DEVELOPMENT OF THE PICMSS NETSOLVE SERVICE

A Report
Presented for the
Master of Science
Degree
The University of Tennessee, Knoxville

D. Matthew Kelleher Jr.
April 2002

Innovative Computing Laboratory Technical Report
ICL-UT-02-04

ABSTRACT

This report describes the installation of the PICMSS (Parallel Interoperable Computational Mechanics System Simulator) computational fluid dynamics (CFD) program as a NetSolve service in NetSolve version 1.4. The PICMSS system uses state-of-the-art techniques for setting up and performing CFD calculations. It consists of a front-end program that runs on user systems and prepares program input, a compute engine that runs on parallel systems and calculates results, and post-processing programs that run on user systems and analyze and present results. The PICMSS compute engine was installed as a NetSolve service on three parallel systems and was tested on four user systems. NetSolve provides the infrastructure required to run the compute engine on the parallel systems and transfer program input and output files between user systems and parallel systems. The results of this project show that using NetSolve to run parallel programs is an effective and productive alternative to directly executing parallel programs on a parallel system.

TABLE OF CONTENTS

1. INTRODUCTION

The University of Tennessee Computational Fluid Dynamics Laboratory (CFDL) and Joint Institute for Computer Science (JICS) are collaborating to develop a new computer software system for performing computational fluid dynamics (CFD) calculations called Parallel Interoperable Computational Mechanics System Simulator (PICMSS) [Wong 2000, Wong 2001]. This CFD code uses state-of-the-art techniques for setting up, performing, and analyzing CFD calculations. PICMSS consists of a front-end program that prepares program input, a compute engine that calculates results, and post-processing programs that analyze and present results. The front-end program uses a graphical user interface and has tools that allow a user to completely specify the problem to be solved. The compute engine uses parallel processing techniques to quickly perform the required calculations. Post-processing programs, for example, convert calculated results to forms that can be used as input to plotting packages.

The PICMSS compute engine is a versatile implicit three-dimensional finite element program designed to solve a wide range of fluid dynamics problems [Wong 1995]. It is designed to run in parallel on a wide range of platforms ranging from single workstations to workstation clusters to large supercomputers. The program is written in C, uses the Message Passing Interface [MPI] protocol to communicate between processes, and uses the Aztec [Aztec] library to perform its linear algebra calculations. The program reads data from fourteen input files and writes its results in a set of output files. The first parallel process reads the input files and distributes the data to the other processes. Each process works with other processes to perform the required calculations and all processes write results separately into output files.

PICMSS is designed to allow front-end and post-processing programs to run on a user's local system and for the compute engine to run on a system built and managed to run parallel programs. This requires development or use of a system that transfers compute engine input files from the user system to the parallel system, runs the compute engine on the parallel system, and delivers output files back to the user system. PICMSS developers determined that NetSolve [Arnold 2001] developed at the University of Tennessee, Knoxville can run the PICMSS compute engine and provide the required file transfer services.

NetSolve was designed to provide high-performance network-based computational services to scientific users. It consists of clients that use NetSolve services, servers accessible across a network that provide services, and agents that connect clients to servers. A NetSolve client program is any program that uses a NetSolve service. It typically runs on a user's local system. Servers run on high-performance networked systems and provide a wide range of computing services such as solving dense and sparse systems of linear equations, parameter optimization, and data sorting. Agents run on

central network systems and use the available servers, the services they provide, their relative performance, and current workload to identify the server best able to provide a service to a client.

The NetSolve client interface is implemented as a set of library routines called from a user's program. To add a NetSolve service to a program, the user adds a call to a NetSolve interface routine and adds the NetSolve client library to the program. The call specifies the name of the service to be performed, pointers to the data required by the service, and pointers to locations where the results will be stored. When the program runs, it calls the NetSolve interface routine in the client library. The NetSolve client library contacts an agent and tells it the name of the service it needs. The agent identifies the server best able to provide that service and returns its network address to the client library. The client library contacts the specified server and gives it the name of the service and the required data. The server performs the service and returns the results to the client library. The client library stores the results in the specified location then returns to the calling program. The program then uses the results.

The NetSolve client and server can also transfer files between client and server systems. This feature is required for the PICMSS compute engine because it gets all its data from files and writes all its results in files.

The PICMSS service was added to NetSolve to meet the needs of the Virtual Human project at Oak Ridge National Laboratory (ORNL). Virtual Human was an ORNL Laboratory-Directed Research and Development (LDRD) project to bring cutting-edge computational techniques to modeling complex biological systems [Ward 2001]. These models require capabilities not available in commercial CFD packages. Virtual Human researchers identified PICMSS as a good package for performing these calculations because it has the flexibility and power required to model complex biological systems and the usability required to efficiently prepare and perform calculations and analyze the results. However, before the project could use PICMSS effectively, the system had to be installed as a NetSolve service.

This report describes the installation of the PICMSS compute engine as a NetSolve service. It lists the service requirements, describes how these requirements were satisfied, and shows that the PICMSS NetSolve service is an efficient and easy-to-use system for performing CFD calculations.

This report assumes the reader is familiar with NetSolve and how the client, agent, and server work together to provide computational services to the client program. A user should be familiar with the material in the NetSolve 1.4 Users Manual [Arnold 2001] before using the material in this report to develop a similar service.

