Communication time for a given map

After discussing the time constraint associated with the data transfer in a particular direction for three different symmetry classes, we continue with a discussion of the total communication time of the entire parallel 3D-FFT algorithm. The total communication time t is the sum of the times t_r and t_c for the data exchange within the rows and the columns of the virtual processor grid

$$t = t_r + t_c \,. \tag{14}$$

The times t_r and t_c can not be shorter than the largest of the times required for the data transfer through the bi-sections

$$t_r \geq \max_i \left(t_{r,i} \right) \,, \qquad t_c \geq \max_i \left(t_{c,i} \right) \,. \tag{15}$$

We restrict the following discussion to maps for which for each of the $t_{r,i}$ and $t_{c,i}$ either of the equations (7), (12) and (13) can be applied. We now investigate which maps from this group are marked out as particularly efficient by our

model. To do so, we rewrite the right hand side of equation (15) as

$$t_r \geq \max_i (m_i) \frac{D_T}{4PB_l} \quad (16)$$

$$m_i = \begin{cases} \frac{n_i}{g_t} \text{ if equation (7) applies} \\ \frac{n_i}{f_i} \text{ if equation (12) applies} \\ 0 \text{ if equation (13) applies} \end{cases}$$

A similar equation holds for t_c. The m_i can be regarded as the effective length of the image in the i-direction. To optimize the performance we have to aim to get the largest of the m_i as small as possible. This can typically be achieved by removing holes from the basic row images or by increasing the m_i in either of the other directions. Hence, optimum performance is obtained if all the m_i are equally small.

The maps for the rows and the columns are not independent. The entire 3D-FFT algorithm requires information to be exchanged through the entire (partition of the) machine. Therefore either t_r or t_c can not perform better than the time $t_{r,i}^{(f)}$ associated with the longest of the n_i. By selecting a good mapping between the virtual processor grid and the physical mesh we can only improve either t_r or t_c but not both.

We note some important observations on equation (16):

- For a given machine geometry n_x, n_y, and n_z there is no explicit dependency on P_c or P_r.
- In the limit of $P \to \infty$ our model agrees with the model presented in [2] for a single All-to-All transformation on a row, plane, or a volume of the physical mesh of the machine. For finite P our model gives longer times.

Our model assumes that all maps are capable of utilizing the full bandwidth B_l of the links at the bi-section at their best. Obviously for very small problems, latency considerations (which do not form part of our model) will dominate.

In Section 7 we will investigate how well our model describes the performance of an 3D-FFT with a two-dimensional processor grid when using an IBM eServer Blue Gene/L. Prior to this, we want to give a short review of the Blue Gene/L system and an overview on the application used for this investigation.

5 Overview of the Blue Gene/L system

5.1 Processors and operational modes

The model described in Section 4 has been tested on the University of Edinburgh's IBM eServer Blue Gene/L, named BlueSky. This section gives a short overview on the features most relevant for this investigation. Further information can be found in [15, 16]. The machine uses IBM PowerPC 440 dual core chips (nodes) with a clock frequency of 700 MHz. The system in Edinburgh offers a total of 1024 chips or 2048 processors (cores). Per chip, 512 MB of main memory are installed.

The Blue Gene/L architecture offers two main operational modes. In co-processor mode (CO), a single MPI task is placed on the chip. In CO mode the second core of the chip is used as a communication co-processor.

In the other operational mode, called virtual node mode (VN), two MPI tasks are placed on the chip. If the application fits into 256 MB of main memory per MPI task, this mode typically offers the better performance when comparing on a per-chip basis, and when the application has not run out of scalability [17].

5.2 Partitions and communication

The Blue Gene/L architecture offers five different networks which are dedicated and optimized for different tasks. For All-to-All communication, only the torus network is relevant. The torus network arranges the chips on a 3D torus, with communications taking place between nearest neighbors. The connecting links of this network offer a bandwidth of 2 bits per cycle per direction, which translates to 167 MB/s when using a clock frequency of $700{\cdot}10^6$ Hz. The maximum length of the torus packets is 256 bytes, with the first 16 bytes being used for routing, software and header information [18]. Additionally, 14 bytes of error control data are sent for each packet that is sent into the torus network [19]. This results in a maximum utilization of the torus network of 89 % and a limit of about 148 MB/s for the bandwidth. For a simple ping-pong benchmark using MPI, a sustained bandwidth of 147 MB/s has been measured and presented in [17] which is remarkably close to that limit.

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 chips or multiples thereof, there is a choice of an open mesh or a full torus, with the latter being the default.

5.3 Parallel runtime environment

The Blue Gene/L 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.

6 Details of the benchmark application

6.1 Calculating the 1D FFT

The parallel 3D-FFT algorithm is reviewed in Section 3. Here we describe the key features of our benchmark application used for this project. We also give a brief summary on initial investigations during the design stage of the benchmark.

