On the Implementation of a Portable, Client-Server Based MPI-IO Interface *

Thomas Fuerle, Erich Schikuta, Christoph Loeffelhardt,
Kurt Stockinger, Helmut Wanek

Institute of Applied Computer Science and Information Systems
Department of Data Engineering, University of Vienna,
Rathausstr. 19/4, A-1010 Vienna, Austria
fuerle@vipios.pri.univie.ac.at

Abstract. In this paper we present the MPI-IO Interface kernel in the **Vi**enna **P**arallel **I**nput **O**utput **S**ystem (ViPIOS), which is a client-server based parallel I/O system. Compared to the already existing parallel I/O systems and libraries the concept of an independent distributed server promises to greatly enhance the usability and acceptance of the I/O system as well as the portability of client applications. The programmer of a client application does not have to deal with details like file layout on disk, sharing of filepointers etc. Instead high level MPI-IO requests may be issued and the server is expected to perform them in a (near) optimal way.
ViPIOS is based on MPI and is targeted (but not restricted) to MPP's using the SPMD paradigm. We describe the current system architecture in general and give a detailed overview of MPI-related design considerations.

Keywords

parallel I/O, server-client, SPMD, ViPIOS, MPI-IO, I/O chapter in MPI-2

1 Introduction

ViPIOS is an I/O runtime system, which provides efficient access to persistent files, by optimizing the data layout on the disks and allowing parallel read/write operations. ViPIOS is targeted as a supporting I/O module for high performance languages (e.g. HPF).

The basic idea to solve the I/O bottleneck in ViPIOS is *de-coupling*. The disk access operations are de-coupled from the application and performed by an independent I/O subsystem, ViPIOS. This leads to the situation that an application just sends disk requests to ViPIOS only, which performs the actual disk accesses in turn. This idea is caught by Figure 1.

* This work was carried out as part of the research project "Language, Compiler, and Advanced Data Structure Support for Parallel I/O Operations" supported by the Austrian Science Foundation (FWF Grant P11006-MAT)

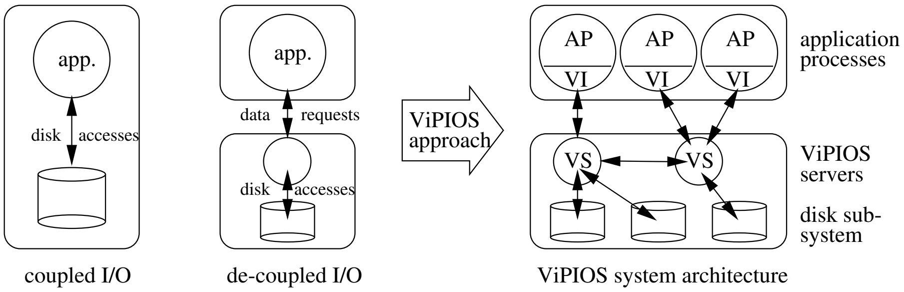

Fig. 1. Disk access decoupling - ViPIOS approach

The ViPIOS system architecture is built upon a set of cooperating server processes, which accomplish the requests of the application client processes. Each application process AP is linked by the ViPIOS interface VI to the ViPIOS servers VS.

The design of ViPIOS followed a data engineering approach characterized by two design principles,

- *Efficiency.* This is achieved by a *Two-Phase Data Administration* method, which aims to minimize the number of disk accesses by both compile time and runtime optimization. It provides a suitable data organization of the stored data on disk to the 'outside world' and organizes the data layout on disks respectively to the static application problem description and the dynamic runtime requirements.
- *Portability.* ViPIOS was designed to run on a large class of computer systems and to allow easy ports. Therefore the system is based on the MPI standard. All systems supported by MPI should provide a platform for ViPIOS.

1.1 The Two-Phase Data Administration method

The management of data by ViPIOS servers is split into two distinct phases, the preparation and the administration phase (see Figure 2).

The *preparation phase* precedes the execution of the application processes (mostly during the startup time). This phase uses the information collected during the application program compilation process in form of *hints* from the compiler. Based on this problem-specific knowledge the physical data layout schemes are defined, the actual ViPIOS server process for each application process and the disks for the stored data are chosen. Further, the data storage areas are prepared, the necessary main memory buffers allocated, and so on.

