Parallel 3D-FFTs for multi-core nodes on a mesh communication network.

Joachim Hein[1,2], Heike Jagode[3,4], Ulrich Sigrist[2], Alan Simpson[1,2], Arthur Trew[1,2]
[1] *HPCX Consortium*
[2] *EPCC, The University of Edinburgh, James Clerk Maxwell Building, Mayfield Road, Edinburgh, EH9 3JZ, UK*
[3] *The University of Tennessee in Knoxville*
[4] *Oak Ridge National Laboratory (ORNL)*

Abstract

Parallel fast Fourier transformations are important for many scientific applications and are difficult to parallelise efficiently on large numbers of processors due to the all-to-all nature of the communications. We will discuss how the topology of a mesh communication network can affect the communication performance and explore how nodes offering several processing cores can be exploited to improve the communication performance. We present benchmarking results from the UK's national supercomputing services HECToR (Cray XT4 with dual core Opterons) and HPCx (IBM p575 cluster with 16-way SMP nodes and HPS interconnect) as well as the University of Edinburgh's IBM BlueGene/L.

1 Introduction

Fast Fourier transformations (FFT) are an important computational kernel used in many scientific applications. FFTs are unfortunately very hard to parallelise efficiently on large numbers of processors. Since individual FFTs can only be parallelised with reasonable efficiency on a small number of processors [1], arrays of dimension D are typically transformed by employing a virtual processor grid of dimension $D-1$ or smaller. Using a processor grid in more than one dimension requires All-to-All type communications within subgroups.

To keep the number of communication operations small, a one dimensional processor grid is typically best, since a single global All-to-All communication is all that is required. However this seriously limits the total number of processors which can be employed to the second largest array extent. It has been demonstrated that on a modern HPC architecture, modest sized arrays in three dimensions can be efficiently transformed on a thousand processors using a two dimensional processor grid [3, 4, 5], without the costs of the communication part becoming excessive.

Today's modern HPC architectures typically feature inhomogeneous communication networks, in the sense that individual processing cores can communicate more efficiently with some processing cores than they can do with others. Examples include hierarchical switches, meshed communication networks, SMP nodes and multicore chips. Many modern machines combine several of these features. In this report we investigate the performance impact of the mapping of a two dimensional virtual processor grid onto the physical machine when used to parallelise an FFT in three dimensions. The performance impact of the meshed network of the University of Edinburgh's IBM BlueGene/L server has been investigated in [5, 6], while [7] studied the effects of the network of the UK's national service HPCx, which is based on 16-way SMP nodes. This report will mostly focus on the UK's latest national service HECToR, based on Cray XT4

hardware. The article by F. Reid et al, gives and overview over the performance of a number of scientific applications on HECToR [8].

2 Computer systems

We start with a short description of the key features, most relevant for this study, of the three super computer systems used for this report.

2.1 HPCx

The HPCx system, one of two UK national supercomputer services, offers 160 IBM eServer p575 computing nodes. These are connected with a communication network, based on IBM's High Performance Switch (HPS), also known as Federation. The entire system offers 2560 processors and delivers a theoretical peak performance of 15.4 Tflops/s and a sustained performance of 12.9 Tflops/s on Linpack. Each eServer p575 node has 16 IBM Power5 processors with a clock frequency of 1.5 GHz. The nodes have 32 GB of main memory installed, shared between their 16 processors.

The nodes on HPCx are connected with a switch network using IBM's HPS. To improve the bandwidth of the network, HPCx features two independent planes, which are essentially two independent networks. Each node can use either plane to communicate with any other node of the machine.

2.2 HECToR

HECToR is the latest national supercomputer service in the UK. The system is based on Cray XT4 hardware, utilising 5664 dual core Opteron processors, with a clock frequency of 2.8 GHz. HECToR offers a theoretical peak performance of 63.4 Tflops/s and a sustained performance of 54.6 Tflop/s on Linpack. The processors are arranged on a meshed network of dimension $20 \times 12 \times 24$, which offers toroidal connections in the two longest dimensions only. The network is based on Cray's SeaStar technology offering a hardware link speed of 7.6 GB/s [9], which translates into a bi-sectional bandwidth of about 3.6 TB/s. The machine is operated under UNICOS/LC and offers SMP capability within a dual core node. Using the standard systems software, the user is only offered minimal control over the placement of the tasks on the machine. The environment variable `MPICH_RANK_REORDER_METHOD` allows the user to control how the tasks are grouped into pairs of two, which are then mapped onto the same dual core node.

2.3 BlueSky

The University of Edinburgh's BlueSky system is based on IBM BlueGene/L technology. The system offers 1024 IBM PowerPC 440 dual core processors, with a clock frequency of 700 MHz. The peak performance of the system is 5.7 Tflop/s and the sustained Linpack performance is 4.7 Tflop/s. The processors are arranged on a torus network of dimension $8 \times 8 \times 16$, which can be subdivided into two tori of dimension $8 \times 8 \times 8$ or even smaller partitions with open meshes. An individual partition can not be shared between several users. The meshed communication network offers a hardware link speed of 175 MB/s. Taking all overheads into account, a bandwidth of 148 MB/s is available to the application [10, 11]. This gives 18.5 GB/s for the bi-sectional bandwidth of the 512 and 1024 node partitions. In contrast to the Cray XT architecture, the BlueGene/L offers the user full control over how the tasks are placed onto the mesh network.

