AC 2012-3853: DEVELOPMENT AND IMPLEMENTATION OF A HIGH PERFORMANCE COMPUTER (HPC) CLUSTER FOR ENGINEERING EDUCATION SIMULATIONS

Dr. Kurt C. Gramoll, University of Oklahoma

Kurt Gramoll is Hughes Professor of Engineering.

Development and Implementation of a High Performance Computer (HPC) Cluster for Engineering Education Simulations

Abstract

With the advancements in high performance computer (HPC) computing, it is only natural that engineering education also utilizes the massive computational capabilities of large server clusters to enhance student learning. This paper presents recent work in developing and implementing complex engineering simulations for engineering education. Key aspects of this work include developing methods to access the simulations through web pages, creating user friendly input modules (web-based), automated job control system for web submission, efficient core utilization for a large number of simultaneous users, and display of results on the user's web page. Each of these issues are critical to engineering education due to the unique environment required for using computers in classroom and lab settings.

A detailed working example for torsional stress of non-circular bars is given in the paper to illustrate the implementation of a server cluster. Currently, up to 40,000 degree of freedom problems can be solved with the system. Execution time varies depending on the number of cores devoted to a given problem. But even if only one core is used, the solution time is 10-50 faster on the cluster than on the client device (laptop, smart phone, tablet, etc.) since the cluster solver is compiled C code instead of less efficient Flash ActionScript. All examples used in the paper are currently available at www.eCourses.ou.edu.

The paper addresses the special needs of engineering education when utilizing HPC systems. All simulations are web-based, and students do not need special knowledge of clusters, job control, or parallel programming. Simulations are accessed through a web page where parameters, such as boundary conditions, geometry constraints, loads, accuracy and grid resolution (FEA) are specified. The web interface is one of the more difficult aspects of the system. The interface needs to be intuitive and accessible on a large number of devices, such as laptops, smart phones, and tablets.

To simplify the development of the user interface, this system used web-enabled Flash Player for both the simulation set up and viewing of the results. This allows most devices connected to the internet to access the system through a common web page. Utilizing Flash also makes it easier to develop advanced user interface graphics such as real-time grid generation, sliders, input boxes and graphical result output.

The paper provides details on how the dedicated 32 node (384-core) engineering education cluster was set up using Windows 2008 HPC Server R2. This includes the job control system, allocation of core resources, cluster solvers, utilization of math libraries, and network communications between the cluster and user during the solution steps. One of the primary goals of this paper is encourage others to pursue developing complex simulations for engineering education so that students can reap the benefits of the recent advances in cluster computing.

Introduction

Even though desktop and laptop computers have continued to increase in computational speed efficiency, smaller devices (tablets and smart phones) have been introduced with significantly reduced computational capabilities. This is only natural since these mobile devices where not designed to perform intense numerical calculations. However, they have become as common as slide rules in the 1960's or hand calculators for the last four decades. There is now hardly an engineer without a smart phone. The question becomes, why is engineering education not using smart phones and tables (mobile devices) for complex engineering analysis? Two main concerns come to mind immediately, their screen sizes are small when compared to traditional desktop or laptop computers, and they are relatively slow (but still faster than desktop computers in the 1980's). While the screen size can be a problem, current mobile devices have incredible screen resolutions (over 300 dpi in some cases). A recently released smart phone by Samsung, the Galaxy Nexus [1], has a resolution of 720x1280 which is the same for most mid-range laptop computers. Granted, the pixels are tightly packed, but they are capable of rendering detailed stress plots or flow fields.

The next issue is the relatively slow CPU speed of mobile devices. They are constantly improving, but they are not designed with engineers in mind and will always be slow for intense numerical calculations like finite element analysis. This paper addresses a solution to this problem by off-loading calculations to a server cluster through a web-based analysis tool. While clusters are not new, they generally are used in batch mode where input programs are submitted through specialized tools, such as PuTTY [2]. But these tools are not conducive for use by engineering students in non-computer science courses. Thus, a new method to interface with the cluster needed to be developed that is designed for engineering students with no experience in cluster computing or finite element method. But this was only one part of the solution. A cluster control system also had to be developed to allocate nodes and cores on the cluster to a particular analysis and user. Next, the engineering problem had to be solved using various numerical methods. Finally, the results needed to be returned to the user and displayed. This all needs to take place in just a few seconds, assuming reasonable network bandwidth.

The project can be separated into three main parts. The first being the web-based tool that allows students the ability to change basic parameters for common engineering problems. The tool also plots the results returned from the. The tool must communicate with the server through a standard network protocol, such as HTTP or RTMP (Adobe format) [3] and send basic information about the design to the server. The second component in the system is the cluster job control program. This program is called from the web tool and allocates the cluster resources, sets run conditions, and sends the problem parameters to one of the cluster compute nodes (attached servers). The actual problem calculations are in the third component of the system, the solver. The solver takes the input parameters from the job control, sets up the gird, constructs the general matrix, applies the boundary conditions, and solves the resulting set of equations. The results are sent back to the job control program (still running), which then sends the results back to the client web tool. These basic components or parts are illustrated in Fig. 1.

