Looking Ahead: SGI's Project Ultraviolet

Project Ultraviolet will build on the multi-paradigm capabilities delivered with Altix. The system architecture will be able to scale from a single node, all the way up to Petascale systems. Key advances will include:

- A new generation of interconnect, that will increase the global addressing reach, and implement communications protocols to increase the efficiency of packet and message level data transport.
- The incorporation of novel processing elements in addition to robust support for the next generation of Intel processors as the general purpose processing elements
- A second generation of the SSP to provide even greater control to these devices to increase data transport efficiency within the system architecture.
- A new data transport capability to deal with algorithms that traditionally mapped onto a vector paradigm. This will be used to supplement the microprocessors when dealing with data items that don't fit the cache-line orientated designs that mass market processors use.

Ultraviolet comprises a truly elegant combination of both flexibility and performance; one that can support everyday workload demands with a new level of productivity, while scaling up to power the next grand challenge problems which can't afford the limitations of today's clustered processor approach.

Summary

The HPC industry is facing new challenges as IC technology continues to deliver on Moore's law growth of transistors, but cores used at the heart of many cluster based systems stall in the delivery of better single thread performance. Novel computing elements, such as FPGAs and highly parallel floating point accelerators, are offering new potential to drive application performance forward. As we move towards Peta-scale computing, what will the programming models be to make effective use of such systems? SGI is taking a multi-paradigm approach, with its globally addressable memory architecture as the foundation, to build cost effective, scalable and versatile systems. SGI is already delivering on this vision, with the technology to build innovative solutions that directly address the problems of building high performance, high productivity systems.

HDF2AVS - A Simple to Use Parallel IO Library for Writing Data for AVS

Craig Lucas and Joanna Leng
Manchester Computing, University of Manchester

Motivation and Background

HDF2AVS is a library of routines to write out data, in parallel, in the HDF5 (Hierarchical Data Format) [3,4] data format, for input into the visualization system AVS/Express (Advanced Visual Systems) [1,2]. It is available as a Fortran 90 module and consists of various routines for writing out different types of array or coordinate data.

HDF uses a tree structure of groups and datasets. A group consists of zero or more datasets and other groups, together with supporting metadata. Datasets contain multidimensional arrays of data elements. The library is still in development at NCSA (National Center for Supercomputing Applications) [6]. We chose HDF for many reasons:

- it is a user defined format like XML
- it is a binary format that allows compression so drastically reducing the size of data files
- it is a format with longevity (NetCDF4 [7] is to be implemented on top of it)
- there is a dump facility that allows users to investigate the contents of binary files easily
- there is a reader for HDF already within AVS/Express
- parallel IO is supported.

There are many advantages to writing out data in parallel. On parallel machines there are two traditional approaches to writing data. One is to collect all data to one processor and then write this to disk. This obviously creates a communication overhead and can be slowed down further if there is not enough local memory on this master processor to hold the entire data set. These problems can be avoided by the other standard approach

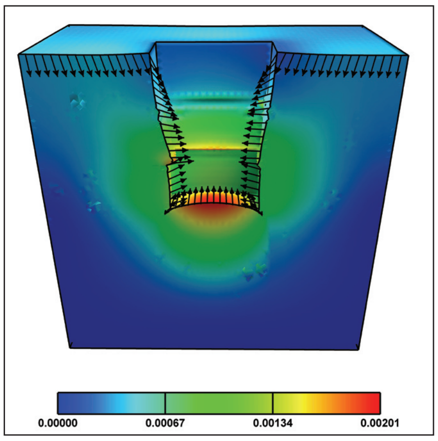

Figure 1: The results of an FEA soil excavation and subsidence experiment using an inhomogeneous soil model, generated by a random field generator.

of having each processor write to its own file. This introduces a post processing overhead, and also limits you to read the files back in on the same number of processors. Parallel HDF is built on top of MPI-IO [5] which allows simultaneous access to the same file, thus removing all of the problems above.

The HDF files are in binary. In fact, MPI-IO itself only supports binary data. Binary (and compressed binary) files are much smaller than ASCII files, easily by a factor of 3 or 4. Our experience shows that many computational scientists write ASCII data, as they are human readable, and binary files, of course, are not. However, HDF supply the excellent utility, h5dump, to dump to the screen all or part of a binary HDF file, thus removing any disadvantages of binary data. Figure 1 shows the visualization of some FEA data [7]. Stored as ASCII the total file size for this data is 57MB. Converted to HDF format the size is 20MB and a further reduction can be obtained by converting to a compressed HDF format giving a file size of 13MB. Although this is a small example, the potential in reducing file sizes is obvious for very large datasets.

We chose to write the HDF2AVS wrapper library for many reasons. Firstly, the AVS reader requires much metadata; for example the dataset requires metadata for the data type, its dimensions and the grid used. In fact, to write out a 2 dimensional array, in a form that the AVSreader understands, requires 87 HDF subroutine calls! However, only one call to HDF2AVS.

Also, some of the concepts in HDF are not straightforward, with data and file spaces, property lists and unique data types. But to use our library the user is not required to understand any of these.

Using HDF2AVS

To write out an array in HDF2AVS you just need to pass the dimensions of your global data, and the part of that global data owned by the calling process. For example consider writing a 2 dimensional array.

Figure 2 shows a possible data distribution for the local arrays distributed over blocks of rows of the global array. The dimension of the global array is *m* x *n*, and the process here owns *t* rows of data, the first of which is in row *s* of the global array. Note HDF numbers its coordinates from zero.