3 All-to-all performance

3.1 Insertion bandwidth

As we will explain in the next section, in a typical implementation the performance of a parallel FFT is directly linked to the performance of all-to-all type communications. We begin with a study of the

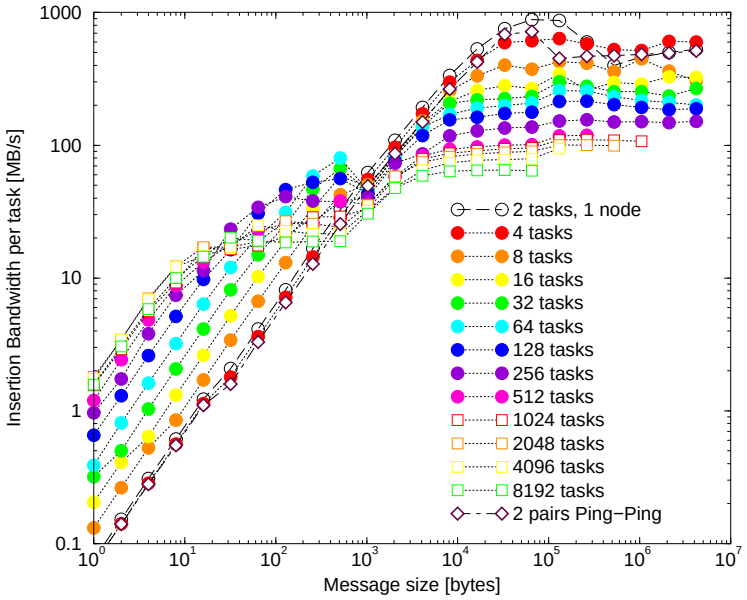

Figure 1: Performance of the all-to-all communication on the HECToR system

`MPI_Alltoall` call, using version 3.0 of the Intel MPI benchmarks (formerly known as Pallas MPI benchmarks) [12].

In Figure 1 we display the results for a large number of different task counts and a wide range of message sizes. All results are for two active tasks per node. In the figure we report the insertion bandwidth I_t per task, under the assumption that the data to stay on the processor does not get inserted into the network

$$I_t = \frac{m(n-1)}{t_{\mathrm{av}}}\,. \tag{1}$$

Here n denotes the number of tasks, m the message size and t_{av} the average time as reported by the benchmark code. The results reported are the best out of number of runs, most of them performed when the machine was draining prior to maintenance but was still scheduling short jobs, which definitely reduced interference by other users jobs. We see three different regimes. For messages below 1 kB of data, we see the performance improving for increasing task count, while for messages above 1 kB of data the performance decreases with increasing task count. The third regime starts at a message size of 128 kB. For message of 128 kB and larger, the plateau values are slightly larger than they are below 128 kB. We also took data with a single task per node. The step is at 128 kB is even more pronounced for one task per node.

When using two tasks, both tasks are placed on the same node and intra-node communication can be used instead of network communication. The measurements for two tasks show an almost linear performance rise up to a message size of about 16 kB. For larger messages the bandwidth saturates. The performance advantage over utilising four task, which is the smallest measurement, that has to utilise the network, is small.

For a comparison, we also include the bandwidth measured for a multi Ping-Ping benchmark. For this we choose two dual core nodes, and placed two pairs on them. Per pair one partner is located on each node. The insertion bandwidth measured with the multi Ping-Ping is again comparable to the result for the four task all-to-all call.

3.2 Bi-sectional bandwidth

We tried to understand why, for messages of 8 kB and larger, the insertion bandwidth of the all-to-all communication drops so dramatically with increasing task count. A potential bottleneck for the performance

	Bi-section	Insertion point
Link speed: Datasheet value	7.6 GB/s	6.4 GB/s
Link speed: Ping-Ping 2 tasks/node	1.4 GB/s	1.4 GB/s
Link speed: Ping-Ping 1 tasks/node	1.4 GB/s	1.4 GB/s
Number of links:	20×24	4096
theoretical bandwidth from datasheet:	3.6 TB/s	25.6 TB/s
scaled bandwidth from Ping-Ping, 2 t/n:	0.66 TB/s	5.6 TB/s
scaled bandwidth from Ping-Ping, 1 t/n:	0.66 TB/s	5.6 TB/s
bandwidth from alltoall, 2 task per node:	0.13 TB/s	0.51 TB/s
bandwidth from all-to-all, 1 task per node:	0.21 TB/s	0.85 TB/s

Table 1: Comparing the bi-sectional bandwidth and the total insertion bandwidth of the HECToR system.

of the all-to-all communication is the bi-sectional bandwidth.

If the processors of (a partition of) a parallel computer are divided into two groups of equal size, the bi-sectional bandwidth denotes the total bandwidth available to transfer data from one half into the other. When considering an all-to-all communication, half of the data is on one side of the bi-section, the other half on the other side. During the communication, each side has to transfer half of its data, which is a quarter of the total, onto the other side. Hence communications can not finish faster than

$$t_{\text{av}} \geq \frac{D_T}{4B} = \frac{mn^2}{4B} \tag{2}$$

D_T denotes the total amount of data, B the bi-sectional bandwidth, t_{av} the average time for the communication, m the message size and n the task count. Eq. (2) can be used to define an effective bi-sectional bandwidth B_{eff} from the measured all-to-all times

$$B_{\text{eff}} = \frac{D_T}{4t_{\text{av}}} \tag{3}$$

On a parallel machine with a meshed communication network, such as the Cray XT4 architecture, the bi-sectional bandwidth is the bandwidth of an individual link, multiplied with the number of links cut by the bi-section. It is interesting to compare the performance results for the all-to-all with the Cray datasheet values and Ping-Ping results for the bandwidth of the network link and the insertion bandwidth when using 2 nodes. We use the performance results obtained on 4096 nodes using one or two tasks per node. 4096 nodes is the largest partition HECToR users can routinely get access to, and equates to 73 % of the machine. The comparison is shown in Tab. 1.

The results show that significantly higher bandwidth results can be realised when using only 1 task per node. However the bandwidth utilisation is substantially smaller than that from the Ping-Ping test scaled with the relevant number of links. It is more than an order of magnitude smaller compared to what the Cray datasheet appears to be promising.

3.3 Comparison between different machines

Having analysed the all-to-all performance on HECToR, it is very interesting to investigate how this compares to the other machines. In Figure 2 we compare the effective bi-sectional bandwidth of HECToR to HPCx and the BlueGene/L system, for 1024 tasks on fully populated nodes. On each machine a few repeat runs have been performed. For each of the machines, we choose the run which gave the best result. We note that in particular the HPCx runs are extremely noisy for message sizes up to about 128 bytes. Below 16 bytes, which is typically not of interest for parallel FFTs, poor runs displayed a performance decay of a factor of about four. Neither of the other systems showed fluctuations on this scale. For messages of 2 kB and larger the fluctuations displayed by HPCx are similar to the other systems.

