Porting AVS/Express to Cray XT4

George Leaver, Martin Turner
Research Computing Services, Devonshire House, University of Manchester, Manchester, UK, M13 9PL

1 Introduction

The visualization of large datasets has become a key bottleneck in applications where validation of results or data acquisition from scientific equipment is required at an early stage. Such validation would allow correctness of methods (such as the set up of a physical experiment) to be determined prior to further computational or imaging machine resources being spent. Despite advances in GPU hardware, researchers are able to produce datasets that are too large to visualize even using modern graphics workstations and clusters.

An alternative to the GPU cluster is to use the large memory and core counts offered by supercomputers even if GPU hardware is not available. This project has ported the commercial visualization application AVS/Express[1] to the Cray XT4 hardware provided by HECToR, the UK National Supercomputing Service[2]. This allows our target end users from Materials Science to visualize large datasets that currently exceed the capabilities of our GPU-based visualization systems. Working with Materials Science we have access to datasets that are typically 500GB in size. The datasets provide material density data acquired by CT scanning equipment, some of which is provided by the Henry Moseley X-Ray Imaging Facility at Manchester[3].

The main development task was to modify the AVS/Express code such that it could operate within the HECToR runtime environment. More specifically AVS/Express uses MPI for communication between distributed compute and rendering processes but also requires X11 for user interface display. The use of MPI and X11 must be separated because the Cray architecture allows X11 only on the login nodes and MPI only on the compute nodes. This will make AVS/Express appear more like the open source Paraview[4, 5, 6] application in structure. The goal was to do this with minimal changes to the AVS/Express source code. This has been achieved by providing an MPI forwarding mechanism for the AVS components that must be executed on the login node. This is discussed in the next section.

2 MPI Forwarding

The existing visualization code comprises of a number of components. The main application is AVS/Express Distributed Data Renderer (DDR) Edition\(^1\). This version of the product is able to render data on distributed compute nodes where no GPU hardware is available using the MesaGL[7] software implementation of OpenGL. It uses sort-last rendering\(^8\) in which a complete image of the dataset is formed by compositing together rendered images of partial datasets. The partial datasets are created by decomposing the data according to some decomposition scheme. For example, sub-domains may be created by decomposing along the Z axis or using some hierarchical decomposition of the dataset.

The main express executable displays the application’s user interface. As shown in Figure 1 this comprises a network editor in which the user can connect visualization modules to form a visualization pipeline\(^9\). A separate window is also available to display an individual module’s user interface (depending on the type of module) and the resulting visualization produced by the pipeline. The user can interact with the visualization through this window. These components are part of the MPI job that is started when running AVS/Express. The other components are two types of MPI executables, namely the pstnode and mpunode executables. The pstnode processes execute parallel module codes according to the modules used in the visualization network. For example, a parallel file reader module will have the filename to read specified in the main express executable but will perform parallel file reading by passing that filename to the pstnode MPI processes. They will execute the parallel file reading code, reading their section of the dataset determined by their MPI rank and total number of MPI processes. Similarly a parallel isosurface module will receive parameters from the user interface (e.g., the isosurface level to compute) but the computation will take place within the pstnode processes on their current sub-domain of data. The visualization network will specify which modules should produce renderable geometry. Any geometry produced by the pstnode processes will be passed directly to an assigned mpunode process. The mpunode MPI processes execute the AVS/Express rendering methods in parallel. They receive global scene graph data from express and insert in to the scene graph the geometry received from their assigned pstnode. Hence each mpunode process only renders a fraction of the total geometry in the scene. The images produced by the mpunode processes are composited together (using either depth testing or alpha blending) and the final image is sent back to the express process for final display in the user interface. All communication between the various MPI processes is performed using MPI point-to-point or collective communication facilities depending on whether the message is domain-specific or common across all domains.

\(^{1}\)Productised using the Parallel Support Toolkit (PST) from JAEA.
In order that the express user interface can be run on the HECToR login node a number of changes to AVS/Express are required. Most significantly all MPI communication must be removed from the executable. Two strategies were considered, the first being to add another communication API to express and the PST framework, removing any dependency on MPI. This strategy was rejected because it would have resulted in a significant rewrite of large sections of AVS code, in particular the framework used to manage the parallel modules and rendering. Also, users developing their own parallel modules would potentially have to be aware of both the MPI and non-MPI communication methods.

The second strategy, implemented in this project, was to provide an alternative MPI library that did not use the Cray MPI layer. The express executable could still make MPI function calls but, when linked against the replacement MPI library, did not make Cray MPI function calls. Instead an implementation of the MPI functions has been developed that communicates with a proxy MPI process via standard tcp/ip sockets. As shown in Figure 2, express sends requests for MPI functions to be called on the compute node on which a proxy xpnode process is running. This process receives the request and any required arguments to the requested MPI function. For example, a request for MPI_Send() requires the buffer, datatype information, destination rank, tag and communicator arguments expected by the MPI function. Upon receiving the request the xpnode process calls the Cray MPI function with these arguments. Any results of the function (return type, buffer content etc.) are sent back to the express process via the socket.

The replacement MPI library and proxy xpnode process provide a representation of MPI types that can be used by a non-MPI process and a mapping of those types to real MPI types within the proxy process. When express creates new MPI objects (communicators, datatypes, statuses etc.) the proxy creates the equivalent objects using the Cray MPI layer and a mapping between the two repesentations is maintained. A caching mechanism has also been implemented to reduce the transfer of data over the socket when the express process makes repeated point-to-point MPI calls.

The pstnode and mpunode MPI processes are unchanged and will communicate with the xpnode process as though it were the express process because they assume express is the rank 0 process (which is true of the xpnode process) and only communicate in response to MPI functions being called by express. Hence they are completely unaware that the xpnode process is a proxy for express. It should be noted that the pstnode and mpunode processes communicate via the Cray MPI layer and so benefit from this optimized library and its use of the Cray interconnect. The amount of data sent by the non-MPI express process via the socket is in general small because it is mainly command-and-control messages from the express user interface. The global scene graph information sent from express to the mpunode render processes is also small because most of the geometry is generated by the pstnode processes. There may be a significantly large amount of geometry transported between the pstnode and mpunode processes but this communication is performed entirely within the Cray MPI domain and so suffers no change in performance as a result of removing the Cray MPI layer from the express user interface.

### 3 Summary

This project has ported a commercial visualization code to the Cray XT4 architecture provided by the HECToR service. Using this platform we have visualized CT data acquired on various CT X-Ray scanning systems using 255 cores (127 for parallel module processing, 127 for parallel rendering, 1 MPI proxy). The system allows the user to interact with the visualization at an approximate rate of 5 frames per second. This is just about sufficient for interactive manipulation of the visualization. The low framerate is mainly attributed to the use of a software implementation of OpenGL. The use of the MPI forwarding layer and proxy has eliminated the need for a significant redevelopment of the communication code within the visualization application, allowing the communication between the parallel module and rendering processes to remain within the vendor MPI domain.

Future work will be examining the use of a hybrid OpenMP / MPI approach to the parallel module processing codes so that the number of MPI processes can be reduced while increasing the use of multiple cores within the remaining processes.

### References

1. [http://www.avs.com](http://www.avs.com)
2. [http://www.hecot.ac.uk](http://www.hecot.ac.uk)
3. [http://www.materials.manchester.ac.uk/research/facilities/moseley/](http://www.materials.manchester.ac.uk/research/facilities/moseley/)
4. [http://www.paraview.org](http://www.paraview.org)
Figure 1: AVS/Express DDR architecture.

Figure 2: Forwarded MPI from login nodes to compute nodes.