CENTER FOR HIGH PERFORMANCE COMPUTING
NEWS

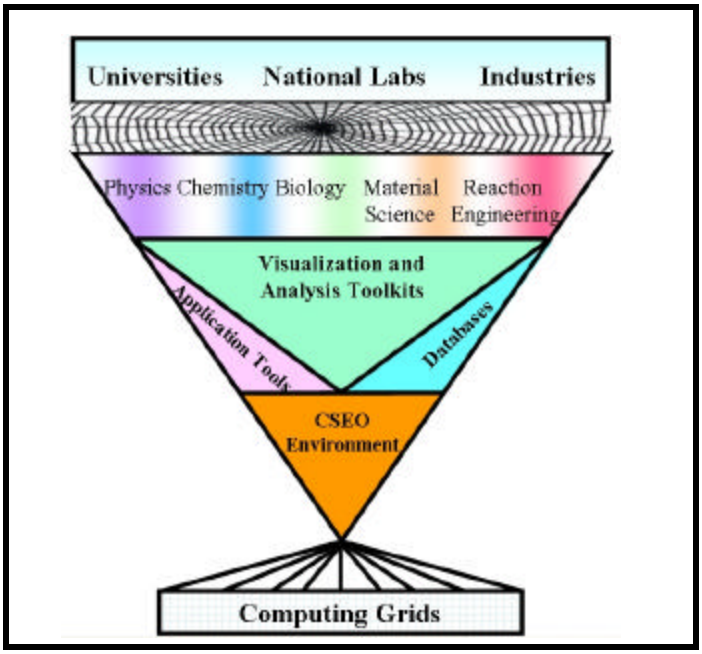
University of Utah Vol. 14 No.2 OCTOBER 2003

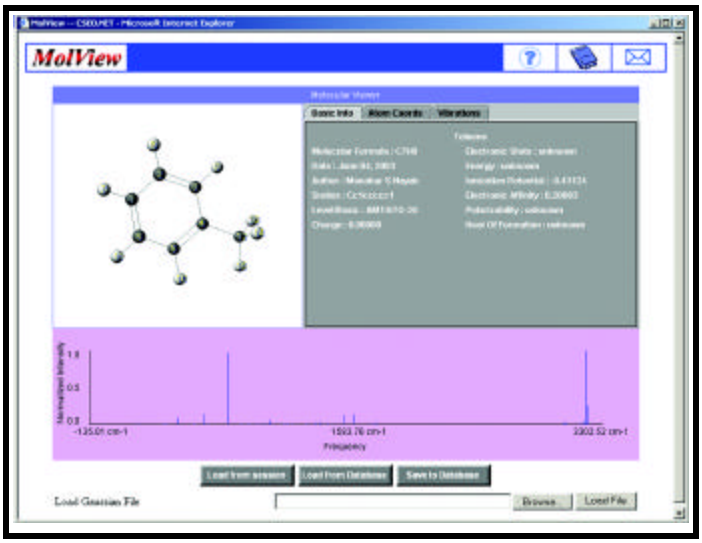
COMPUTATIONAL SCIENCE AND ENGINEERING ON-LINE

Thanh N. Truong, Professor, Henry Eyring Center for Theoretical Chemistry, Dept. of Chemistry, University of Utah

The World Wide Web (Web) has changed the way we communicate in the last decade. Research in Prof. Thanh N. Truong's group in the Department of Chemistry in the last couple years has been taking advantage of the Web information technology to develop a research environment called the Virtual Kinetic Laboratory in which students and researchers can access forefront research tools in chemical kinetics and computational resources regardless of geographical location and personal schedule. Recently the National Science Foundation Information Technology Research Initiative awarded a grant to Prof. Truong in collaboration with Prof. Chuck Wight also from the department of Chemistry, Prof. Tom Cheatham from the Medicinal Chemistry Department, Dr. Julio Facelli from CHPC, and Prof. James Lewis from the Physics Department at the Bringham Young University to evolve the above efforts into a broad-based collaboratory, a laboratory without walls, for computational science and engineering called Computational Science and Engineering Online (CSEO).