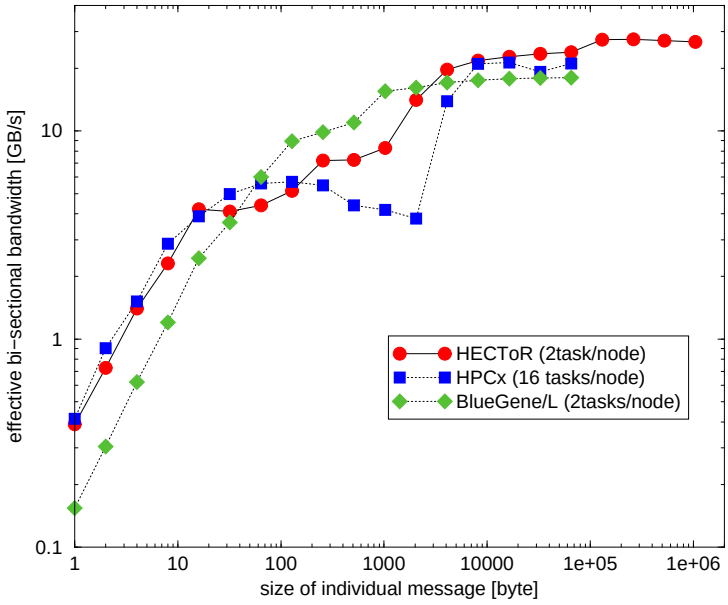

Figure 2: Comparison of the all2all performance using 1024 tasks on fully populated nodes

Ignoring these fluctuations for small messages, HECToR and HPCx show similar performance. Starting from about 64 byte to 2048 byte, the BlueGene/L system shows the best performance. For large messages, HECToR shows the best performance, despite the system not living up to its potential as discussed above. The best result for the effective bi-sectional bandwidth on HECToR is 27.5 GB/s, while on HPCx we observe up to 21.3 GB/s and on the BG/L we see 18.1 GB/s. Considering the discussion of subsection 3.2, we would like to emphasise how close the BlueGene/L operates to its specification of 18.5 GB/s.

4 Parallel FFT algorithms using multidimensional processor grids

We start this section with a short overview on parallel fast Fourier transformations using multi dimensional processor grids.

4.1 Basic algorithm

To explain the algorithm, we focus on transformations of arrays in three dimensions. The Fourier transformation $\tilde{X}$ of an array X in three dimensions is defined as

$$\tilde{X}(k_x,k_y,k_z) := \sum_{x=0}^{N_x-1}\left(\sum_{y=0}^{N_y-1}\left(\sum_{z=0}^{N_z-1} X(x,y,z)\exp\left(\tfrac{2\pi \mathrm{i}}{N_z}k_z z\right)\right)\exp\left(\tfrac{2\pi \mathrm{i}}{N_y}k_y y\right)\right)\exp\left(\tfrac{2\pi \mathrm{i}}{N_x}k_x x\right) \qquad (4)$$

When interested in a numerical solution, each of the three sums in eq. (4) is typically evaluated by using the fast Fourier transform algorithm (FFT) in one dimension. Unless N_x, N_y and N_z are very large it is presently not efficient to distribute an FFT in one dimension onto more than one processor and even for large N-values it is not possible to employ very many processors [1]. On a parallel machine offering thousands of processors, we are interested in utilising hundreds or thousands of processors for a single multidimensional FFT.

To achieve a decent parallel speedup of the 3D FFT, the array $X(x,y,z)$ is best distributed onto a one or two dimensional processor grid, leaving at least one dimension of the array local to the processor. This

allows the first one or two sets of FFTs to be evaluated locally without any communication. Before further FFTs in the originally distributed directions can be evaluated, the processor grid needs to be rearranged. When using a one dimensional processor grid, this can be accomplished in a single All-to-All communication over the entire processor grid. When using a two dimensional processor grid, two groups of All-to-All type communications are required. Assuming the data array is already distributed onto the two dimensional processor grid of dimension $n_c \times n_r$, with the z-direction being local to the processors, the basic algorithm looks as follows:

- Each task performs the 1-dimensional FFTs in the z-direction on its data.
- Perform an all-to-all communication within each of the rows of the processor grid, to get the y-direction processor local. These n_r all-to-all communications are performed simultaneously, each in a disjoint communicator.
- Each task performs the 1-dimensional FFTs in the y-direction on its data.
- Perform an all-to-all communication within each of the columns of the processor grid, to get the x-direction processor local. These n_c all-to-all communications are performed simultaneously, each in a disjoint communicator.
- Each task performs the 1-dimensional FFTs in the x-direction on its data.

Please see [2, 3, 5, 7] for a more detailed description of the algorithm. Due to the second communication step, this is typically slower than using a one dimensional processor grid, when using the same number of processors [2, 5]. However the second processor grid dimension allows for more processors to be utilised than a simple one dimensional processors grid.

For our investigations we use two benchmark suites, written in C, which use version 2.1.5 of FFTW [13]. The communication is performed by calling `MPI_Alltoall` of the relevant MPI library. Both benchmarks utilise explicit buffer packing in the application space instead of MPI derived data types. We use `MPI_Cart_create` and `MPI_Cart_sub` to generate the processor grid and split communicators.

The key aim of this report is to investigate whether the performance of this algorithm on HECToR can be improved by changing the processor grid and whether the placement with respect to the dual cores matters.

4.2 Basic Performance

To obtain an overview on the basic performance of the benchmark on HECToR, we used a processor grid as suggested by `MPI_Dims_create` and investigated over a range starting from 2 tasks up to 8192 tasks, when using two tasks per node. We used symmetric problem sizes in a range from 16^3 to 512^3. See Figure 3 for the results.

