ViPIOS: The Vienna Parallel Input/Output System*

Erich Schikuta, Thomas Fuerle and Helmut Wanek

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

Abstract. In this paper we present the Vienna Parallel Input Output System (ViPIOS), a novel approach to enhance the I/O performance of high performance applications. It is a client-server based tool combining capabilities found in parallel I/O runtime libraries and parallel file systems.

1 Introduction

In the last few years the applications in high performance computing (Grand Challenges [1]) shifted from being CPU-bound to be I/O-bound. Performance can not be scaled up by increasing the number of CPUs any more, but by increasing the bandwidth of the I/O subsystem. This situation is commonly known as the I/O bottleneck in high performance computing ([5])

In reaction all leading hardware vendors of multiprocessor systems provided powerful concurrent I/O subsystems. In accordance researchers focused on the design of appropriate programming tools to take advantage of the available hardware resources.

1.1 The ViPIOS Approach

Conventionally two different directions in developing programming support are distinguished: Runtime libraries for high-performance languages (e.g. Passion [6]) and parallel file systems, (e.g. IBM Vesta [4]).

We see a solution to the parallel I/O problem in a combination of both approaches, which results in a dedicated, smart, concurrently executing runtime system, gathering all available information of the application process both during the compilation process and the runtime execution. Initially it can provide the optimal fitting data access profile for the application and may then react to the execution behavior dynamically, allowing to reach optimal performance by aiming for maximum I/O bandwidth.

* 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)

This approach led to the design and development of the Vienna Input Output System, ViPIOS ([2,3]).

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).

2 System Architecture

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 general I/O requests to ViPIOS, which performs the actual disk accesses in turn. This idea is caught by figure 1.

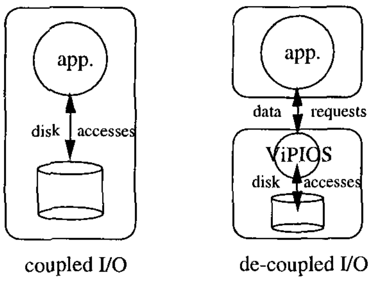

Fig. 1. Disk access de-coupling

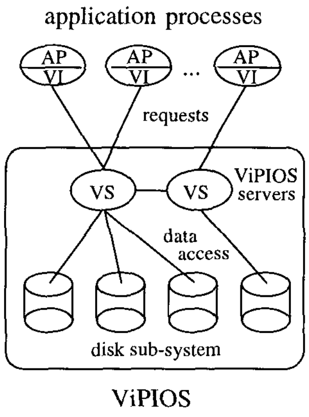

Fig. 2. ViPIOS system architecture

Thus ViPIOS's 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 (see figure 2).

The server processes run independently on all or a number of dedicated processing nodes on the underlying MPP. It is also possible that an application client and a server share the same processor.

Generally each application process is assigned exactly one ViPIOS server (which is called the *buddy server* to the application), but one ViPIOS server can serve a number of application processes, i.e. there exists a one-to-many relationship between the application and the servers (see figure 3). The other ViPIOS servers are called *foe server* to the application.

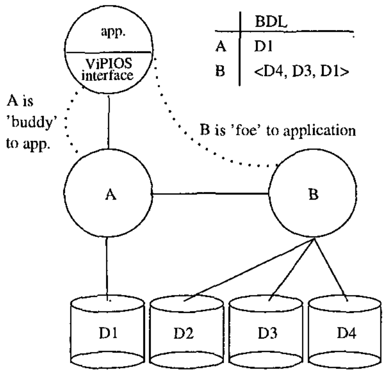

Fig. 3. "Buddy" and "Foe" Servers

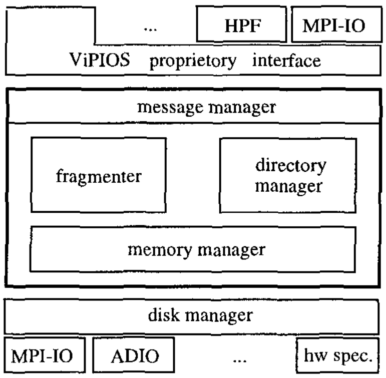

Fig. 4. ViPIOS server architecture

2.1 ViPIOS Server

A ViPIOS server process consists of several functional units as depicted by figure 4.

