Lawrence
Livermore
National
Laboratory

LLNL-TR-519831

Creating a Parallel Version of VisIt For Microsoft Windows

Brad Whitlock, Kathleen Biagas, and Paul Rawson

December 7, 2011

DISCLAIMER

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Creating a Parallel Version of VisIt for Microsoft Windows

Brad Whitlock, Kathleen Biagas, and Paul Rawson
Lawrence Livermore National Laboratory

Introduction

VisIt is a popular, free interactive parallel visualization and analysis tool for scientific data. Users can quickly generate visualizations from their data, animate them through time, manipulate them, and save the resulting images or movies for presentations. VisIt was designed from the ground up to work on many scales of computers from modest desktops up to massively parallel clusters. VisIt is comprised of a set of cooperating programs. All programs can be run locally or in client/server mode in which some run locally and some run remotely on compute clusters. The VisIt program most able to harness today's computing power is the VisIt *compute engine*. The compute engine is responsible for reading simulation data from disk, processing it, and sending results or images back to the VisIt viewer program. In a parallel environment, the compute engine runs several processes, coordinating using the Message Passing Interface (MPI) library. Each MPI process reads some subset of the scientific data and filters the data in various ways to create useful visualizations. By using MPI, VisIt has been able to scale well into the thousands of processors on large computers such as dawn and graph at LLNL.

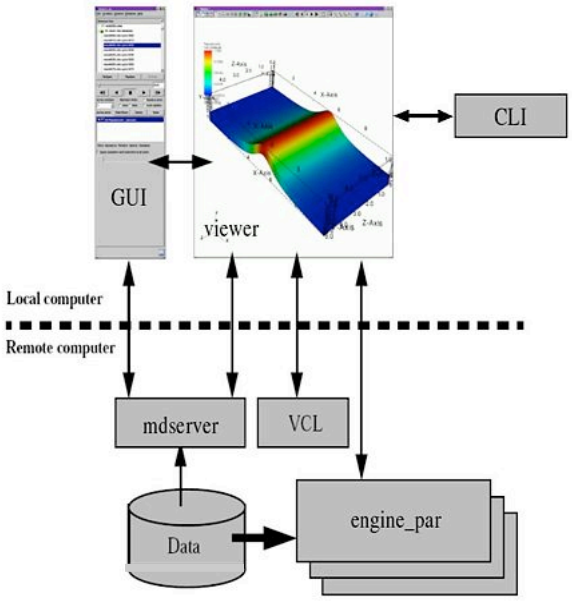

Figure 1 - VisIt architecture

The advent of multicore CPU's has made parallelism the "new" way to achieve increasing performance. With today's computers having at least 2 cores and in many cases up to 8 and beyond, it is more important than ever to deploy parallel software that can use that computing power not only on clusters but also on the desktop. We have created a parallel version of VisIt for Windows that uses Microsoft's MPI

implementation (MSMPI) to process data in parallel on the Windows desktop as well as on a Windows HPC cluster running Microsoft Windows Server 2008. Initial desktop parallel support for Windows was deployed in VisIt 2.4.0. Windows HPC cluster support has been completed and will appear in the VisIt 2.5.0 release. We plan to continue supporting parallel VisIt on Windows so our users will be able to take full advantage of their multicore resources.

Single-node Parallelism

The first deliverable in this project was a parallel version of VisIt capable of running on a single desktop computer, using a user-settable number of cores. As the schedule for this project coincided with a major update to the VisIt source code to update it to VTK 5.8, we chose to begin once we completed the VTK upgrade. We also made this decision because we wanted to deliver this project using libraries built with Microsoft Visual Studio 2010, which was made available for this work. VisIt's reliance on many open source libraries makes the process of upgrading compilers on Windows a difficult task since many open source projects lack an adequate build system for Windows. After we rebuilt VisIt's core dependencies using Visual Studio 2010, we were free to concentrate on making a parallel version of VisIt's compute engine.