Starting from a task count of two, we see an initial performance improvement when increasing the task count. For most problem sizes, there is an intermediate point, when the benchmark ceases to go faster when increasing the task count. Interestingly, when increasing the task count further, the benchmark speeds up again. Studying the message sizes, it turns out that this intermediate point of poor performance occurs when messages of 1 kB are sent. This is the point in Figure 1 where the performance is essentially independent from the task count.

5 Task placement, processor grid dimensions and performance

In this section we will look into the scope for performance improvement by changing the processor grid and changing the placement of the tasks with respect to the dual core processors.

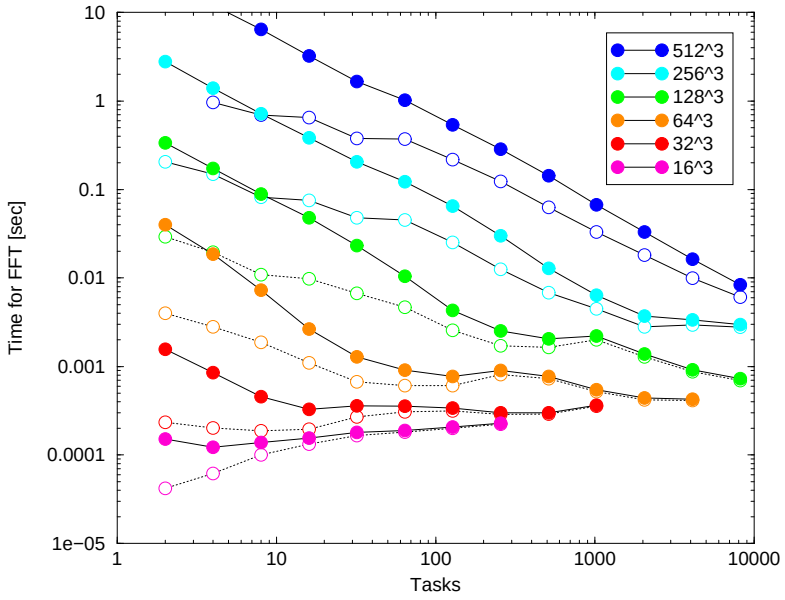

Figure 3: Basic performance of the parallel FFT algorithm on the HECToR system. Full symbols give the time spend in the entire algorithm (communication, buffer copies and communication), while open symbols represent the time spend in communication only.

5.1 Processor grids and dual core nodes on HECToR

For two of the task counts, 256 tasks and 4096 tasks, we looked in detail how the performance for different problem sizes is affected when the processor grid is changed, and the effect of placing two task from different communication groups on a single dual core node.

The communication results, again corrected for the fact that data for the same processor doesn't need to be inserted into the network, are shown in Figure 4 for 256 task and Figure 5 for 4096 tasks. Each test of benchmark runs the problem repeatedly and averages the results over these repeats. Depending on problem size, the benchmark varies the number of repeats, from sixteen for 64^3 to two for 512^3. For each problem size and `MPICH_RANK_REORDER_METHOD` value, we perform at least five tests. The results on the graphs are from the test which offered the shortest time for the entire FFT. Typically these results were in good agreement with the averages over the repeat runs, with the exception of the results for 128^3 on 256 tasks. For this test we show the results from the fastest FFT and the averages. The latter are shown by the small symbols in Figure 4. For the 256 tasks run, there is a significant effect of the `MPICH_RANK_REORDER_METHOD` and hence we show both results. For 4096 tasks these difference was insignificant and hence to make the figure less busy, we only give the results for `MPICH_RANK_REORDER_METHOD`=0. Both figures show the results for the first communication only, since the results and findings from the second communication are very similar. The figures also include a subset of the results from Figure 1 to allow an easy comparison.

For 256 processors, we varied the processor grid from 2×128 to 128×2 and for 4096 processors, the grid was varied from 8×512 to 512×8. Obviously, for the small problem sizes, the end-points of this variation could not be reached. The processor grid variation gives a corresponding change in message size. Messages become larger if the grid is smaller in this direction. In the figures the individual measurements are plotted against their message size.

When considering the first all-to-all communication, for `MPICH_RANK_REORDER_METHOD`=0 two tasks from the same communication group are placed on each dual core node. For `MPICH_RANK_REORDER_METHOD`=1 two tasks from different communication groups are placed on a node, again with respect to the first communication call. For the second all-to-all the roles are reversed.

One observes the performance in terms of insertion bandwidth per task for groups of all-to-all, as used in

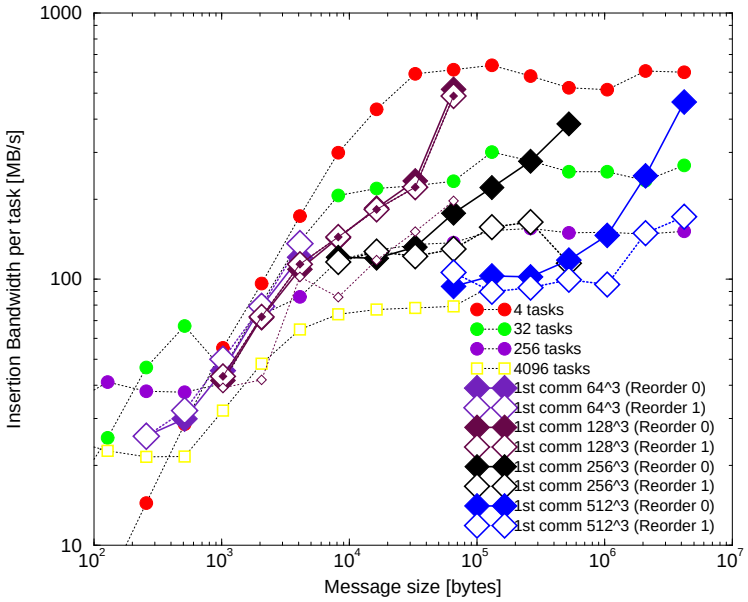

Figure 4: Comparing the performance of the individual all-to-all communications for a FFT algorithm using 256 tasks. Closed diamonds use `MPICH_RANK_REORDER_METHOD = 0` while open diamonds use `MPICH_RANK_REORDER_METHOD = 1`. The circles repeat a subset of the results from Figure 1. For the 128^3 problem there is a significant qualitative difference between the minimum times and the average over all the trials: the average results are shown by smaller symbols of the same colour and type as the corresponding results from the minimum times.