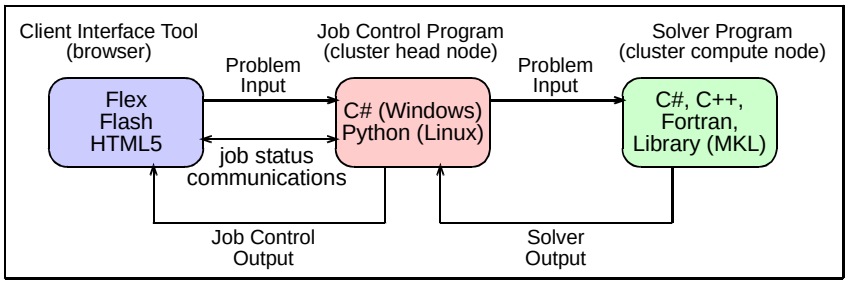

Figure 1. Problem Set and Solution Process

The three step process is conceptually straight forward, but the actual communications between the three components was challenging. The following sections will describe in detail each of these components and how they work together to provide a method for solving complex engineering problems on any network connected device.

To illustrate the complete system, a basic demo example will be given. The example is the torsion of non-circular bars or shafts. This is a common problem in engineering mechanics but is rarely taught since there are few closed form solutions. The tool is also integrated into the eBook at www.eCourses.ou.edu [4] for student use. The theory and application of non-circular bars is provided at the web site, and the tool allows students to solve advanced configurations.

User (Client) Interface Tool

The user interface tool (client) is the key link between the user and the cluster and is the only component that the student will interact with. The client can be developed using a variety of program application tools, including Flash, Flex, HTML5, and Silverlight. These are the four most common web-based application development tools. It is also possible to develop a full “app” that can be run outside the browser. “Apps” are common for smart phones and tablets, but this route is more complex and will only work on the device that it was developed for. On the other hand, web-based applications (not native) can theoretically run in any web page. This allows the developer to program it once for all platforms.

This project focused on web-based applications for the ability to reach a large audience while only programming one interface. Each of the four common development tools for web-based applications (Flash, Flex, HTML5, and Silverlight) has advantages and disadvantages. Microsoft Silverlight [16] has the lowest installed base, and is quickly losing ground to the other tools. Flash and Flex are similar tools, both from Adobe [5]. In fact, they both produce a common file format type, swf, that can be played in any browser using the Flash Player plugin (>98% installed). The main drawback for Flash/Flex is that Apple mobile devices do not support the Flash Player, which means iPhones and iPads cannot be used. The final tool, HTML5, is quickly growing in popularity, and is support by most browsers. There are two difficulties with HTML5. First, it is still a young format, and it not as sophisticated as Flash/Flex or even Silverlight. Particularly, the graphics are difficult to program and lack many basic drawing capabilities.

Second, HTML5 communication libraries and classes for sending data back and forth between servers and clients are limited. HTML5 can work with server sockets, but each browser has implemented sockets slightly differently, and this makes it near impossible to have all users experience the same high level of performance without developing for multiple socket types.

After surveying all tools, it was decided that Flex (or Flash) would be the best tool even though it excludes use with Apple mobile products. In particular, Flex has advanced graphics for user interaction, and a variety of communication functions to interface with the cluster. It should be noted, even though Flex and Flash are similar (both use ActionScript programming language), their component libraries and third party components are different. Flex is designed to be a full application running in a web page. On the other hand, Flash is targeted more for web-based animations and simple user interaction. However, either one can perform as the client for the front end to the cluster.

Flex itself can do some calculations, but it is slow and not designed for numerical methods. Similar to Java programming language, Flex uses byte-code that must be re-compiled at run time. This is important so that it can run on any platform but the penalty is speed. It is an order of magnitude slower than a normal C#, C++ or FORTRAN compiled executable program. For a small engineering problem, where the number of simultaneous system of equations (degree of freedom, DOF) is less than one thousand, most computers can handle it within a few seconds or less. However, for large DOF cases, many computers (especially smart phones and tablets) will take minutes (or hours) and will time-out. Detailed times are given in the results section.

The Flex interface program serves two functions. First, it gives the user control over the engineering problem such as dimensions, similar to any pre-processor. Second, it serves as a post-processor when the calculations are completed. Since the front end functions (pre- and post-processing) are separated from the core solver routines, it is possible to change the front end tool later with another program developed with another language (i.e. HTML5). An example of the front end for the torsion example, before and after the solver, is illustrated in Fig. 2.

