

File I/O from multi processor jobs

Joachim Hein
HPCx Terascaling Team
EPCC
The University of Edinburgh

June 23, 2003

Abstract

In this study we investigate the I/O performance on HPCx when several tasks are accessing the disks at the same time. Going through more than one pair of switch adapters when accessing the disks leads to a substantially improved I/O rate. As a direct consequence, a master/worker model for disk-I/O can not be recommended on HPCx if I/O performance is crucial for your application.

1 Introduction

Some scientific codes running on modern massively parallel super computers produce large amounts of data which needs to be written to disk and read back into the machine on a subsequent run. Often, this data is distributed over the individual nodes involved in the run. The easiest solution for writing or reading this data is, each task reads/writes its own file or all tasks read/write to different positions in the same random access file¹. A widely used alternative is to use a master/worker model. The processors send their data to a master node, which in turn communicates with disk. Compared to the above, this is more complex to set up. The performance of the even more sophisticated MPI-I/O is discussed in [1].

In this investigation we study three benchmark style codes in which each task writes/reads its data directly to/from disk. In the first benchmark each task has its own file, in the second benchmark all tasks write to a single random access file. The case of all tasks reading the same data file is investigated in the third benchmark. This study aims to investigate whether a large number of task accessing the disks simultaneously will degrade I/O performance. This is of particular importance for task farms, where several copies of the same code are started within a short time span.

In section 2 we review the relevant pieces of the hardware. Section 3 gives details on the benchmarks and presents the results for the case of each processor holding different data. The case of each processor reading the same data is discussed in section 4. Our findings are discussed in section 5.

2 Hardware details

The HPCx system consists out of 40 IBM p690 Regatta H frames. Each frame has 32 POWER4 processors with a clock of 1.3 GHz, a total of 1280 processors. This provides a peak performance of 6.6 Tflop/s and up to 3.2 Tflop/s sustained performance. Per frame, these processors are grouped into 4 *multi chip*

¹Random access files are also know as direct access files

modules (MCM). Each MCM has 8 processors and 8 GB of main memory installed. In order to increase the communication band width of the system, the frames have been divided into 4 *logical partitions* (lpar), coinciding with the MCMs. Each lpar is connected to the network via two PCI cards, also known as switch adapters. Each switch adapter is connecting the lpar to one of the two planes of the communication network. The disks of HPCx are attached to 4 designated I/O-lpars, which are not used for the execution of user codes. For a user code to access the disk, it has to use the communication network to transfer the data to the I/O-lpars, which in turn send it to the disks.

3 Processors hold different data

As noted above, we studied the performance of two benchmark codes for the case each processor holds different data. In the first code, each task writes its own file. This was implemented in Fortran using MPI to synchronise the tasks directly before the opening call for the files. Using MPI_Wtime we measured the time for the open, the write/read and the close statement. In this study we report the time taken for the slowest processor to finish the whole sequence including the open and close. Individual processors can finish substantially faster.

In the second benchmark, written in C, the tasks write to different positions in a single random access file. In this case the tasks get synchronised before the read/write statement and we measure the time for the read or write only.

For both benchmarks each task writes or reads 500 MB of data. Choosing 8 active tasks per lpar, this is about half of the main memory available on the lpar. This test has been performed in a production environment, with other users using the network and disk system for their needs. We performed at least 5 runs for a given set of run time parameters. For each set we report on the fastest run observed. For these fastest observed times no significant difference between accessing single or multiple files was seen. There is also no significant difference between writing and reading. Obviously slow results depend on the usage of the network and I/O-lpars by other users and are difficult to reproduce.

In figure 1 we show the best results obtained when writing multiple files. The numbers give the quotient of the total exchanged data volume exchanged between all the processors and the disk system² divided by the time for the slowest processor to finish. For this test we choose 8 tasks per lpar, resulting in one pair of switch adapter for 8 tasks. When using 8 tasks, we observe a speed of 315 MB/s. Our tests of the communication rate between processors on different lpars give similar numbers, suggesting the write/read speed is limited by the rate at which the switch adapters can pump the data into the network. The I/O speed increases when using more lpars, to plateau at about 700 MB/s for 4 or more lpars involved in the I/O. The rate at which the data can be pumped into the network is no longer the limiting factor for the I/O speed. To check whether an even larger number of switch adapters does not make a difference, we repeated the test with 32 lpar, 64 lpar and 128 lpar. In these runs we reduced the number of task per lpar such that each run had a total of 128 tasks. The observed rates are compatible with the ones in figure 1 for more than 32 processors.

4 Read same data into different processors

In this section we report on the case of all processor simultaneously reading the same data from disk. This benchmark was again implemented in Fortran using MPI to synchronise before the read. Each MPI-task reads 500 MB of data, all tasks read the same file. Again the reported rates are the total volume of exchanged data divided by the time for the slowest processor to finish. The results depend on the number of active tasks per lpar and are reported in figure 2. The observed rates for 8 or more lpars and 1 task per lpar are about the same as reported in figure 1, perhaps slightly larger. This might or might not be significant in a more detailed comparison study between the two benchmarks. Having

