Implementation of A Parallel NetCDF Interface for Seamless Collective Remote I/O

Yuichi Tsujita

Department of Electronic Engineering and Computer Science,
Faculty of Engineering,
Kinki University

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.
Outline

- Objective of implementation
- Overview of Stampi
- Overview of PnetCDF
- Support of a PnetCDF interface in Stampi’s remote I/O system
- Preliminary performance results
- Related work
- Summary
Objective of implementation

- Parallel computation
  - requires huge amounts of resources
  - Data intensive
- NetCDF
  - provides a common data format and a simple I/O interface.
- Parallel netCDF (PnetCDF)
  - A parallel I/O interface which supports a netCDF data format
  - An MPI-I/O interface is used as a parallel I/O layer.
  - Derived data types are used in its MPI-I/O operations.

Realization of remote I/O with a PnetCDF interface

Implementation of Stampi-I/O in PnetCDF functions

Support of derived data types for the PnetCDF functions
Overview of Stampi (1)

MPI libraries (both commercial and non-commercial)
- MPI communications and MPI-I/O operations inside a computer
- MPI communications, dynamic process creation, and MPI-I/O operations among different MPI libraries

Development of a seamless intermediate library

- **Stampi** (developed by CCSE, JAEA)
  - Intermediate library for MPI operations among different MPI libraries
  - Both intra-machine and inter-machine MPI communications
  - Dynamic process creation to a remote computer
  - MPI-I/O operations both inside a computer and to a remote computer
Overview of Stampi (2)

- Architecture of an MPI-I/O mechanism in Stampi

- Intra-machine MPI communications / MPI-I/O operations
  - High performance communications and I/O operations by using a native MPI library

- Inter-machine MPI communications / MPI-I/O operations
  - Communication by using TCP/IP
  - Flexible communication by using router processes
  - I/O operations on a remote computer by MPI-I/O processes

- Automatic selection of local/remote operations
1. Stampi’s start-up command initiates a Stampi starter.
2. The Stampi starter kicks off a native MPI starter and a router process if it required.
3. User processes are created, and they call MPI_File_open().
4. Another Stampi starter is invoked on a remote computer and it creates MPI-I/O processes and a router process if it is required.
5. A communication path is established among the user processes and the MPI-I/O processes.
Overview of PnetCDF (1)

- **netCDF and parallel netCDF**
  - **netCDF**
    - NetCDF supports a view of data as a collection of
      - self-describing,
      - portable, and
      - array-oriented objects that can be accessed through a simple interface
    - Its I/O operations do not provide parallel I/O mechanisms.

- **Parallel netCDF (PnetCDF)** (Li et al., SC2003 paper)
  - PnetCDF provides a parallel I/O interface which supports a netCDF data format in I/O operations.
  - An MPI-I/O interface is used as an underlying parallel I/O interface.
### Overview of PnetCDF (2)

#### Typical PnetCDF functions and used MPI functions

<table>
<thead>
<tr>
<th>PnetCDF function</th>
<th>Used MPI functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ncmpi_create(),</td>
<td>MPI_File_open(), MPI_File_delete(), etc.</td>
</tr>
<tr>
<td>ncmpi_open()</td>
<td></td>
</tr>
<tr>
<td>ncmpi_put_var_int()</td>
<td>MPI_Comm_rank(), MPI_File_set_view(), MPI_Type_hvector(),</td>
</tr>
<tr>
<td></td>
<td>MPI_Type_commit(), MPI_Type_free(), MPI_File_write(), etc.</td>
</tr>
<tr>
<td>ncmpi_put_vars_int_all()</td>
<td>MPI_Comm_rank(), MPI_File_set_view(), MPI_Type_hvector(),</td>
</tr>
<tr>
<td></td>
<td>MPI_Type_commit(), MPI_Type_free(), MPI_File_write_all(), etc.</td>
</tr>
<tr>
<td>ncmpi_close()</td>
<td>MPI_Allreduce(), MPI_File_close(), etc.</td>
</tr>
</tbody>
</table>

Because used MPI functions are supported in Stampi, it is possible to realize remote I/O operations through a PnetCDF interface with the help of the Stampi library.
Effective remote I/O with derived data types (1)

Execution steps in creation of a derived data type

1. Create a derived data type in each user process.
2. Store the data type in a list-based table of each user process.
3. Associated parameters such as a primitive data type and stride length are packed in a buffer.
4. The buffer is transferred to a corresponding MPI-I/O process.
5. Each MPI-I/O process also creates the same derived data type according to the transferred parameters.
6. The data type is also stored in a list-based table of each MPI-I/O process.
7. MPI-I/O processes return status value to corresponding user processes.
8. User processes receive them.
Effective remote I/O with derived data types (2)

Execution steps of MPI-I/O functions (MPI_File_write_all())

1. **Synchronization** among user processes by MPI_Barrier()
2. I/O requests and associated parameters are packed in a buffer using MPI_Pack() on each user process.
3. The buffer is transferred to a corresponding MPI-I/O process.
4. Bulk data is also transferred to the MPI-I/O process.
5. MPI-I/O processes carry out I/O operations using PVFS2 functions via ROMIO MPI-I/O API.
6. MPI-I/O processes return associated parameters to user processes.
7. User processes receive them.
Preliminary performance results (1)

Setup for measurement
- **OS:**
  - Cluster-I, Cluster-II: Fedora Core 3
  - MPI library: MPICH version 1.2.7p1
  - PVFS2 file system (version 1.3.2) was available on a server node of a PC cluster-II.
  - TCP_NODELAY was activated for inter-machine data transfer.