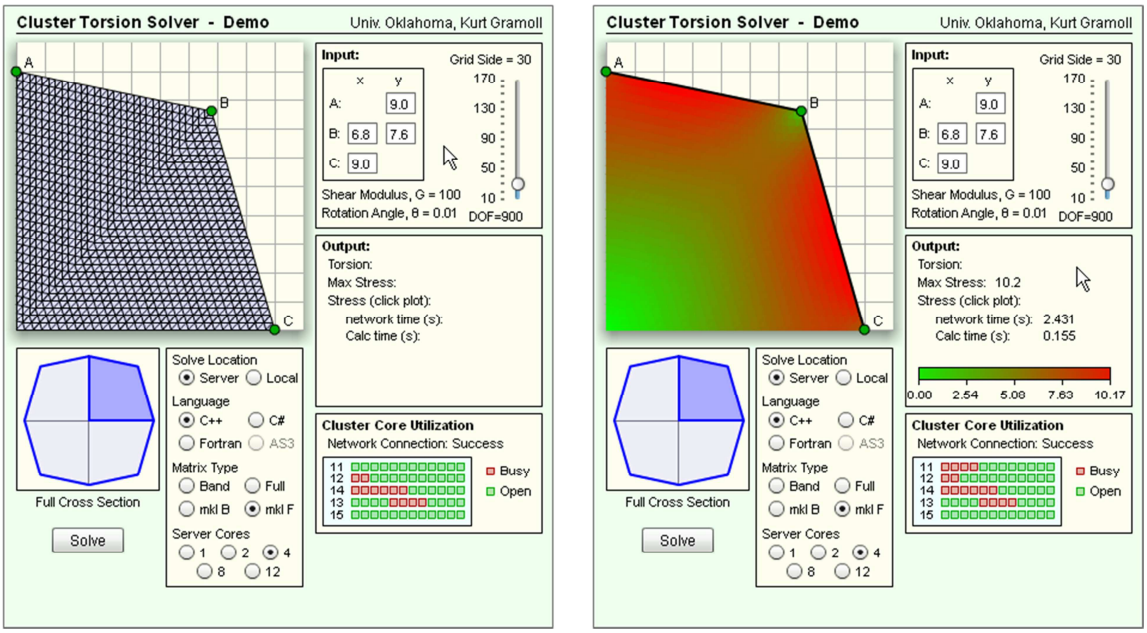

Figure 2. Torsion Simulator Example a) User Setting Problem Parameters b) Solution Results

The key programming detail with the client program is its communication with the server cluster. Flex can make simple HTTP function calls to the website to run server scripts (PHP, Perl, ASP.NET, etc.), but this has two major limitations; text is transmitted as strings and it is synchronous. A better option is to use a media server on the cluster such as Red5, WebOrb, LiveCycle, Wowza, etc. These tools allow Flex to communicate with the server through dedicated channels using sockets. This means communications are asynchronous (and in binary) which allows the server to communicate with the client at any time, and the client can make multiple function calls. This project uses WebOrb [6] since it works with ASP.NET and is free for university community licenses.

Basically, the client invocates a function on the server and transmits the problem parameters. The function on the server is part of a compiled DLL (HPC Windows 2008 operating system) that acts as the job control program (see next section). At the same time, the remote DLL on the server communicates back to the client to update how many cores are available on the cluster. The small red squares in Fig. 2 above indicate that particular core is currently solving a problem. Each core on the cluster system used for this project has 12 cores (2 cpu's with 6 cores each).

Server Cluster and Job Control

A server cluster is a group of servers that are linked together to perform large scale calculations using parallel processing techniques. The capabilities of clusters have increased tremendously over the last several decades and are now in the price range of individual researchers or small labs. For example, the cost for a moderate size, blade-based cluster of 16 nodes (a node is an individual server) that has two cpu's and each cpu has 6 cores is in the range of $60,000-$75,000 depending on the memory, cpu speeds and hard disk space. Such a system would likely have a

theoretical TFLOPs (tera-floating-point operations per second) of about 2. In 1997, the top cluster in the world, Sandia National Lab, [7] was rated at approximately 2 TFLOPs (and filled a room and cost millions). The current cluster used for this project at the University of Oklahoma has 32 nodes with a total of 384 cores and has a theoretical 4.1 TFLOPs capability.

A basic cluster configuration requires one node designated as the head node, which controls the computer nodes. Depending on the operating system, Linux or Windows, there can be other designated nodes for specialized functions. This project used Windows HPC 2008 R2 system for all nodes, and is configured for basic operation. The choice of Windows was due, in part, to the author's past experience with Windows, an available site license for Microsoft software, and the availability of third party communication software between Windows and Flex. Linux is more common for pure cluster configurations where the job submission is not done through web pages.

The Windows operating system installation for a cluster is similar to any Windows server operating system, but there are several special features (or roles) that need to be activated [8]. For example, the head node must also work as a DNS (Domain Name System) server, DHCP (Dynamic Host Configuration Protocol) Server, Active Directory Server, File Services server and an IIS (Internet Information Services) Web Server. These are all used to organize and communicate between the head node and compute nodes (and between compute nodes themselves). One of the more challenging aspects of this project was learning all these different servers and how they are used to facilitate full utilization of the cluster. After the head node operating system is set up, a special add-on, HPC (High Performance Computing) Pack 2008, needs to be installed. This is separate from the operating system and has a separate license from Microsoft. It installs the control programs that turn the basic server into a cluster server. It also has utilities that help the cluster administrator organize and maintain the cluster.

