VIPAR: A Case Study for UKHEC - January 2001

J. Leng, K. Roy, J. M. Brooke

MRCCS
Manchester Computing
University of Manchester

Introduction

VIPAR was originally a 2 year EPSRC research project, Research Grant Reference Number: GR/K40390. The project was started in April 1996 and concluded in December 1997 and was carried out at the Manchester Visualization Centre, University of Manchester. The funding came as part of the Portable Software Tools for Parallel Architectures (PSTPA) initiative. This project also had industrial partners AVS Inc. and Meiko Ltd.

Visualization systems like AVS/Express, Iris Explorer, IBM Data Explorer and Khoros are described as Application Builders or Modular Visualization Environments (MVE). These tools provide a highly developed GUI which is easy for new or non-technical personnel to operate. The user drags and drops modules from the systems libraries and connects them to produce a network which in turn produces an associated application and visualization. These visualization tools are extendable and experienced users can add their own coded modules to the libraries.

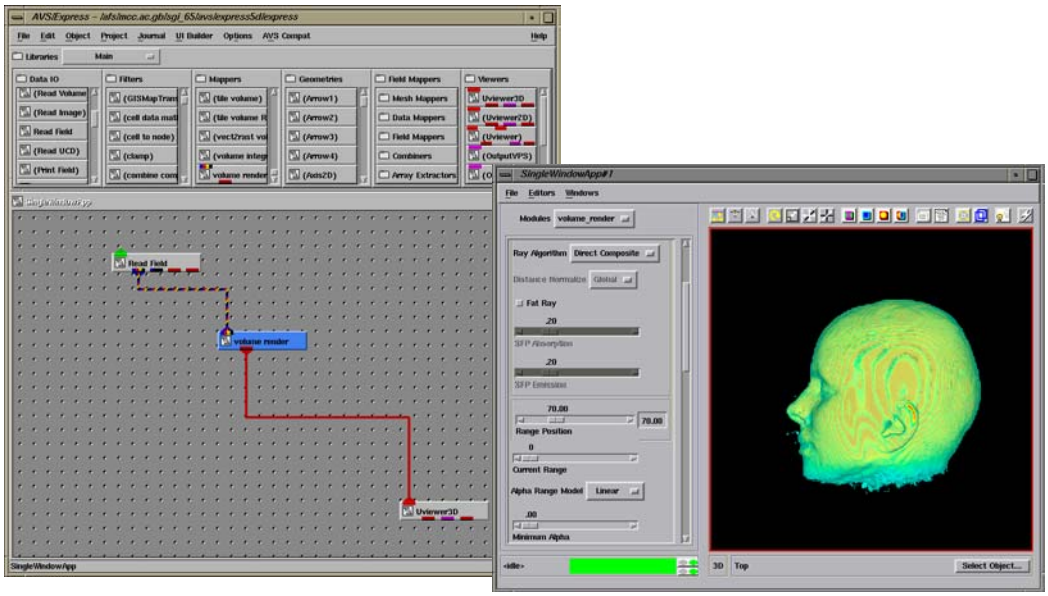

Fig 1: AVS/Express showing it's main library and a network. It's associated application and visualization are to the right.

There were two main reasons why researchers needed parallel solutions to tackle performance bottlenecks within a visualization pipelines. Firstly because some highly computational modules caused bottlenecks. Secondly because large data sets could not fit into the real memory of a single compute node.

At that time many of the parallel solutions that had been developed to deal with these problems had the disadvantage of being specific to the hardware and/or the parallel support libraries used

were application dependent. The effect was that the code was unusable for other applications, or too time consuming to change. An overview of the solutions that had been developed at this time can be obtained from two references (2,3).

VIPAR had two particular goals which it aimed to implement without being limited to one visualization system or one parallel support library:

- An automatic parallel module generator,
- A dataflow network editor for building parallel visualization modules

The VIPAR environment was intended to be portable between networks of workstations and MPP systems and to utilise both virtual shared memory and distributed memory parallel machines. Techniques such as tagged data paths were used to minimise de-/re-composition of data and to assist global parallelisation.

This was a research project aimed at proof of concept. The deliverables of the project were for the development of original research ideas and proof of their solution by the production of prototype tools and libraries.

