Proceedings of the International MultiConference of Engineers and Computer Scientists 2008 Vol I
IMECS 2008, 19-21 March, 2008, Hong Kong

Feasibility Study of Effective Remote I/O Using a Parallel NetCDF Interface in a Long-Latency Network

Yuichi Tsujita *

Abstract—**NetCDF provides portable and self-describing I/O data format for array-oriented data in scientific computation domains. Its parallel I/O interface named parallel netCDF (hereafter PnetCDF) provides parallel I/O operations with the help of an MPI interface. To realize such operations among computers which have different MPI libraries through a PnetCDF interface, a Stampi library was introduced as an underlying MPI library. Decomposition in multi-dimensional data leads to complex I/O operations in non-contiguous parallel I/O pattern with the help of a derived data type. However, times for data communications between computers are independent of the complexity because the transfered data are contiguous in memory buffer. Although it succeeded in effective remote I/O operations in a LAN environment, throughput is degraded due to unoptimized configuration for TCP sockets if interconnection among computers has long latency like WAN. We have addressed the degradation and we observed that applying an appropriate size in a socket buffer minimized I/O times effectively.**

keywords: parallel netCDF, MPI-I/O, Stampi, MPI-I/O process

1 Introduction

NetCDF [1, 2] is a popular package for storing and retrieving data files in scientific computation domains. It provides a view of data as a collection of self-describing, portable, and array-oriented objects that can be accessed through a simple interface on a wide variety of platforms. For example, atmospheric science applications use netCDF to store a variety of data types that encompass single-point observations, time series, regularly spaced grids, and satellite or radar images [1]. PnetCDF, which is a parallel I/O interface for netCDF data, was developed with the help of an MPI library such as MPICH [3], and the PnetCDF succeeded in scientific computation domains [4]. Although it supports parallel I/O operations inside the same MPI implementations, seamless I/O operations to a remote computer have not been available for visualization of remote calculated results.

Stampi was developed to support seamless MPI operations including MPI-I/O operations among different MPI implementations without paying attention to complexity and heterogeneity in underlying communication and I/O systems [5]. The seamless operations are realized by deploying a wrapper interface library on top of each MPI library. MPI communications among different implementations are realized by using TCP sockets. It is available on a wide variety of supercomputers and UNIX based PC clusters. To realize remote I/O operations with a PnetCDF interface, MPI interfaces of the Stampi library were implemented in an underlying MPI layer of the PnetCDF library.

In scientific applications, non-contiguous data are usually created. The more the data format becomes complex, the more the number of data communications between user processes increases. As a result, this leads to performance degradation. However, I/O times do not increase so much in this system because such the communications are carried out among MPI-I/O processes on a remote computer.

The implemented system provided sufficient performance for remote I/O operations in a LAN environment. However, throughput is degraded due to unoptimized socket buffer size when we carry out the same operations among computers which are connected via a network with long latency like WAN. It is well known that applying twice the product of bandwidth and latency in socket buffer size provides better performance [6]. To improve throughput, we have studied optimization for TCP connections to find a desirable solution for effective remote I/O by tuning socket buffer size.

In the rest of this article, architecture of Stampi and implementation of it in a PnetCDF's MPI layer are explained in Section 2. Performance measurement is reported in Section 3. Related work is mentioned in Section 4, followed by conclusions in Section 5.

*Department of Electronic Engineering and Computer Science, School of Engineering, Kinki University, 1 Umenobe, Takaya, Higashi-Hiroshima, Hiroshima 739-2116, Japan
Email: tsujita@hiro.kindai.ac.jp

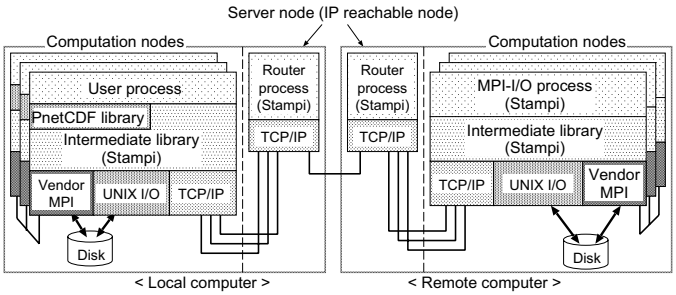


Figure 1: Architecture of a seamless remote I/O system.