2. PICMSS SERVICE REQUIREMENTS

The PICMSS compute engine was designed to run as a stand-alone program on a parallel system. To run the program by itself, a user first prepares a set of input files that describes the problem. If these files cannot accessed by PICMSS on the parallel system, the user must move them to the parallel system. The user then logs into the parallel system, runs PICMSS, and waits for the results. After the program stops, the user either analyzes the results on the parallel system or moves results files to another system for later analysis.

The overall requirement of the PICMSS NetSolve service is that it must be at least as easy to use as the stand-alone version. It must also provide a service comparable to or better than the service now provided by the compute engine running on a parallel processing system. The nine requirements listed in this section ensured that the PICMSS NetSolve service met this general requirement.

1. The PICMSS service will be provided by a client that runs on the user's machine and a service associated with a NetSolve server. The client will be a standard NetSolve client that uses the PICMSS service by calling NetSolve routines in the NetSolve client library. The service will be a standard NetSolve service and will consist of all the software required to run the PICMSS compute engine under the control of a NetSolve server. The service software will be as similar as possible to existing NetSolve services.

This requirement ensured that the PICMSS service is a standard NetSolve service. Users familiar with NetSolve will know what to expect and new users can learn to understand and use the PICMSS service from NetSolve documentation and examples. This requirement also helps ensure that PICMSS can be ported to new NetSolve releases.

2. The client must be a stand-alone program that does nothing but provide the PICMSS service.

The PICMSS front-end software is currently being developed so it cannot now serve as the client program. Until this program is ready for use, the PICMSS compute engine must be run under NetSolve the same way it is run as a stand-alone program. This requirement allows the user to develop the input files by hand on a local system, run the program as a NetSolve service, and get the results files back on the local system. When the front-end software is completed, this stand-alone client can be integrated into it.

3. The client must run on every system for which NetSolve client libraries are available.

NetSolve client libraries are currently available for common UNIX systems and Windows family systems. The client must run on these systems.

4. The PICMSS service must be executed from a shell script that is run by NetSolve and supplied by the user.

The PICMSS compute engine is still under development so versions can be expected to be created frequently and multiple versions can coexist. This allows users to specify the versions of the PICMSS compute engine to use when they request PICMSS services. It also makes the PICMSS service different from other NetSolve services. However, the code used to execute programs and shell scripts is similar, so the shell script can easily be replaced by direct program execution similar to other NetSolve parallel services when development ends and a production version is created.

A second reason to use a shell script is to allow a user to specify a file containing a list of the processors to use during a run. Some operating systems such as AIX on the ORNL Eagle system automatically allocate available nodes to a submitted job. However, NetSolve and MPI do not provide this service. A user on systems that just use MPI (such as the SInRG systems) must tell MPI which nodes to use for a run. If the job specifies an unavailable node, the job fails. This allows a user to include a file that tells the PICMSS service which nodes to use for a run.

5. The client must get the number of processors required from an input file and supply that number to the server.

This number is needed on the server side because number of processors required for a calculation depends on the problem being solved. It allows the user to use the same shell script for all calculations and makes it easier to eventually run the program directly from the PICMSS service program. The number is already present in the PICMSS compute engine input file inpara.indat so no input file changes are required.

6. Client and service must work together to move all PICMSS input and output files between the client and server. The client must get all input data files required by the PICMSS compute engine from the user's current directory and transmit them to the PICMSS service. The service must receive the input files sent by the client and put them in the PICMSS compute engine default directory. The PICMSS service must get all output files produced by PICMSS compute engine and send them to the client. The client must receive the output files produced by the PICMSS compute engine service and put them in the user's current directory. Transmitted input and output files must be written in the destination directory using end-of-line characters standard for text files on the destination machine.

Program users create and the PICMSS compute engine program expects to read and write standard system text files. The client and service must ensure that the transmitted files appear to all users as standard text files.

7. The client must transmit all required input files to the service and transmit optional files only if present.

The PICMSS compute engine has fourteen required and one optional input file. The client must fail if a required file is not present. It must transmit the optional file only if present.

8. The PICMSS compute engine service must direct stdout and stderr output to a file and deliver it to the client.

The NetSolve service captures output written by its services to stdout and stderr and delivers them to the client. NetSolve client library routines automatically write them on the clients' stdout and stderr units. This requirement ensures that they are delivered instead as files that can be later examined by the user.

9. The PICMSS compute engine cannot be changed.

This requirement helps ensure that new versions of the program will work with the NetSolve environment developed by this project.

3. PICMSS SERVICE SOFTWARE

The PICMSS NetSolve service is implemented as a stand-alone client program that runs on the user's machine, a PICMSS service integrated with NetSolve server software, and the combined file module that creates the combined file used by both client and service software to contain the files transmitted between client and server.

The client, service, and combined file module are written in as much as possible in standard C and use commonly available extensions to provide system services not standardized in the C language. The client and combined file module have been compiled and run successfully on IBM AIX, Sun Solaris, Linux, and Microsoft Windows systems. The service has been compiled and run successfully on IBM AIX, Sun Solaris, and Linux systems.

COMBINED FILE MODULE

The major technical challenge in creating the PICMSS NetSolve service was in managing the program input and output files. The PICMSS compute engine reads fourteen required and one optional input files and the PICMSS service has one required and one optional input file. Each parallel process produces its own set of output files. As a result, the client must transmit to the service fourteen to seventeen input files and expects back many more output files.