²500 MB times the number of active tasks

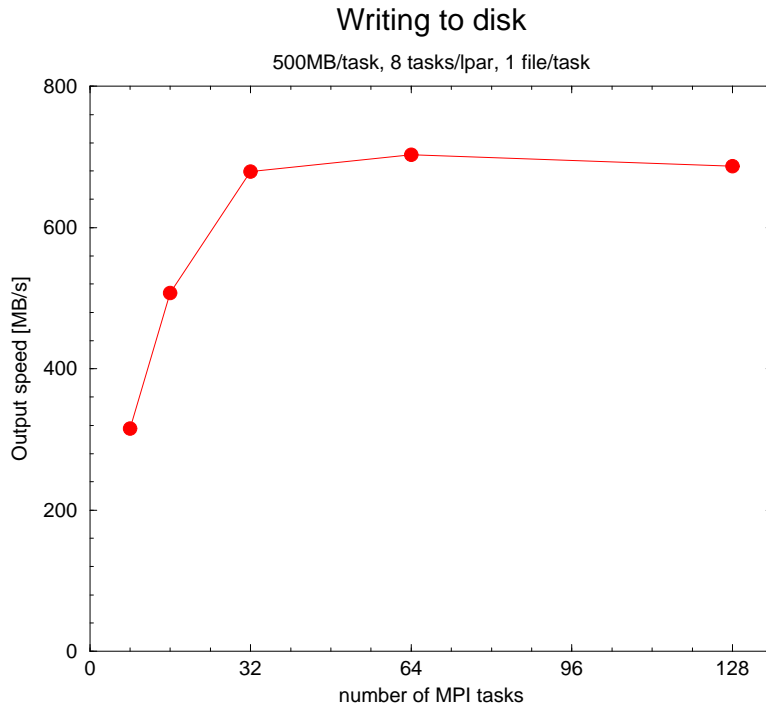


Figure 1: Write speed vs number of MPI tasks on HPCx. For the test we used 8 task per lpar, which is a switch adapter pair per 8 tasks. Each task writes its own file.

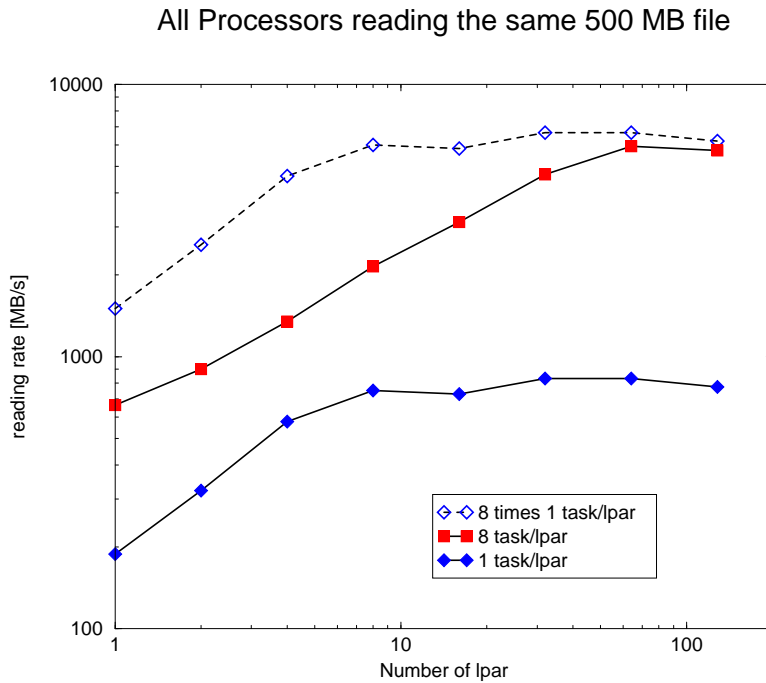


Figure 2: Reading rate for all processes reading the same file against the numbers of active lpar. The curve at the bottom is for 1 active task per lpar, the one in the middle for 8 active tasks per lpar and the top curve is 8 times the rate for 1 active task per lpar.

more than one active task per lpar results in a substantially larger rate. We expect this is due to caching in the lpar. This view is supported by the rate saturating at 8 times the rate observed for 1 task per lpar when using 64 and more lpars. Each lpar reads the data only once and distributes internally to all of its 8 task. For fewer than 64 lpar, the data streams into the lpars at a rate at which it can not distribute internally. Hence some of the data has to be send to the lpar more than once. The resulting rate is still faster than the single task per lpar rate. Increased rates due to caching of the I/O inside the lpars have also been reported in [1].

5 Discussion

In this investigation we study the File-I/O performance on HPCx for several processors writing or reading simultaneously. We did not observe a significant difference between writing or reading. Neither did it make a difference whether each processor writes its own file or all write into the same random access file. If all active processors are located on the same lpar, they have to share a switch adapter pair to the network and the I/O rate is significantly reduced compared to going through several switch adapters. When all active tasks are reading the same data, we observe substantially larger rates when using more than one active processor per lpar. We believe this is due to caching of the I/O inside the lpar.

As a direct consequence of these findings, on HPCx the use of a master/worker model for disk-I/O can not be recommended. In this model all processors would need to send their data through a single pair of switch adapters to single master processor, which in turn writes it to disk, using the same pair of switch adapters. If each processor would be writing its own data directly to disk, a much higher rate can be realised.

Once the writing tasks are associated with at least 4 different lpars, I/O rates of around 700 MB/s are possible. This rate does not decay when increasing the number of writing tasks to 128. This is important for a task farm. Especially when starting the farm, it has to be expected that all task want to initialise themselves from disk within a short time span. In this case the start-up will be slowed down by the maximum possible rate, but not beyond this point.

References

- [1] Elena Breitmoser, "Investigating MPI-IO on HPCx", HPCx TR0304, to appear.