2 Remote I/O in Stampi and Its PnetCDF support

In this section, details of architecture and execution mechanism of the remote I/O system are explained and discussed. Its architecture is illustrated in Figure 1. Since MPI functions are used in PnetCDF functions, almost all the MPI functions have been replaced with Stampi's MPI functions to realize seamless remote I/O operations with a netCDF data format. In the next sections, we explain a mechanism of a remote I/O system with an MPI-I/O API and PnetCDF support in the remote I/O system.

2.1 Remote I/O with an MPI-I/O API

We have adopted a Stampi library to support remote I/O operations. It supports seamless accesses in both local and remote I/O operations. Inside a local computer, a Stampi's start-up command (Stampi starter; `jmpirun`) calls a native MPI start-up command (MPI starter) such as `mpirun`. Later, user processes are initiated by the native MPI start-up command. Stampi's MPI function calls are translated into native MPI function calls in this case. This leads to high performance MPI operations including MPI-I/O operations by using a vendor's MPI library through the Stampi's MPI functions. If the vendor's one is not available in MPI-I/O operations, UNIX I/O functions are used instead of it.

On the other hand, remote I/O operations are realized as illustrated in Figure 2. The Stampi starter calls an MPI starter and the MPI starter initiates user processes. Moreover, a router process is initiated by the Stampi starter to relay message data from/to the user processes if computation nodes where the user processes are initiated can not communicate outside directly. When Stampi's `MPI_File_open()` is called in an MPI program to open a remote file, another Stampi starter is invoked on the remote computer by the Stampi starter or the router process by using a remote shell command (rsh or ssh). The invoked Stampi starter kicks off MPI-I/O processes on computation nodes by using a native MPI starter. If the computation nodes can not communicate outside directly, a router process is invoked by the starter. Finally, a communication path is established among the user processes

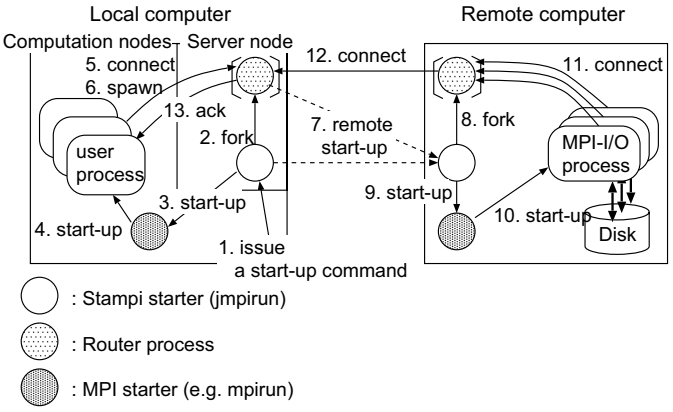


Figure 2: Execution steps of remote I/O operations.

and the MPI-I/O processes.

Each I/O request of Stampi's MPI functions is transfered from the user processes to the MPI-I/O processes so that the MPI-I/O processes play the requested I/O operations. The operations are carried out on a target computer by using a vendor's MPI-I/O library or UNIX I/O functions if the vendor's one is not available. Associated parameters for derived data types such as a unit data type, the number of units in each block, and the number of bytes between the start of each block are stored in a linked list based table in the user processes. The same parameters are transfered to the similar table of MPI-I/O processes. The same derived data type and file view are created in both the user processes and MPI-I/O processes.

Once the derived data type is created, each user processes can start remote I/O operations with the derived data type. Figure 3 shows typical remote collective read operations with a derived data type support by four processes. A target data file is read by all the MPI-I/O processes in the collective manner. Each MPI-I/O process sends the data to a corresponding user process. The user processes carry out all-to-all data communications to reorder the read chunks of the data file. It is remarked that data to be transfered between the user processes and the MPI-I/O processes are contiguous in memory. Thus, the data transfer time is dependent of data size, however, independent of complexity of a derived data type.