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. This allows 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 and the one measured on a hot L3 cache and yielding the best performance regarding the total amount of time, taken for the entire three-dimensional forward FFT computation, has been presented in this paper.

The application uses version 2.1.5 of the open-source "Fastest Fourier Transform in the West" (FFTW) library [20] to perform the required one-dimensional FFTs. The Vienna University of Technology offers a FFTW 2.1.5 version specifically optimized for the double floating point unit of the Blue Gene processors and is known as FFTW-GEL [21, 22]. A detailed comparison by one of the present authors confirms the FFTW-GEL library yields a substantial performance improvement over a version of the library directly compiled from the source [13]. Hence FFTW-GEL is used for this project.

Our application calls the routine `fftw()` from the FFTW-GEL library which can perform several one-dimensional transformations within a single call. This routine takes a single array for the input of all the transformations and returns a single output array. For both arrays there is a choice of having the data for each individual transformation contiguous, followed by the data for the next transformation, or having regular strides between the individual data points of a single transformation. When using FFTW-GEL, best performance is achieved by using contiguous data in the `fftw()`-calls and copying the data into this format within the application if required by the algorithm [13].

6.2 Communication inside the application

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 3.2. This is divided into sub-communicators for the rows and the columns of the grid. For the communication kernel of our parallel three-dimensional FFT computation the `MPI_Alltoall` routine is used.

Bypassing the MPI layer and using the low-level system programming interface (SPI) of the Blue Gene for the communications would give additional benefits when the data per processor is small [2]. Since the reported improvements from accessing the SPI are small or negligible for the problem sizes and processor counts we are interested in, it was decided that accessing the SPI is beyond the scope of this project. Furthermore, for a general application, access to the SPI would make portability more difficult.

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

7 Results of the experimental investigations

7.1 Process mapping patterns

The following investigations have been carried out on a 512-chip partition with an $8 \times 8 \times 8$ topology using the toroidal network. Incidentally, this is the smallest partition that offers a torus network. A similar investigation using the mesh network has recently been published [13]. The investigation shows that for a partition offering a full torus network, the use of an open mesh network yields inferior performance for all mapping patterns and problem sizes included in this investigation. For this reason this investigation focuses on the case of a full torus network.

Our model, which we presented in Section 4, predicts best performance when using cubes as the basic mappings for either the rows or the columns of the virtual processor grid. For this reason, this study concentrates on two cube-shaped customized mappings. Additional mappings were investigated in [13]. Figure 4 illustrates the two cube-shaped customized mappings versus the default mappings in CO mode. We call the mapping used for the customized mapping of the rows for the case (a) an "8-cube" and the one for case (b) a "64-cube". For the purpose of clarity, all the figures depict only one basic image. 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 4 (a) the 8-cube is displaced 64 times and in (b) the 64-cube is displaced 8 times to fill the entire partition. The same applies to all other mapping patterns.

In Table 1 we summarize the m_i values for these mappings, required to obtain predictions on the communication times from equation (12). We would like to recall that we have to aim to get the largest of the m_i values as small as possible to optimize the performance. For a 512-chip partition these suggest a performance improvement for the 8-cube only, the