The following *administration phase* accomplishes the I/O requests of the application processes during their execution, i.e. the physical read/write operations, and performs necessary reorganization of the data layout.

The Two-Phase data administration method aims for putting all the data layout decisions, and data distribution operations into the preparation phase,

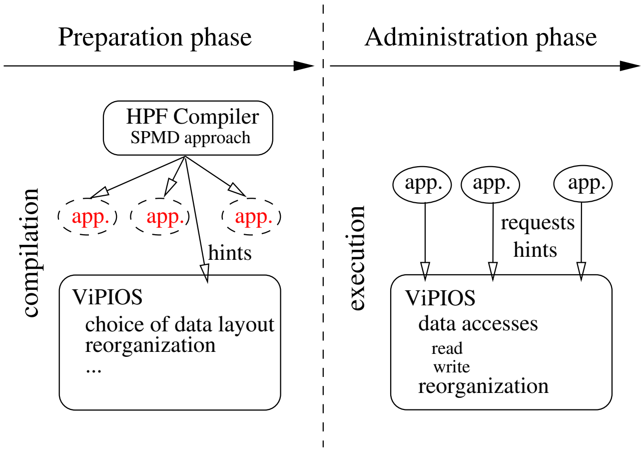

Fig. 2. The Two-phase data administration method

in advance to the actual application execution. Thus the administration phase performs the data accesses and possible data prefetching only.

1.2 System Modes

ViPIOS can be used in 3 different system modes, as

- runtime library,
- dependent system, or
- independent system.

Runtime Library. Application programs can be linked with a ViPIOS runtime module, which performs all disk I/O requests of the program. In this case ViPIOS is not running on independent servers, but as part of the application. The interface is therefore not only calling the requested data action, but also performing it itself. This mode provides only restricted functionality due to the missing independent I/O system. Parallelism can only be expressed by the application (i.e. the programmer).

Dependent System. In this case ViPIOS is running as an independent module in parallel to the application, but is started together with the application. This is inflicted by the MPI-1 specific characteristic that cooperating processes have to be started at the same time. This mode allows smart parallel data administration but objects the Two-Phase-Administration method by a missing preparation phase.

Independent System. In this case ViPIOS is running as a client-server system similar to a parallel file system or a database server waiting for application to connect via the ViPIOS interface. This is the mode of choice to achieve highest possible I/O bandwidth by exploiting all available data administration possibilities, because it is the only mode which supports the Two-phase data administration method.

Therefore we have to strive for an efficient implementation of the independent mode, in other words, ViPIOS has to execute as a client-server system.

2 Implementation Aspects of ViPIOS

2.1 Introduction

Unfortunately the client-server architecture described above can not be implemented directly on all platforms because of limitations in the underlying hard- or software (like no dedicated I/O nodes, no multitasking on processing nodes, no threading, etc.). So in order to support a wide range of different plattforms ViPIOS uses MPI for portability and offers multiple *operation modes* to cope with various restrictions.

2.2 Restrictions in Client-Server Computing with MPI

Independent Mode is not directly supported by MPI-1. MPI-1 restricts client-server computing by imposing that all the communicating processes have to be started at the same time. Thus it is not possible to have the server processes run independently and to start the clients at some later point in time. Also the number of clients can not be changed during execution

Clients and Servers share MPI_COMM_WORLD in MPI-1. With MPI-1 the global communicator MPI_COMM_WORLD is shared by all participating processes. Thus clients using this communicator for collective operations will also block the server processes. Furthermore client and server processes have to share the same range of process ranks. This makes it hard to guarantee that client processes get consecutive numbers starting with zero, especially if the number of client or server processes changes dynamically.

Simple solutions to this problem (like using separate communicators for clients and servers) are offered by some ViPIOS operation modes, but they all require, that an application program has to be specifically adapted in order to use ViPIOS.

Public MPI-Implementations (mpich, lam) are not MT-Safe. Both public implementations mpich and lam are not mt-save, thus non-blocking calls (e.g. MPI_Iread, MPI_Iwrite) are not possible without a workaround. Another drawback without threads is that the servers have to work with busy waits (MPI_Iprobe) to operate on multiple communicators.

2.3 Operation Modes of ViPIOS

ViPIOS can be compiled for the following different operation modes.