We have improved the process of building VisIt on Windows greatly over the past 2 years by transitioning the project's entire build system to CMake. Prior to the transition, VisIt maintained 2 separate build systems: an autoconf-based build system for UNIX and Visual Studio project files for Windows. VisIt itself is built from dozens of shared libraries and hundreds of plug-ins. Maintaining 2 build systems was not sustainable and it posed difficulties for creating a parallel build of VisIt because of the large number of additional parallel libraries. After the CMake transition, VisIt had a single build system and Visual Studio project files could be generated directly from the build logic contained in the CMake scripts. Since VisIt was already building in parallel on other platforms, the process of generating extra project files for the parallel components came for free.

Fortunately, minimal source code modifications were needed in order to support MSMPI since VisIt had already been built using various MPI implementations. The main changes needed to support an MPI-based compute engine on Windows were in the VisIt launch program. On all platforms, there is a VisIt launch program that sets up the environment and performs some other initialization before calling the real executable programs. On Windows, the launch program is a small C++ program and we had to modify it to accept some command line arguments that VisIt expects when running a parallel compute engine. In addition, we changed the launch program so that when a parallel compute engine is requested, the actual executable in invoked using the *mpiexec* launch program.

After enhancing the VisIt launch program, we made other improvements to VisIt's installer program. For instance, the VisIt installer now will prompt the user to install MSMPI if MSMPI is not detected on the system. Once MSMPI support is detected, the installer creates VisIt host profiles that let it launch in in parallel using some number of

cores. The installer also creates Start menu shortcuts for running VisIt in serial and parallel.

Multiple-node Parallelism

Creating a version of VisIt that can operate on a Windows HPC cluster involved more substantial changes to VisIt. When VisIt wants to run its compute engine in client/server mode on a remote computer, it typically starts listening for socket connections on a port. Then VisIt invokes a secure shell (SSH) program to run the VisIt launch script on the remote computer, passing arguments that tell the remote program how to connect back via a socket. The launch script runs the VisIt Component Launcher program (VCL) on the remote computer and it coordinates subsequent program launches. When VisIt runs client/server, the remote computer ends up running VCL, a metadata server, and VisIt's compute engine. Since Windows does not supply an SSH daemon, we decided to change how VisIt talks to the Windows HPC cluster, dropping SSH in favor of interacting directly with the job scheduler.

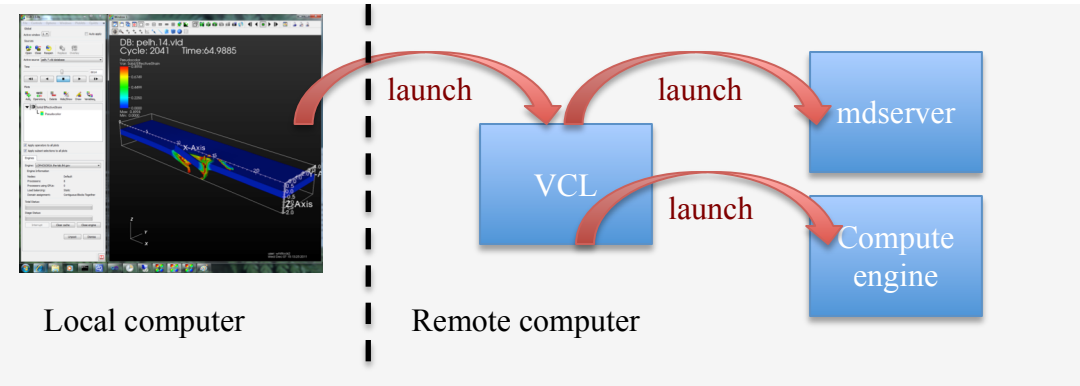

Figure 2 - Normal VisIt launch procedure

The role of starting our compute engine was delegated to the job scheduler instead of using SSH and our VCL program. We wanted to simplify the problem of job submission. Instead of having to coordinate the launch of multiple jobs or create multiple tasks in a job for each of the VisIt programs we need on the remote cluster, we decided to reorganize VisIt's launching somewhat. We added a launch mode whereby the metadata server and the compute engine can share the same batch job. When running client/server in this mode, VisIt launches the compute engine on the remote cluster first. When a metadata server is required, the engine spawns a process for it within its allocated compute nodes. This simplification of the job launching process allowed VisIt to create an HPC job with a single mpiexec/compute engine task. During this reorganization phase, we also discovered and fixed minor errors with how VisIt creates program command lines destined for remote computers, adding better support for Windows.