After the head node is set up, the basic Windows HPC 2008 R2 operating system is installed on each individual compute node. No special services, features or roles need to be turned on. But the add-on HPC Pack 2008 needs to be installed on each compute node. The HPC Pack ties the compute nodes back to the head node so that the head node knows they are now part of the cluster, and the head node can assign tasks to them.

The head node has a built in job control program, Cluster Manager, that allows individual jobs to be scheduled and submitted. The term "job" refers to the execution of another program such as a executable (.exe) file constructed separately. Generally, users submit the jobs; they run, write results to a file, and then the user accesses the file to view the results. This is similar to the 1970's batch submission concept. (Sadly the author still remembers those days along with the frustration of not getting immediate response and having to wait overnight in many cases.)

One of the main goals of this research was to avoid the "batch" method of interacting with the cluster by using a web-based interface. This would allow anyone, anywhere, to utilize the system without installing any additional software. To do this, job submission had to be done through a server job control program and not the built-in Cluster Manager program (Fig. 3). The developed job control program accepts problem parameters from the client program (Flex); it sets up the job dynamically, assigns tasks to the job (generally, there is only one task per job for these simple simulations), specifics the resources (1-12 cores), and set time limits. Once this is set up, then the

job is submitted to a compute node for execution. It should be noted, the term “task” will be running the actual solver program (third component of the system). The solver program is developed separately so that multiple instances of the solver can be run at any given time. This also allows the solver to use any programming language such as Fortran, C++ or even C#, whereas the control job program must be done in C# so that it can access all HPC classes built into the cluster HPC extensions.

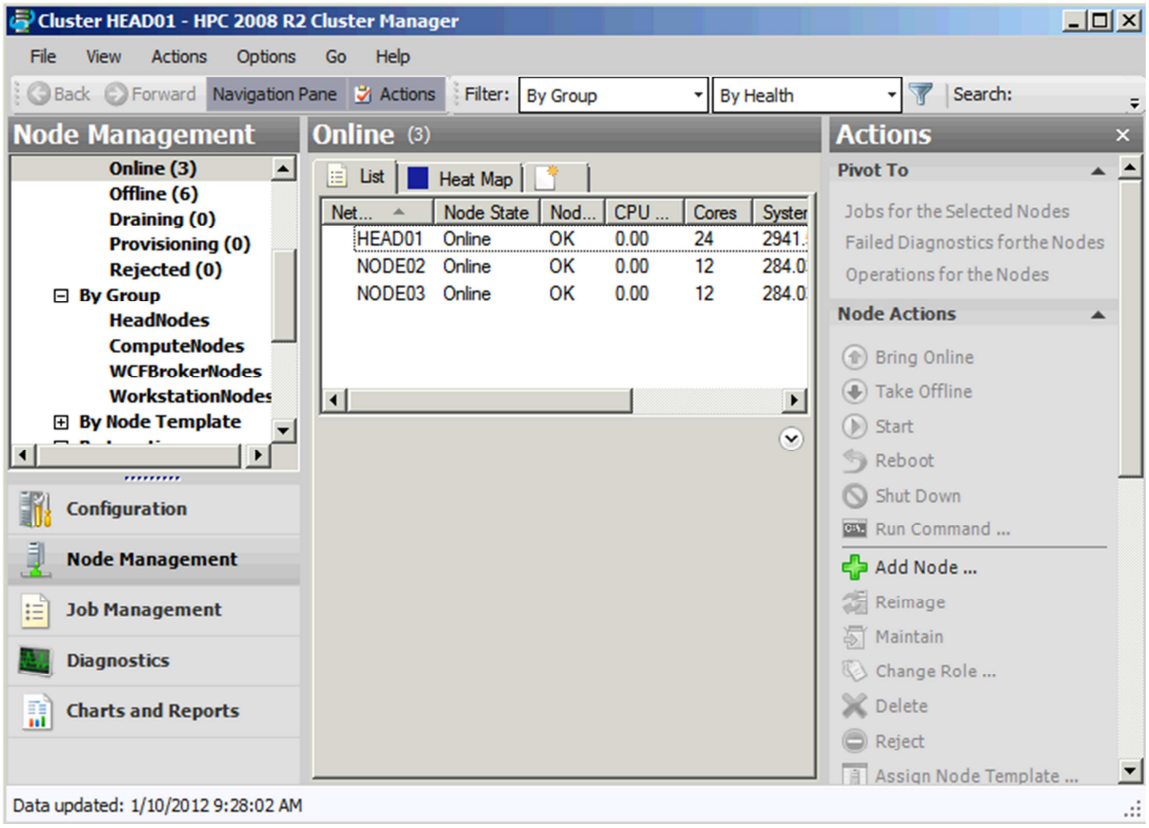

Figure 3. Windows HPC 2008 R2 Cluster Manager

To summarize, the user sets up the simulation parameters using the client on a web page (Flex). Flex sends the parameters (dimensions, grid density, material constants, etc.) to the cluster head node by making a remote function call (also called “remoting”) to the server to start the job control program. That program then sets up the job for submission to the actual solver executable. Now, the final piece of the puzzle is the actual solver (turns out to the simplest part of the whole project), and that is discussed in the next section.

