



Parallel Visualization of Petascale Simulation Results from GROMACS, NAMD and CP2K on IBM Blue Gene/P using VisIt Visualization Toolkit

Dr. Valentin Pavlov*, Dr. Miroslav Iliev, Anton Tomov, Veslin Slavchev, Dimitar Dimitrov, Nina Ilieva

NCSA, Acad. G. Bonchev str., bl. 25A, Sofia 1113, Bulgaria

Abstract

Visualization is a key post-processing activity for petaflops simulations. In this paper we have researched, identified and implemented a methodology suitable for carrying out this activity, based on the VisIt open-source visualization tool by LLNL. We have installed the toolkit on a specialized hardware in NCSA's Tier-1 facilities in Sofia, integrating it with our primary Tier-1 system, an IBM Blue Gene/P. We have researched the possibility to support GROMACS, CP2K and NAMD data formats and proposed best practice procedures. The experience and methodology is documented and can be used to integrate the tools into other PRACE facilities.

1. Introduction

Post-processing of Petascale simulation results – and in particular visualization – is in itself a hard task that requires a substantial amount of resources. In Petascale simulations, life science software packages like GROMACS and NAMD deal with millions of atoms whose trajectory footprint may require hundreds of MB per frame. Computational chemistry packages like CP2K will produce similarly sized output for even smaller systems. Just the movement of this data from the computing center to the scientist's workstation can become prohibitive. Moreover, workstation hardware may not be powerful enough to render the millions of atoms and bonds produced in a single frame. This outlines the kind of support requested from this project and consequently its goals:

- To provide parallel visualization post-processing solution and methodology that supports at least GROMACS, NAMD and CP2K output formats;
- The visualization package should be based on a distributed architecture that allows the post-processing engine to run on the remote computing center, using its resources to process/render the scenes, while the actual images are viewed on a local workstation close to the scientist;
- The parallel post-processing engine must operate on the computing nodes of an IBM Blue Gene/P machine, as a representative of a PRACE RI Tier-1 resource;

The objective of the project is to research the means and provide the methodology and tools for users of the above mentioned packages to be able to visualize Petascale simulation results using the computing power of Tier-1 machines, and more specifically the IBM Blue Gene/P, and without the need to physically transfer the whole output to their workstations.

* Corresponding author. *E-mail address:* vpavlov@rila.bg

2. Methods

The VisIt open source package from LLNL [1] supports to some extent the above requirements. Still, some improvements and optimizations had to be done in order to achieve the objectives of the project:

- The IBM Blue Gene/P platform is not officially supported *a priori*, so a porting exercise was needed. This proved to be quite a demanding job and accounts for the most part of the effort spent on the project. Several issues were encountered:
 - Architecturally, IBM Blue Gene/P computing nodes cannot adopt rendering hardware, which leaves off-screen rendering as the only option. Although this is somewhat supported by VisIt, it is not straightforward to remove all dependencies on X11 libraries and objects and to come up with correct dependencies to other rendering libraries as Mesa, GLEW, VTK, etc.
 - VisIt is a very complex, loosely coupled, dynamically loaded and componentized software, parts of which should run on the front-end node of the Blue Gene/P, while other parts must run on the compute nodes. This component mix-in introduces additional complexity and it is very time consuming to produce the correct compile-time configuration.
- Implement input methods that support the GROMACS, CP2K and NAMD output formats. For GROMACS and CP2K these include a set of instructions to the simulators regarding output file generation, while in the NAMD case an additional plug-in for VisIt is needed.
- Provide a methodology for the users of these packages that includes instructions how to structure their output and best practices how to use the package.
- Provide a methodology and reference binary package for other centers to integrate the same package in their software stack.