CSEO provides a Web-based grid-computing environment in which a researcher from an university, an industry, or a national lab can perform research using a variety of state-of-the-art scientific application tools, access and analyze information from public databases and personal electronic notebook, share and discuss results with colleagues, access computational resources on the computing grids that are far beyond those available locally, and master subjects in other areas of computational science and engineering without regard to geographical location and schedule.





Furthermore, it provides an extensible web-computing environment that allows a seamless interface and data flow between different areas of computational science and engineering and thus allows collaborations on multi-scale modeling of complex scientific problems. There are no limitations on what aspects of computational science and engineering can be incorporated into CSEO. The initial efforts are on providing a workflow environment to bridge fundamental chemistry and reaction engineering and on creating an integrated environment for atomistic simulations of biological and macro-molecular systems.

CSEO is not meant to be a central web portal but rather a world-wide extensive network of many mirror sites, hosted by different universities, computer centers, national laboratories, even industries, that share their public databases via a secure network. This will maximize resource utilization, data generation and sharing.

The first version of CSEO was officially released in May of 2003. Interested researchers and students can access it from http://cseo.net. Although this version is more of a proof-of-concept of the whole CSEO environment, it already provides many useful features. In particular, it inherited all research tools for calculating different types of rate constants for elementary chemical reactions from the Virtual Kinetic Laboratory but with an improved graphic-user-interface. Furthermore, it provides an interface to quantum chemistry packages such as the Gaussian98 program and an information management system that allows storage and retrieval of electronic structure properties of molecular systems and transfer of such data to another application tool for calculating their thermodynamic properties. The figure aside

is a snapshot of the GUI for molecular properties where data can be uploaded from a client computer or retrived from the CSEO database.

NEW CLUSTER AT CHPC FOR HIGH-THROUGHPUT BLAST SEARCHING

Anita Orendt, Molecular Sciences, Center for High Performance Computing

A new cluster, sequence.chpc.utah.edu, has been built at CHPC for the single purpose of running thousands of BLAST searches on a regular basis. BLAST is a package containing a set of search routines to look for similarities between a query sequence, either a nucleotide or a protein, and entries in a number of sequence databases. Both the BLAST search engines and the databases are available at the NCBI (National Center for Biotechnology Information) at the NIH. The new cluster was built to meet the needs of the Alejandro Sanchez Alvarado research group in the Department of Neurobiology and Anatomy in the School of Medicine. Working with the Sanchez group, CHPC personnel designed and built the system as well as wrote the scripts necessary to run the searches. This cluster, while built with the express purpose of the Sanchez group needs, is available for other University researchers with similar high-throughput BLAST searching needs and can be scaled to meet the demand of the workload. In the following article, the scientific problem is discussed, followed by a discussion on the design of the system and information on the implementation.

This project was a joint effort between the staff at CHPC in conjunction with Sofia Robb and Alejandro Sanchez Alvarado of the Neurology and Anatomy in the School of Medicine. For interested readers, a more detailed description of the cluster design, implementation and deployment has been submitted for publication in the *Journal of Cluster Computing*.

Description of Scientific Problem: Professor Sanchez is working on the genome of the planarian *Schmidtea mediterranea*. Part of the effort is to understand the relationship between a sequence and its biological function. This is often done by looking for similarities between the target sequence and other genetic sequences for which the function has already been established. Currently, there are over 6500 sequences they are analyzing. The input sequences are uploaded to sequence where the similarity search is done using a set of search engines (BLAST) and databases available at the NCBI. The results of the BLAST searches are then downloaded to a system in the Sanchez labs for incorporation into the database they maintain on the genetic information on this organism, called SmedDb. Due to the exponential growth in the amount of sequence information available in the databases, the search must be repeated on a regular basis or the information in the SmedDb would quickly become outdated. Therefore, the search on their entire collection of sequences is repeated on a weekly basis. A flow chart of the search and SmedDb update is shown in *Figure 1*.