After I/O operations, the MPI-I/O processes close the opened file when a function to close the file is called. Finally, they are terminated and whole I/O operation finishes.

2.2 PnetCDF Support in Remote I/O Operations

We have realized PnetCDF support by introducing the Stampi library into a PnetCDF's underlying MPI interface. Sequence of function calls in typical write and read programs is illustrated in Figure 4. When user processes

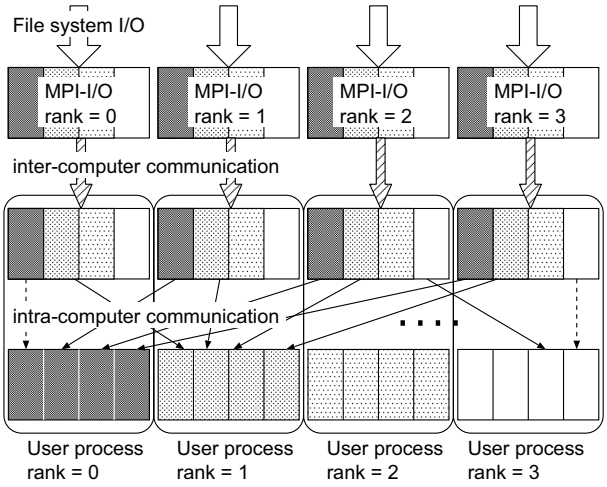
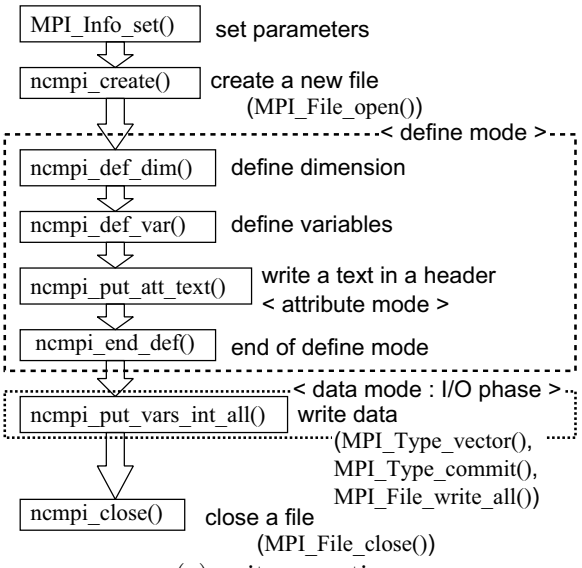


Figure 3: Data flow of remote collective read operations using a derived data type. **MPI-I/O** denotes an MPI-I/O process on a remote computer.

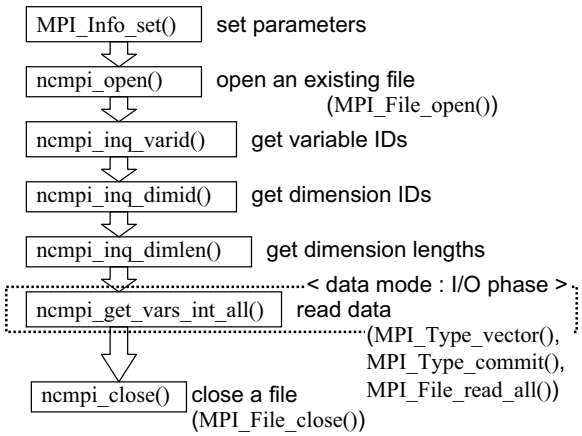
call PnetCDF functions, those function calls are translated into several MPI function calls based on the original PnetCDF implementation. In the write operation, parameters which are associated with the I/O operations such as a hostname of a remote computer, a user ID, a working directory, and a file name are specified in an `MPI_Info` object with the help of `MPI_Info_set()` prior to I/O operations. Stampi's MPI functions identify which operation is requested, local or remote I/O operations, according to them. When `ncmpi_create()` is called, Stampi's `MPI_File_open()` is called to invoke MPI-I/O processes on a remote computer. Later they creates a new netCDF file according to the parameters. Information values for array data are written in a record header of it in a define mode by using several PnetCDF functions such as `ncmpi_def_dim()`. The array data are written in it by using `ncmpi_put_vars_int_all()` in the collective manner. Inside the function, several MPI functions including MPI-I/O functions are used. Finally the file is closed by `ncmpi_close()`, followed by calling `MPI_File_close()`, and all the I/O operations finish. On the other hand, the read operation is carried out in the same manner except that inquiry of parameters from the record header and reading data from the file.