Solver Program Component

The core program that does the actual calculations is the third component of the system. It is called by the job control program on the head node and executed on one of the compute nodes. The solver program can be developed in any language and is independent of the job control

program. It is just a basic executable program that accepts input, solves the problem, and then writes out the results. However, there are a number of special considerations when running the solver through a job control program.

The solver receives the problem parameters from the job control (which received them from the client) and must pass them onto the solver. This can be done by using arguments input string when calling the solver. For example, when running the torsion example, a typical job task statement would be, “Torsion_cpp.exe 6 6 6 6 50 mklb 4”. This is set by the job control program, and it references the solver input parameters including solver language (C++ version), the dimensions of the problem (6,6,6,6), grid density (50), solver routine (mklb), and number of cores (4).

Since the solver is actually run on a compute node, a mechanism is needed to return the results to the head node so that the job control program can return the results to the client. This a bit convoluted, but the solver cannot communicate directly to the client; it is only solving the problem. Thus, the job control program must wait until the solver is finished, then fetch the results from the solver, and send it back to the web-based client which then presents the results to the user. To facilitate transferring the results between the solver and the job control, a shared volume was set up that can be accessed by all nodes. The solver program resides on the share volume which makes it easy for the job control program to run from any node. (They all have access to the same executable.) The job control program also sets up the file name where the solver results are written (on the shared volume) and changes the solver console output to that file. When the solver program finishes, the job control is notified and reads in the solver output file. The job control packages the data into a data string and sent back to the client. In Flex, there is a listener event set up for the return of data which is activated when the data is received.

Thus, the job control program on the head node sets up a job with a single task of solving the problem. The job is submitted to a compute node. Then the solver performs the analysis and writes the results to a file on a shared volume in the cluster. The job control reads the file (It knows the file name since it redirected the solver output to the file specified in the job control program.) and sends the data back to Flex.

While the solver is working on the problem, the job control program communicates back to the client using a dedicated communication channel through RTMP (Real Time Message Protocol). This is made possible through the use of WebOrb which is a third party server-based program that allows socket communications between the server and Flex clients, or even between different Flex clients. It has been used previously to construct real time lecture tools [9].

Solver Types

The solver is the core program that does the actual program calculations. Since it is compiled and on the server with a fast cpu, it is generally 1-2 orders of magnitude faster than using the local computer. Remember, the local computer, even if it has a fast cpu, is running the Flash Player in a browser using ActionScript which not compiled. So by just compiling the solver, the user will experience an order of magnitude in execution time reduction. However, the negative side of

executing the solver on a server or cluster is added time for the network transmission and large string manipulation between the job control, solver and back to the client.

Various different solvers were developed to test different speeds and conditions. Four different computer languages (ActionScript, C#, C++ and FORTRAN) were used, all using the same two algorithms to solve the finite element based torsion problem. ActionScript (native language of Flex) was included to compare calculations times with cluster-based solver times. The other three, C#, C++, and FORTRAN were all developed and compiled for cluster execution. The two algorithms used are the Cholesky decomposition for a fully populated matrix (full) and for a banded matrix (band) [10]. While all 2D torsion problems can be solved efficiently with the band method, the full method was included to compare times when many calculations are needed. Both C++ and FORTRAN used Intel compilers, and C# used the native ASP.NET compiler. Visual Studio 2010 [15] development tool was used for all languages.

Neither the band matrix nor full matrix algorithm will work on multiple cores, since the Cholesky method is inherently solved sequential and difficult to apply parallel programming techniques. But the main goal of this project was to utilize multiple cores to quickly solve problems. Thus, two additional methods were added using the Intel MKL (Math Kernel Library) [11]. The Intel MKL has hundreds of efficient algorithms for most numerical math routines. For solving fully populated matrices, the LAPACKE_dposv routine was used. For banded matrices, the LAPACKE_dpbsv routine was used. The implementation of both routines requires the matrix to be converted to a single dimensional column array. The number of cores that can be used varies from 1 to 12 (maximum cores per node) since this project currently does not use special shared memory techniques between nodes to allow for more than 12 cores. This would add more overhead to the programming, and it was found most problems only need 2-4 cores to solve quickly. Also, it is expected that dozens of users will be using the system simultaneously and thus no one user should use more than 12 cores so that all users will experience a quick response. However, future work will involve moving beyond 12 cores for particularly complex simulations.

After testing solution times for various matrix sizes (degrees of freedom for torsion example problem), it was found that C++ was fastest for the full matrix solution method as shown in Fig. 4. For the full matrix, it was substantial. The actual code algorithm was exactly the same in all three test cases. Some speed difference could be due to array allocation. No special language-specific optimization was done to enhance the speed. The banded solution method was not as dramatic. All tests were done using a server with Intel x5650 cpu operating at 2.67 Ghz.

