Presented and Published as part of the HPCN '96, Brussels, Belgium.

"Vipar" Libraries to Support Distribution and Processing of Visualization Datasets

Steve Larkin, Andrew J Grant, W T Hewitt

Computer Graphics Unit, Manchester Computing
University of Manchester, Manchester M13 9PL, UK.
Tel: +44 161 275 6095, Fax: +44 161 275 6040
Email: s.larkin@mcc.ac.uk, a.j.grant@mcc.ac.uk, w.t.hewitt@mcc.ac.uk

Abstract

The aims of the "Visualization in Parallel" (Vipar) project is to produce a comprehensive environment for the development of parallel visualization modules in systems such as the Application Visualization System (AVS), Iris Explorer, IBM Data Explorer (DX) and Khoros. This paper presents an overview of the project and describes the libraries developed to support the first phase of the work which is a tool to describe parallel visualization modules. This work is funded as part of the EPSRC project (GR/K40390) Portable Software Tools for Parallel Architectures (PSTPA).

1 Introduction

This paper first describes the aims of the Visualization in Parallel (Vipar) project and, as background material, an overview of the problems associated with producing parallel visualization systems. Section 2 describes the Vipar system architecture and related components which include the support libraries. The terms and concepts used in the support libraries are covered in section 3 with more specific detail on the routines and an example application being covered in section 4 and 5. The paper finishes with some conclusions and an outline of the future work in section 6.

1.1 Aims of the Project

Current visualization systems such as the Application Visualization System (AVS) [1], Iris Explorer [2], IBM Data Explorer (DX) [3] and Khoros [4] allow users to construct visualization applications by connecting a number of *modules* together into a *network* or *map*.

One of the key features of these systems is the ability of users to integrate their own application code into the system to perform tasks that are not supported within the standard package or to embed and tightly couple a simulation code or interface to a data acquisition device.

As these systems have become more widely used for a variety of problems researchers have moved towards parallel solutions when building applications using these systems. The main reasons being:

- bottlenecks created by highly computational modules in an application
- large datasets which cannot easily fit into the real memory of a single compute node.

Many of the parallel solutions which have been developed to tackle the above problems have the disadvantage of being specific to the hardware and parallel support libraries being used and are application dependent. This has the effect of either rendering the code unusable for other applications, or too time consuming to change.

The aim of the Vipar project is to provide a software environment to support users who are developing parallel modules for use within applications constructed using current visualization systems. The tools will support the generally used schemes for implementing these modules and will insulate users from the underlying hardware and support libraries. The phases of the project include the development of:

- an automatic parallel module generator
- a network editor for building and managing parallel visualization applications

The environment will be portable between networks of workstations and MPP systems and will be made available for both virtual shared memory and distributed memory parallel machines.

1.2 Background

There has been some work carried out by various research groups to exploit potential parallelism in visualization systems. This can be catergorised into three classes [5], [6].

A. Functional/Task: a number of modules in the system can be executed concurrently on separate processors within machines (see figure 1A). Most of the current application builders provide a facility to support the execution of modules on other remote heterogenous machines with the visualization systems handling the communication and data transfer. This is a coarse grain solution as each individual module is still executed sequentially.

B. Parallel Modules: this approach targets the most computationally expensive modules in an application and parallelises/vectorises them for specific platforms (see figure 1B). There are many examples of work in this class [7], [8], [9], [10], [11]. The problem with this approach is that the data distribution and resulting composition of results carries an overhead and can sometimes outweigh the performance speedup gained.

C. Parallel Systems: The application is constructed in a visualization system which has been developed to support and handle parallel modules. Interaction between parallel modules is managed by the system and communication between individual modules is performed through the sharing of data or parallel communication (see figure 1C). There are a number of issues relating to the development of parallel systems and some of these are summarised below. A more detailed discussion can be found in [12].

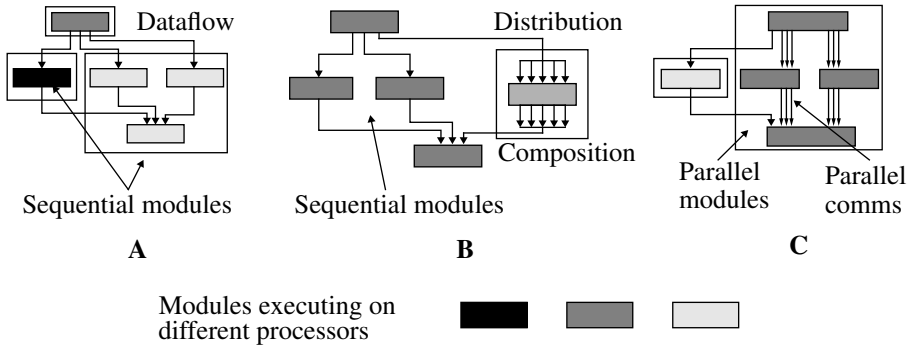

Fig. 1. Exploiting potential parallelism

Issues relating to Parallel Systems

Data Decomposition: there is a requirement to reduce or eliminate the unnecessary data composition and redistribution between modules and to take advantage of parallel communications or the sharing of data.