NetSolve can transfer files from the client to the server and from the server to the client. This capability is used by NetSolve, for example, to get routines that calculate objective function values required by optimiztion services from the user. The client provides the name of the file containing the objective function calculation in the NetSolve parameter list. The NetSolve client and server work together to send the file to the server where it is compiled and linked with the parameter optimization service. The service with the user-provided function work together to calculate the optimum value which is returned by NetSolve to the client.

In NetSolve, the fact that files will be transmitted from client to server and from server to client when a service is executed is specified when the service is defined. The names of the files to be transferred are parameters in the NetSolve interface routine call. Although predicting the number and names of the files to be transferred for a PICMSS calculation is possible, the resulting parameter list would be very long and have different lengths for each problem solved. This simply is not practical.

The solution was to have the client combine all input files into a single combined input file and send that file from the client to the server. The PICMSS service unpacks the combined file and writes the input files in the default directory used by the PICMSS

compute engine. After the compute engine finishes, the PICMSS service combines all output files produced by the engine into a single combined output file and sends that combined file to the client where it is unpacked to reproduce the original output files. This way only two files are transferred so only two file names are required in the parameter list for every PICMSS service call.

The combined file module has routines that open a new combined file, add files to the combined file, close the combined file, and extract all files from a combined file. To send input files to the service, the client creates a new combined file, adds the required and optional (if present) input files to the combined file, then closes the combined file. When it calls NetSolve, it puts in the parameter list the name of the combined file that contains all input files and the name of the combined file that will contain all output files. The server works with the client to transfer the input combined file to the server. The server starts the PICMSS service which uses the combined file module extraction routine to extract all files from the input combined file. The service executes the PICMSS compute engine which creates a set of output files all ending with the extension ".outdat". After the engine is finished, the PICMSS service creates a new combined file, gives it the output file name specified in the client parameter list, and adds to it all files with names ending in .outdat. When the service is finished NetSolve sends that file back to the client where it is unpacked.

The combined file module also handles all text file end-of-line character conversions. UNIX systems mark the end of a line in a text file using a line feed character and Windows-based systems use a carriage return character followed by a line feed character. To ensure that end-of-line characters are consistently and correctly converted, the module removes all end-of-line characters and records their locations when files are added to the combined file. When the file is extracted, the module adds the end-of-line characters for the local system in their correct locations. Removing all end-of-line characters from the file also ensures that lines are not modified by format conversion transformations that may be applied to the file in transit between client and server.

PICMSS FILES

Two new input files were added to the set of files already required by the PICMSS compute engine. File picmss.sh contains the commands required to execute the PICMSS compute engine on all parallel systems the service is installed on. This shell script must contain the mpirun command (for MPI systems) or the equivalent command on other systems that runs parallel programs. NetSolve environment variable NETSOLVE_ARCH can be used to make sure the correct command is executed. The script can also contain any other programs or shell script commands. This script takes one parameter, the number of processors required.

The second file is named hostfile and contains the names of the processors to use for MPI systems and other systems where the user must specify the parallel processors to use. This file is optional so it only needs to be provided where required.

PICMSS CLIENT

The client is a simple stand-alone program that runs on the user's machine. The client first reads the number of processors required from file inpara.indat. It then creates the input combined file and adds all of the input files to be sent to the server to it. If a required file is missing, it reports the missing file and stops. The client then calls NetSolve with these parameters: PICMSS service name, input combined file name, output combined file name, and number of processors required. After the NetSolve service finishes, the client splits the output combined file returned by NetSolve into individual output files then stops.

PICMSS SERVICE

The NetSolve server calls the PICMSS service module to run the PICMSS compute engine and provides it these three parameters: input combined file name, output combined file name, and number of processors required. The service extracts the input files from the input combined file then calls the C system() routine to execute shell script picmss.sh. The system call gives execute permission to file picmss.sh, executes it using the number of processors as its single parameter, and redirects its standard output to stdout.outdat and its standard error output to stderr.outdat. After shell script execution ends, the PICMSS service creates the output combined file, adds all files ending in .outdat to it, then returns to NetSolve.

COMBINED OPERATION

The client program, NetSolve, and the PICMSS NetSolve service work together to run the PICMSS compute engine and deliver its results to the client. The client program assembles the input files into the combined input file then calls NetSolve. The NetSolve client contacts the agent and gets from it the address of the NetSolve server it determines is best able to provide the PICMSS service to the client. The NetSolve client contacts the server, tells it that it wants to execute the PICMSS service, and gives it the input and output combined file names and number of processors required. The server creates a temporary directory for the files and the NetSolve client and server work together to transfer the input combined file to that directory. The NetSolve server calls the PICMSS service to perform the calculations. The PICMSS service splits the input combined file, executes picmss.sh, and adds all output files into the output combined file. It returns to the NetSolve server which works with the client to transfer the output combined file to the client's directory. The client NetSolve call returns and the client extracts the results files from the output combined file.

4. PICMSS SERVICE EVALUATION RESULTS

The PICMSS client and NetSolve service using NetSolve version 1.4 were installed and tested on the ORNL Center for Computational Sciences Eagle and Bearcat systems and on the Neo and Cypher clusters, part of the SInRG (Scalable Intracampus Research Grid) network at the University of Tennessee Knoxville. Eagle is an IBM RS6000-based parallel processing system running the IBM AIX operating system. Bearcat is a dual-processor RS6000 system running the AIX operating system that operates as a single serial processor system. The PICMSS client and NetSolve agent runs on the Bearcat system and the NetSolve server with PICMSS service runs on the Eagle system.