Current Interest In VIPAR

When the original VIPAR project was completed the results of the project were not taken up by a user base although the code has been available through the MVC web pages. Since this time there have been several changes, the one that has probably had the greatest impact is the first one in the list below:

- Development of hardware rendering systems has meant that the major visualization bottlenecks have moved away from the rendering components into other parts of the visualization pipeline. It makes sense to parallelize just the bottlenecks, i.e. the CPU intensive components.
- Now there are more and larger MPP systems allowing users to increase the size of data sets.
- More application areas are using visualization.
- Computational steering/monitoring is becoming more important and VIPAR could aid this.

Recently a number of interested parties have approached Manchester Visualization Centre and asked for VIPAR or a system with similar functionality. One of these groups which is based at Queen Mary and Westfield in the Physical Chemistry Department is a partner in a large CSAR consortium. At their request the VIPAR project was re-opened.

Specifically their requirements are:

- VIPAR to run with the new/current versions of software on their current platform, a SGI Onyx.
- A parallel isosurface and FFT module.
- The group’s programming skills lie with MPI. They want to use the module generation tool to produce MPI coded modules, although the template code produced by this tool could be for other parallel libraries.
- In the long term they also wish to exploit the ability of VIPAR to run in a heterogeneous environment, the ability to link directly to existing code, monitor its progress and for a user feedback into the program based on existing results.

The Structure of VIPAR and Its Libraries

The VIPAR libraries are a suite of routines that provide an interface between the visualization system and the message passing system. The three libraries are, a visualization library to provide an interface to the visualization system (VPRvsi), an intermediary library that performs the necessary calculations and provides the interface between the two other libraries (VPRidd), and the lower level message passing and parallel process control library (VPRdd). These libraries are designed to sit on top of one another and provide increasing levels of complexity.

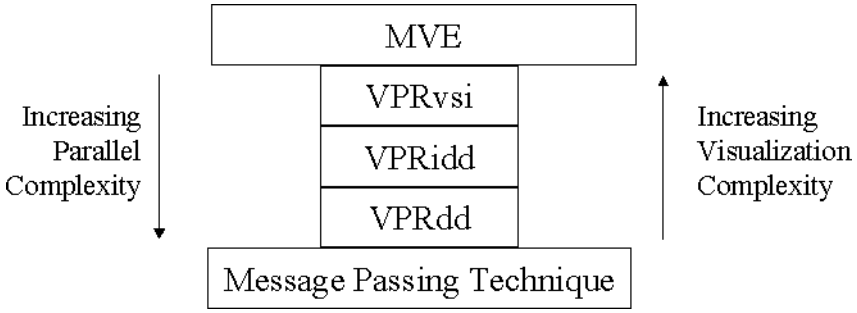

Fig 2: Shows how the VIPAR libraries are layered to achieve portability between MPI libraries and the visualization system.

The functionality of the libraries is as follows:

VPRvsi: The Visualization System Interface library provides an interface to the other support libraries by handling the data in the form that the native MVE uses and mapping it onto the structures used by the independent and general support libraries (VPRidd and VPRdd). A specific version of the libraries must be written for each visualization system which VIPAR supports.

VPRidd: routines to calculate data distribution patterns and other useful utilities. These functions are characterised by being independent of the underlying parallel architecture. A specific version of the libraries must be written for each parallel support library.

VPRdd: routines to distribute, manage and composite data. They are used to implement the mechanisms made available in the VPRvsi. This library is dependant upon the underlying parallel support libraries being used to distribute data and form communications.

The highest and lowest level VIPAR libraries are designed to be interchangeable, this is how the portability is achieved. VIPAR allows different hooks from its modules, VPRvsi and VPRdd, so that VIPAR can work with different visualization systems and MPI libraries.

VIPAR is also a construction kit for producing parallel visualization modules (4) quickly and easily. The tool that does this is called the DDTool, data decomposition tool, that is used to generate template code for parallel modules. The template modules may be tailored for a variety of visualization systems, parallel support libraries and data de-/re-composition methods.