### System Specifications

<table>
<thead>
<tr>
<th></th>
<th>PC cluster-I</th>
<th>PC Cluster-II</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Server node</strong></td>
<td>DELL PowerEdge800 x 1</td>
<td>DELL PowerEdge1600SC x 1</td>
</tr>
<tr>
<td><strong>Computation nodes</strong></td>
<td>DELL PowerEdge800 x 4</td>
<td>DELL PowerEdge1600SC x 4</td>
</tr>
<tr>
<td><strong>CPU</strong></td>
<td>Intel Pentium-4 3.6 GHz x 1</td>
<td>Intel Xeon 2.4 GHz x 2</td>
</tr>
<tr>
<td><strong>Chipset</strong></td>
<td>Intel E7221</td>
<td>ServerWorks GC-SL</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>1 Gbyte DDR2 533 SDRAM</td>
<td>2 GByte DDR 266 SDRAM</td>
</tr>
<tr>
<td><strong>Disk system</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Server node</strong></td>
<td>80 GByte (Serial ATA) x 1</td>
<td>73 GByte (Ultra320 SCSI) x 1</td>
</tr>
<tr>
<td><strong>Computation nodes</strong></td>
<td></td>
<td>73 GByte (Ultra320 SCSI) x 2</td>
</tr>
<tr>
<td><strong>NIC</strong></td>
<td>Broadcom BCM5721 (on-board)</td>
<td>Intel PRO/1000-XT (PCI-X)</td>
</tr>
</tbody>
</table>
## Preliminary performance results (2)

**Program codes for performance measurement**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Example Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MPI_Info_set()</code></td>
<td>Set parameters in an info object</td>
<td><code>MPI_Info_set()</code></td>
</tr>
<tr>
<td><code>ncmpi_create()</code></td>
<td>Create a new file</td>
<td><code>ncmpi_create()</code></td>
</tr>
<tr>
<td><code>ncmpi_put_att_text()</code></td>
<td>Write a text in a header</td>
<td><code>ncmpi_put_att_text()</code></td>
</tr>
<tr>
<td><code>ncmpi_def_dim()</code></td>
<td>Define dimension</td>
<td><code>ncmpi_def_dim()</code></td>
</tr>
<tr>
<td><code>ncmpi_def_var()</code></td>
<td>Define variables</td>
<td><code>ncmpi_def_var()</code></td>
</tr>
<tr>
<td><code>ncmpi_end_def()</code></td>
<td>End of define mode</td>
<td><code>ncmpi_end_def()</code></td>
</tr>
<tr>
<td><code>ncmpi_begin_indep_data()</code></td>
<td>Beginning of independent I/O mode</td>
<td><code>ncmpi_begin_indep_data()</code></td>
</tr>
<tr>
<td><code>ncmpi_put_var_int()</code></td>
<td>Write data</td>
<td><code>ncmpi_put_var_int()</code></td>
</tr>
<tr>
<td><code>ncmpi_end_indep_data()</code></td>
<td>End of independent I/O mode</td>
<td><code>ncmpi_end_indep_data()</code></td>
</tr>
<tr>
<td><code>ncmpi_close()</code></td>
<td>Close a file</td>
<td><code>ncmpi_close()</code></td>
</tr>
</tbody>
</table>

### Write operations

- `MPI_Info_set()`, `ncmpi_create()`, `ncmpi_put_att_text()`, `ncmpi_def_dim()`, `ncmpi_def_var()`, `ncmpi_end_def()`, `ncmpi_begin_indep_data()`, `ncmpi_put_var_int()`, `ncmpi_end_indep_data()`, `ncmpi_close()`

### Read operations

- `MPI_Info_set()`, `ncmpi_open()`, `ncmpi_inq_varld()`, `ncmpi_inq_cimid()`, `ncmpi_inq_dimlen()`, `ncmpi_begin_indep_data()`, `ncmpi_get_var_int()`, `ncmpi_end_indep_data()`, `ncmpi_close()`
Preliminary performance results (3)

- Performance of remote I/O (PC Cluster I ↔ PC Cluster II)
  - Total I/O time
  - Data type: integer (NC_INT)

- With a huge amount of data, I/O times for collective functions in four user processes case are less than those for non-collective ones.

(a) Read operations

(b) Write operations
Preliminary performance results (4)

- Performance of remote I/O (PC Cluster I ↔ PC Cluster II)
  - Pure I/O time
  - Data type: integer (NC_INT)

Pure I/O times for collective operations are shorter than those for non-collective ones.

The times for collective ones became short with an increase in the number of user processes.

(a) Read operations

(b) Write operations
Related work

- **ROMIO**
  - MPI-I/O implementation in MPICH
  - ADIO: I/O interface to many kinds of I/O systems

- **RFS**
  - Remote I/O operations by an RFS request handler on a remote computer

- **HDF5**
  - Hierarchical data format
  - Two objects
    - “Dataset” for management of multi-dimensional data
    - “Group” for relational mechanisms among objects.
  - Parallel I/O with the help of ROMIO and PVFS I/O (R. Ross et al., SC2001 paper)
Summary

- Remote I/O operations through PnetCDF functions have been realized with the help of Stampi’s remote I/O mechanism.
- Through performance measurement, collective PnetCDF functions provided sufficient performance with a huge amount of data.
- Times to open a netCDF file were shorter than those for non-collective functions.
- Pure I/O times were minimized in collective functions with an increase in the number of user processes (the number of MPI-I/O processes were the same with the number.).
- It is considered that the PnetCDF interface provided sufficient performance in collective operations on two interconnected PC clusters.
- As a future work, implementation in real applications is planed to evaluate its performance.