2.1. VisIt architectural overview

VisIt is componentized software adhering to the client/server architectural model. Figure 1 shows a simplified diagram depicting its major components and their interrelations. Two of the components – the GUI and the Viewer – reside on the local computer – the scientist's workstation. Through the GUI he/she is able to select data for viewing, schedule draws, apply drawing operators and control the viewport and display settings. The Viewer is the window displaying the actual drawing of the data, according to the selections in the GUI. The actual processing, including data input, scene composition and possibly rendering occur on the remote computer, which in this case is the supercomputing machine itself. This leads to two important benefits: 1) the whole batch of data stays close to where it was generated and is not transferred in its entirety to the client machine; only the elements of the scene currently visible in the Viewer window is put on the wire; and 2) the parallel execution environment of the supercomputing machine (or the cluster) may be used to greatly speed up the scene composition itself and possibly also the rendering.

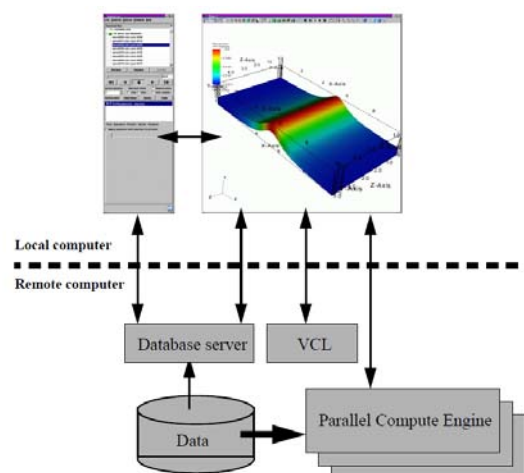


Fig. 1. Overview of VisIt architecture (Source: [1])

The main technical goal if this project is to adapt the server components of VisIt to the Blue Gene/P architecture. The client-side components can be installed straightforward on the scientist's machine from binary distributions off the package web-site.

The main server-side components are:

- Database server;
- VCL
- Parallel Compute Engine;

VCL serves as the main component launcher. The client-side contacts VCL in order to launch different server-side components. As such it runs only on the Blue Gene Front-End Node.

The Database server's responsibility is to provide Input/Output services to the client-side. It is launched by the VCL and runs only on the Blue Gene Front-End Node. It provides a pluggable architecture through the use of Shared/Dynamic Libraries. The public API that is exposed can be used to write new plug-ins to read/write data from/to file formats not originally supported by the distribution. The data sources are referred to as *databases* and are usually files or collections of files, but can also be relational databases, network streams or other exotic devices.

The Parallel Compute Engine is the main processing component of VisIt. As its name implies, it runs in parallel and as such it is the only component that needs to be compiled for the Blue Gene Compute Nodes. As with the Database server, it provides a pluggable architecture through Shared/Dynamic Libraries and a public API. Each plug-in for the Database server has a corresponding plug-in for the Parallel Compute Engine. While the Database server plug-in is concerned with physically reading/writing the data source, the Compute Engine plug-in is concerned with building in-memory structures that correspond to the data being displayed *and* the method used to display the data.

The method used to display the data, along with available operators upon the drawing is subject to another set of plug-ins for the Compute Engine. The distribution supports the main drawing methods and operations used in the community and can be enlarged with additional ones.

The main workflow proceeds as follows: the GUI sends the Compute Engine the parameters of the scene – viewpoint, scale, coordinates of the bounding box, method for display, etc. – and the engine computes which data objects fall inside the scene. It uses the Database server to extract information about the objects currently being displayed, and through the plug-ins for display methods and operators constructs a list of graphical primitives to be displayed.

The engine supports two modes of operation: normal mode and scalable rendering mode. In normal mode, the engine uses the off-screen rendering provided by the Mesa library to construct a list of graphical primitives that need to be displayed (because they appear in the current scene as viewed from the corresponding viewpoint) and sends this list to the GUI via the VCL. This is the only information that is transmitted over the wire from the server center to the client machine. The workstation's graphical card is then used to actually render the scene on the screen.

The scalable rendering mode – supported on clusters and supercomputers which can adopt GPUs in their configuration – works by delegating off-screen rendering to the connected GPUs; this can lead to substantial performance gains. In this case, the scene is actually rendered by the GPUs and the engine sends to the GUI the pixel data that should be displayed. The workstation graphical card treats the bitmap image and just displays it (no additional rendering).