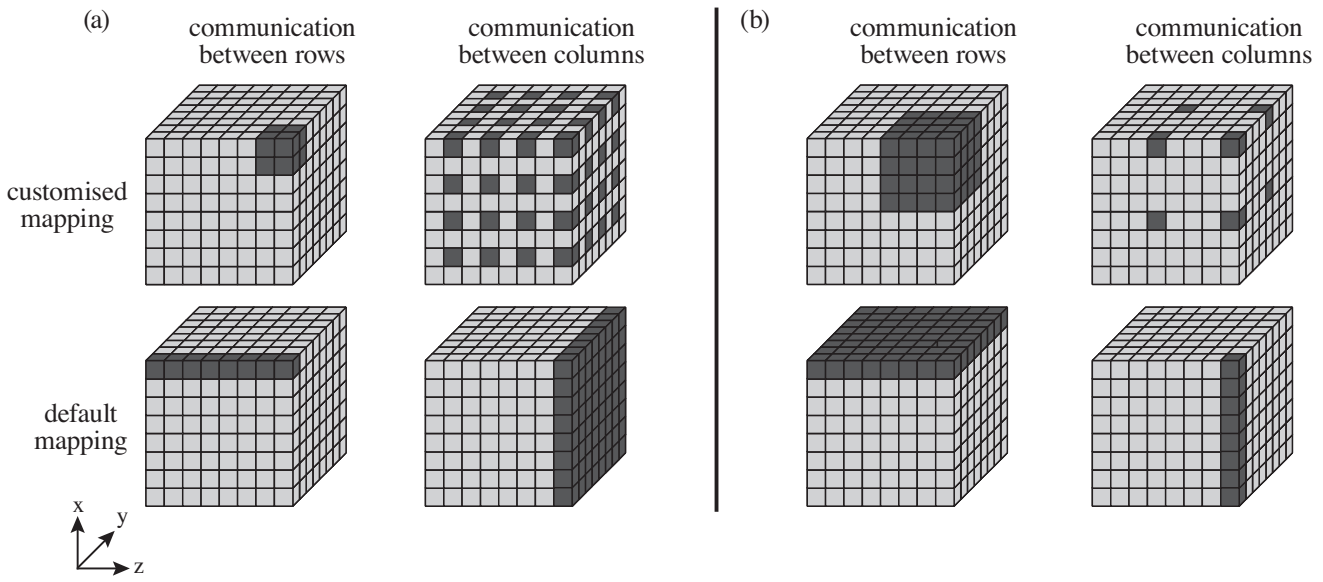

Figure 4: Customized vs default mapping on a 512-chip partition in CO mode and a division of processors in a Cartesian grid $P_r \times P_c$ of (a) 8×64 and (b) 64×8

	Rows			Columns		
	m_x	m_y	m_z	m_x	m_y	m_z
8×64 customized	2	2	2	4	4	4
8×64 default	0	0	4	4	4	0
64×8 customized	4	4	4	4	4	4
64×8 default	0	4	4	4	0	0

Table 1: m_i numbers for the mappings shown in Figure 4

64-cube is in all probability rather interesting for a 16^3 partition which would be available on larger Blue Gene/L systems. One would also expect the customized mapping in case (a) to offer better latency properties, since the 8-cube is smaller than the matching default. Hence the 8-cube requires fewer "hops" for the messages to reach their destination.

In addition to these mappings in CO mode, we investigated mappings in VN mode, which show the same shape for the customized and default cases. This means that for the mapping using the 8-cube we considered two VN mode mappings. One VN mode mapping uses a virtual processor grid with 16×64 nodes while the other mapping uses 8×128 nodes. In the first case we map a single row of 16 compute tasks onto an 8-cube, while in the second case we map two rows with 8 compute tasks each onto an 8-cube. Since the default mapping with the division of processors 8×128 shows a different shape compared to the default mapping with 8×64 in CO mode, we do not further consider this case. The same applies to the default case using the processor division 64×16, since it shows a different shape compared to the default mapping of a processor division 64×8 in CO mode. For the sake of clarity, Figure 5 illustrates the two VN mode mappings. Also here only one basic image is depicted, however, with one exception. If the two cores per node are divided between two different communicators, a second displacement of the basic image is presented in white. Again, 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.

Two different virtual processor grids, 128×8 and 64×16 and a single default are also used for performance investigation of the 64-cube in VN mode. These are constructed similarly to the maps of the 8-cube in VN mode and illustrated in Figure 6.

7.2 Analysis of overall performance

Figures 7 (a) and (b) present the influence of the 8-cube and 64-cube mapping described in Section 7.1 on the total performance of the entire 3D-FFT algorithm. To obtain more readable figures, the results of both figures have been normalized with the performance of an arbitrarily chosen default mapping. More precisely, the performance result of the default mapping for the 64×8 virtual processor grid - as our default choice - has been divided by the results of all other mappings. The measured times for the overall performance together with the times for the two All-to-All type communications can be found in Appendix A.

8-cube and 64-cube: For small problems and hence small messages, the VN mode does not show any benefits. CO mode is more efficient for small problems such as 64^3, since the shorter time spent in communications cancels out the advantage of using more processors for the computations.