System Design: The system was built in order to satisfy several considerations. Due to constraints on cost, it was determined that all components be commodity, "off-the-shelf" items. The initial target of workload turnaround was based on the ability to process the SmedDb sequences in about 48 hours; however, the computing capacity should be able to be increased as the need arises, both in terms of the growth of the Sanchez group requirements as well as the addition of other users. The searches, including job submission and retrieval of results as well as database updates should be as automatic as possible to allow for high-throughput with minimal human intervention. Also, the database updates should not interfere with ongoing searches.

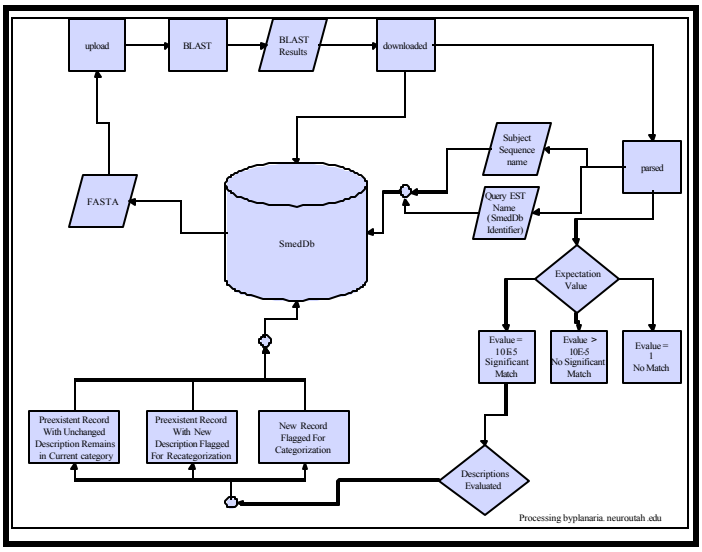


Figure 1: Flow Chart of Search and SmedDb update process

The configuration of the cluster is shown in *Figure 2*. It is composed of eight dual processor AMD Athlon MP 2000+ search nodes each with 2 GB of RAM and 60 GB local disk space and a core file server/administrative node, a dual AMD Athlon MP 2000+ with 1GB of RAM and 240 GB of usable space in a RAID array configuration, optimized for NFS read performance. In addition, there are two other machines: an interactive node, sequence, for user access and a node, seqdat, dedicated to the database updates. All of the nodes in the cluster are internally connected via a GigE network using a Foundry Big Iron 15000 switch supporting jumbo frames. The internal connection of the search and file server nodes via a private vLAN makes them inaccessible from outside the cluster sequence and seqdat are multi-ported, connected to both to the campus wide area network and to the private network with the rest of the nodes. This scheme provides a higher degree of security by concentrating the access points to two nodes and eliminating the possibility of external threads on the rest of the nodes of the cluster. All nodes are running LINUX Redhat 8.0 as well as PBS, Maui and QBank for resource management, scheduling, and accounting. The latest version of BLAST (2.2.6, Apr 2003) is installed and accessible to all compute nodes via NFS.

Implementation: Database updates are performed on seqdat via a nightly cron job. Using ncftpget (http://www.ncftp.com) the files are downloaded from the NCBI's ftp site only when an update is available. The downloaded compressed tar files are held on a disk local to seqdat, which are then unpacked and stored on the NFS space available to all nodes of the cluster. Users can perform their BLAST search directly from this copy of the database if it is relatively short; if their search is going to

overlap a possible database update they can make a static copy of the needed database files on either local scratch on the nodes they are using or on global scratch space.