the FFT algorithm, to be very similar to the performance of a global all-to-all. In particular the behaviour for 1 kB sized messages seems almost "universal" in the sense that the task count, the number of all-to-all communication operations running concurrently on neighbouring processors, or even the other core of the same processor, has very little influence on the performance, which is always around 40 MB/s. Again for messages up to 256 byte we see a marked performance improvement over the straight line of a global four processor all-to-all. However the most interesting behaviour is seen for messages of 8 kB, and larger for the 256 task run. For each problem size, the data point corresponding to the larges message size utilises only two task. Here we see a strong dependence on the `MPICH_RANK_REORDER_METHOD`. For `MPICH_RANK_REORDER_METHOD`=0 we have two tasks of a single all-to-all on one node. The data for the second task of the node is not required to be inserted into the network, but even if the MPI-library is not multi-core aware and does insert this data into the network, the data does not need to travel far and will not cause severe congestion inside the network. If the number of nodes involved in an individual all-to-all is small, this significantly reduces the amount of data inside the network. When using 4096 tasks the data point corresponding to the larges messages size still involve eight tasks. For eight tasks the scope for reduced congestion due to data staying local is not very large. It is not surprising that there is no significant dependence on the value of `MPICH_RANK_REORDER_METHOD`.

When using `MPICH_RANK_REORDER_METHOD`=1, at least for the two largest problems investigated for 256 tasks, the observed performance with respect to data inserted into the network is comparable between the FFTs in groups and a global FFT using 256 tasks. The better performance realised for a global all-to-all communication with a smaller processor count is not obtained when running several all-to-all communications concurrently.

Overall, the insertion performance for a given message size when using groups of all-to-all transformations is very similar to the performance observed for a single all-to-all. Throughout this investigation we did not observe a performance level outside the range marked by the performance of the global all-to-all transformation.

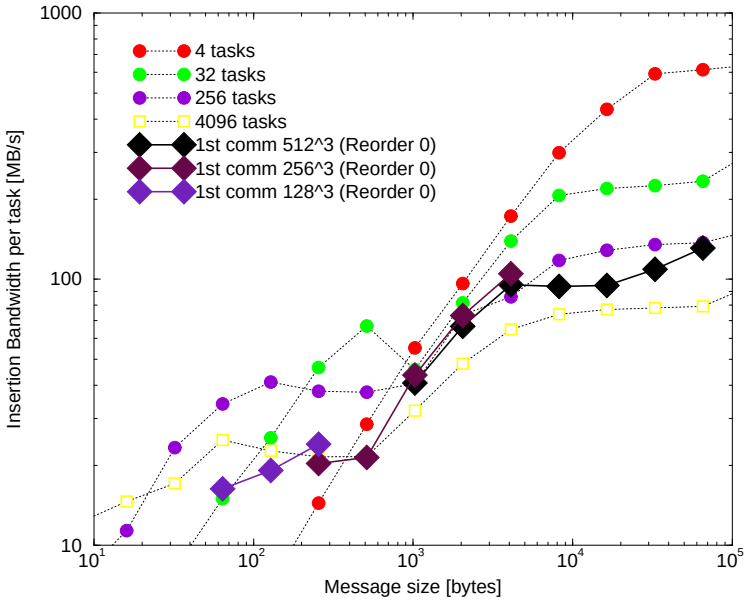

Figure 5: Comparing the performance of the individual all-to-all communications for a FFT algorithm using 4096 tasks. Closed diamonds use `MPICH_RANK_REORDER_METHOD = 0`. The circles repeat a subset of the results from Figure 1.

5.2 Communication times

For the results presented in the figures 4 and 5 we have assumed that data remaining on the task does not get inserted into the network. While this is a good metric in the quest to understand the performance results, it is not the best metric if one is interested in the time taken for the communication to finish. For small communicators, that is a small number of tasks working on a row or a column of the virtual processor grid, the "non-insertion" of data to stay local reduces the amount of data travelling in the network, leading to additional advantages with respect to a time metric. The discussion of the previous subsection does not show that clearly enough. For this reason we would like to discuss the timings for the two of the cases, which we investigated.

Both cases use 256 tasks for the parallel FFT. The first case is for the 128^3 sized problem, while the second case is for the 512^3 sized problem. Figure 6 displays the results against the processor grid. The figures show the time for the first and the second communication phase, the total communication time (which is the sum of these) and the total time of the algorithm. All results are give for both rank reorderings, showing the effect of the intra-node communication.

One notices the individual communications are fastest when there are only a few processors assigned to the communication group. However the two discussed cases have different reasons, which then lead to opposite conclusions with respect to what is best. For the smaller problem, we operate in the regime where the available bandwidth is highly dependent on message size. Increasing either communicator occurs a substantial bandwidth penalty, which can not be compensated for by the other communication phase being faster. The clear result in this case is that, it is best to use a decomposition as symmetric as possible, which is 16×16 for 256 tasks.

For the larger problem (512^3) the situation is different. The message sizes are in the range where the performance is essentially independent of the message size. Hence using more tasks for either the rows or columns of the virtual processor grid does not incur a significant bandwidth penalty. On the other hand, reducing the size of either communication group leads to increasingly larger amounts of data staying on the task. This reduces the data inserted into the network and hence the time required for the communication. We get a situation when we can gain an advantage on one of the communications by reducing its task count without penalising the other group due to the necessary increase of its communicator. We see best