Synchronisation: The dataflow paradigm on which the visualization systems are based restricts modules in the network to only start processing when the complete dataset is available from an earlier stage. Typically in most datasets there are regions where less processing is required and in a parallel visualization application these portions could be passed onto later stages in the network. This can be counteracted if the processed portion requires data from its neighbouring portions before the next stage can be started. If a number of time steps or stages of a simulation are being processed then the system needs to ensure it has a method of correctly grouping the portions when they reach the final stage.

Mapping: the mapping of processes to physical machines (in a Network of Workstations) or a group of processes within a single environment (MPP) needs to addressed. An important factor in this decision is the provision of feedback on performance monitoring to allow both the user and system to perform load-balancing.

There has been some earlier work on developing complete parallel visualization systems. The PISTON system [13] was designed to aid the development of data parallel image processing and visualization algorithms. The NEVIS project [12], [14] investigated the use of modular visualization systems in a distributed memory MIMD environment.

The support libraries described in this paper have been designed to provide the mechanism which allows the tools built on top of these libraries to address the above issues.

2 System Architecture

There are a number of projects underway in the Computer Graphics Unit with which the Vipar project is intended to collaborate as they have similar needs for the development of some libraries. These are the Wanda (Wide Area Networks Distributed Applications) project [15] which is investigating the issues related to applications over wide area networks and the PVV (Parallel Volume Visualizer) project [16], [17] which is producing a programming environ-

ment for writing parallel volume visualization algorithms. Figure 2 shows the relationship between different systems and the Vipar tools and support libraries.

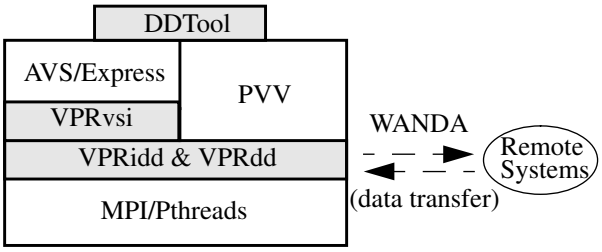

Fig. 2. System architecture

DDTool: An automatic parallel module generator. DDTool [19] allows the user to describe the data decomposition strategy and other criteria when constructing a parallel module and produces a skeleton module structure.

VPRvsi: The Visualization System Interface library [20] provides an interface to the other support libraries by handling the data in the form the native visualization system uses and mapping it onto the structures used by the independent and general support libraries (`VPRidd` and `VPRdd`).

VPRidd: a library of routines to calculate data distribution patterns and other useful utilities. These functions are characterised as being independent of the underlying system.

VPRdd: a library of routines to distribute, manage and composite data. It is used to implement the mechanisms made available in the `VPRvsi`. This library is system dependent.

The `VPRidd` and `VPRdd` routines do not perform any intelligent partitioning or processing but rather provide the mechanism for an application or tool, such as DDTool, to define these groupings and have the actions carried out across different platforms.

To support both distributed and shared memory environments the libraries will be implemented using MPI [26], [27] and Pthreads [31]. Some preliminary work experimented with the use of PVM (Parallel Virtual Machine) [22] and the authors were aware of other portable message passing libraries [23], [24], [25] but it was decided to adopt MPI for the following reasons:

- To ensure the code is future proof and portable;
- The mechanisms provided for building safe parallel libraries in MPI;
- Derived Data Types in MPI for extracting data directly from arrays;
- Problems of packing/unpacking buffers for sending data;

2.1 A Prototype of DDTool

The prototype version of DDTool is being developed for the AVS/Express programming environment [18] and AVS6 visualization system [21]. AVS/Express is designed for developers who are creating technical applications for distribution to customers and AVS6 is the next release of AVS, replacing AVS5. Both AVS6 and AVS/Express share the same common architecture.