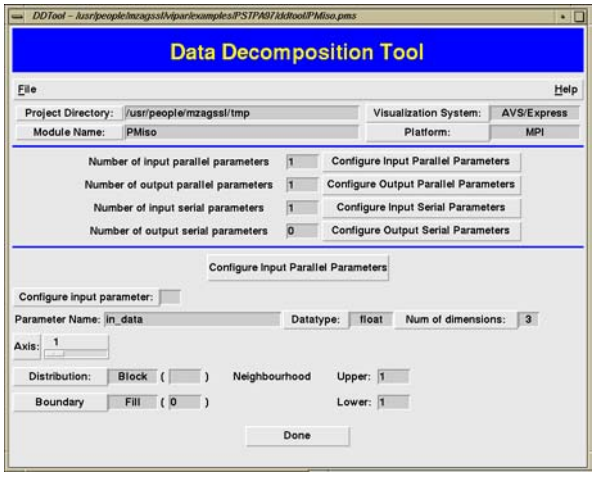

Fig 3: The DDTool produces template code that interfaces with the visualization system.

Internal Structure of The Parallel Isosurface Module

The isosurfacing module produces a surface of a given value within a scalar volume of data. The surface is in effect a 3D contour. You would use an isosurface if you wished to see the graphical depiction of a particular data value within a volume of data. The figures below show an isosurface and its associated network in AVS/Express. When the parallel isosurface module, PMiso is expanded you can see that it is made up of a distributor, a harness and a user interface.

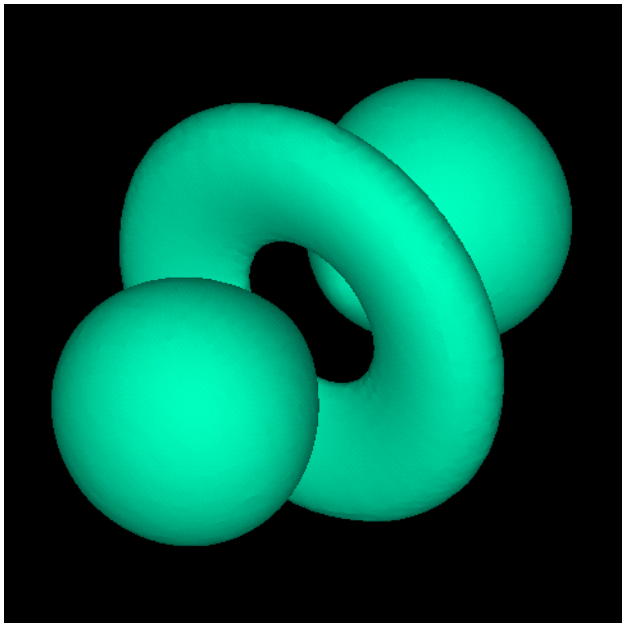

Fig 4: Isosurface of the electron potential of a hydrogen atom.