Basically we differentiate between 3 layers:

- The *Interface layer* provides the connection to the "outside world" (i.e. applications, programmers, compilers, etc.). Different interfaces are supported by *interface modules* to allow flexibility and extendibility. Until now we implemented an HPF interface module (aiming for the VFC, the HPF derivative of Vienna FORTRAN) a (basic) MPI-IO interface module, and the specific ViPIOS interface which is also the interface for the specialized modules.
- The *Kernel layer* is responsible for all server specific tasks.
- The *Disk Manager layer* provides the access to the available and supported disk sub-systems. This layer too is modularized to allow extensibility and to simplify the porting of the system. At the moment ADIO [7], MPI-IO, and Unix style file systems are supported.

The ViPIOS kernel layer is built up of four cooperating functional units:

- The *Message manager* is responsible for the external (to the applications) and internal (to other ViPIOS servers) communication.
- The *Fragmenter* can be seen as "ViPIOS's brain". It represents a smart data administration tool, which models different distribution strategies and makes decisions on the effective data layout, administration, and ViPIOS actions.
- The *Directory Manager* stores the meta information of the data. We designed 3 different modes of operation, centralized (one dedicated ViPIOS directory server), replicated (all servers store the whole directory information), and localized (each server knows the directory information of the data it is storing only) management. Until now only localized management is implemented.
- The *Memory Manager* is responsible for prefetching, caching and buffer management.

Requests are issued by an application via a call to one of the functions of the ViPIOS interface, which in turn translates this call into a request message which is sent to the buddy server.

The *local directory* of the buddy server holds all the information necessary to map a client's request to the physical files on the disks. The *fragmenter* uses this information to decompose (*fragment*) a request into sub-requests which can be resolved locally and sub-requests which have to be communicated to other ViPIOS-servers (foe servers). The *I/O subsystem* actually performs the necessary disk accesses and the transmission of data to/from the AP.

2.2 System Modes

ViPIOS can be used in 3 different system modes, as

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

These modes are depicted by figure 5.

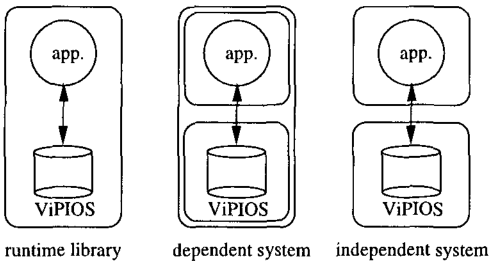

Fig. 5. ViPIOS system modes

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 ViPIOS 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

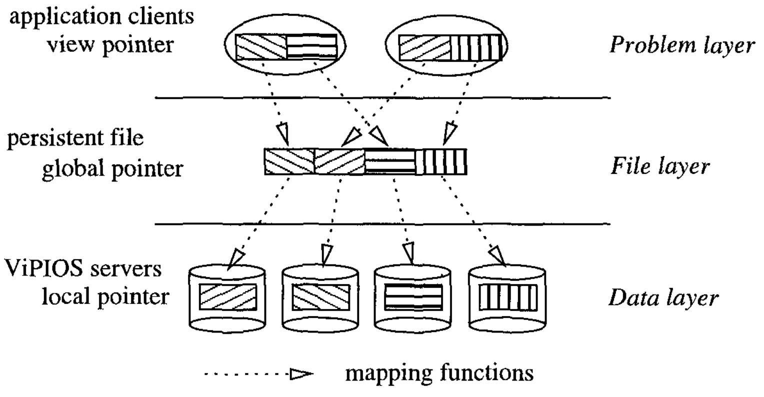

Fig. 6. ViPIOS data abstraction

is inflicted by the MPI[1] specific characteristic that cooperating processes have to be started together in the same communication world. Processes of different worlds can not communicate until now. This mode allows smart parallel data administration but objects a preceeding preparation phase.