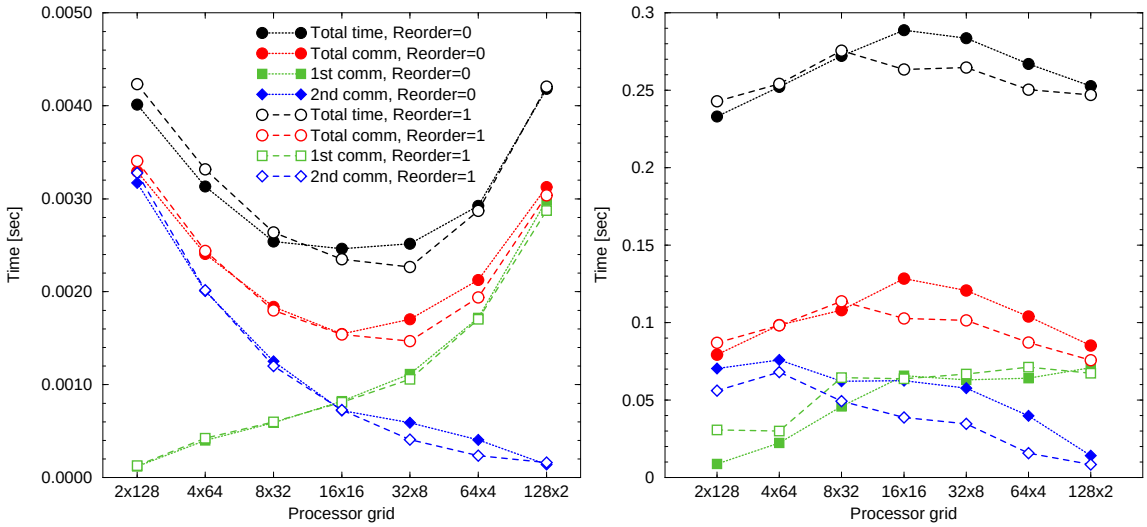

Figure 6: Timings for the parallel FFT algorithm when using 256 tasks for a 128^3 sized problem (left hand side) and a 512^3 size problem (right hand side). Full symbols show timings for `MPICH_RANK_REORDER_METHOD`=0, while open symbols represent `MPICH_RANK_REORDER_METHOD`=1.

performance when the grid is chosen as unbalanced as possible.

For the larger 512^3 sized problem we see clear effects from internode communication. Whenever both cores of a node are taking part in the same communication group, this gives better performance than assigning them to different communication groups. However these effects are small. Overall there appears to be a small advantage to have both cores of a processor within the smaller communication group.

5.3 Using fat SMP nodes on HPCx

The present HECToR system offers dual core nodes, offering faster intra node communication between only two tasks. With the present trend to larger core counts per processor, future systems will allow for larger task counts per processors. To illustrate the effects one might see on a system utilising quad-core or even octo-core nodes, we re-analyse the results from [7]. In this reference the effect of different virtual processor grids on the performance of the parallel FFT has been studied on the HPCx system, which is a cluster using 16-way SMP nodes.

In Figure 7 we show the results for a 256 processor run. Unfortunately, for these tests the results proved to be noisy. This could have a large number of causes, including disturbance by other users' jobs, since the measurements have been conducted during normal user operation. The graphs give results for the minimum observed times and the averages over all measurements.

On HPCx we placed our tasks as follows: Starting with the first row, the tasks have been placed onto the first 16-way SMP node. If there were fewer than 16 tasks per row, we continued with placing the second row onto the first node and so on until this node was full. If there were more than 16 tasks per row, we placed the first 16 task of the row on the first node and the next 16 on the second node and so on until the entire first row has been placed. This way, intra-node communication is always part of the row communications. The communications within the columns of the virtual processor grid always contains communication via the switch network.

On HPCx we notice a very substantial performance difference between intra- and inter-node communication, marked by the difference between the blue and the green curves of the figure for up to 16 tasks per grid row. It is interesting to see that once the grid rows exceed 16 tasks, the performance does not directly drop to the performance level of the inter-node communication. Most likely the all-to-all library is well aware of the 16-way nodes and does not insert data destined for a different task on the same SMP-node into the

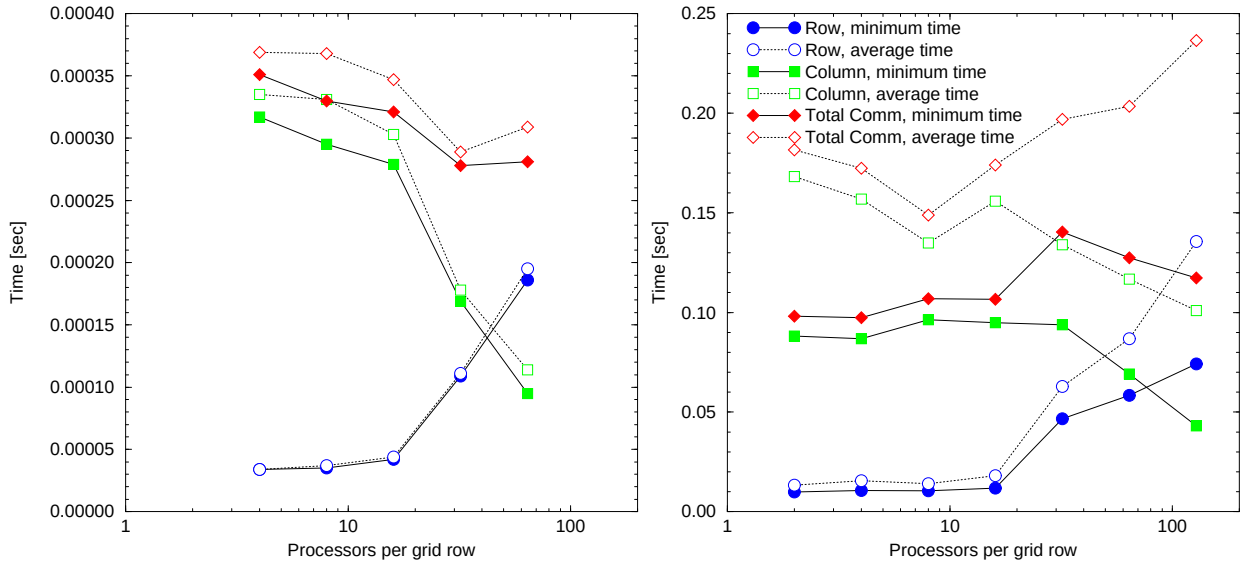

Figure 7: Timing results for different virtual processor grids on the HPCx system. The left hand side shows results for a small 64^3 sized problem, while the right hand side results for a larger 512^3 problem. The total task count is 256 tasks. Full symbols are for the best observed times, while open symbols are for the averages.

network. This would save precious insertion bandwidth.