Neo is a network of sixteen Sun SPARC workstations running Solaris 7 and Cypher is a network of sixteen Dell systems with Intel Pentium processors and the Linux operating system. The PICMSS client, NetSolve agent, and NetSolve server with the PICMSS service were installed on Neo system neo1 and on Cypher system cypher01. The client was also installed and tested on a Pentium Pro system running Windows 98 and used the agents and servers running on the Neo and Cypher systems.

TEST CASES

The tests were performed using three test cases that each calculate the flow through a duct. These test cases have analytical solutions and were originally used by the code author to validate the PICMSS compute engine. These cases were used because the data files were available and the results known to be correct. Case 2DCASE-2 and 2DCASE-3 are two-dimensional calculations with 2DCASE-2 representing a 5x5 structured grid and 2DCASE-3 representing an unstructured grid. Case 3DCASE-0 is a three-dimensional calculation using a 5x5x5 grid.

ACCURACY TESTS

A set of test cases was used on each system to verify that the PICMSS service produced the same results as the PICMSS compute engine run separately. Four test cases were used on the Bearcat and Eagle systems and three used on the Neo and Cypher systems. All test cases were provided by the PICMSS developers.

Each test was conducted following the same procedure. First, the PICMSS compute engine executable file was created for each of the three systems using a makefile and source code provided by the developers. This executable was used for both the stand-alone program runs and for the NetSolve PICMSS service.

The stand-alone results were produced by running each of the test cases using the PICMSS executable file and the test case input files. PICMSS was run on the Neo and

Cypher machines using a makefile provided by the developers and on the Bearcat and Eagle systems using a shell script provided by the developers. The output written by the PICMSS compute engine to stdout and stderr was saved in files stdout.outdat and stderr.outdat so they could be compared to the stdout.outdat and stderr.outdat files produced by the NetSolve PICMSS service. The output for each case was saved in a separate directory.

The input files were copied to a separate directory on the same system and each case was run again using the PICMSS client and NetSolve PICMSS service. For the ORNL systems the client and NetSolve agent ran on the Bearcat system and the NetSolve server with PICMSS service ran on system eagle163. For the Neo tests the client, NetSolve agent, and NetSolve server with PICMSS service ran on system neo1. Likewise, for the Cypher tests, the client, NetSolve agent, and NetSolve server with PICMSS service ran on system cypher01. Shell script picmss.sh executed the same PICMSS executable file used to create the stand-alone results for that system.

The UNIX diff command was used to compare each output file produced by the NetSolve PICMSS service with the same file produced by the PICMSS compute engine run as a stand-alone program. All output files were compared including the stdout.outdat and stderr.outdat files. The only differences in the files were in lines reporting timing results. All reported calculation values were identical.

The Neo system test cases were copied to the Bearcat system and the Neo tests repeated to confirm that the results did not change when the client and server ran on two different systems. The output produced by PICMSS running as a stand-alone program and the output produced by the PICMSS service were compared. The results were identical except for lines reporting timing results.

PERFORMANCE TESTS

The Neo test cases on the Bearcat system were also used to measure the additional time required to run the NetSolve PICMSS service compared to the time required to run the PICMSS compute engine as a stand-alone program. A terminal session was established with system neo1 and the NetSolve agent and NetSolve server with the PICMSS service were started on that system. A performance test directory was set up that contained copies of the test cases and execution scripts used to create the stand-alone output files. This directory was created to allow the PICMSS compute engine to be run repeatedly and to create test case output files without affecting the stand-alone output files used to confirm the accuracy of the NetSolve calculations.

A separate terminal session was established with the Bearcat system. A performance directory was created on this system to contain copies of the files used to test the NetSolve PICMSS service on the Neo system. This directory was created to allow the

NetSolve service to be run repeatedly without affecting the files used to confirm the accuracy of the NetSolve calculations.

Together these two terminal sessions were used to compare the times required to execute PICMSS as a stand-alone program and as a NetSolve service on the Neo system. The three Neo test cases were used as the test data. The stand-alone version and the NetSolve service were each run five times for each test case and the elapsed times recorded. The runs were interleaved to reduce the effects of other activities present in the systems. The NetSolve service was started as soon as the stand-alone version finished and the stand-alone version was started as soon as the NetSolve service finished. The times were collected by the Korn shell time command. The output produced by each run was also checked to verify that the calculated results were correct.

The times recorded for each of the test cases are listed in table 4.1. These results show that for these short runs NetSolve added a significant amount of time to the length of the run. They also show that the time required by NetSolve is proportional to the length of the run. These are the only test cases provided by the developer so longer runs could not be examined.

Table 4.1
Run Time Added by NetSolve

Case	Batch Time, s	NetSolve Time, s	Difference, s
2DCASE-2	51	76	25
2DCASE-3	49	72	23
3DCASE-0	16	23	7

5. CONCLUSIONS

These results show that the NetSolve PICMSS service is an effective and productive alternative to directly executing the PICMSS compute engine on a parallel system. The calculation results were identical but the wall-clock times were higher. However, with the NetSolve service there is no need to log into the parallel system. Program files can be created, the PICMSS compute engine program run, and results analyzed directly on a user's home machine. This allows the user to use the user's own editors, plot packages, and other analysis tools. It also allows the program that generates input files to be designed to run on user-oriented systems rather than on parallel systems better suited for computational tasks. This convenience factor far outweighs the cost of the added time required to perform the calculations.