Object Manager

The dataflow paradigm used by many visualization systems is restrictive as it means generating multiple copies of the data as it passes through the visual data analysis pipeline. This overhead is greatly increased when large datasets are being processed. AVS/Express and AVS6 have moved away from the dataflow paradigm to the idea of an object manager with modules using object references as handles to datasets. Work is underway to develop a distributed version of AVS/Express (called MP/Express) [28] which will involve the implementation of a Distributed Object Manager (DOM) to handle the management and distribution of AVS/Express data.

3 Support libraries terms and concepts

There are a number of terms and concepts used by both the independent and general data distribution libraries. These are explained in the following sections.

3.1 Distribution Map

This structure contains the distribution scheme for a particular dataset over a number of processes. Distribution schemes in other systems were examined [3], [29], [30] and the following set is used to specify the distribution of each dimension in a dataset:

Preserve: this is not subdivided;

Block: subdivide into equal blocks among the processes;

Cyclic: subdivide using a cyclic distribution;

Application: a user/application defined subdivision.

A combination of these distribution schemes can be used to define common methods of distributing data for visualization tasks, see figure 3.

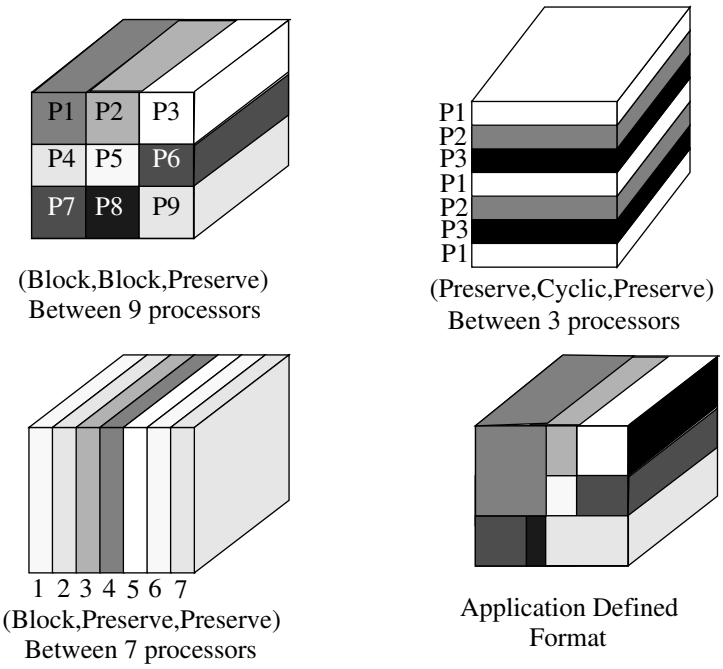

Fig. 3. Combining different distributions

This map is used in combination with other data structures which provide more specific implementation information for distributing/compositing and processing the data portions.

Neighbourhood and boundary processing

For some applications groups of worker processes will require data stored in neighbouring portions of the dataset. If a data portion is on the boundary of the complete dataset then accesses outside the boundary needs handling.

To specify this the distribution map has the following extra information:

Neighbourhood: whether the region should be grown or shrunk and the extent in each dimension for this operation.

Boundary: if an access goes outside the boundary the choice of actions are: no expansion, expand with a value, use nearest boundary value or assume the dataset is cyclic.

3.2 Data Source and Sink

The data source/sink is a reference for the process working on a particular portion of data.

Data Source: location or copy of the data portion to process;

Data Sink: destination for the processed portion or resulting data;

There are two scenarios for the data source/sink:

1. *Process identifier:* process which sends/receives the data portion. It allows multiple number of processes to act as data source/sinks and they can be processes associated with other parallel modules.
2. *Object identifier:* reference to an object which contains the particular data portion. An object can be a shared memory segment or one which is handled by the DOM.

Both the master-slave and SPMD paradigms can be supported as the data source/sink maps allow data to be distributed by multiple processes, passed on by other worker processes or handled by the DOM or something similar (see figure 4).

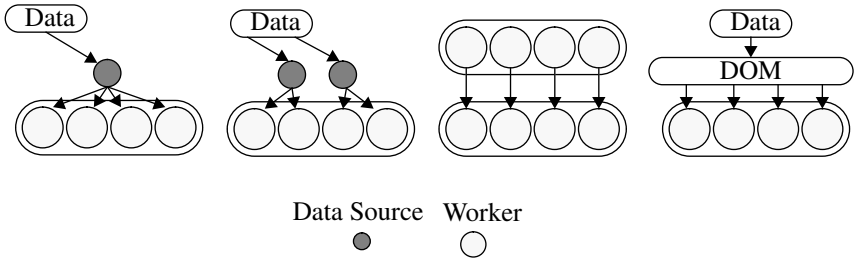

Fig. 4. Different data source and worker patterns

3.3 Implementation Specific Maps

These structures add extra implementation information to the data distribution and data source/sink maps. The ones we will discuss relate to the MPI implementation.