The search sequences are provided by the user as a set of input files, each containing a number of sequences in FASTA format. Each of these sequences needs to be compared to the sequences in the database. These input files are held in a directory $HOME/search; search results are also kept in this location in directories given by the search date "$HOME/search/yyyy-mm-dd". The name of the files must somehow signify which BLAST code to use (blastn, blastx, tblastx). In the case of the Sanchez group this is accomplished by the presence of an n, x, or t in the input file name. A perl script is used to create the PBS batch scripts needed to perform the search, with minimal user supplied information. Before creating the PBS batch scripts the user is prompted as to whether this is a first time or a repeat search. The only other user input is the type of search (as the Sanchez group performs two different searches of the data). If it is a repeat search then the script compares the date of the database in the last search with the date of the current database. If the date of the current database is later than the date of the database used in the last search the batch script files are then made and subsequently submitted; otherwise the user is notified that the databases have not been updated since the last search. The perl script also queries to determine the number of idle search nodes. Using only a file containing the input file names the perl script generates and submits the PBS batch scripts. A round robin scheme is used to divide the searches among the nodes available. After the batch jobs have completed, the user can download the results back to their local machine for incorporation into the SmedDb. New matches are flagged for further analysis by the researcher.

Performance: Please keep in mind that the following data is specific to the searches that the Sanchez group is performing. The current SmedDb contains approximately 6500 input sequences. The most common search is to perform a blastx search on each of these sequences. This is the search that they run on a weekly basis and it takes about 30 node hours. The second search, performed about once a month, does the blastx search and then checks the output to determine if any matchs were found. If none were found a second, more time consuming tblastx search is also completed. This "no hits found" situation currently occurs for about 1800 of the input sequences. This second search takes about 650 node hours to complete.

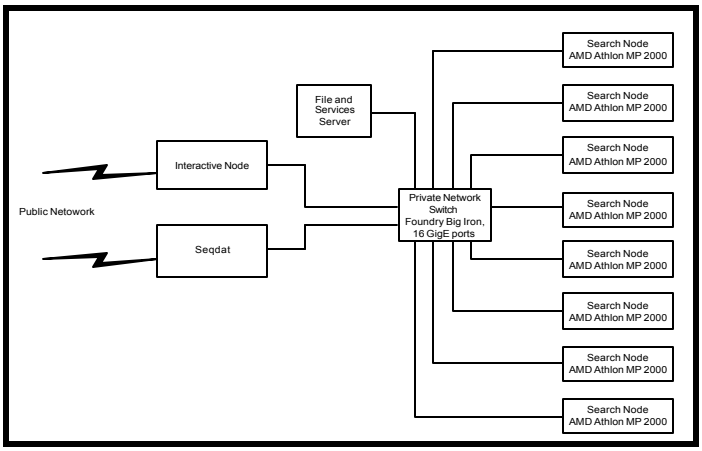


Figure 2: Architecture of Sequence Cluster

As can be seen from these numbers, there are plenty of computer cycles remaining for an increase workload, either from growth of the Sanchez search or from other research groups. Interested researchers are encouraged to contact me at orendt@chpc.utah.edu or 587-9434.

PART 3: DEBUGGING TIPS AND TRICKS FOR PROGRAMMING ON ICEBOX AND SIERRA

Martin Cuma, Scientific Applications Programming, Center for High Performance Computing

In the last two newsletter articles [1,2], we have concentrated on ways to measure and increase application's performance. However, even the most optimal code is worthless if it does not work correctly. In this article, we will explain ways one has to debug an application on both a workstation and on the CHPC's systems. We will try to present the material from a practical point of view, starting with simple methods and tools and progressing to those that are more complex.

Code compilation: Using various code-checking flags that the compilers have to offer is very useful during the code development. One of the most common errors is an array overflow, that is, writing into an array index larger than then declared size of the array. This error usually causes memory corruption resulting in a segmentation fault. Virtually every compiler has a flag that checks the array bounds (the default is to disable the array bounds checking, as it can significantly reduce the performance). Most compilers, including those from Compaq and PGI, take -C flag to turn on the array bounds checking. In the Compaq C and Fortran, one can also use -check_bounds. In the GNU compilers (gcc, g77), this flag is -fbounds-check.