This project also shows that NetSolve can be an effective platform for delivering general parallel processing services to users. Many parallel applications operate the same way as the PICMSS compute engine: They read input files, perform calculations, and write output files. In fact, the PICMSS client and service programs can be modified to run almost any parallel program even though they were designed specifically to run the PICMSS compute engine. Of course, it wouldn't be simple. The input files needed by the program would have to be renamed to the names used for the PICMSS compute engine then renamed to their original names by the shell script. The shell script would have to run the other program and would have to give all its output files the .outdat extension. The inpara.indat file would have to specify the number of processors needed. Yet the fact that it would work shows how simple adapting this software to run other parallel applications would be.

6. FUTURE WORK

Extending this work to other applications that use parallel computations is relatively simple. The PICMSS client and service can already be used without change for any parallel application as described in the conclusions. The PICMSS client can be changed into a general purpose parallel program client by giving it a list of input files to add to the input combined file and by giving it the number of processors required. The PICMSS service can be changed into a general purpose parallel processing service by giving it a list of the output files to return in the output combined file. The parallel program would continue to be specified in the shell script sent from the client to the server. If these actions are taken, a NetSolve server with this service installed could run any non-interactive parallel program installed on that system for users on the network. This new service would be a change from the services NetSolve currently provides but it would provide a valuable new service to parallel program users.

If NetSolve will continue to be used to provide parallel processing services, the calculations used to measure server workloads should be redesigned to measure workloads on parallel systems. The current load-balancing algorithms used by NetSolve agents to identify the server best able to perform a calculation for a user assume that the service will be provided by a serial processor. If a server machine is lightly loaded but other machines in the cluster are heavily loaded, an agent may assign the job to the server. As a result, this job may take much longer than it would have if assigned to another server.

For those MPI systems where the user must specify the nodes to be used for a calculation, the server should be able to tell the service which nodes to use. NetSolve now uses a fixed list of available nodes. If one of these nodes is down and a parallel program uses this list to get available nodes, the job will fail when it attempts to use the unavailable node. Currently this list must be manually maintained by a system administrator. It would be much better if the NetSolve server could monitor cluster performance and use the information to dynamically maintain the MPI host list. When a parallel service starts, the server could provide it with a list of nodes to use sorted by workload. This would allow the service to use the nodes with the lightest workload to perform its calculations.

LIST OF REFERENCES

LIST OF REFERENCES

[Arnold 2001]
Dorian Arnold, S. Agrawal, S. Blackford, J. Dongarra, M. Miller, K. Sagi, Z. Shi, S. Vadhiyar, *Users' Guide to NetSolve V1.4*, Computer Science Dept. Technical Report CS-01-467, University of Tennessee, Knoxville, July 2001.

[Aztec]
R. S. Tuminaro, et. al., *Official Aztec User's Guide: Version 2.1*, Sandia National Laboratories, December, 1999.

[LAPACK 1999]
SIAM, *LAPACK Users' Guide, Third Edition*, 0-89871-447-8, E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A.McKenney, and D. Sorensen.

[MPI]
William Gropp and Eugene Lusk, *Users Guide for mpich, a portable implementation of MPI Version 1.2.2*, Report ANL/MCS-TM-ANL-96/6 Rev. D, Argonne National Laboratory, undated.

[Ward 2001]
R. C. Ward, et. al., *Integrated Respiratory System Model for the Virtual Human*, Oak Ridge National Laboratory, LDRD Final Report, November 2001.

[Wong 1995]
Kwai L. Wong, *A Parallel Finite Element Algorithm for 3D Incompressible Flow in Velocity-Vorticity Form*, Ph. D. Dissertation, University of Tennessee, Knoxville, 1995.

[Wong 2000]
Kwai L. Wong and A. J. Baker, Computer-Based Simulation Maturation in Y2K, available from www.picmss.org, (January 21, 2002).

[Wong 2001]
Kwai L. Wong, *A Parallel Interoperable Computational Mechanics System Simulator(PICMSS)*, February 2, 2001, http://www.picmss.org (January 21, 2002)

APPENDIX

A. INSTALLING AND USING NETSOLVE 1.4 PICMSS SERVICE ON SINRG NEO AND CYPHER CLUSTERS

NetSolve version 1.4 and the PICMSS service were installed and tested on the SInRG Neo and Cypher clusters and on the ORNL Bearcat and Eagle systems. The procedures used for the Neo and Cypher clusters are described in this appendix; the procedures used on Bearcat and Eagle are similar but not described further as these systems are not likely to be available much longer. The makefiles used to create them are included in the PICMSS directories on the SInRG system.

INSTALL NETSOLVE 1.4

NetSolve version 1.4 is installed by downloading the tar file from the NetSolve web site and unpacking it in the directory that will be the NetSolve root directory. The README and INSTALL files unloaded from the tar file describe the procedures that must be followed from that point forward. These procedures require the installer to run the configure script then run the makefile produced by the configure script. NetSolve is ready for use after the makefile ends.

On Neo the following directory must be permanently added to the PATH variable before the configure script is run:

```
/usr/local/SUNWspro/bin
```

This file is required to find the Fortran compiler. The current directory must also be in the PATH variable for the NetSolve configure script to work. Once configure has been run the current directory can be removed from the PATH variable. No path changes are required on Cypher.

On Cypher the makefile produced by configure contains a flaw that causes the make to fail. The makefile reports it cannot find /usr/local/bin/ar which is supposed to be the UNIX archive program. Change variable AR which specifies the archive program location in $NETSOLVE_ROOT/conf/Makefile.i686_pc_linux_gnu.inc to correct directory /usr/bin/ar then repeat the make command to complete the installation.