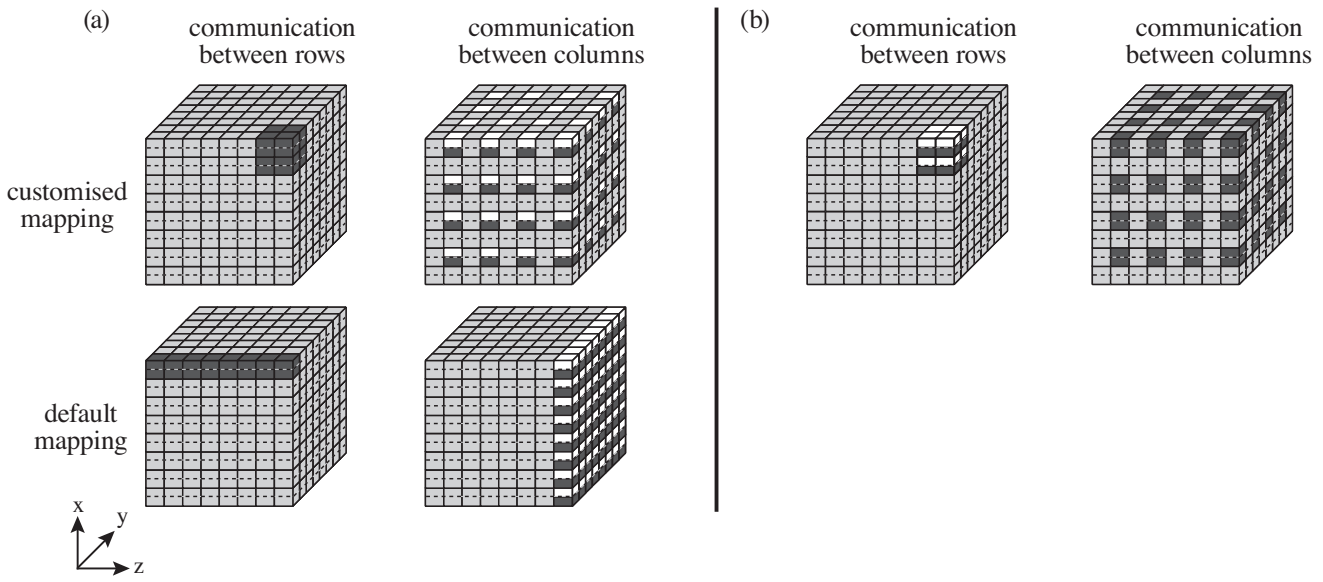

Figure 5: Customized vs default mapping on a 512-chip partition in VN mode and a division of processors in a Cartesian grid $P_r \times P_c$ of (a) 16×64 and (b) 8×128

This performance degradation in VN mode is most likely caused by the two cores per chip locking each other out from the network, as access to the network is shared between the two cores. Aside from that, in some cases, for VN mode the problem size is too small to divide the problem up between the number of processors. For larger problem sizes VN mode becomes attractive, independent of whether the customized or the default mapping is used.

8-cube: The expected performance improvement for small and dense cubes which is discussed in Section 4.3 can only be observed in VN mode. In CO mode the 8-cube mapping yields only a minor improvement. In VN mode the performance improvement due to the 8-cube mapping is quite substantial. For problems of 128^3 and 256^3 and a processor grid of 16×64 we observe a performance improvement of 16% over the default. This decreases for larger problems, but even for 1024^3 we see a performance increase of 10%.

The figures for VN mode show an explicit drop between 256^3 and 512^3 problems when compared to the CO mode. This is a result of additional time spent in the FFTW library functions, rather than in communication routines.

64-cube: For the 64-cube we do not observe any performance improvement when compared to the default mapping. This was expected from Table 1. For CO mode we see a performance degradation of less than 8%, while in VN mode the performance is virtually unchanged by the customized mapping. This is promising news for a Blue Gene/L system with 4096 chips and will be discussed below in Section 7.4 in greater detail.

Figure 7 (a) shows an unexpected performance difference of 6% between the two default cases for a problem size of 128^3. Both mappings use rows and planes of the physical processor mesh for the basic images. However these basic images have a different orientation and a different ordering, see Figure 4.

7.3 Analysis of average bandwidth for the 8-cube

After discussing the impact of the customized mappings on the overall performance, we now addresses the question how well this performance is described by the model in Section 4. For this we calculate the average bandwidth utilization per wire from the measured communication times by solving equation (16) for B_l and inserting the m_i from Table 1. A comparison to the hardware limit of 148 MB/s for B_l shows how well the measured times agree with our model, which is derived from considering the bi-sectional bandwidth. Here we focus on the mappings using the 8-cube and the corresponding default, Section 7.4 gives this comparison for the 64-cube.

Figure 8 (a) presents the bandwidth utilization for the communication within the rows of the virtual processor grid. The grid rows are mapped onto small and dense 8-cube patterns. The results for the communications within the columns of the virtual processor grid are presented in Figure 8 (b). These are mapped with the non-contiguous pattern, complementary to the 8-cube, with small gaps equally distributed over the entire 512-chip partition. Both figures contain the results for CO and VN modes and the prediction of our model corresponds to the horizontal lines marked as "HARDWARE LIMIT".