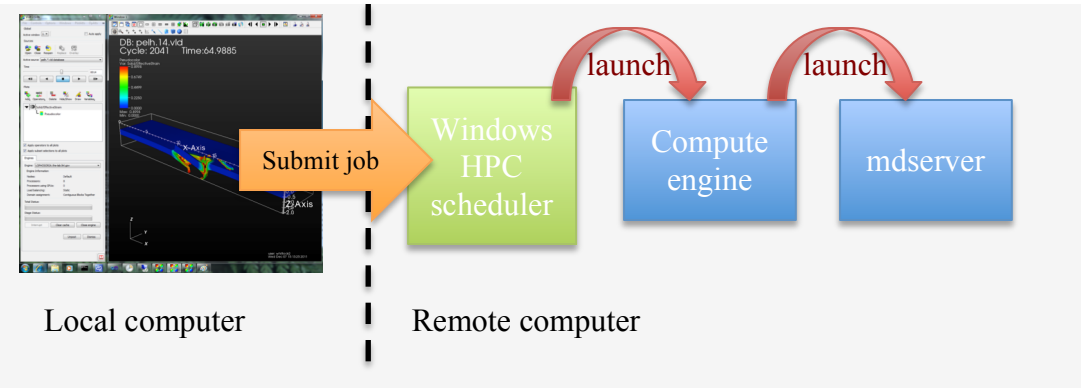

Figure 3 - Shared job launched via Windows HPC scheduler

After creating the new "shared job" launch mode that enables the VisIt compute engine and metadata server to share the same compute nodes, we needed a way for VisIt to automatically submit its jobs to the Windows HPC scheduler. Fortunately, the Windows HPC scheduler provides a COM interface so we were able to write a new function for VisIt's viewer that uses COM to connect to the scheduler and submit a custom job. This approach allows us to bypass our need for SSH since the COM interface to the job scheduler lets us queue our job on the cluster without our code needing to know the details about how the communication takes place. Once the VisIt compute engine runs on the Windows HPC cluster, it connects back to the viewer on the local computer.

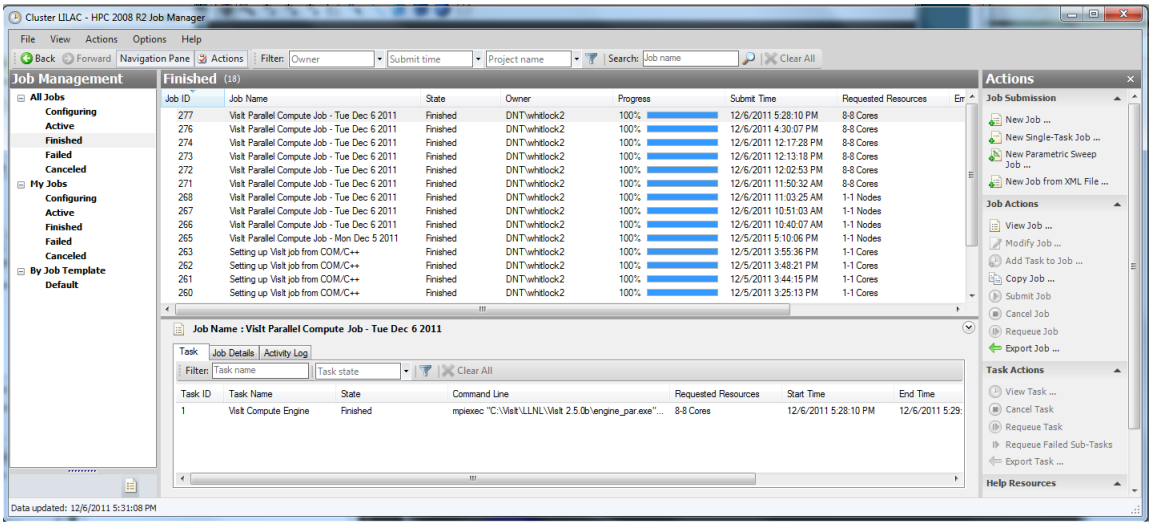

Figure 4 - HPC Job Manager showing VisIt parallel engine jobs

We are currently assuming that the cluster's compute nodes can connect a socket back to the Windows computer that is running VisIt's viewer. These changes accomplish phase 2 of the project, getting VisIt to work with Windows HPC clusters.