To test the NetSolve installation, set the NETSOLVE_AGENT environment variable to point to the system, edit the server_config file to tell the server to use the local system as the NetSolve agent, start the agent then the server, run the Test script in the bin/sparc_sun_solaris2_7 or bin/i686_pc_linux_gnu directory, and follow the instructions written by the script. If all tests return success, NetSolve is working correctly.

CREATE PICMSS FILES

The PICMSS files are stored in a directory structure with the PICMSS directory at the top and directories CFMODULE, CLIENT, SERVER, and TEST at the next level. CFMODULE contains the combined file module, CLIENT contains the files required to create the PICMSS client, and SERVER contains the files required to create the PICMSS service. The level below these directories identifies the program version number, currently V01.00.001. The files required to create the programs are in the version directories. Below the version directories are directories used to test the program and to store Neo and Cypher executable files.

The combined file module must be created before the client or service can be created. To create this module, change to the PICMSS/CFMODULE/V01.00.001 directory and run make. This program creates the CFModule object file required by the other two programs and programs combfile and splitfile that can be used to test the program. Program combfile is run in directory COMBTEST and creates a combined file. Program splitfile is run in directory SPLITTEST and splits a combined file back into its individual files. A script in SPLITTEST can be used to confirm that the files split from the combined file are identical to the original files in COMBTEST.

The client is created next from the source files in directory PICMSS/CLIENT/-V01.00.001. The makefiles used to create the client on Neo and Cypher are different, so first copy makefile.neo or makefile.cypher to makefile.sys. If the NETSOLVE_ARCH variable is not defined, it must be defined to be the name of the architecture subdirectory of the $NETSOLVE_ROOT/lib directory. For Neo NETSOLVE_ARCH must be sparc_sun_solaris2_7 and for Cypher it must be i686_pc_linux_gnu. Run "make" by itself to create the picmss client program and "make picmsstest" to create a program that can be used to test the picmss client program. To test the client program, change to the TEST subdirectory and execute picmsstest. This program creates a combined input file, simulates a NetSolve execution, and splits an output combined file back into individual files.

The server software is created from the files in PICMSS/SERVER/V01.00.001. The makefile in this directory creates archive file libPICMSS.a and program picmsstest. To test the service, copy the input combined file created by the client picmsstest program to the TEST directory and run picmsstest in the TEST directory. This program splits an input file into separate files, runs the picmss.sh shell script, then copies all the *.outdat files to the output combined file. The difffile script in the TEST directory can be used to confirm that the input files are the same as the files in PICMSS/CLIENT/V01.00.001/-TEST. The output combined file can be copied to the client program test directory and picmsstest run again in that directory. Shell script difffile in that directory can be used to verify that the output files are the same.

ADD PICMSS SERVICE TO NETSOLVE

PICMSS is installed as a customized service in NetSolve similar to other NetSolve services that perform parallel processing. All parallel processing services in NetSolve are installed as customized services because, due to MPI limitations, these services cannot use a directory descended from the system /tmp directory. These three changes must be made to NetSolve file generateservice.c to add PICMSS as a customized service:

1. Add PICMSS to the list of services that use a working directory descended from a permanent directory instead of a directory descended from /tmp. All current NetSolve services that perform parallel processing use directories not descended from /tmp, so just add PICMSS to this list of services.
2. Change the temporary directory used by PICMSS and the other parallel services to a directory descended from $NETSOLVE_ROOT/tmp. The default is to place these files in a directory descended from $NETSOLVE/src/Server which is an inappropriate location.
3. Add PICMSS as a customized service that calls the standard file service. File generateservice.c was written assuming each customized service would have a special service routine.

Now execute this procedure to add PICMSS to NetSolve version 1.4:

1. Define $NETSOLVE_ROOT to point to the NetSolve root directory.
2. Define environment variable NETSOLVE_ARCH to be the name of the directory below the $NETSOLVE_ROOT/bin that contains the executable files for this architecture. For the Neo systems NETSOLVE_ARCH must be sparc_sun_solaris2_7 and for the Cypher systems it must be i686_pc_linux_gnu.
3. Copy NetSolve problem definition file picmss from PICMSS/SERVER/-V01.00.001 to $NETSOLVE_ROOT/problems.
4. Add problem ./problems/PICMSS to $NETSOLVE_ROOT/server_config.
5. Copy PICMSS library routine libPICMSS.a in PICMSS/SERVER/V01.00.001 to $NETSOLVE_ROOT/lib/$NETSOLVE_ARCH.
6. Execute make server in the $NETSOLVE_ROOT directory.
7. When the make fails because target libPICMSS.a is not found or reports permission denied, copy $NETSOLVE_ROOT/Makefile.numerical to src/Makefile.numerical. This has to be done to get around a bug in NetSolve.
8. Create a NETSOLVE_ROOT/tmp directory to contain the PICMSS input and output files while the service is running.

When the make completes, the PICMSS service is ready for use. The tmp directory in the $NETSOLVE_ROOT directory is used for temporary directories that store PICMSS files while the PICMSS compute engine is running. These directories are created by NetSolve when the PICMSS service starts and are supposed to be deleted by NetSolve when the

service ends. However, on Neo and Cypher the empty directories remain after the NetSolve service ends and must be manually deleted using the rmdir * command. They are deleted correctly on IBM AIX systems, so this may be an artifact of the network file service used on these systems. The location of this directory is different from the standard NetSolve 1.4 distribution which places these files in the $NETSOLVE_ROOT/-src/Server directory, an inappropriate location.