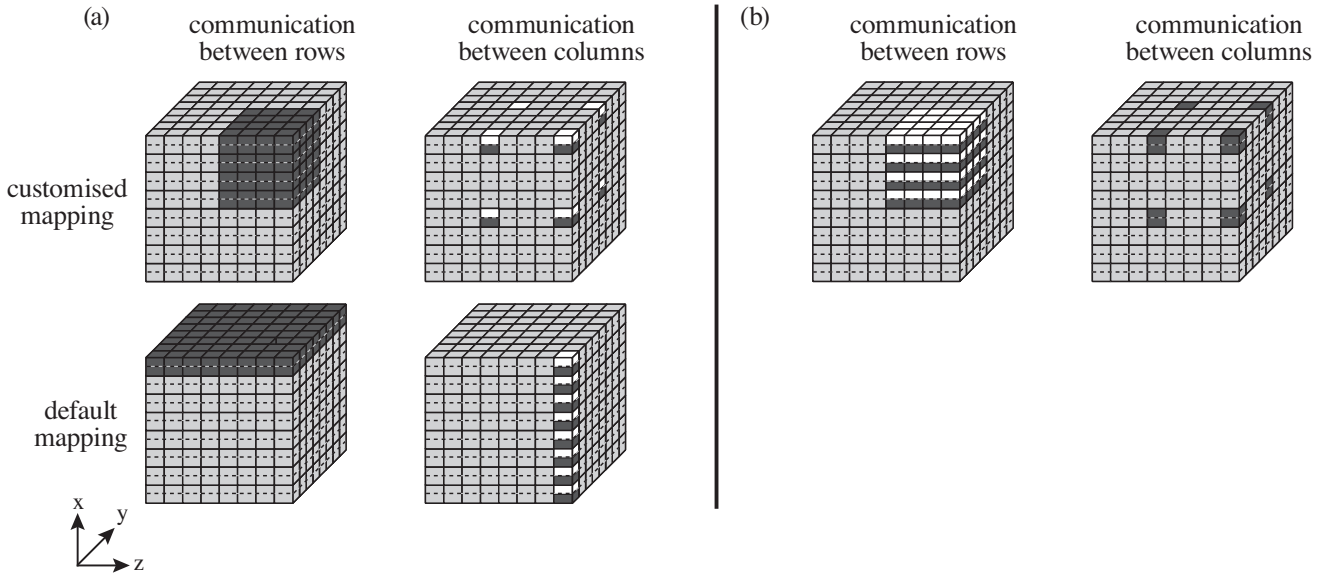

Figure 6: Customized vs default mapping on a 512-chip partition in VN mode and a division of processors in a Cartesian grid $P_r \times P_c$ of (a) 128×8 and (b) 64×16

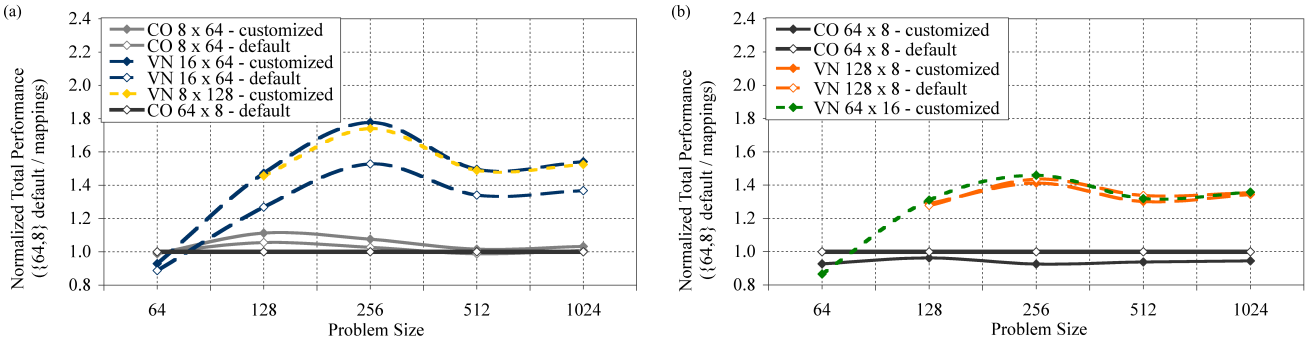

Figure 7: Comparison of the relative performance of the entire 3D-FFT algorithm on a 512-chip partition when deploying (a) 8-CUBE maps and (b) 64-CUBE maps between the virtual processor grid and the physical processor mesh

The figures demonstrate clearly the efficiency of the system with respect to the utilization of the bandwidth through the links of bi-section. Given a large enough problem, for most of the investigated maps, the average bandwidth utilization is amazingly close to the hardware limit and our model's prediction for the communication time is reasonably accurate. The most important exception to this is the performance of the 8-cube in CO mode. This saturates at an average bandwidth per wire of around 90 MB/s, which is substantially below the hardware limit. This shortfall is the key reason for the 8-cube not delivering the expected performance improvement for the entire 3D-FFT as discussed in Section 7.2. By contrast, in VN mode the performance is close to the hardware limit and the 8-cube delivers a sizable boost to the performance. This is independent of whether one or two communicators are placed on the 8-cube.

We can only speculate about the reasons for this behavior. 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 the L1-caches on the chip. In VN mode, when each core manipulates its own private data per chip, the problems associated with the lack of cache coherency go away. The same hardware as in the case of the CO mode is now capable of saturating the links of the bi-section.