3 Performance Results

To evaluate the implemented remote I/O system, its performance was measured on two interconnected PC clusters. Specifications of the clusters are summarized in Table 1. Each cluster had one server node and four computation nodes. All the PC nodes were connected via each Gigabit Ethernet switch. One FreeBSD PC server was located between the clusters as a gateway with 1 Gbps connections.



(a) write operation



(b) read operation

Figure 4: Sequence of function calls in collective (a) write and (b) read programs.

In a PC clusters I and II, MPICH [3] version 1.2.7p1 was used as a native MPI library. In a PC cluster-II, PVFS2 [7] version 1.4.0 was available on its server node by collecting disk spaces (73 Gbyte each) of four computation nodes. Thus 292 Gbyte (4×73 Gbyte) was available for the file system. During this test, default stripe size (64 Kbyte) of it was selected. A Stampi library was available on both the clusters. Several network latencies were applied to the network connections by dummynet [8] of the FreeBSD PC server.

User processes were executed on the PC cluster-I and MPI-I/O processes were invoked on the PC cluster-II by rsh. A router process was invoked on each server node of both the clusters because each computation node was able to communicate outside via own server node only.

PnetCDF functions were evaluated using three-dimensional data. The data set with $16 \times 16 \times 16$

Table 1: Specifications of PC clusters which were used in performance evaluation, where **server** and **comp** in bold font denote server node and computation nodes, respectively.

	PC cluster-I	PC cluster-II
server	DELL PowerEdge800 × 1	DELL PowerEdge1600SC × 1
comp	DELL PowerEdge800 × 4	DELL PowerEdge1600SC × 4
CPU	Intel Pentium-4 3.6 GHz × 1	Intel Xeon 2.4 GHz × 2
Chipset	Intel E7221	ServerWorks GC-SL
Memory	1 Gbyte DDR2 533 SDRAM	2 Gbyte DDR 266 SDRAM
Disk system	80 Gbyte (Serial ATA) × 1 (all nodes)	73 Gbyte (Ultra320 SCSI) × 1 (**server**) 73 Gbyte (Ultra320 SCSI) × 2 (**comp**)
NIC	Broadcom BCM5721 (on-board)	Intel PRO/1000-XT (PCI-X card)
Switch	3Com SuperStack3 Switch 3812	3Com SuperStack3 Switch 4900
OS	Fedora Core 3	kernel 2.6.12 (**server**)
	Fedora Core 3	kernel 2.6.11 (**comp**)
NIC driver	Broadcom tg3 version 3.71b	Intel e1000 version 6.0.54

(16 Kbyte), $64 \times 64 \times 64$ (1 Mbyte), $100 \times 100 \times 100$ ($\sim$ 3.8 Mbyte), $128 \times 128 \times 128$ (8 Mbyte), and $256 \times 256 \times 256$ (64 Mbyte) were prepared with an integer data type. In this test, times to issue the I/O functions in the data mode were measured using `MPI_Wtime()` in a test program.