Cluster configuration

For the purposes of testing our code modifications, we created a 3 node Windows HPC cluster running Microsoft Windows Server 2008 R2 Standard on each of the nodes. The cluster nodes are Dell Optiplex computers with Intel Core i7-2600 CPU at 3.4GHz, 8Gb RAM, 1Gb/s networking, and 500Gb hard drives. We roughly followed the cluster-

building instructions at http://social.technet.microsoft.com/wiki/contents/articles/2539.aspx. Instead of configuring one of the nodes with 2 network interfaces for the purpose of creating a private network for the compute nodes (as suggested in the article) we connected each node to the enterprise network, as required by our IT policies.

One caveat of this setup is that since all nodes are on the enterprise network, it means that all nodes have outward connectivity to other machines on the enterprise network. This is actually needed since the rank 0 process in VisIt's compute engine creates sockets back to VisIt's viewer, which would be running on another computer. If compute nodes were to lack connectivity to the enterprise network then VisIt's rank 0 process might need to run on the cluster head node. Future work might investigate alternate modes of network communication between the rank 0 compute engine process and the viewer process.

Results

We performed some simple timings of VisIt's compute engine processing a synthetic particle dataset on 1, 8, and 16 cores. The particle dataset contains 8M particles with 8 variables per particle, divided into up to 32 Silo/HDF5 data files for each of the 170 time steps. The number of actual files varies per time step because each file represents a spatial block and the particles move through the blocks and some blocks might not have particles. The size of the data in each time step is roughly 680Mb. We replicated the data on each of the compute nodes in our cluster since our cluster has no shared file system. Then we ran VisIt and timed how long it took to render a Pseudocolor plot of the particles for each of the time steps.