The Blue Gene/P architecture supports only normal mode, since the Compute Nodes cannot adopt GPUs.

2.2. Porting

We based the porting effort for the latest version of VisIt (2.3.2) on an article in the VisIt user forum [2]. The wiki page describes the steps taken to build a previous version of VisIt (1.12) for IBM Blue Gene/P. Although it does provide important guidelines, it explicitly states that it is not applicable for recent versions, because of several important changes in the build system and in the way OpenGL is recognized and worked with.

VisIt's web-site provides a build script – `build_visit2.3.2` – which automates the creation of a binary package. It works perfectly well for building the Blue Gene Front-End Node executable tree, but is incapable of building the Compute Nodes version. In the original version, the build script sequentially builds:

- Third-party required libraries:
 - CMAKE build system;
 - Python 2.6 programming environment;
 - Qt 4.2 GUI library;
 - Mesa 7.8.2 library – OpenGL reference implementation, including off-screen rendering;
 - VTK 5.0.0i – Visualization Toolkit library for visualization primitives;
- Third-party optional libraries – a whole lot of libraries for data I/O like HDF5, NetCDF, silo, CGNS, etc.
- VisIt 2.3.2, including all client- and server-side components and selected plug-ins.

The Blue Gene Compute Nodes run CNK – Compute Node Kernel – an IBM proprietary OS. The build script needs to be modified in order to cross-compile the Parallel Compute Engine for the CNK. The following changes are required:

- Third-party required libraries:
 - CMAKE build system is compiled as usual, producing executable for the Front-End Node. This is required, because the build script and configuration scripts of the components search for the cmake executable in a pre-configured location created by the build script itself;
 - Python 2.6 programming environment is compiled as usual, producing executable for the Front-End Node. Again, this is required because the build script and configuration scripts of the components search for Python in a pre-configured location created by the build script itself. This python environment however is never used.
 - Qt 4.2 GUI library is compiled as usual, producing a library for the Front-End Node. As with CMAKE and Python, this is required in order to satisfy the dependencies of the build system itself. The built Qt library is never actually used, since the Blue Gene compute nodes does not have on-screen visualization capabilities and thus a GUI library is not used.
 - Mesa 7.8.2 library is cross-compiled to produce a CNK version. The original build script compiles two versions of Mesa – a mangled X11/GLX version and a mangled off-screen rendering version. This is needed to satisfy the VTK requirements. Mangled versions are built in order to allow VTK to use both optimized OpenGL and Mesa simultaneously. Since the Blue Gene CNK does not support optimized OpenGL, Mesa is used for all rendering. Our experiments show that by building a single non-mangled off-screen version of Mesa, dependencies are easier to satisfy and so the build script is changed accordingly. Also, VisIt uses the GLEW library to *dynamically* load Mesa during runtime, thus a shared library needs to be built.
 - VTK 5.0.0i is cross-compiled to produce a CNK version. There are a few places in which VTK depends on X11 explicitly and since X11 is not supported on Blue Gene CNK these need to be removed. The situation is described in [2]. Following the advice therein and adapting the changes to the currently used release, a set of patches is included in the build script that allows for building VTK without any X11 support. Also, since CNK does not offer support for OpenGL, the VTK build system is changed in order to require and support only non-mangled off-screen Mesa rendering.
- The third-party optional libraries are not ported as part of this project – they are not needed for the file formats we are targeting to and can be built in a future extension of this project.
- VisIt 2.3.2 is cross-compiled to produce a CNK version. This was the hardest part of the porting process, since the new CMAKE-based build system differs wildly from the build system described in [2] and some of the more advanced options necessary were actually lacking. The following patches were made to the build system and the source code:

- A new CMAKE variable, `VISIT_OFFSCREEN_ONLY` was introduced. By setting it to `TRUE`, the build system creates a distribution that only supports off-screen rendering and all dependencies to X11, GLX, OpenGL, Qt, etc. are guarded by appropriate conditional compilation directives.
- The build system is changed so that it only requires and links against the single non-mangled off-screen Mesa library built earlier.
- Instead of searching of an optimized OpenGL implementation, the build system locates and uses the Mesa library built earlier.
- The GLEW library included in the VisIt distribution needs to be changed in order to properly find and dynamically bind to the non-mangled off-screen Mesa, and to remove any dependencies on GLX found in its source code.
- Some of the display methods (e.g. Rubber Band Mapper 2D) are only supported in on-screen rendering mode. They produce a compilation error in case no suitable implementation is found. The build system needed to be changed in order to skip building these methods.
- The `internallauncher` perl application that is called by the VCL to launch the Parallel Compute Engine had to be changed in order to adopt the parameters of the `mpirun` to the format required by the MPICH2 implementation found on the Blue Gene. Currently, due to the nature of visualization jobs, it is envisioned that they would be run in a separate reserved partition by calling `mpirun` directly. However, if the submission of visualization jobs through TWS LoadLeveler is to be supported, this same script needs to be changed in order to prepare the job control file and submit the job in the LL queue.
- The linker scripts that are used to link the binaries and shared libraries is changed to remove some implicit assumptions contained in them. For example, the build script assumes that the linker produces a shared executable, while the cross-linker produces a static executable by default.
- In addition to those, several improvements to the build system itself are introduced like better debug support and fixing an error in BOV display method compilation.

All these changes are implemented in a modified build script which, along with the pre-compiled binary package is available from the authors upon request. As seen fit, the build script and/or binary package will also be distributed through the regular PRACE publication facilities.

2.3. Handling required inputs

We require the ported installation of VisIt to provide support for the output formats of GROMACS, CP2K and NAMD. These applications produce time-series frames of the evolution in time of the system under study. It is desirable that such time-series are viewed as animations. In VisIt, by default the Database server supports animations by expecting the different frames stored in different files, all having a common prefix and a numeral suffix that contains the corresponding sequence number of the frame. This setup is achievable on all three packages through command line options to the simulators or through post-processing scripts like GROMACS's `trjconv`.

GROMACS and NAMD can both be instructed to supply their output in PDB (Protein Data Bank) format. It is supported in VisIt's default distribution through the built-in PDB plug-in. Similarly, CP2K's output is supported through the built-in XYZ plug-in. The standard distribution also provides the Molecule drawing method, which is useful for displaying the required kind of data.

3. Results

The objective of this project is to enable supercomputing centers that operates an IBM Blue Gene/P installation to provide its users the option for scalable visualization solution for Life science simulations. The port described in this paper achieves the goal. The main results of the project are:

- A modified build script that produces a build for the IBM Blue Gene/P CNK;
- Instructions for other centers how to use the build script in order to integrate the same package in their software stack;
- A reference implementation binary package build using the above script;
- Instructions for the users about structuring their output so as to satisfy the default VisIt requirements.

Above results are available from the authors upon request and will be distributed through the official PRACE publication facilities.

4. Future Work

There are several directions in which future work can commence:

- Investigate the Scalable Rendering mode of operations on a GPU-enabled cluster and compare the performance to the IBM Blue Gene/P implementation we have. This can be useful for decision making for centers looking for visualization solution for their users;
- Implement additional plug-ins for VisIt that support the native binary output formats of the packages (e.g. TRR for GROMACS, DCD for NAMD);
- Integrate VisIt as an *in situ* library into one or all of the packages in order to support real-time display of simulation results while frames are actually being generated by the corresponding software. This online mode of visualization will allow online monitoring of the simulation progress which can save time and resources.

Acknowledgements

This work was financially supported by the PRACE project funded in part by the EUs 7th Framework Programme (FP7/2007-2013) under grant agreement no. RI-211528 and FP7-261557. The work is achieved using the PRACE Research Infrastructure resource IBM Blue Gene/P Tier-1 system at NCSA, Sofia, Bulgaria.

References

1. VisIt package home page. <http://visit.llnl.gov>
2. Building VisIt on Blue Gene/P. http://www.visitusers.org/index.php?title=Building_on_BlueGeneP