A few other common errors that can be caught by using an appropriate flag are floating point exceptions, such as overflows or division by zero. Some compilers, such as those from Compaq, stop at the exception by default and display an information message. If there is a need, this behavior can be modified with -fpex series of flags, -fpe1 attempts to recover from the exception and continue the execution. Conversely, both PGI and GNU compilers on the Linux platform by default allow the program to recover from the exception and continue running. This behavior is possibly fatal as the errorneous value propagates through the calculation. To enable program termination at the floating point exception, use -Ktrap=fp flag in the PGI compilers. The program terminates with a core dump, which can then be examined in a debugger and the offending instruction located. Unfortunately, there is no such option in the GNU compilers. There are numerous other compiler options that can be used for both compile time and run time checks, which can be occasionally useful for bug hunting. Refer to man pages and to web-based documentation of each of the compilers [3-5] for more details.

Debugger basics and requirements: In case the compiler flags did not help in finding the error, it is time to call the debug-

ger to the rescue. Debuggers are programs that enable the user to load the executable in it and control the execution. As such, the user can step through the program, stop the execution at the desired location (breakpoint), examine values of the variables, etc. It is generally useful to recompile the whole code with a -g flag before loading it into the debugger. This flag inserts debugging symbols into the executable, which enables the debugger to print detailed information about the program behavior. Debuggers can be also used with executables compiled without the -g flag, but in that case, only limited information is available, such as list of machine code instructions instead of the native language source code. Also, although debugging of an optimized code is generally permitted, since the optimizer does various code rearrangements, the debugger may display the program flow incorrectly. For these reasons, one should insert the -g flag and disable optimizations before loading a program into a debugger.

Text-based debuggers: Let's first talk about text-based debuggers. Although they are not as suitable for detailed debugging as debuggers with a graphic user interface, they are present in several forms on each platform and can be useful in quick determination of a crash point in case the program crashes hard, e.g. with a segmentation fault. Every Linux machine and also many non-Linux platforms (e.g. Compaq Sierra) have a GNU debugger, GDB, installed. To load an executable into the debugger, just type gdb followed by the executable name. Optionally, core file produced by the executable can be loaded as the third argument.

In case only the executable is loaded, one can use command run typed on the command line to run the executable. Once the program crashes, GDB displays information on where the crash occurs. Command backtrace then lists the whole hierarchy of functions that called the function where the crash occurred. A few other useful GDB commands include print to display a value of a variable, and help to list all the commands and their syntax. When the core file is available, GDB recreates the situation at which the crash occurred. One can then just type backtrace to find the position of the crash. ***Figure 1*** shows an example of a Fortran program that contains a division by zero. Upon compilation with the exception trapping flag and execution, it crashes producing a core dump. We then load the executable and the core into the GDB to find out where the exception occurred. This way one can very quickly find source of many errors that crash the program.

```
program divz

        implicit none
        real*8 a,b,c

        a = 1.0d0
        b = 0.d0
        c = a/b

        write(*,*)'c = ',c
        end

icebox:~/>%pgf77 -Ktrap=fp -g divz.f -o divzp   #compile the code using
PGI
icebox:~/>%./divzp                              #run the executable
Floating exception (core dumped)
icebox:~/>%gdb divzp core                       #load the executable
and
GNU gdb Red Hat Linux (5.1-0.71)                #core into GDB
....
Loaded symbols for /lib/ld-linux.so.2
#0  0x08049112 in divz () at divz.f:8           #GDB displays the
location
8               c = a/b                         #of the crash
```