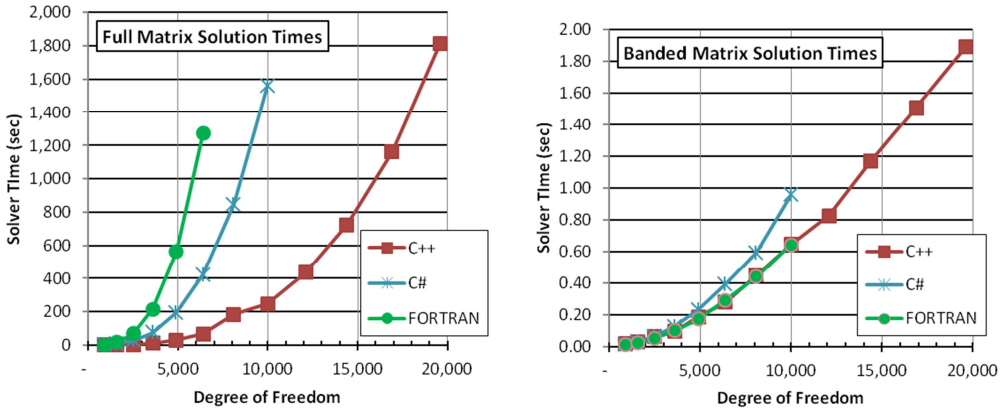

Figure 4. Solver Solution Times a) Full Matrix, b) Banded Matrix

The next comparison is between Flex using ActionScript (AS) and C++. As expected, ActionScript is an order of magnitude slower than compiled C++ (Fig. 5). This is one of the reasons a cluster server is needed to solve complex engineering problems. This issue is even more problematic when the user tries using the torsion example problem on a slower smart phone or tablet.

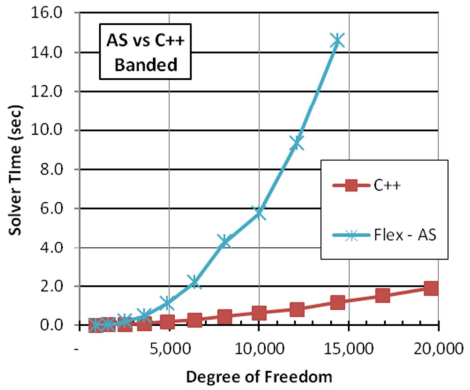

Figure 5. Solver Solution Times Between Flex (ActionScript - AS) and C++

The final set of test cases involves solving the torsion problem using the Intel Math Kernel Library (MKL) that is available from Intel for its FORTRAN and C++ compilers. It is a large collection of routines for numerically intensive calculations. Furthermore, many of the routines can utilize multiple cores if they are present. The LAPACKE_dposv routine used to solve the full matrix did allow the program to set the number of cores to be used. The test case looked at the solution times for 1, 2, 4, 8 and 12 cores in solving degree of freedom problems up to 19,600. As expected, when more than one core is used, the solution time does decrease but it is not a linear relationship. In Fig. 6a, the raw times are given. To better understand this nonlinear relationship,

the times were plotted as a ratio of single time to the multiple core time (Fig. 6b). If it was a perfect system, then using 2 cores were decrease the time by 50%. But there is always inefficiencies and overhead with using multiple cores, especially for smaller problems. It is interesting to note, as the problem size increases, the efficiencies increases, especially for low number of cores. When using 12 cores, even at 19,600 degree of freedom, the efficiency only approached 8 (not the theoretical possible 12). To maximize the use of the cluster, 4 to 8 cores should be used. More cores will have less effect (law of diminishing return).

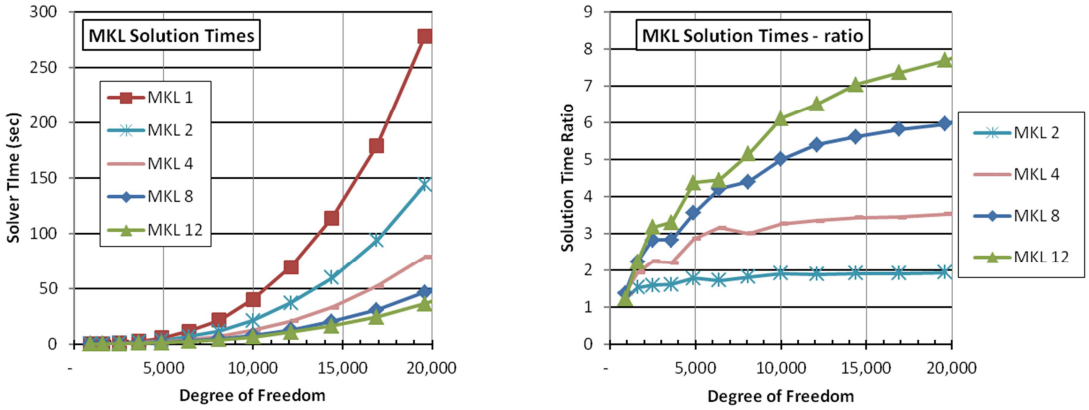

Figure 6. Solver Solution Times Using MKL Library
a) Actual Times, b) Ratio of One-Core Time to Multi-Core Time

It should be noted, no attempt was made to utilize more than 12 cores on a single problem. To do this requires additional programming and the use of Message Passing Interface (MPI) between physically separate nodes. Since it is expected to have dozens of users requesting cores at any given time, it would be unwise to allow each of them to use more than 12 cores at a time.