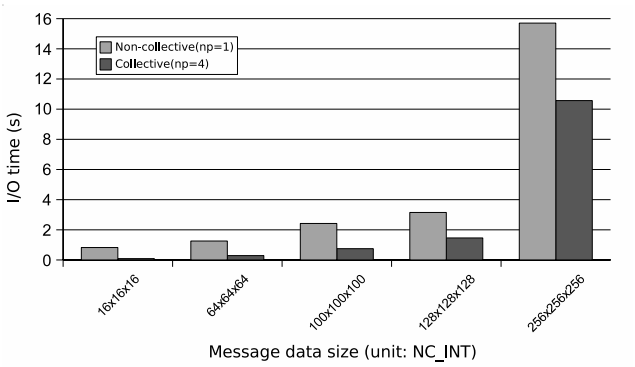
Firstly, we compared I/O times of non-collective and collective PnetCDF functions with 5 ms latency in the interconnected network. In the non-collective case, we executed a single user process on the cluster-I and the same number of MPI-I/O process was invoked on the cluster-II. Data to be written or read were managed by the single process, thus a derived data type was not used. On the other hand, four user processes were executed on the cluster-I and the same number of MPI-I/O processes were invoked on the cluster-II. Data were split evenly into four pieces and each piece was managed by one of the user processes in the collective manner. MPI-I/O processes carried out I/O operations according to a created derived data type.

Measured times were shown in Figure 5. In both the operations, the collective case quite outperformed the non-collective one. This was due to data buffering of the MPICH library in the collective operations. Each MPI-I/O process managed data of the corresponding user process, and the process was able to cache the data in the collective case. While a single MPI-I/O process was not able to cache all the data due to memory size constraint for the data buffering in the non-collective case. As a result, the non-collective case took longer time than that in the collective case.

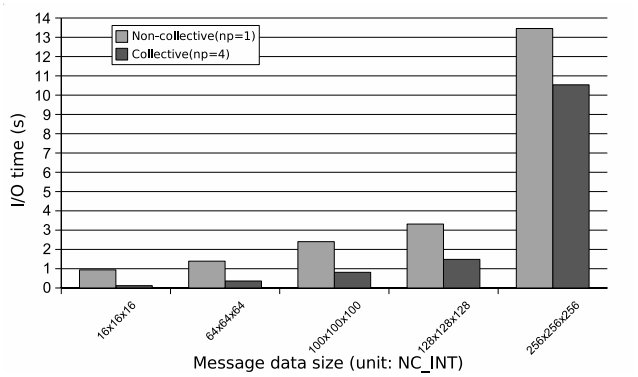
Secondly, times for remote collective I/O operations were measured with respect to maximum socket buffer size. The size was adjusted by `rmem_max` and `wmem_max` in `/proc/sys/net/core/`. I/O times for four user processes are shown in Figure 6. It is noted that applying the twice the product of network latency and bandwidth (2×5 ms $\times 125$ Mbyte/s = 1.25 Mbyte and 2×50 ms $\times 125$ Mbyte/s = 12.5 Mbyte for the 5 ms and 50 ms cases, respectively.) in the maximum socket buffer size was sufficient to minimize the times. From these results, it is concluded that optimization in the socket buffer size was quite effective for long message data. However, the minimization effect in the read operations was smaller than that in the write operations in the case of 50 ms in latency. In the read operations, every user processes had to wait until whole data were transfered to an own user space memory buffer. While in the write operations, I/O function calls finished once the whole data were transfered to a kernel-space memory buffer. In this case, it is considered that data transfer from the PC cluster-I to the PC cluster-II and I/O operations on the cluster-II were continuing after the I/O function call finished. As a result, visible I/O times became shorter than the real I/O times.

4 Related Work

Providing a common data format makes data I/O operations not only portable but also tolerate for application programmers. This kind of implementations such as netCDF [1] and HDF5 [9] has been proposed. NetCDF provides a self-describing and common multi-dimensional data format and a simple interface. Its parallel I/O operations have been realized in parallel netCDF (PnetCDF), which is an extension of the interface, by introducing MPI-I/O functions as an underlying parallel I/O library [4]. On the other hand, HDF5 provides hierarchical data format so as to access a huge amount of data effectively. An HDF5 interface has two objects, one is "Dataset" and the other is "Group". The Dataset manages multi-dimensional array data, while the Group provides relational mechanisms among the objects. Parallel I/O operations are also available with this interface by introducing MPI-I/O functions as an underlying parallel