On the other hand we notice that for the columns the performance improves when the number of tasks (typically located on different nodes) is decreased. We identified two underlying effects, which we have already discussed for the HECToR system. A decreased task count per grid column leads to an increased message size. This in turn, can lead to an increased insertion bandwidth, speeding up the communication. From the two cases discussed this is relevant for the 64^3 sized problem only. In addition, both problem sizes, 64^3 and 512^3, appear to benefit from smaller column communicators, for which the data to stay on the processor becomes sizable. As a result of these effects, we notice that for the small problem it is best to use a row communicator which is larger than the size of the nodes. This leads to substantially shorter times for the slow inter-node column communications, while the row communications still show a sizable benefit from intra-node communication. For the larger problem of 512^3 the message is not fully clear, partly due to the large variations in run time. To a large extent the different effects cancel each other out. This leads to a performance which is not overly sensitive to different virtual processor grids.

5.4 Processor mesh on BlueGene/L

In subsection 3.2 we discussed that for messages larger than 8 kB the performance of an all-to-all communication decreases with increasing task count. If this is caused by congestion in the network, the situation might be improved by clever task placement, if the Cray XT architecture would allow users to do so. We want to illustrate this with results measured on the BlueSky system and reported in [6].

In contrast to the Cray XT architecture, the IBM BlueGene architecture schedules user jobs on dense partitions of cuboidal shape, with no holes inside due to service nodes or nodes utilised by other users' jobs. It further allows the user to take full control over the placement of the task onto the machine.

We compare the performance of the parallel FFT benchmark on a 512 dual core node partition on the BlueSky system for the default mapping and a customised mapping. A 512 node partition of a BlueGene/L system comes with a $8 \times 8 \times 8$ mesh network. The comparison is performed in CO mode, placing a single compute task on a dual core node, leaving the second core as a communication co-processor, and VN mode, which places two compute tasks on a node, one on each core.

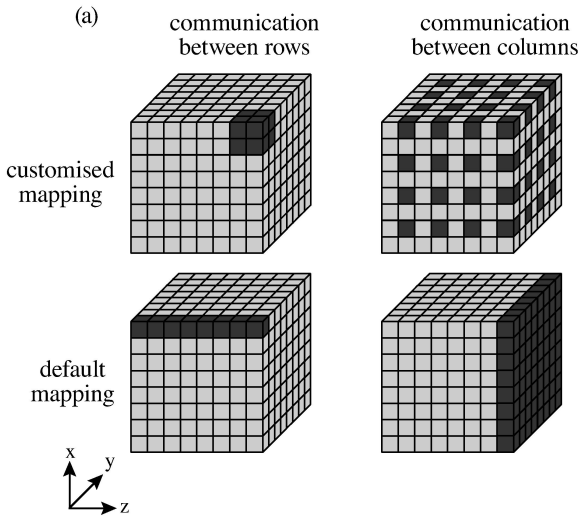

Figure 8: Illustration of the mapping of the rows and columns of the virtual processor grid onto the physical hardware of a BlueGene/L system. The dark parts of the figure shows how a single grid row or column is placed. To place the remaining rows and columns the entire machine has to be filled with translated versions of the shown basic maps.

In CO-mode, we investigated a 8×64 virtual processor grid. By default the machine will map the row-communicators of size eight onto sticks of eight nodes on the meshed network, while the 64 sized column-communicators will be mapped onto 8×8 planes of the mesh. Based on considerations of the bi-section bandwidth, reference [6] argues that mapping the rows of size eight onto small cubes of size $2 \times 2 \times 2$ should offer a significantly improved performance. When placing the rows of the virtual processor grid onto small cubes, the images of the columns consist of isolated processors, which fill the entire mesh of the machine in a regular pattern. The basic images of a single row or column for the default and customised mappings are shown in Figure 8.

We also investigated similar mappings in VN mode offering 1024 tasks on a 512 node partition. For this we investigated virtual processor grids of 16×64 and 8×128. For the processor grid of 16×64 we placed grid rows of 16 tasks onto the small 2^3-cubes or the sticks of eight nodes in Figure 8, two tasks on each node of the basic image. For the 8×128 two rows of eight tasks got placed on each 2^3-sized cube, two tasks from different rows on each node of the cube. In all cases the remaining rows and columns get placed on translated copies of the basic images.

Figure 9 gives an overview on the results. To obtain a more readable figure, the results have been normalised with the performance obtained with a 64×8 virtual processor grid in CO mode using the default mapping. The results show that in CO mode, there is only a small advantage to using the customised mapping. This advantage is substantially smaller than expected. In [6] this was traced to the machine not being able to saturate the links at the bi-section when using a single task per node for the small 2^3 sized cubes. However in VN mode, placing two tasks per node, the system architecture is able to saturate the links at the bi-section and one obtains a very sizable performance boost of up to 16% from using the customised mapping when compared to the default mapping.

6 Conclusion

The performance of a multidimensional parallel FFT using a large number of processors relies critically on the performance of the all-to-all communication available on the computer architecture. For a small task count, the all-to-all communication operation available as part of the standard system software on the Cray XT4 shows a performance comparable to the performance level observed with a simple Ping-Ping test.