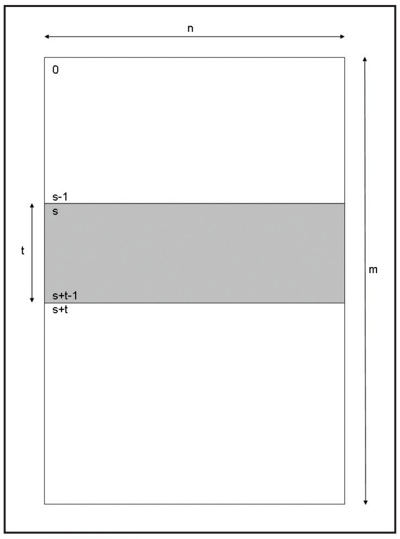

Figure 2: An example data distribution for HDF2AVS. The global array has *m* rows and *n* columns of data. The local array has *t* rows of data, and is shown shaded.

There is no constraint in where or how big the local array is. For instance, local arrays can overlap, and need not span the whole width and not all of the global array needs to be written to. We do, however, check that you don't try to write outside of the global array, which is not permitted in HDF.

```
PROGRAM using_hdf2avs
USE MPI
USE HDF2AVS
.
! Could be REAL etc
INTEGER, ALLOCATABLE, DIMENSION(:,:) :: my_data .
INTEGER(KIND = 8), DIMENSION(2)  :: global_dims
INTEGER(KIND = 8), DIMENSION(2)  :: local_dims
INTEGER(KIND = 8), DIMENSION(2)  :: global_origin
.
global_dims = (/m,n/)
! Data split in rows
local_dims = (/t,n/)
global_origin = (/s,0/)
ALLOCATE( my_data(t,n) )
.
CALL MPI_INIT(error)
.
CALL HDF2AVS_WRITE_2D_ARRAY(my_data, global_dims,
&
           local_dims, global_origin, 'filename.
h5' ) .
CALL MPI_FINALIZE(error)
END PROGRAM using_hdf2avs
```

Figure 3: A code fragment, in Fortran, calling HDF2AVS to write out a 2D array.

In Figure 3 we give a code fragment, in Fortran, that calls the HDF routine HDF2AVS_WRITE_2D_ARRAY. Note the generic routine name for our supported data types: `integer, real` and `real*8.`

The routine takes just five arguments:

- `my_data` is the local data on the process
- `global_dims` is the dimension of the global dataset distributed over all processes
- `local_dims` are the dimensions of `my_data`
- `global_origin` gives the position of the local data in the global array
- the file name is also required.

This is a collective call made by all processes.

Performance

In writing the library we were not driven by performance issues. We have concentrated on functionality rather than performance. We wanted to help users to write their data out easily to a useful visualization format. It is not intended to replace your bespoke IO routines.

Consider the example where you have data on each processor and you wish for this data to be written to a single file. We have 1GB of data on each process, so the more processors used, the larger the file. We can write out the data using HDF2AVS with a single collective call. Alternatively, a master process receives the arrays from each of the other processors and writes them individually to the file.

Figure 4 shows the time to write the HDF file and also the time to receive and write the individual files in a binary format. Note that the time taken to write ASCII data is very much increased, and it not shown. You can see that the times vary enormously for different runs, up to a factor of 10. There are many issues that affect the performance at a particular time: how busy the machine is, where the processors are on the machine, whether there is enough memory available to buffer the data before writing and what other codes running are demanding OS services etc. We used the CSAR machine Newton for these timings, which was heavily loaded.

Further Work and Information

The library is still under development, planned improvements include:

- Support for cell data for FEA applications.
- Optional arguments for altering the metadata.
- MPI-IO optimizations.
- Routines for reading files.
- Other data distributions.
- Documentation.

The aim of this article is to gauge interest. The development can be driven by user requirements.

If you think this library could be useful to your applications please contact either Craig Lucas or Joanna Leng. We can advise on how the library could be used or adapted for your application and dataset.

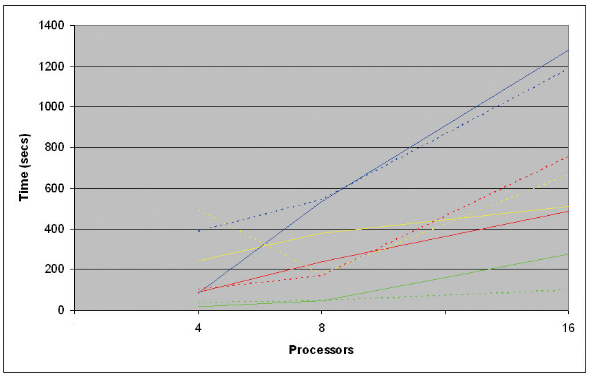

Figure 4: Comparison of HDV2AVS and gathering data on one process to write to a single file, for a 2D array. Each colour relates to the same 16 processors for both methods run concurrently. HDF2AVS is shown solid and gathering dashed.

References

[1] Advanced Visual Systems Inc. *AVS/Express Developer's Reference*, Release 3.1.

[2] Advanced Visual Systems Inc. Homepage: http://www.avs.com/

[3] HDF Homepage: http://hdf.ncsa.uiuc.edu/index.html

[4] HDF5 API *Specification Reference Manual*

[5] MPI Forum Documentation: http://www.mpi-forum.org/docs/docs.html

[6] NCSA Homepage: http://www.ncsa.uiuc.edu/

[7] NetCDF Homepage: http://www.unidata.ucar.edu/software/netcdf/

[8] Smith I.M., Leng J. and Margetts L., *Parallel Three Dimensional Finite Element Analysis of Excavation*, ACME, Sheffield, March 2005.