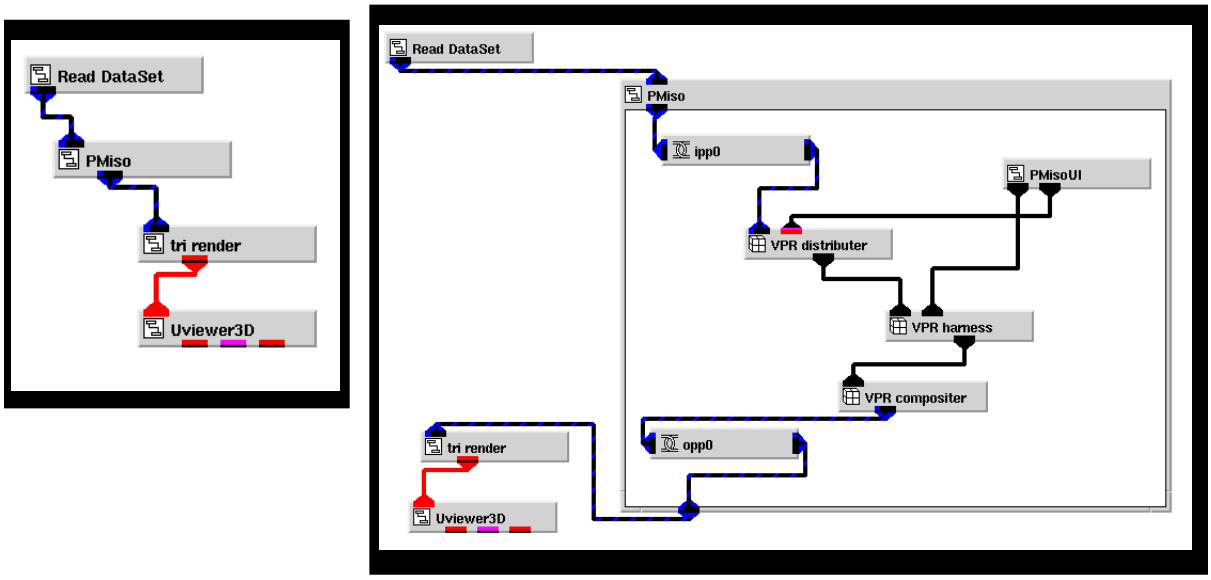

Fig 5: The network containing the VIPAR isosurfacer. The image on the right has the VIPAR module expanded to show the internal structure.

The parallel module is essentially split into 3 sub-modules. The distributor calculates, based on the strategy defined in the source code, the portions of the data to send to each processor. It is the task of the harness to send this data to the worker processes and receive the data back. The compositor then collects this data into a single array again. The sub-modules communicate with each other via tags which are stored in AVS/Express.

At the completion of the VIPAR project a number of timings were produced of the isosurfacer to see how effective this solution could be. Details of the timing experiments with a triangle decimation module are given on the web at:

http://www.man.ac.uk/MVC/students/summer97/kaffka/serial/timing.html
http://www.man.ac.uk/MVC/students/summer97/kaffka/parallel/timing.html

A graph and table of results for one set of isosurface timings are given below.

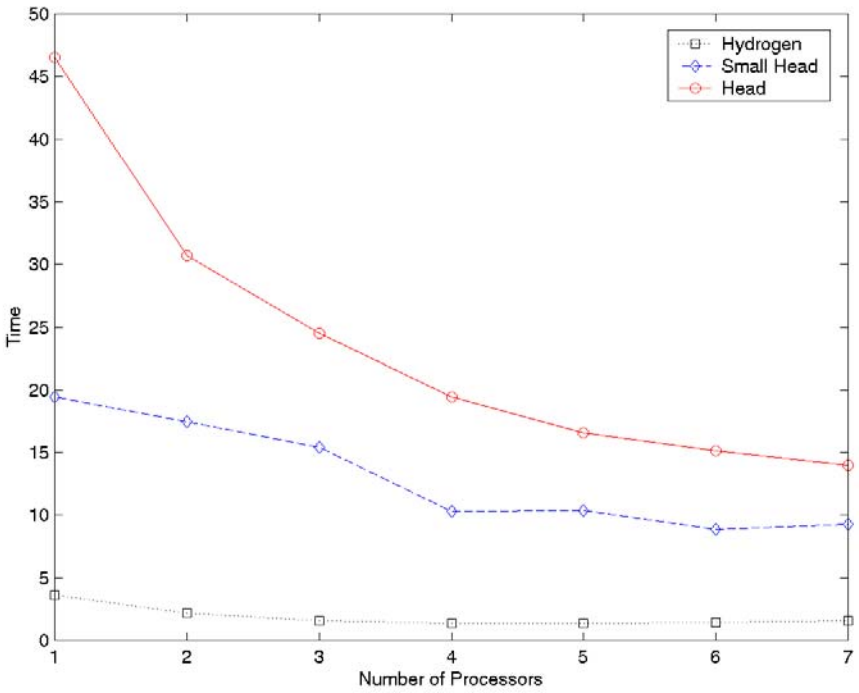

Fig 6: Timings for isosurface module in seconds

Table 1: Timings for parallel code in seconds

Processors	Hydrogen	Small Head	Head
1	3.61	19.42	46.56
2	2.16	17.48	30.67
3	1.59	15.45	24.52
4	1.38	10.28	19.44
5	1.39	10.39	16.58
6	1.41	8.88	15.13
7	1.54	9.28	13.96

All were volumes of scalar byte data, the hydrogen was 64x64x64, the small head was 128x128x56 and the head was 256x128x109. The time is given in seconds for all input data to be distributed to each processor. The trial was done over a cluster of SGI Indy workstations connected via an ATM-based LAN connecting at 155 Mb/s.