Runtime Library Mode behaves basically like ROMIO [7] or PMPIO [4], i.e. ViPIOS is linked as a runtime library to the application.

- *Advantage*
  - ready to run solution with any MPI-implementation (mpich, lam)
- *Disadvantage*
  - nonblocking calls are not supported. Optimization like redistributing in the background or prefetching is not supported
  - preparation phase is not possible, because ViPIOS is statically bound to the clients and started together with them
  - remote file access is not supported, because there is no server waiting to handle remote file access requests, i.e. in static mode the server functions are called directly and no messages are sent

Client Server Modes allow optimizations like file redistribution or prefetching and remote file accesses.

Dependent Mode. In Client-Server mode clients and server start at the same time using application schemes, see mpich.

- *Advantage*
  - ready to run solution (e.g with mpich)
- *Disadvantage*
  - preparation phase is not possible, because the ViPIOS servers must be started together with the clients

Independent Mode. In order to allow an efficient preparation phase the use of independently running servers is absolutely necessary.

This can be achieved by using one of the following strategies:

1. MPI-1 based implementations.
   Starting and stopping processes arbitrarily can be simulated with MPI-1 by using a number of "dummy" client processes which are actually idle and spawn the appropriate client process when needed. This simple workaround limits the number of available client processes to the number of "dummy" processes started.
   This workaround can't be used on systems which do not offer multitasking because the idle "dummy" process will lock a processor completely. Furthermore additional programming effort for waking up the dummy proccesses is needed.
   - Advantage
     - ready to run solution with any MPI-1 implementation

- Disadvantage
  - workaround for spawning the clients necessary, because clients cannot be started dynamically

2. MPI-2 based implementations.
   Supports the connection of independently started MPI-applications with ports. The servers offer a connection through a port, and client groups, which are started independently from the servers, try to establish a connection to the servers using this port. Up to now the servers can only work with one client group at the same time, thus the client groups requesting a connection to the servers are processed in a batch oriented way, i.e. every client group is automatically put into a queue, and as soon as the client group the servers are working with has terminated, it is disconnected from the servers and the servers work with the next client group waiting in the queue.
   - Advantages
     - ready to run solution with any MPI-2 implementation
     - No workaround needed, because client groups can be started dynamically and independently from the server group
     - Once the servers have been started, the user can start as many client applications as he wants without having to take care for the server group
     - No problems with MPI_COMM_WORLD. As the server processes and the client processes belong to two different groups of processes which are started independently, each group has implicitly a separated MPI_COMM_WORLD
   - Disadvantage
     - The current LAM version does not support multi-threading, which would offer the possibiliy of concurrent work on all client groups without busy waits
3. Third party protocol for communication between clients and servers (e.g. PVM).
   This mode behaves like MPI-IO/PIOFS [2] or MPI-IO for HPSS [5], but ViPIOS uses PVM and/or PVMPI (when it is available) for communication between clients and servers. Client-client and server-server communication is still done with MPI.
   - Advantage
     - ready to run solution with any MPI-implementation and PVM
     - Clients can be started easily out of the shell
     - no problems with MPI_COMM_WORLD, because there exist two distinct global communicators
   - Disadvantage
     - PVM and/or PVMPI is additionally needed

2.4 Sharing MPI_COMM_WORLD

So far, the independent mode using PVM(PI) or MPI-2 is the only ones which allows to use ViPIOS in a completely transparent way. For the other modes one of the following methods can be used to simplify or prevent necessary adaptations of applications.

1. Clients and servers share the global communicator MPI_COMM_WORLD.
   In this mode ViPIOS offers an intra-communicator MPI_COMM_APP for communication of client processes and uses another one (MPI_COMM_SERV) for server processes. This also solves the problem with ranking but the application programmer must use MPI_COMM_APP in every call instead of MPI_COMM_WORLD.
2. Clients can use MPI_COMM_WORLD exclusively.
   This can be achieved patching the underlying MPI-Implementation and also copes the problem with ranking.

A graphical comparison of this solutions is depicted in Figure 3.