Neighbourhood Map

This data structure is generated and used if a group of processes working on a datatset need to update neighbourhood information.

In the case of the MPI implementation this is an MPI communicator with an associated cartesian grid. This allows the implementation to make use of the utility functions supplied with MPI for handling cartesian grids when requesting neighbourhood data. It also separates the worker processes from the data sources, if any are present.

When the cartesian grid is generated the flag to permit an MPI implementation to reorder the process ranks is enabled. This allows the implementation to choose the best distribution which will reflect the underlying physical hardware and process connection.

Process Map

This is used specifically by the MPI implementation to augment the distribution map with which process ranks will process a particular data portion. The process map is used to locate and place the data portions but is not intended for neighbourhood data processing.

The main reason for its inclusion is due to the fact that we cannot predict prior to the neighbourhood map formation how the process ranks will be reordered. Also if we are using any data source/sink processes they will not be part of this new communicator (neighbourhood map) and will need this information for distributing/collecting data.

Derived Datatype Map

When data portions are being distributed from or gathered into a larger array the MPI implementation can make use of the derived data type facility to directly extract or insert the data and thus avoid the need of a temporary storage to send/receive the data.

The generation of the derived datatypes requires a type to be created and then committed. Instead of the data source/sinks continually performing this action and then releasing the datatype a derived datatype map can optionally be generated.

4 VPRidd and VPRdd Routines

4.1 Main routines

VPRidd_CalcDist: calculates the data distribution for a dataset across a number of processes. The function returns this information in the form of a distribution map.

VPRdd_FormNbr: creates a communicator which just contains the pool of processes for working on the data partitions. The MPI implementation is allowed to reorder the processes to reflect the underlying topology. This function must be invoked by all processes in the original communicator but any processes which are allocated as data source/sinks are split from the new communicator.

A collective operation is used to generate a process map for any data source/sink processes involved.

VPRdd_SwapNbr: If the worker processes need to update neighbourhood information then they all call the collective routine to swap the data between processes.

VPRdd_DistReg, VPRdd_CollReg: used by data source/sink processes to distribute/collect data portions.

VPRdd_RecvReg, VPRdd_SendReg: used by worker processes to receive and send data portions. These can be from any type of data source or sink.

4.2 Other routines

There are a number of utilities routines which are used by the main routines in section 4.1 to pass portions of arrays between MPI processes. Some of these routines also handle growing and shrinking neighbourhood regions and boundary processing.

5 Conclusions and future work

The first phase of the project has been addressing the need to provide tools which aid the generation of parallel visualization modules.

The second phase of the project will handle inter module parallelism addressing the issues related to implementing a parallel visualization system. The tools developed during this phase will manage the parallel modules providing facilities to control the placement and characteristics of the modules. An important part during this phase is providing useful performance feedback to aid the users decisions.

6 Acknowledgments

The authors of the paper would first like to thank EPSRC for funding the project under the PSTPA initiative. They would also like to acknowledge Gary Oberbrunner, Advanced Visual Systems Inc., for the ideas and comments he has input to the project. We are also grateful for the support from our industrial collaborators AVS/UNIRAS Ltd. and Meiko Ltd. Finally thanks to our colleagues in the Computer Graphics Unit, Manchester Computing and LSI, University of São Pãulo.

7 References

[1] Upson C et al, "The Application Visualization System: A Computational Environment for Scientific Visualization", IEEE Computer Graphics and Applications, 9(4), pp 30 -42, 1989.

[2] "IRIS Explorer - Technical Report", Silicon Graphics Computer Systems.

[3] Lucas B, Abram G D, Collins N S, Epstein D A, Gresh D L, McAuliffe K P, "An Architecture for a Scientific Visualization System", Proceedings of Visualization '92, IEEE Computer Society Press, 1992.

[4] Rasure J, Young M, "An Open Environment for Image Processing Software Development", SPIE/IS&T Symposium on Electronic Imaging Proceedings, Vol. 1659, February 1992.

[5] Whitman S, "Survey of Parallel Approaches to Scientific Visualization", Computer Aided Design, Volume 26, Number 12, pages 928-935, December 1994

[6] Grant A J, "Parallel Visualization", Presented at EASE Visualization Community Club seminar on Parallel Processing for Visualization, University of Manchester, November 1993.