Technical Aspects of Reviving VIPAR

Before any new developments could be added to the VIPAR system the libraries needed to be compiled and run with an appropriate AVS/Express project. The original system that VIPAR had been developed on was a cluster of SGI indy running Irix 5.3 (the code was developed 4 years ago) and the current target system is a SGI Origin 2000 running Irix 6.5. The isosurface example application was the AVS/Express project that was used and it was ported to run with the new versions of LAM MPI and AVS/Express.

To get this example to run both the AVS/Express project, written in AVS/Express's own language v code, and the VIPAR libraries, written in C, needed to be debugged. Some functions within the VIPAR libraries needed to be re-written or added. And finally the way the AVS/Express picked up functions from the VIPAR libraries needed to be redefined. Details of how and why these changes were made are given below.

The VIPAR libraries and the AVS/Express projects are compiled separately. The AVS/Express project picks up the VIPAR libraries by having the right path. Initially some of the path were set as environment variables but several were set as absolute paths within the code. These paths were extracted from the code and set as environment variables so they could be easily updated globally. This produced a number of environment variables to be set. As development has continued and the functionality of the libraries and the paths have became clearer some environment variables were found to be redundant.

LAM MPI had changed its interface to some routines. This has had two effects, it meant there is now a greater dependence on the environment variables. Also some of the VIPAR library functions no longer worked and needed to be re-coded.

There had been an extensive use of wrappers within the VIPAR libraries, in many cases there was a hierarchy of wrappers. This caused particular problems because the LAM MPI interface had changed and the call of a function needed to be traced through the libraries. For example the AVS/Express project has a harness module which is associated with the source code harness.c. This code spawned processors with the function VPRvsi_StartWorkers a function from the VPRvsi library. This function then called the function VPRdd_StartWorkers in the VPRdd library which then called the function MPI_Comm_spawn from the LAM MPI library.

Another problem that occurred is that in the isosurface example we could not find the initialisation calls for MPI. The initialisation of MPI could have been done through AVS/Express, it may well have been initialised and finalised as AVS/Express start up and exit functions or in another methodology. We decided to use the MPI initialise and finalise routines in the VPRdd library as calls within the module's source code. This meant we had to include tests to check that MPI was initialised only once.

The parallelisation strategy was defined within the AVS/Express project, in its v code, it was not hard coded into the C source code. This caused complications because the values defined in the v code could be altered in an AVS/Express application at run time, when an application was saved or read into AVS/Express. Control of how these parameters are set, saved, read and fired are defined in the v code as properties. The values and how they were propagated could vary depending on whether an application was read in at start up, read in during a session or produced in a session through the Network Editor. Some of these properties were not correctly set for all ways of starting and running an application. The original application was run by a script which

set environment variables, started AVS/Express and read in the example application on start up. The properties were not set correctly for building an application in the Net Work Editor or reading in an application after start up. Proper settings were found to give the flexibility we needed to debug the code and in the long term to make it more suitable for users.

The module generation tool was developed as a prototype tool in the original version of VIPAR. The tool that does this is called the DDTool, data decomposition tool, it is designed to generate template code for parallel modules for a variety of visualization systems, parallel support libraries and data de-/re-composition methods. Initially we used this tool to develop source code to test and run VIPAR with however there were too many changes in the system for this to work and instead we picked the isosurface example application to work with. Some work was done on the DDTool but it requires further development.

The MPI Libraries

When the original VIPAR was designed it was assumed that MPI libraries would move to the MPI-2 standard and this would have spawning included. If this had happened the complexity of VIPAR could now be reduced, however, in generally available MPI libraries much of the MPI-2 functionality is not supported. There have been recent exceptions to this, most notably, Fujitsu and Sun have implemented process spawning and SGI will soon.

In practice the lack of a general MPI product led to the use of LAM MPI. The reason why LAM MPI was chosen was because its supported process spawning even before MPI-2 standardised it. When the MPI-2 standard was released the LAM extensions were converted to MPI-2 routines.