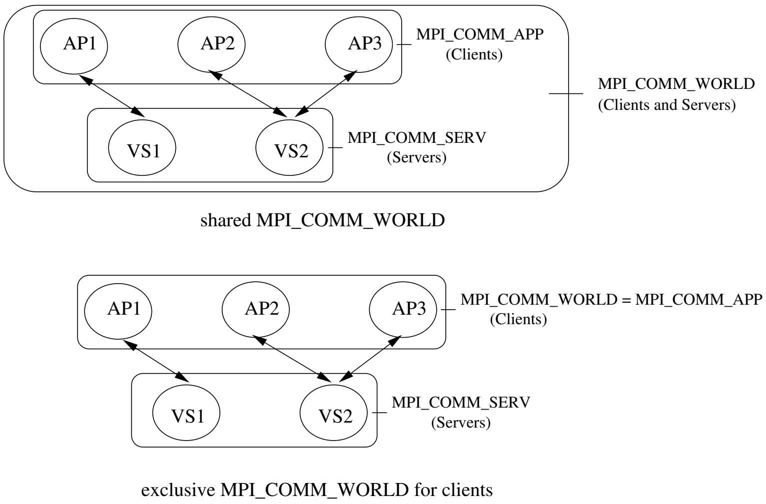

Fig. 3. shared MPI_COMM_WORLD versus exclusive MPI_COMM_WORLD

2.5 Implemented Modes

So far we have implemented the following modes

- runtime library mode with MPI-1 (mpich)
- dependent mode with MPI-1 with threads (mpich and patched mpich)
- independent mode with the usage of PVM and MPI-1 (mpich)
- independent mode with MPI-2 without threads (lam)

3 Conclusion and Future Work

This paper presents a practical application, which shows a deficiency of MPI. MPI-1 does not support the communication between independently started process groups. In the case of our ViPIOS client server architecture this is a strong drawback, which results in poor or non optimal performance. The introduction of ports in MPI-2 copes only to a limiting factor with this situation. Thus we showed some workarounds to handle this problem. Furthermore the public implementations of MPI (e.g. lam, mpich) do not support multithreading. We expect in the near future that new developments will provide us with similar capabilities in the MPI-standard as recognizable by the PVMPI [3] approach and that public implementations of MPI will be MT-Safe.

However in the near future the first running implementation of ViPIOS offering all presented operation modes will be finished. Currently we aim for the VFC compiler [1] (support for other HPF compilers is planned).

We also intend to add optimizations like prefetching and caching, and a port to a threaded (partly already implemented) version is on the way.

For preliminary performance results of ViPIOS refer to [6].

References

1. S. Benkner, K. Sanjari, V. Sipkova, and B. Velkov. Parallelizing irregular applications with the vienna hpf+ compiler vfc. In *Proceedings HPCN'98*. Springer Verlag, April 1998.
2. Peter F. Corbett, Dror G. Feitelson, Jean-Pierre Prost, George S. Almasi, Sandra Johnson Baylor, Anthony S. Bolmarcich, Yarsun Hsu, Julian Satran, Marc Snir, Robert Colao, Brian Herr, Joseph Kavaky, Thomas R. Morgan, and Anthony Zlotek. Parallel file systems for the IBM SP computers. *IBM Systems Journal*, 34(2):222–248, January 1995.
3. G. Fagg, J. Dongarra, and A. Geist. Heterogeneous mpi application interoperation and process management under pvmpi. Technical report, University of Tennessee Computer Science Department, June 1997.
4. Samuel A. Fineberg, Parkson Wong, Bill Nitzberg, and Chris Kuszmaul. PMPIO—a portable implementation of MPI-IO. In *Proceedings of the Sixth Symposium on the Frontiers of Massively Parallel Computation*, pages 188–195. IEEE Computer Society Press, October 1996.
5. Terry Jones, Richard Mark, Jeanne Martin, John May, Elsie Pierce, and Linda Stanberry. An MPI-IO interface to HPSS. In *Proceedings of the Fifth NASA Goddard conference on Mass Storage Systems*, pages I:37–50, September 1996.
6. E. Schikuta, T. Fuerle, C. Loeffelhardt, K. Stockinger, and H. Wanek. On the performance and scalability of client-server based disk i/o. Technical Report TR98201, Institute for Applied Computer Science and Information Systems, Juli 1998.
7. Rajeev Thakur, Ewing Lusk, and William Gropp. Users guide for ROMIO: A high-performance, portable MPI-IO implementation. Technical Report ANL/MCS-TM-234, Mathematics and Computer Science Division, Argonne National Laboratory, October 1997.