The PICMSS/TEST directory contains three test cases that can be used to verify that the NetSolve service produces the same results as the PICMSS compute engine run by itself. The TEST directory is divided into a BATCH directory containing the results from executing the PICMSS compute engine as a stand-alone program and the NETSOLVE directory that contains the results from executing the NetSolve service. Each of these two directories contains directories 2DCASE-2, 2DCASE-3, and 3DCASE-0 for the three test cases provided by PICMSS developers.

The BATCH output files were created by executing a makefile provided by the developers in the case file directories. The “make mprun5" command was used for the 5-processor cases (2DCASE-2 and 3DCASE-0) and the “make mprun3" command was used for the 3-processor test case 2DCASE-3. The PICMSS compute engine executable file and hostfile are in another subdirectory in BATCH as required by the makefile.

The NETSOLVE output files were created by running PICMSS client program picmss in each of the test case directories. The picmss program used is in the NETSOLVE directory. The shell scripts used are in the case file directory and reference a PICMSS compute engine executable and host file that are identical to the executable and hostfile used to create the BATCH directory output. Shell script diffoutdat in each of the test case directories compares the output in the NETSOLVE test case directory with the same output in the BATCH test case directory. These scripts can be used to verify that the calculation results are the same.

RUNNING THE PICMSS NETSOLVE SERVICE

The PICMSS NetSolve service is run from a user’s machine simply by executing the PICMSS client program in the directory containing the input files. The PICMSS client creates the input combined file, gets the server name from the agent, sends the input combined file and other information to the NetSolve server, then waits for the server to finish. The NetSolve server executes the PICMSS compute engine, creates the output combined file, and sends it back to the client machine. When NetSolve server finishes, the client program breaks the output combined file into individual files.

The PICMSS NetSolve client program gets the address of the NetSolve agent from environment variable NETSOLVE_AGENT so this variable must be set to the IP name or

address of the NetSolve agent before the client is run. For example, to use an agent on neo1 set NETSOLVE_AGENT to neo1.sinrg.cs.utk.edu.

The fourteen required PICMSS input files listed below must be present in the directory before the PICMSS client can be started:

inbcon.indat	ineqnt.indat	initer.indat	inproc.indat
inbindx.indat	ingeom.indat	inpara.indat	inwall.indat
indmsr.indat	ingrid.indat	inphys.indat	
inelem.indat	inicon.indat	inpinf.indat	

If the files are not needed, they can be completely empty but they must be present. This standard PICMSS input file is optional:

restart.indat

It will be sent to the server only if present in the directory. Please see PICMSS documentation or contact the developers for more information about the data required in PICMSS input files.

The PICMSS service uses two additional files that are not standard PICMSS input files:

picmss.sh
hostfile

File picmss.sh is required and file hostfile is optional.

File picmss.sh is a shell script that executes the PICMSS compute engine on the NetSolve server. This command must contain the operating system command used to execute the PICMSS compute engine and any parameters required by that program. The shell script executes using the default shell on the server, so the appropriate command must be present in the first record of the script to execute another shell. The shell script has one parameter, the number of processors required. For example, this shell script might be used to execute the PICMSS compute engine on the Neo cluster:

```
/usr/local/mpich/bin/mpirun -np $1
-machinefile hostfile
/home1/user/PICMSS/NEO/mpexe
```

Script picmss.sh can contain any legal shell commands and execute any program on the system.

File hostfile is provided to allow users to specify the hosts to use for the calculations on those systems where the parallel processing environment does not assign jobs to

processors. The example above uses a user-supplied hostfile. A standard hostfile can also be used on the system.

The PICMSS client program produces the following messages reporting its progress:

```
PICMSS NetSolve Client Version 1.00.001
Processors Required:  5
Input Files Read
Initializing NetSolve...
Initializing NetSolve Complete
Sending Input to Server neo1.sinrg.cs.utk.edu
Downloading Output from Server neo1.sinrg.cs.utk.edu
PICMSS service version 1.00.001 started
Input files created from combined input file
Shell script execution started
Shell script execution completed
Value returned: 8192
Combined output file created
Returning combined output file
See stdout.outdat and stderr.outdat for program
messages
PICMSS service completed
NetSolve Returned:  0
Output Files Written
```

The client writes the messages reporting the program version, number of processors required, and the successful creation of the input combined file. The NetSolve client library code writes the initializing NetSolve messages while it contacts the agent and the sending input to server message after it has established contact with the server. The remaining messages are written after the PICMSS service ends. These messages report the PICMSS service version number, the value returned by the system call that executed picmss.sh and the creation and return of the output combined file. The program generally pauses after it reports the value NetSolve returned while it breaks up the output combined file into individual files before ending.

NETSOLVE PROBLEMS

Four problems were encountered with NetSolve version 1.4 when NetSolve was installed on the four systems and when the PICMSS service was added to NetSolve. These problems and their work-arounds are listed below:

1. The NetSolve code generator produced a Makefile.numerical file that did not correctly link the PICMSS compute engine service. This problem occurred on all systems and was corrected by manually adding the required code to the makefile.

2. The configure program does not work correctly if the current directory is not in the path. This problem was encountered on the Neo system and was corrected by adding the current directory to the path.
3. The configure script on the Cypher system recorded the wrong location for the UNIX ar command. This was manually corrected in file Makefile.-i686_pc_linux_gnu.inc.
4. NetSolve on Neo and Cypher empties but does not delete the temporary directories it creates to run parallel services. The empty directories have to be manually deleted. It does delete them on AIX systems.