Figure 8 (a) shows that the best bandwidth utilization is achieved when all processors of a virtual grid row are mapped onto a line along a single row of the physical mesh in CO mode. This is most significant for small problem sizes such as 64^3 and 128^3. The superior latency offered by this simple map leads to the very good overall performance of the default map in CO mode as shown in Figure 7 (a).

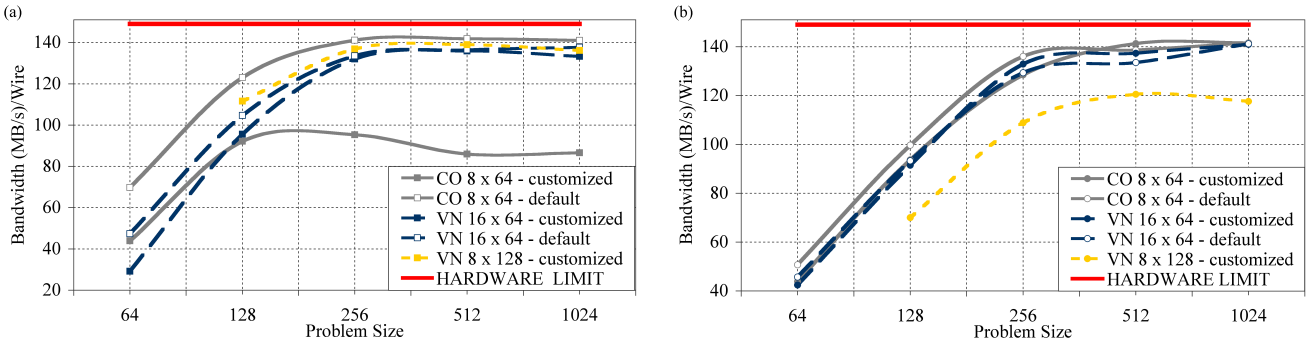

Figure 8: Average bandwidth utilization per link of the bi-section for the communication between (a) ROWS and (b) COLUMNS of the 2D-virtual processor with the division 8×64

The reason might be the simpler message routing on a line of processors. For each message there is only one shortest path to reach its destination. If this knowledge is used in the communication libraries, the overheads can be reduced.

It is interesting to note that the effect of VN mode over CO mode on the latency is substantially more pronounced for the row communications than the column communications. This is really surprising. For the communications within the columns and the smallest problem size, the individual messages are smaller than a single package. This holds for both modes and is obviously independent of the map. Switching from CO mode to VN mode increases the number of messages by a factor of four, since the number of tasks is doubled. Assuming single package mode for each message, this would lead to a fourfold increase in the number of packages on the network. The results in Figure 8 (b) are not consistent with such a scenario. The algorithm employed for the All-to-All must be able to combine more than one message into single packages.

7.4 Analysis of average bandwidth of the 64-cube

Figure 9 (a) shows the average bandwidth utilization for the communication within the rows of the virtual processor grid, which refers to the dense 64-cube pattern in CO and VN mode. Figure 9 (b) shows it for the communication within the columns of the virtual processor grid, which refers to the complementary non-contiguous pattern with large gaps equally distributed over the entire 512-chip partition. Again, on both figures, the prediction of our model corresponds to the horizontal lines marked as "HARDWARE LIMIT".

For problem sizes larger than 128^3, the available bandwidth at the bi-section is better utilized for 64-cube in CO mode than it was for the 8-cube pattern. When switching to VN mode we do not observe a significant performance improvement when compared to the CO mode. Again this is different from the 8-cube, where a dramatic improvement was seen when switching to VN mode. Obviously, for the 64-cube, the ratio of links at the bi-section to processors is halved when compared to the 8-cube. This is in support of our above hypothesis, that in the case of an 8-cube in CO mode, a single core is not capable of simultaneously dealing with the handling of the MPI call and the data insertion into the network.

The average bandwidth at the bi-section of the 64-cube saturates at around 120 MB/s, which is significantly less than the hardware limit. 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 centre of the bi-section is among the shortest paths than there are pairs for the links at the corners of the bi-section. In this context it is interesting to note that the communication times for the 64-cube are unchanged whether we configure the communication network as a torus or as an open mesh. This confirms that the 64-cube cannot benefit from torus connectivity of the partition.

The matching default mapping in CO mode, where the tasks are spanned over an entire plane along two axes, shows the best bandwidth utilization. The pattern here utilizes torus links in two dimensions, which results in more symmetric communication. This symmetry makes avoiding hot spots easier.

When using a large number of small communicators, that is for the columns communications in this case, we observe an increased latency for the smallest problem size when we switch from CO to VN mode. When using a small number of large communicators, this sensitivity is very small. This is the same as discussed for the 8-cube.