The design behind VIPAR allows serial computationally expensive modules to be replaced with parallel versions. VIPAR implements this by spawning processes (or possibly threads) from the serial MVE, these processes are then available to run the parallel code. The number of processes available to run the parallel code can be changed on a module by module basis, so that knowledge of the dataset can be factored in, e.g., it may be known that the dataset is small and scalability could be poor.

This solution seems to limit portability since LAM-MPI is not the most widely used public domain library, especially compared to MPICH. In addition, MPICH-G, is an extension of MPICH that is used by the Globus Toolkit for message-passing in a Grid environment. Given the current strategic importance of Globus for the UK eScience effort, it is desirable to enable VIPAR to integrate with Globus and an MPICH-G port would be an important step towards this.

There is also the consideration that the vendor-supplied MPI libraries may be more efficient on certain architectures than public domain libraries such as MPICH or LAM-MPI. Given the current importance of SGI machines in the field of visualization and virtual reality, a port to the SGI version of MPI for Origin 2000 and Origin 3000 machines would be very desirable. Such machines, and the machines from other manufacturers (e.g., SUN), can also be programmed using a shared memory model of parallelism. Thus VIPAR should be extended to run under Open MP which became a de Facto standard for parallelisation on shared memory systems.

The conclusion is that a solution which requires a specific MPI library can not be considered truly portable. It is to be hoped that dynamic process spawning will become more generally implemented in MPI libraries. In the meantime, other methods of communication and launching

processes (e.g., sockets) should be investigated.

The Visualization System

Most of the VIPAR work was developed with AVS/Express version 3.1/4.1, the current version is 5.1. Many changes have occurred in the software over this time, the two most important are firstly that now there is a 64 bit version as well as a 32 bit version and secondly that now there is a version which supports Multi-Pipe Rendering.

With the introduction of 64 bit machines there has been a demand for a 64 bit version of AVS/Express. The 64 bit version was introduced 18 months ago and is routinely installed and used now on most SGI machines. LAM MPI now also has a 64 bit version but it has known bugs which make it unsuitable for this project. If AVS/Express is installed as the 64 bit version it uses 64 bit addressing and if LAM MPI is installed as the 32 bit version it uses 32 bit addressing but VIPAR works by AVS/Express calling LAM MPI functions, through a VIPAR wrapper. Each uses a different address space so any address that is passed is altered and the and the function call will not work. If the 64 bit version of AVS/Express is used then the LAM MPI library should also be in 64 bit or an incompatibility will be introduced.

Like many visualization systems AVS/Express can now take advantage of new graphics hard ware, in the case of AVS/Express this includes the production of a new Multi-Pipe Rendering edition. This vastly increases the speed it takes for a visualization system to render it's images and in many cases will turn an application that used not to be interactive into one that now is. This has taken the visualization bottleneck out of the renderer into the visualization pipeline and has increased the users needs for parallel systems like VIPAR.

The original VIPAR modules were not robust because there were not enough error checks in the code and because the User Interface allowed selection of inappropriate values. We put a reasonable amount of effort into getting them into a sturdy project structure (the International AVS Centre's project structure), making the linking between AVS/Express and the VIPAR libraries non-specific (connected via environment variables rather than absolute paths), restructuring so that 2 or more VIPAR modules could be used in one session and making the user interface robust and crash free.

The advantages of the IAC (6) project structure are that it:

- Separates the module functionality into that of a a low level module, a user macro and a user interface.
- Keeps the source code well structured so that code from one module is kept separate from that of others.
- It is a well known format that is supported by the International AVS Centre, details and templates of the format can be downloaded from the IAC web site.
- It is designed for users to be able to set up the project structure easily from the template files.
- It is easier for modules to be submitted to the IAC if they are in IAC format.

The IAC project structure is the preferred structure. This meant that not only did the isosurface example need restructuring but also the template code produced by the DDTool. Despite these alterations to the DDTool, it is still only in the prototype stage.

AVS/Express comes in two editions the Developer's Edition and the Visualization Edition. The

Developers Edition comes with access to all libraries and allows a large amount of adaptability. The Visualization Edition is much cheaper and does not have the same flexibility. VIPAR was originally produced in the Developer's Edition for two reasons, firstly because development work is more productive with the Developer's Edition but secondly because VIPAR was a research project that relied on proof of concept and not end usability. However most academic users only have access to the Visualization Edition so this work had to include a port from the Developer's Edition to the Visualization Edition.