[7] Woys K, Roth M, "AVS Optimisation for Cray Y-MP Vector Processing", Proceedings of AVS '95, pages 145-162, Boston MA, US, 1995.

[8] Ford A, Grant A J, "Adaptive Volume Rendering on the Meiko Computing Surface", Parallel Computing and Transputer Applications Conference, Barcelona, 1992

[9] Cheng G, Fox G C, Mills K, Marek Podgorny, "Developing Interactive PVM-based Parallel Programs on Distributed Computing Systems within AVS Framework", Proceedings of AVS '93, pages 171-179, 1993.

[10] Chen P C, "Climate Simulation Case Study III: Supercomputing and Data Visualization", Proceedings of AVS '95, pages 373-384, Boston US, 1995.

[11] Krogh M, Hansen C D, "Visualization on Massively Parallel Computers Using CM/AVS", Proceedings of AVS '93, Orlando, USA, 1993.

[12] Thornborrow C, Wilson A J S, Faigle C, "Developing Modular Application Builders to Exploit MIMD Parallel Resources", Proceedings of Vis '93, pages 134-139, IEEE Computer Society Press, 1994.

[13] Tsui K K, Fletcher P A, Hutchins M A, "PISTON: A Scalable Software Platform for Implementing Parallel Visualization Algorithms", CGI '94, Melbourne, Australia, 1994.

[14] Thornborrow C, "Utilising MIMD Parallelism in Modular Visualization Environments", Proceedings of Eurographics UK '92, Edinburgh, March 1992.

[15] Lever P, Grant A J, Hewitt W T, "WANDA: Wide Area Network Distributed Applications", *In Preparation*, 1995.

[16] Zuffo M K, Grant A J, "RTV: A system for the visualization of 3D medical data", SIBGRAPH '93, Pernambuco, Brazil 1993.

[17] Zuffo M K, Grant A J, Santos E T, Lopes R de D, Zuffo J A, "A Programming Environment for High Performance Volume Visualisation Algorithms", *In Preparation*, 1995

[18] Vroom J, "AVS/Express: A New Visual Programming Paradigm", Proceedings of AVS 95, pages 65-94, Boston MA, 1995.

[19] Larkin S, Grant A J, Hewitt W T, "A Data Decompositon Tool for Writing Parallel Modules in Visualization Systems", *In Preparation*, 1996.

[20] Larkin S, Grant A J, Hewitt W T, "A Generic Structure for Parallel Modules in Visualization Systems", *In Preparation*, 1996.

[21] Lord H, "AVS/Express Product Family Overview", Proceedings of AVS 95, pages 3-13, Boston MA, 1995.

[22] Begulin A, Dongarra J, Geist A, Manchek R, Sunderam V, "Users Guide to PVM: Parallel Virtual Machine", ORNL Report TM-11826, July 1991.

[23] Butler R, Lusk E, "Users Guide to the p4 parallel programming system", Technical Report ANL-92/17, Argonne National Laboratory, October 1992.

[24] Gropp W D, Smith B, "Chameleon parallel programming tools users manual.", Technical Report ANL-92/93, Argonne National Laboratory, March 1993.

[25] Geist A, Heath M T, Peyton B W, Worley P H, "Users Guide for PICL: A Portable Instrumented Communications Library", Technical Report ORNL/TM-11616, Oak Ridge National Laboratory, Oak Ridge, TN, October 1990.

[26] Message Passing Interface Forum, "MPI: A message-passing interface", Computer Science Department Technical report No. CS-94-230, University of Tennessee, Knoxville, TN, April 1994 (Also in International Journal of Supercomputer Applications, Volume 8, Number 3/4, 1994).

[27] Gropp W, Lusk E, Skjellum A, "Using MPI: Portable Parallel Programming with the Message-Passing Interface", MIT Press, 0-262-57104-8, 1995.

[28] Oberbrunner G, "MP/Express Preliminary Specification", Internal Technical Report, Advanced Visual Systems

Inc. December 1994.

[29] Chapple S, "Parallel Utilities Libraries-RD Users Guide", Edinburgh Parallel Computing Centre (EPCC), UK, Technical Report, 1992.

[30] "HPF: language definition document", published in Scientific Programming, Vol. 2, no. 1-2, pp. 1-170, John Wiley and Sons.

[31] "Pthreads: POSIX threads standard", IEEE Standard 1003.1c-1995.