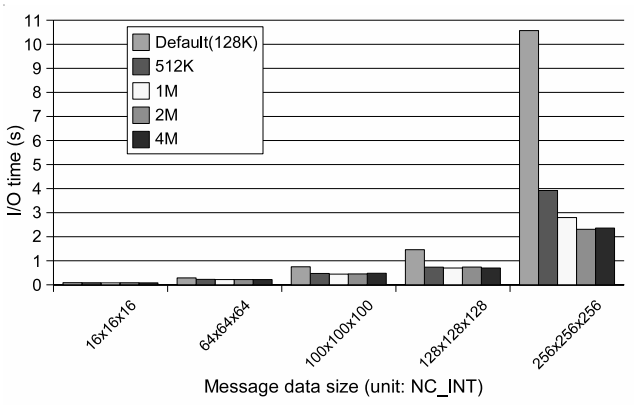


(a) read

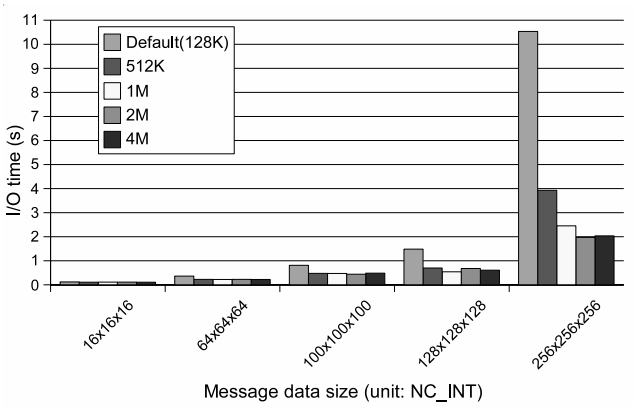


(b) write

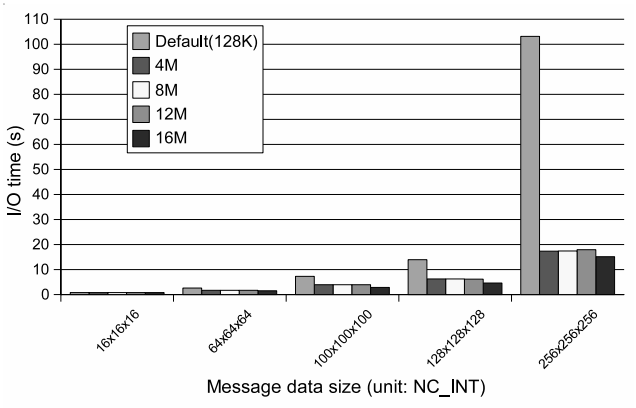
Figure 5: I/O times of non-collective and collective PnetCDF functions for 5 ms latency network.



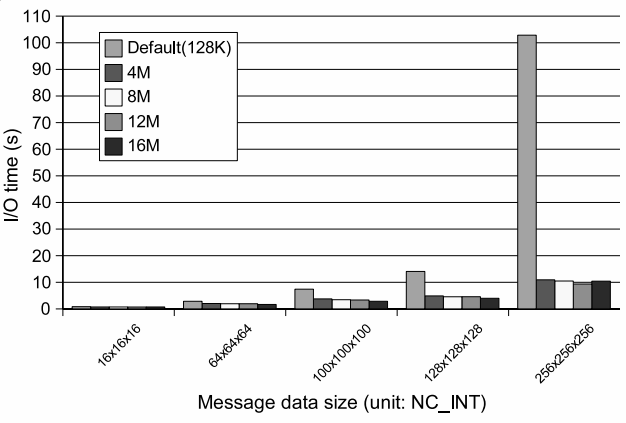
(a) read, latency = 5 ms



(b) write, latency = 5 ms



(c) read, latency = 50 ms



(d) write, latency = 50 ms

Figure 6: I/O times for collective PnetCDF functions using an integer data type. (a) and (b) show I/O times for read and write operations using a network with 5 ms latency, respectively. While (c) and (d) show the times for the same operations using a network with 50 ms latency. Number of each plot denotes a maximum socket buffer size for TCP sockets.

I/O interface library [10].