VIPAR Today and in The Future

VIPAR has currently been ported so that it will run the isosurfacing example with the 32 bit version of AVS/Express and LAM-MPI on Fermat (a SGI Origin 2000). The user interface, error handling and structure have all been improved to make the it more robust to use and install. The work done to reach this point has allowed us to gain a good understanding of the VIPAR system. VIPAR development will now be funded as part of the Reality Grid eScience project.

Future possible developments are given below. They are listed in the order they will be implemented:

1. The parallel module within the MVE is currently made of 3 component modules (a distributer, a harness and a compositor) which are called sequentially. Data is read into the distributor along with settings that define a parallel strategy. Output data is passed back into the MVE as tagged fields, which hold information about how the data in the MVE will be split and how it will be passed into the parallel process. The harness then reads in all the information in the distributor read in as well as the tagged fields then it passes the data to the parallel process and receives the results which it outputs to the MVE. The compositor reads in the settings for the parallel strategy, in the form of tagged fields, which also holds information on the re-composition and the out put of the harness and composites it in an appropriate way.

   This means source code is needed for each of the 3 component modules. We aim to condense these down into one module. Overall there will be less code produced and data will have to pass less times through the interface to the MVE.

2. The data distribution strategy is defined in the MVE, for example in AVS/Express it is defined in v code. In the original VIPAR the parameters that define the distribution strategy were set when the module was defined, the user had no GUI to change these values through although they could be altered at run time. We intend to expose these parameters to the users so that they can configure the distribution strategy through the GUI of the module. This improve the flexibility of the module and allows the users to adjust it to an optimum for a particular dataset. This improvement will require greater error handling capacities to deal with "bad" inputs.

3. Complexity of the VIPAR libraries will be reduced in 2 ways. Firstly the number of wrapper functions will be reduced, this allows the VPRvsi library to deal with interfacing with the MVE and VPRdd to deal with communication with the parallel processes. Secondly complexity can be further reduced by making routines independent of data dimensions.

4. The range of MPI routines used within the VIPAR libraries is limited to send and receive operations, spawning and a few select broadcast routines. Thus the dependence on MPI is

quite limited, in essence all that is required is for a generic VPRdd library to be created, to spread the data across threads or processors. This should have no dependence on other software and will aid cross machine boundary communication. This unique communications protocol could then be hidden away from the end programmers and be independent of parallel programming techniques such as OpenMP, pthreads and indeed MPI.

This is even more important if the implementation of process spawning in MPI-2 libraries is not universally available.

5. Neither the use of a LAM MPI based VPRdd library or a generic one is the most optimal solution for every platform, for example neither would make use of shared machines such as SGI's Origin. Continued optimization of VIPAR VPRdd libraries will be necessary for new platforms.

6. Development of a system that allows users to develop their own modules. There are 2 ways to do this, firstly by further developing the DDTool or by producing a cook book of methods. Either of these would be best developed so that it works with the final implementation of VIPAR and with many of the different data types/structures dealt with in the MVE.

Future news about VIPAR will be passed on via web pages:

http://www.man.ac.uk/mrccs/research/vipar/

References

1. VIPAR project web pages: http://www.man.ac.uk/MVC/research/vipar/
2. Larkin, S., Grant, A., Hewitt, W.T., Libraries to Support Distributed Processing of Visualization Data. In Future Generation Computer Systems (Special Issue HPCN 96) Vol. 12, no. 5, North-Holland. pp. 431-440, April 1997.
3. Larkin, S., Grant, A.J., Hewitt, W.T., Vipar Libraries to Support Distributed Processing of Visualization Data. In High Performance Computing and Networking (Proceedings of HPCN Europe '96). Springer. pp. 711-721. Brussels, Belgium, April 1996.
4. Larkin, S; Grant, A; Hewitt, W.T., A Data Decomposition Tool for Writing Parallel Modules in Visualization Systems. In Proceedings of the 1996 Eurographics UK Conference, London. pp205-216 (Vol 1), March 1996.
5. Advanced Visual Systems (AVS) home page: http://www.avs.com/
6. International AVS Centre (IAC) home page: http://iavsc.org/