Independent System. This is the mode of choice to achieve highest possible I/O bandwidth by exploiting all available data administration possibilities. In this case ViPIOS is running similar to a parallel file system or a database server waiting for applications to connect via the ViPIOS interface. This connection is realized by a proprietary communication layer bypassing MPI. We implemented two different approaches, one by using PVM, the other by patching MPI. A third promising approach is just evaluated by employing PVMPI, a possibly uprising standard under development for coupling MPI worlds by PVM layers.

3 Data Abstraction in ViPIOS

ViPIOS provides a data independent view of the stored data to the application processes.

Three independent layers in the ViPIOS architecture can be distinguished, which are represented by file pointer types in ViPIOS.

- Problem layer. Defines the problem specific data distribution among the cooperating parallel processes (View file pointer).
- File layer. Provides a composed view of the persistently stored data in the system (Global file pointer).
- Data layer. Defines the physical data distribution among the available disks (Local file pointer).

[1] The MPI standard is the underlying message passing tool of ViPIOS to ensure portability

Thus data independence in ViPIOS separates these layers conceptually from each other, providing mapping functions between these layers. This allows *logical data independence* between the problem and the file layer, and *physical data independence* between the file and data layer.

This concept is depicted in figure 6 showing a cyclic data distribution.

4 Conclusions and Future Work

In this paper we presented the Vienna Parallel Input Output System (ViPIOS), a novel approach to parallel I/O based on a client server concept which combines the advantages of existing parallel file systems and parallel I/O libraries. We described the underlying design principles of our approach and gave an in-depth presentation of the developed system.

References

1. A Report by the Committee on Physical, Math., and Eng. Sciences Federal Coordinating Council for Science, Eng. and Technology. *High-Performance Computing and Communications, Grand Challenges 1993 Report*, pages 41 – 64. Committee on Physical, Math., and Eng. Sciences Federal Coordinating Council for Science, Eng. and Technology, Washington D.C., October 1993.
2. Peter Brezany, Thomas A. Mueck, and Erich Schikuta. Language, compiler and parallel database support for I/O intensive applications. In *Proceedings of the International Conference on High Performance Computing and Networking*, volume 919 of *Lecture Notes in Computer Science*, pages 14–20, Milan, Italy, May 1995. Springer-Verlag. also available as Technical Report of the Inst. f. Software Technology and Parallel Systems, University of Vienna, TR95-8, 1995.
3. Peter Brezany, Thomas A. Mueck, and Erich Schikuta. A software architecture for massively parallel input-output. In *Third International Workshop PARA'96 (Applied Parallel Computing - Industrial Computation and Optimization)*, volume 1186 of *Lecture Notes in Computer Science*, pages 85–96, Lyngby, Denmark, August 1996. Springer-Verlag. Also available as Technical Report of the Inst. f. Angewandte Informatik u. Informationssysteme, University of Vienna, TR 96202.
4. Peter F. Corbett and Dror G. Feitelson. The Vesta parallel file system. *ACM Transactions on Computer Systems*, 14(3):225–264, August 1996.
5. Juan Miguel del Rosario and Alok Choudhary. High performance I/O for parallel computers: Problems and prospects. *IEEE Computer*, 27(3):59–68, March 1994.
6. Rajeev Thakur, Alok Choudhary, Rajesh Bordawekar, Sachin More, and Sivaramakrishna Kuditipudi. Passion: Optimized I/O for parallel applications. *IEEE Computer*, 29(6):70–78, June 1996.
7. Rajeev Thakur, William Gropp, and Ewing Lusk. An abstract-device interface for implementing portable parallel-I/O interfaces. In *Proceedings of the Sixth Symposium on the Frontiers of Massively Parallel Computation*, pages 180–187, October 1996.