Torsion Simulation Development and Results

The test case for this paper is calculating the stresses for a non-circular bar or shaft. This is a basic structural mechanics problem that has been solved for only a few cases. The most common is the circular bars or shafts. Other closed form solutions include elliptical, rectangular and triangular cross section [12]. To allow students to visualize and try different cross sections, the torsion simulator (or Torsion Solver) was developed (Fig. 7).

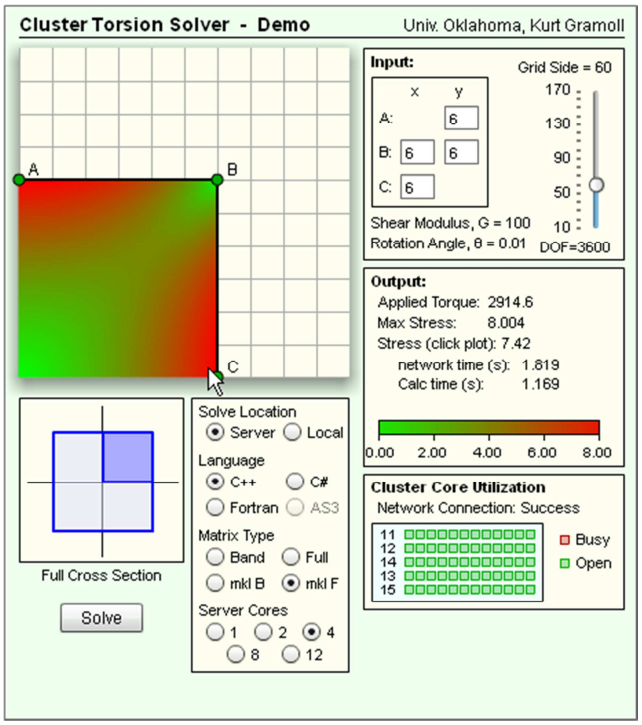

Figure 7. Torsion Solver Test Case (12 by 12 Square Cross Section)

The Torsion Solver uses the finite element method to solve the problem. Only a basic triangular element type is implemented for this example. The development and programming for this type of element can found in most finite element textbooks [13]. The core equation that is solved is Poisson equation of the form,

$$\nabla^2\phi = -2G\theta \tag{1}$$

where ϕ is the stress function, G is the shear modulus, and θ is the angle of rotation. The two shear stress terms can be found from

$$\tau_{xz} = \frac{\partial\phi}{\partial y} \qquad \tau_{yz} = -\frac{\partial\phi}{\partial x} \tag{2}$$

The boundary conditions require ϕ to be zero at all free edges.

The Torsion Solver allows students to solve any non-uniform cross section by moving three control points or typing in numerical values for the control points. The final cross section boundaries are straight lines between these control points. To minimize the number of elements needed, it is assumed that the cross is symmetrical in both directions. The shear modulus and rotation angle are set. No units are used, and it is assumed students will scale the results to match the shear modulus, input torque, and rotation angle, as needed.

The user can also control the number of grid points on a given edge. This is directly related to the total degree of freedom for the problem. When a high grid density is used, the Flex graphics can become sluggish.

The current version of the solver also allows the user to specify where it is solved (locally or on the cluster server), what language is used, the type of matrix solver, and how many cores are used (only available when using C++ and MKL routines). In the future, these choices will be removed, and only the most efficient solver routine will be available to the user.

There is also a graphic that shows how many cores are currently available on the server. This is not critical for the solution process, but it is interesting to know the current status of the server cluster.

To validate the Torsion Solver, a square cross bar was analyzed. The dimensions are 12 by 12 with a shear modulus of 100 and a rotation angle of 0.01 radians. Units are assumed to be consistent, such as inch for length and psi for shear modulus. Using the closed form solution [12], the maximum shear stress is given as

$$\tau_{max} = 1.350G\theta a \tag{3}$$

where a is half the height or width for a square cross section. Substituting the example parameters give the maximum shear stress as 8.10 (psi if inch and psi are used for dimensions and modulus, respectively). The solver gives the maximum stress as 8.004 for a 3,600 degree of freedom (DOF) system. However as the DOF increases, the stress slowly converges to 8.1. A sample result is given in Fig. 7. Another check is the total torque needed to cause a 0.01 angle of rotation. The closed form solution gives this as 2,915 (lb-in in this example), which is almost exactly what the Torsion Solver gives.

The actual total solution times for the Torsion solver, using the full matrix method, shows dramatic increase after only 500 DOF. For even moderate size problems (5,000-10,000 DOF), solving locally inside any client (Flex, HTML5, Flash, Java) is not feasible. Furthermore, the results shown in Fig. 8 were done on a moderately fast laptop. They are even slower on smart phones or tablets.