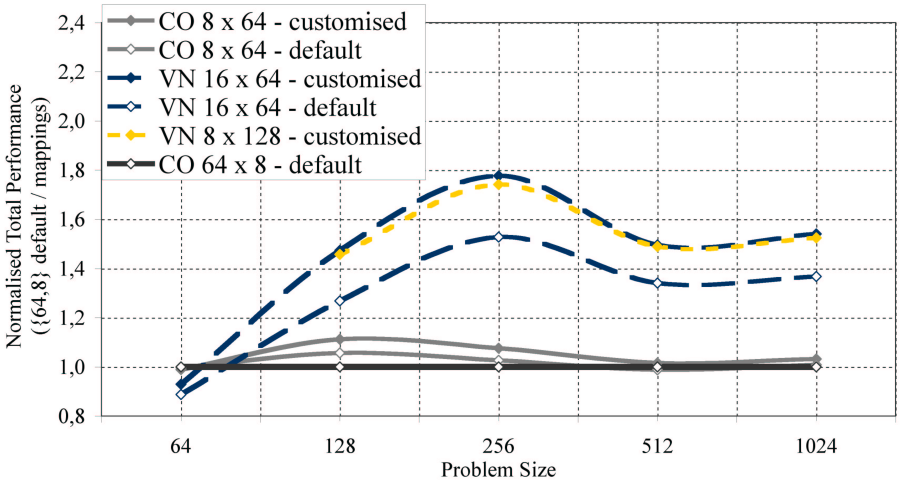

Figure 9: Performance results for the full parallel 3D-FFT using different mappings between the virtual processor grid and the physical meshed communication network on a BlueGene/L system.

When using hundreds or even thousands of tasks for messages of several kilo bytes, this performance level can not be maintained. In particular, the observed performance does not match expectations raised by the Cray marketing materials. Despite these shortcomings, for an all-to-all communication of 1024 tasks, the HECToR system, based on the Cray XT4 architecture, still manages to give a better performance than the other systems used in this investigation. However this does not hold for every message size.

In order to utilise hundreds or thousands of processors in a parallel FFT of a moderately sized array, the array has to be decomposed onto a virtual processor grid of dimension two or higher. These virtual processor grids offer additional parameters, such as the extent of the grid in the different dimensions, which can be utilised to optimise the performance of the FFT. When using a processor grid of dimension two or higher, the algorithm has to exchange its data via all-to-all communications, which are executed concurrently on disjoint sub-communicators of the underlying communicator.

On the HECToR system, we have shown that the performance of many all-to-all communications operating concurrently does not yield a performance outside the range observed for a single global all-to-all communication. When using thousands of tasks to run hundreds of small task all-to-all operations concurrently, as required for the parallel FFT algorithm, we observe a performance which is more comparable to the inferior performance associated with hundreds of tasks than the superior perfomance we observe for an individual all-to-all communications with a task count matching the size of the subgroup. This might indicate that the above mentioned performance problems are caused by network congestion.

We show that there is no general rule as to whether symmetric or asymmetric virtual processor grids offer better performance. For messages for which there is a strong dependence of the all-to-all performance on the size of the messages, it is typically best to use more symmetric processor grids. If the dependence of the all-to-all performance on the message size is small, it is better to use asymmetric processor grids to reduce the data volume.

The investigations show that using nodes with SMP capablity (clustered SMP or modern multicore processors) can be exploited to improve the performance of individual communications. They even show performance improvements for communication groups involving more tasks than can be placed on a single node, as long as the number of nodes involved in the communication is a single digit figure. However in practical experiments, it turns out that the gains for the total performance of the FFT are modest. This is mostly due to the second communication, which has to utilise the communication network, involving a large number of tasks and hence having to insert almost all data into the network. Naive expectations of dramatic performance improvements due to individual communication groups using fast intra-node communication

did not materialise in our tests. However in special cases, e.g. FFT of a 64^3 array on 256 nodes on HPCx, advantages can prevail.

As shown on a BlueGene/L system for a system using a mesh network, mapping communication groups onto small cubes inside the larger mesh can lead to significant performance advantages. Unfortunately the standard system software on the Cray XT4, at the time of writing, does not offer the required control to the user. If this were implemented, it might help with the performance problems we observed when running many small all-to-all communications concurrently on a very large number of nodes, at least for one of the communication groups of the FFT algorithm.

6.1 About the Authors

Dr Joachim Hein works as computational architect for the HPCX consortium and EPCC. Heike Jagode is a Sr. Research Associate at the University of Tennessee in Knoxville and the Oak Ridge National Laboratory. Ulrich Sigrist was a student in EPCC's MSc program in High Performance computing. Dr Alan Simpson is the service director of the HPCX consortium and the technical director of EPCC and Prof Arthur Trew is senior industry executive of the HPCX consortium and the director of EPCC.

6.2 Acknowledgements

We would like to thank Mark Bull (EPCC) and Stephen Booth (EPCC) for useful discussions. The continued support from David Tanqueray, Jason Beech-Brandt, Kevin Roy and Martyn Foster from Cray with respect to our queries on the architecture is gratefully acknowledged.

References

[1] F. Franchetti, Y. Voronenko, M. Püschel, "FFT Program Generation for Shared Memory: SMP and Multicore", Paper presented as SC06, Tampa, FL, http://doi.acm.org/10.1145/1188455.1188575

[2] 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 194-203. Springer-Verlag, 2003.

[3] M. Eleftheriou, et al., "Scalable framework for 3D FFTs on the Blue Gene/L supercomputer: Implementation and early performance measurements", IBM Journal of Research and Development, Vol. 49, page 457, 2005.

[4] M. Eleftheriou, et al., "Performance Measurements of the 3D FFT on the Blue Gene/L Supercomputer", in J.C. Cunha and P.D. Medeiros, editors, volume 3648 of Lecture Notes in Computer Science, page 795. Springer-Verlag, 2005.

[5] 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/

[6] H. Jagode, J. Hein, A. Trew, "Task placement of parallel FFTs on a mesh communication network", Technical Report No. ut-cs-08-613, 2008, http://www.cs.utk.edu/ library/2008.html

[7] U. Sigrist, "Optimizing parallel 3D Fast Fourier Transformations for a cluster of IBM POWER5 SMP nodes", MSc thesis, The University of Edinburgh, 2007,
http://www.epcc.ed.ac.uk/msc/dissertations/2006-2007

[8] "Application performance on the UK's new HECToR service", Fiona Reid, et al., this volume.

[9] Cray XT4 datasheet, http://www.cray.com/downloads/Cray_XT4_Datasheet.pdf

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

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

[12] Intel® MPI Benchmarks, Version 3.0,
http://www.intel.com/cd/software/products/asmo-na/eng/219848.htm

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