B. PICMSS PROBLEM DEFINITION FILE

The PICMSS NetSolve service problem definition file (PDF) is listed below:

```
@PROBLEM PICMSS
@LIB $(NETSOLVE_ROOT)/lib/$(NETSOLVE_ARCH)/libPICMSS.a
@FUNCTION PICMSS
@LANGUAGE C
@MAJOR ROW
@PATH /PICMSS/PICMSS/
@DESCRIPTION
This service is the compute engine for PICMSS (Parallel Interoperable
Computational Mechanics System Simulator) developed by the UTK Joint Institute
for Computational Science (JICS) and the Computational Fluid Dynamics
Laboratory (CFDL).  PICMSS is a versatile implicit finite element 3D-CFD
platform that can admit various formulations of fluid simulations.  This
service performs the finite element calculations for PICMSS.
@INPUT 2
@OBJECT FILE Input Combined File
This file is a combined file that contains all program input files and shell
script picmss.sh.  The combined file is a single file created by the PICMSS
client program that contains the contents of other files.  This file is
sent by NetSolve to the server system where it is unpacked by the PICMSS
service program into separate files before the PICMSS service is run.  For
more information about the combined file and the files required to run the
NetSolve PICMSS service, please see the PICMSS service documentation.  For
more information about the PICMSS input files, please see the PICMSS
documentation.
@OBJECT SCALAR I Number of processors required
The PICMSS client program gets the number of processors required from file
inpara.indat and supplies it to the PICMSS service using this parameter.
@OUTPUT 1
@OBJECT FILE Output Combined File
This file is a combined file that contains all of the output files
written by PICMSS and the standard out and standard error output produced by
PICMSS and the shell script.  This combined file is created by the PICMSS
service, transmitted by NetSolve to the client, and unpacked by the PICMSS
client program into individual files on the client machine.  For more
information about the combined file and the files produced by the NetSolve
PICMSS service, please see the PICMSS service documentation.  For more
information about the PICMSS input files, please see the PICMSS documentation.
@COMPLEXITY 1,1
@CUSTOMIZED PICMSS
@CALLINGSEQUENCE
@ARG I0
@ARG O0
@ARG I1
@CODE
extern void PICMSS_Service (char *, char *, int *);
PICMSS_Service (@I0@, @O0@, @I1@);
@END_CODE
```

C. COMBINED FILE LAYOUT

The combined file and its format were created to allow the PICMSS client to send the input files required to run the program to the NetSolve PICMSS service in a single file and to allow the PICMSS service to send all the output files produced by the program back to the client in a single file. The combined file is a file that contains one or more files stored within it using a format that allows them to be extracted from the file. The format of the file also allows text files extracted from the combined file to be stored using the end-of-line character convention used on the machine on which they are extracted. The file is designed to be transmitted as a binary file between systems. However, all end-of-line characters have been removed from the file so the file will not be changed if end-of-line character transformations are applied to the file.

FILE STRUCTURE

The combined file has the structure of an XML file but is not formally defined as an XML file. The file consists of records defined by start and end tags. Within these records are other records also delimited by start and end tags. Start tags also have attributes that describe the records they enclose.

The top level structure is the entire file and is delimited by <combined-file> and </combined-file> tags. The <combined-file> tag has attribute version which currently has value 1 which means this is the first version of the file.

The <combined-file> tags enclose a set of files, each delimited by <file> and </file> tags. The <file> tag has required attribute type and optional attribute size. Attribute type has values “binary” or “text.” Binary files are transferred without translation. Text files are transferred without translation except that newline characters are removed from the file and their locations recorded. Binary files also have the size attribute which specifies the length of the file in bytes. The entire file content appears between the <file> and </file> tags.

Text files do not use the size attribute. Instead, the individual lines that make up text files are delimited by <line> and </line> tags. These tags are used to ensure that newline characters are translated correctly. Each <line> tag has attributes length and newline. When a line is read, all characters through the next newline character, end of the read buffer, or end of the file, whichever is smallest, are read. If all characters through the next newline character are read, the newline is removed from the line, the newline attribute set to “y,” and the length is set to the number of characters read minus the newline. Otherwise the length is set to the total number of characters read and the newline attribute set to “n.” When the line is written, the number of characters between the tags is written in the file. Then, if newline is “y” a newline character is written in the

file. The routines used to read and write the newlines are required by standards to translate the newline to and from the end-of-line convention used on the computer.

A simple example of a combined file formatted to look good in print is shown below:

<combined-file version="1">
<file type="binary" size="5">abcde</file>
<file type="text"><line length="13" newline="y">line with end</line></file>
<file type="text"><line length="16" newline="n">line with no end</line>
</combined-file>

The combined files written by the client and service program is a single stream without embedded newlines.

COMBINED FILE MODULE

Module CFModule contains routines that create, add files to, and close combined files and a routine that extracts files from a combined file. Three routines are used to create a combined file: CFCreateFile which creates a new combined file, CFAddTextFile which adds a text file to the combined file, and CFCloseFile which closes the combined file. Routine CFSplitFile splits a combined file back into its component files. All routine names start with the CF prefix to help prevent name clashes with routines in other packages. Both the PICMSS client and service use CFModule to create the input and output combined files and to extract the files contained within them.