Note, if the banded solution method is used, the results are not as dramatic. In fact, this particular example (rectangular grid) has an extremely low bandwidth which is generally unrealistic. Thus, banded solution times would actually be acceptable for the client up to 10,000 DOF.

It is interesting to note, that for low DOF (Fig. 8a.), local solution times are actually faster. This is due to the time overhead of contacting the server, pushing data to the server, and then returning a result file back to the client. Generally, this takes 1-3 seconds, depending on the file sizes (a function of DOF).

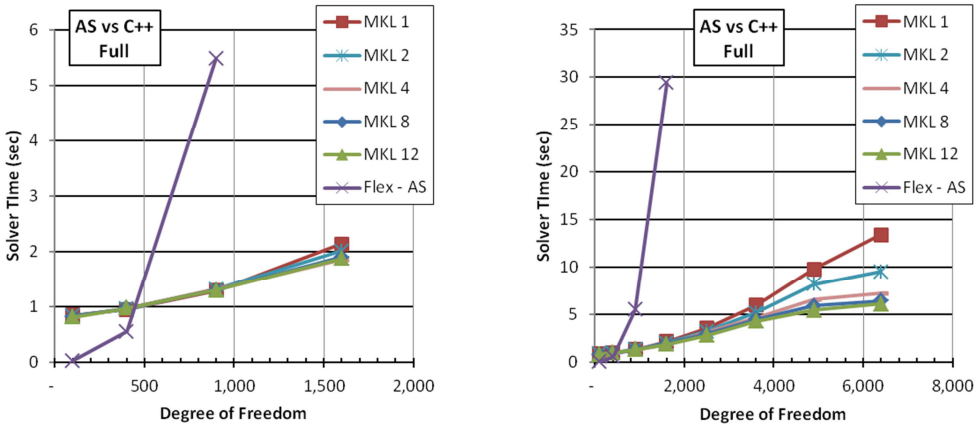

Figure 8. Torsion Solver Total Times (Network and Solution Times)
a) Low DOF, b) High DOF

An additional example of using a cluster server to solve complex problems was recently completed for three dimensional stress analyses [14]. In that case, the bandwidth is 10-25% the size of the full matrix, and then banded solutions are also unmanageable on the local client and the server cluster needs to be used.

Conclusion

This paper has demonstrated the benefits of using cluster server technology to solve complex engineering problems for engineering education. The capabilities and power of multi-core processing can and should be applied to assist students' visual learning and design advanced systems in undergraduate courses. The paper also layouts how others can implement a cluster and link it to a web page for ease of access and availability. This and other simulations can be accessed through the eBooks at www.eCourses.ou.edu.

Bibliography

1. "Galaxy Nexus", en.wikipedia.org/wiki/Galaxy_Nexus, retrieved on 12 Jan 2012.
2. "PuTTY", en.wikipedia.org/wiki/PuTTY, retrieved on 12 Jan 2012.
3. "RTMP", en.wikipedia.org/wiki/Real_Time_Messaging_Protocol, retrieved on 12 Jan 2012.
4. Gramoll, K. *Multimedia Engineering Mechanics of Materials*, retrieved from ecourses.ou.edu on 12 Jan 2012.
5. "Adobe Flash Platform", www.adobe.com/flashplatform/, retrieved on 12 Jan 2012.
6. "WebORB for .NET", www.themidnightcoders.com/products/weborb-for-net/, retrieved on 12 Jan 2012.
7. "TOP500 List - June 1997 (1-100)", www.top500.org/list/1997/06/100, retrieved on 12 Jan 2012.
8. "Windows HPC Server 2008 Getting Started Guide", technet.microsoft.com/en-us/library/cc793950%28WS.10%29.aspx, retrieved on 12 Jan 2012.
9. AlRamahi, Mohammad and Kurt Gramoll, “Online Collaborative Drawing Board for Real-time Student-Instructor Interaction and Lecture Creation", 2004 ASEE Annual Conference, Salt Lake City, UT, June 20-23, 2004.
10. Weaver, W., and J.M. Gere, (1990). "Matrix Analysis of Framed Structures", Kluwer Academic Publishers, 1990.
11. "Math Kernel Library (MKL)", software.intel.com/en-us/articles/intel-mkl/, retrieved on 12 Jan 2012.
12. Timoshenko, S.P. and J.N. Goodier, "Theory of Elasticity", 3rd Edition, McGraw-Hill, 1970.
13. Reddy, J.N., "An Introduction to the Finite Element Method", McGraw-Hill, 1984.
14. Vick, Zachary and Kurt Gramoll, "Online, Interactive, 3D Finite Element Stress Analysis Using High Performance Computing (HPC) Cluster", ASEE Annual Conf. Proc., San Antonio, TX, 10-13 Jun 2012
15. "Microsoft Visual Studio", en.wikipedia.org/wiki/Microsoft_Visual_Studio, retrieved on 23 Feb 2012.
16. "Microsoft Silverlight", en.wikipedia.org/wiki/Silverlight, retrieved on 23 Feb 2012.