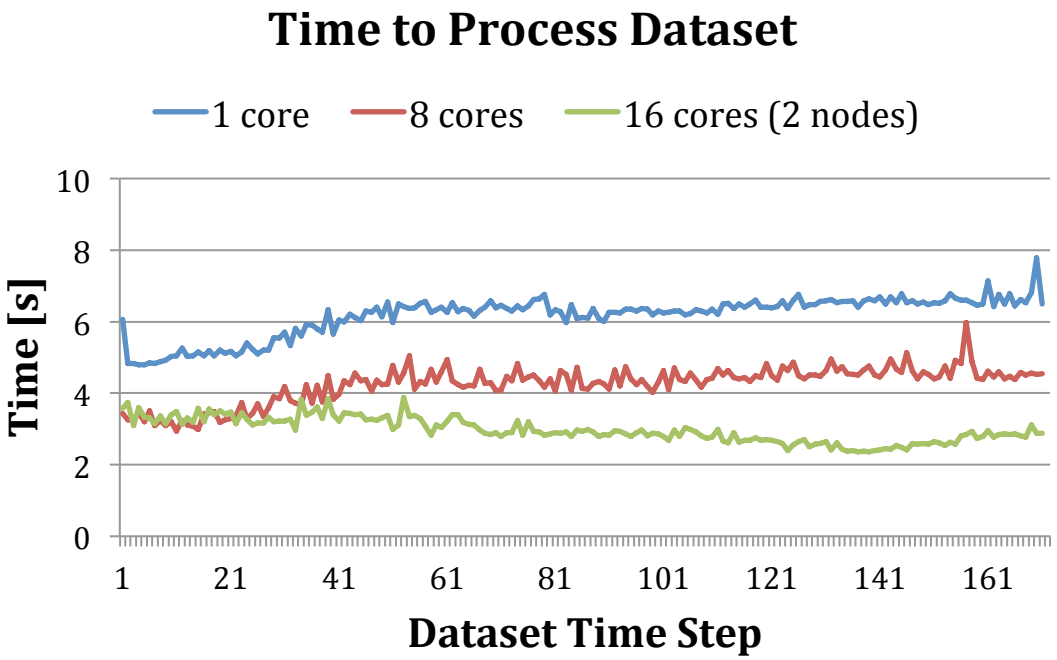

Figure 5 - Time to Process Dataset Using Different Numbers of Cores

When run in parallel, VisIt's compute engine outperforms the single-core version by roughly 2 seconds. The parallel speed advantage comes mainly from having more processes perform I/O since this test is not very computationally demanding. Although the same amount of data is being read in aggregate for each dataset time step, the parallel decomposition varies as particles move from one file to another. Initially, nearly all particles are contained in 4 files, leaving many of the processors idle. As time advances and particles become more evenly balanced among the number of files, the load balance improves and parallel processing becomes more advantageous. This pattern is best demonstrated by comparing the timings for the 8 and 16 core cases. While the timings begin similarly since particles are contained in relatively few files, as the particles distribute, less I/O work is being done per file and VisIt is better able to exploit the I/O bandwidth offered by the second compute node.

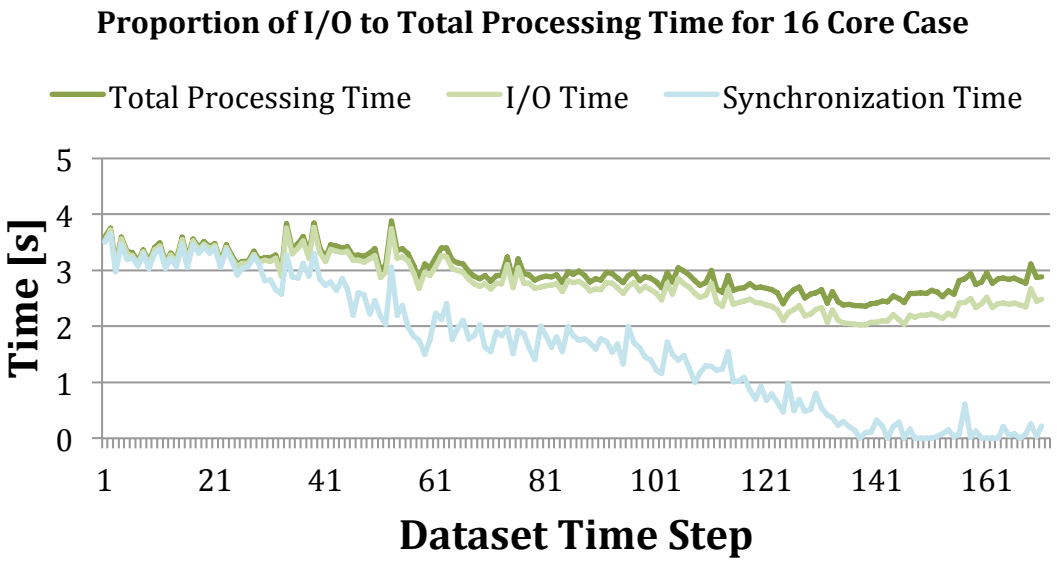

Figure 6 - I/O for the 16 core case

When examining the 16 core results further, we break down the total time process a dataset into I/O time and synchronization time. The I/O time is the total time spent performing I/O and synchronization. The synchronization time is time spent waiting for all processors to catch up so the visualization routines can proceed. For all dataset time steps, I/O dominates the time needed to process the dataset. The synchronization time decreases though since particles are better distributed among files in the later dataset time steps. This redistribution of particles among files leads to faster I/O since processors have more uniform data to read from disk.

Summary

We have enhanced VisIt, a visualization and data analysis code, so it can run in parallel on the Microsoft Windows operating system. The parallel version of VisIt can operate on a single desktop computer or it can submit jobs to a Windows HPC cluster. We plan to continue building parallel versions of VisIt for Windows and to continue improving our

support for the Windows operating system. Source code and binaries are available from the following locations:

- http://portal.nersc.gov/svn/visit/trunk/src/
- http://portal.nersc.gov/svn/visit/trunk/releases/2.4.0/

Binaries featuring support for the Windows HPC scheduler will be released when VisIt 2.5.0 is released. Until then, source code may be used for testing.

Acknowledgements

We wish to thank Wen-ming Ye of Microsoft Corporation for sponsoring this project. This work fulfills the *Work for Others Agreement L-13620* between Lawrence Livermore National Laboratory and Microsoft Corporation.