For the 512-chip partition with an $8 \times 8 \times 8$ topology our model does not predict a performance benefit from using the 64-cube. The key result of this subsection is that neither the 64-cube nor its complementary pattern with large

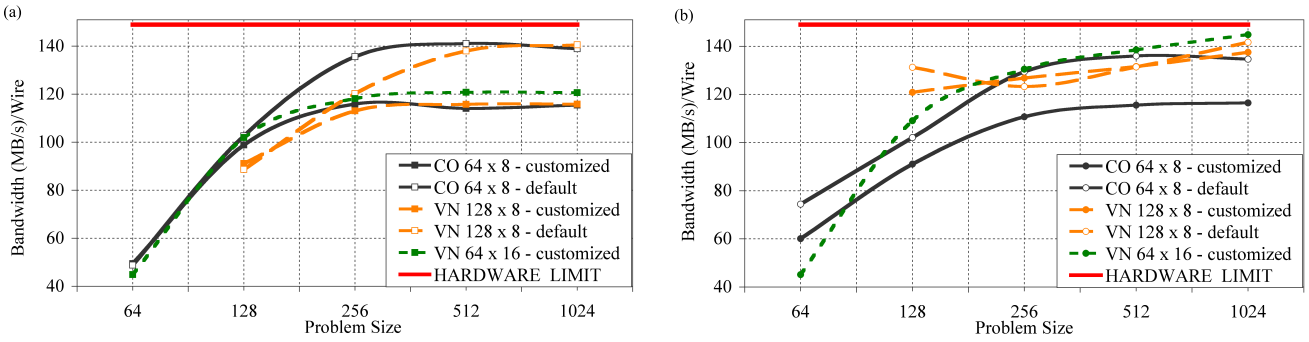

Figure 9: Average bandwidth utilization per link of the bi-section for the communication between (a) ROWS and (b) COLUMNS of the 2D-virtual processor with the division 64×8

gaps between its processors show severe under-performance when compared to the predictions from our model and the matching defaults. This holds for both CO mode and VN mode. The observed decrease in average bandwidth utilization is small enough to make the 64-cube an interesting mapping pattern for a larger Blue Gene/L with 4096 chips in a $16 \times 16 \times 16$ partition. On such a machine one could hope for improvements similar to the ones we have reported for the 8-cube on the 512-chip partition available to us. Obviously this needs experimental verification on such a machine.

8 Conclusions

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. From a detailed discussion of the communication bandwidth through the bi-section of the communication network, we build a simple model for the communication times of the algorithm. The model can be applied to a large number of MPI mappings between the virtual processor grid of the algorithm and the physical mesh of the machine. From the considered maps, our model predicts best performance if either the rows or the columns are mapped onto small cubes.

Our experimental results show also that performance benefits of up to 16% for the entire 3D-FFT algorithm are possible when using cubes for the images on a 512-chip partition of the machine. The observed performance increase of the communication part due to task placement is as large as 33%. This indicates the remaining scope of performance improvement due to task placements if the serial part of the algorithm, such as the deployed FFT routine, would be further optimized. For small problem sizes, our investigations do not show a benefit from using cubes for the images. The reason for this quite likely lies within the communication library and its implementation.

Furthermore, our results show that for larger installations than were available for this study, the 64-cube pattern looks promising with respect to performance improvements. In our investigation this map and its complementary map do not show any critical performance degradation. Obviously this needs experimental confirmation on such a larger machine.

Our measurements also show that for small problems, utilizing the VN mode which places two computational tasks on a dual core chip, is detrimental to the performance of the 3D-FFT. This has been traced to a large number of small communicators performing exceptionally well when only one of the cores is active. If two cores were active they might be locking each other out from shared access to the network.

Acknowledgements

We would like to thank Mark Bull (EPCC) for valuable comments on an earlier version of the manuscript.

A Numerical results of the 3D-FFT computation

The following table gives the total times used by our benchmark for a forward transformation.

Problem size:	64^3	128^3	256^3	512^3	1024^3
CO 8×64 cust.:	0.359 ms	2.20 ms	18.9 ms	188 ms	1.78 s
CO 8×64 def.:	0.356 ms	2.31 ms	19.8 ms	193 ms	1.82 s
VN 16×64 cust.:	0.382 ms	1.66 ms	11.4 ms	128 ms	1.21 s
VN 16×64 def.:	0.400 ms	1.93 ms	13.3 ms	142 ms	1.34 s
VN 8×128 cust.:	—	1.68 ms	11.7 ms	128 ms	1.20 s
CO 64×8 cust.:	0.383 ms	2.54 ms	21.9 ms	203 ms	1.94 s
CO 64×8 def.:	0.355 ms	2.44 ms	20.3 ms	191 ms	1.84 s
VN 128×8 cust.:	—	1.89 ms	14.4 ms	147 ms	1.37 s
VN 128×8 def.:	—	1.91 ms	14.1 ms	143 ms	1.36 s
VN 64×16 cust.:	0.410 ms	1.86 ms	13.9 ms	145 ms	1.35 s

Table 2 summarizes the times for the communication within the rows of the two-dimensional virtual processor grid used by our 3D-FFT application.