An MPI-I/O interface in the MPI-2 standard [11] realizes parallel I/O operations in an MPI program. Several implementations of it are available such as ROMIO [12]. Its MPI-I/O operations to many kinds of file systems are realized through an ADIO interface [13]. It hides heterogeneity in architectures of each system and provides a common interface to an upper MPI-I/O layer. Remote I/O operations using the ROMIO are available with the help of RFS [14]. An RFS request handler on a remote computer receives I/O requests from client processes and calls an appropriate ADIO library. On the other hand, Stampi itself is not an MPI implementation but a bridging library among different MPI implementations. It realizes seamless MPI operations among them by using TCP socket communications.

5 Conclusions

We have studied optimization effects in collective PnetCDF functions for remote I/O on a network with long latency. In this study, we have observed that collective operations outperformed non-collective ones in remote I/O operations. This was due to data buffering in an MPICH library layer of MPI-I/O processes. We also noticed that optimization in a maximum socket buffer size was quite effective. Applying the twice the product of bandwidth and network latency provided good performance.

However, network communication is still bottleneck in blocking operations. As a future work, we are planning to implement a non-blocking API to overlap of computation by user processes and I/O operations by MPI-I/O processes.

Acknowledgments

The author would like to thank the staff at Center for Computational Science and e-Systems (CCSE), Japan Atomic Energy Agency (JAEA), for providing a Stampi library and giving useful information.

This research was partially supported by the Ministry of Education, Culture, Sports, Science and Technology (MEXT), Grant-in-Aid for Young Scientists (B), 18700074 and the CASIO Science Promotion Foundation.

References

[1] Rew, R., Davis, G., Emmerson, S., Davies, H., Hartnett, E., *NetCDF User's Guide*, Unidata Program Center (available at `http://www.unidata.ucar.edu/software/netcdf/docs/netcdf/`), June 2006.

[2] Rew, R. K., Davis, G. P., "The unidata netCDF: Software for scientific data access," *Sixth International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology*, pp. 33-40, American Meteorology Society, February 1990.

[3] Gropp, W., Lusk, E., Doss, N., Skjellum, A., "A high-performance, portable implementation of the MPI Message-Passing Interface standard," *Parallel Computing*, V22, N6, pp. 789-828, 1996.

[4] Li, J., Liao, W. K., Choudhary, A., Ross, R., Thakur, R., Gropp, W., Latham, R., Siegel, A., Gallagher, B., Zingale, M., "Parallel netCDF: A high-performance scientific I/O interface," *The 2003 ACM/IEEE Conference on Supercomputing*, pp. 39, November 2003.

[5] Tsujita, Y., Imamura, T., Yamagishi, N., Takemiya, H., "Flexible message passing interface for a heterogeneous computing environment," Guo, M., Yang, L. T. editors, *New Horizons of Parallel and Distributed Computing*, Chapter 1, pp. 3-19, Springer, 2005.

[6] Tierney, B., "TCP tuning guide for distributed application on wide area networks," *Usenix; login*, pp. 33, February 2001.

[7] PVFS2, `http://www.pvfs.org/pvfs2/`

[8] dummynet, `http://info.iet.unipi.it/~luigi/ip_dummynet/`

[9] The National Center for Supercomputing Applications, `http://hdf.ncsa.uiuc.edu/HDF5/`

[10] Ross, R., Nurmi, D., Cheng, A., Zingale, M., "A case study in application I/O on Linux clusters," *The 2001 ACM/IEEE Conference on Supercomputing*, pp. 11, November 2001.

[11] Message Passing Interface Forum, *MPI-2: Extensions to the Message-Passing Interface*, July 1997.

[12] Thakur, R., Gropp, W., Lusk, E., "On implementing MPI-IO portably and with high performance," *The Sixth Workshop on Input/Output in Parallel and Distributed Systems*, pp 23-32, 1999.

[13] Thakur, R., Gropp, W., Lusk, E., "An abstract-device interface for implementing portable parallel-I/O interfaces," *The Sixth Symposium on the Frontiers of Massively Parallel Computation*, pp. 180-187, 1996.

[14] Lee, J., Ma, X., Ross, R., Thakur, R., Winslett, M., "RFS: Efficient and flexible remote file access for MPI-IO," *The 6th IEEE International Conference on Cluster Computing (CLUSTER 2004)*, pp. 71-81, September 2004.