Problem size:	64^3	128^3	256^3	512^3	1024^3
CO 8×64 cust.:	0.089 ms	0.339 ms	2.622 ms	23.25 ms	185 ms
CO 8×64 def.:	0.112 ms	0.508 ms	3.544 ms	28.21 ms	227 ms
VN 16×64 cust.:	0.134 ms	0.327 ms	1.894 ms	14.71 ms	120 ms
VN 16×64 def.:	0.165 ms	0.597 ms	3.743 ms	29.31 ms	232 ms
VN 8×128 cust.:	—	0.280 ms	1.827 ms	14.39 ms	117 ms
CO 64×8 cust.:	0.158 ms	0.632 ms	4.314 ms	35.07 ms	277 ms
CO 64×8 def.:	0.160 ms	0.609 ms	3.687 ms	28.34 ms	230 ms
VN 128×8 cust.:	—	0.686 ms	4.423 ms	34.54 ms	276 ms
VN 128×8 def.:	—	0.705 ms	4.159 ms	28.98 ms	228 ms
VN 64×16 cust.:	0.174 ms	0.613 ms	4.231 ms	33.12 ms	265 ms

Table 3 summarizes the times for the communication within the columns of the two-dimensional virtual processor grid used by our 3D-FFT application.

Problem size:	64^3	128^3	256^3	512^3	1024^3
CO 8×64 cust.:	0.178 ms	0.667 ms	3.891 ms	28.32 ms	226 ms
CO 8×64 def.:	0.154 ms	0.627 ms	3.675 ms	28.85 ms	226 ms
VN 16×64 cust.:	0.184 ms	0.683 ms	3.761 ms	29.13 ms	227 ms
VN 16×64 def.:	0.171 ms	0.670 ms	3.865 ms	29.95 ms	227 ms
VN 8×128 cust.:	—	0.732 ms	4.023 ms	30.66 ms	223 ms
CO 64×8 cust.:	0.130 ms	0.686 ms	4.516 ms	34.60 ms	274 ms
CO 64×8 def.:	0.105 ms	0.612 ms	3.863 ms	29.41 ms	237 ms
VN 128×8 cust.:	—	0.517 ms	3.943 ms	30.40 ms	232 ms
VN 128×8 def.:	—	0.476 ms	4.057 ms	30.44 ms	225 ms
VN 64×16 cust.:	0.173 ms	0.573 ms	3.830 ms	28.88 ms	221 ms

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

[5] F. Franchetti, et al., "FFT Program Generation for Shared Memory: SMP and Multicore", SC2006, Tampa, Florida, USA, 2006.

[6] A. Ali, et al., "Scheduling FFT Computation on SMP and Multicore Systems", ICS'07, Seattle, WA, USA, 2007.

[7] D. Bailey, "FFTs in External or Hierarchical Memory*", The Journal of Supercomputing, 4, page 23, 1990.

[8] A. Nukada, et al., "High Performance 3D Convolution for Protein Docking on IBM Blue Gene", ISPA 2007, LNCS 4742, page 958, 2007.

[9] A. Na'mneh, et al., "Parallel implementation of 1-D fast Fourier transform without inter-processor communications", System Theory, 2005. SSST '05. Proceedings of the Thirty-Seventh Southeastern Symposium on, page 307, 2005.

[10] G. Bhanot, et al., "Optimizing task layout on the Blue Gene/L supercomputer", IBM Journal of Research and Development, Vol. 49, page 489, 2005.

[11] B. R. de Supinski, et al., "Blue Gene/L applications: parallelism on a massive scale", The International Journal of High Performance Computing Applications, Vol. 22, page 33, 2008.

[12] J. Schumacher, et al., "Turbulence in Laterally Extended Systems", Parallel Computing: Architectures, Algorithms and Applications, Vol. 38, pp. 585-592, 2007.

[13] 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/

[14] MPI Standard, http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html.

[15] "User Guide to EPCC's BlueGene/L Service", available: http://www2.epcc.ed.ac.uk/~bluegene/UserGuide/BGuser/BGuser.html

[16] O. Lascu, et al., "Unfolding the IBM eServer Blue Gene Solution", IBM Redbook, 2006, ISBN 0738493872, http://www.redbooks.ibm.com/abstracts/sg246686.html

[17] M. Bull, et al., "Application performance on the Blue Gene architecture", Preprint, EPCC, The University of Edinburgh, 2007

[18] X. Martorell at al., "Blue Gene/L performance tools", IBM Journal of Research and Development, Volume 49, page 407, 2005.

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

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

[21] 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

[22] S. Kral, FFTW-GEL Homepage, http://www.complang.tuwien.ac.at/skral/fftwgel.html