Figure 1. Example of a faulty code and locating of the exception using GDB

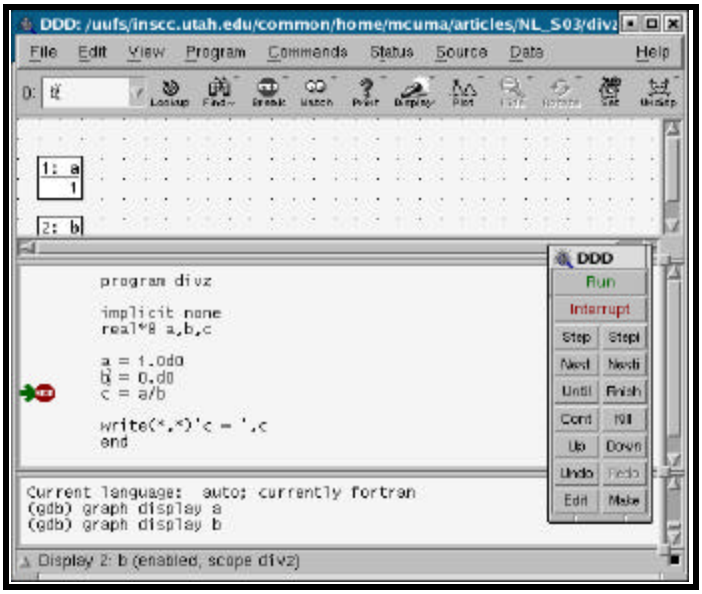


Figure 2. A screenshot of DDD

Graphical debuggers: While text-based debuggers will do most of the work that one needs for debugging, they are cumbersome to use as one needs to remember and know how to use various text based commands. Graphical debuggers enable one to do the same using menus and mouse clicks, which is more intuitive for most users. This way, it quite simple to step through the program, analyze the data at the runtime, insert breakpoints,... Graphical debuggers also come in many flavors, most UNIX distributions come with one, however, in this article, we will describe only two. DDD (Data Display Debugger) is a common Linux based debugger, which can everybody install for free on his Linux machine. Totalview is a popular commercial debugging cross-platform product, which CHPC has installed on the Icebox and Sierra clusters.

DDD: DDD [6] installs by default with many Linux distributions, and acts as a graphical front end for text-based debuggers, such as GDB. It would me my first choice of a graphic debugger on a personal workstation. DDD does all a basic debugger should do with a decent interface. The disadvantages include only serial debugging support (in fact, one could debug a parallel program, as GDB allows parallel debugging, but, only the master process data are displayed), and somewhat awkward data display. *Figure 2* displays a screen shot of the same program presented in *Figure 1*. The program code is in the middle window, data in the upper window (note the awkward vertical display of the variables, which is one of my complaints, it should also learn to display the data horizontaly), GDB text output window is in the lower part.

Totalview: Totalview [6] has become a de-facto debugging standard in the UNIX world, thanks to its complex features and multi-platform support. Apart from basic debugging features, such as stepping through the code, breakpoints, variable display, etc., there are numerous advanced functions. These include various types of conditional breakpoints (stop execution at a certain location when a condition is met) and watchpoints (stop the exe-

cution when the watched variable changes its value), intuitive data structures display and possibility of on-the-fly patches of the code inside the debugger without recompilation. However, the most powerful features are the unrivalled parallel debugging support. Totalview debugs most shared and distributed parallel environments, including threads, OpenMP, MPI and PVM, and mixed parallel codes, e.g. MPI-OpenMP. User can separately work with all parallel processes and threads, execute instructions and display variables both separately and collectively. Switching between the processes and threads is a matter of a mouse click. Totalview's intuitive GUI makes it possible to learn its basic use in several minutes, the user can then learn its more advanced features as needed in the debugging process. A screenshot of Totalview with the simple program referred to in *Figure 1* is shown in *Figure 3*. Similarly to DDD, the program code is in the middle, four windows in the upper and lower sections of the window display stack trace (nested list of subroutines), stack frame (variables and their values in the current context), threads (in this case, there is only one thread as this is a serial application) and action points (breakpoints, watchpoints,…).

Refer to the CHPC's Totalview webpage[8] for instructions on how to use Totalview on the CHPC systems. We would also like to take this opportunity to invite the users to attend a series of lectures that CHPC annually presents in the fall, one of which gives a hand-on experience with Totalview. Also, CHPC has a Quick Start booklet, which is very useful for the beginners. Please, contact the author of this article regarding this booklet or anything else related to this article.

Conclusions: In this article we briefly described options one has in debugging his/her program. First, compiler flags should be used to try to catch the most common errors, such as array overflows. Then, a debugger is recommended for more complex and detailed debugging. We noted a most common text-based debugger, GDB, and two graphic debuggers, free and relatively simple DDD, and a complex and advanced Totalview, installed on the CHPC systems.

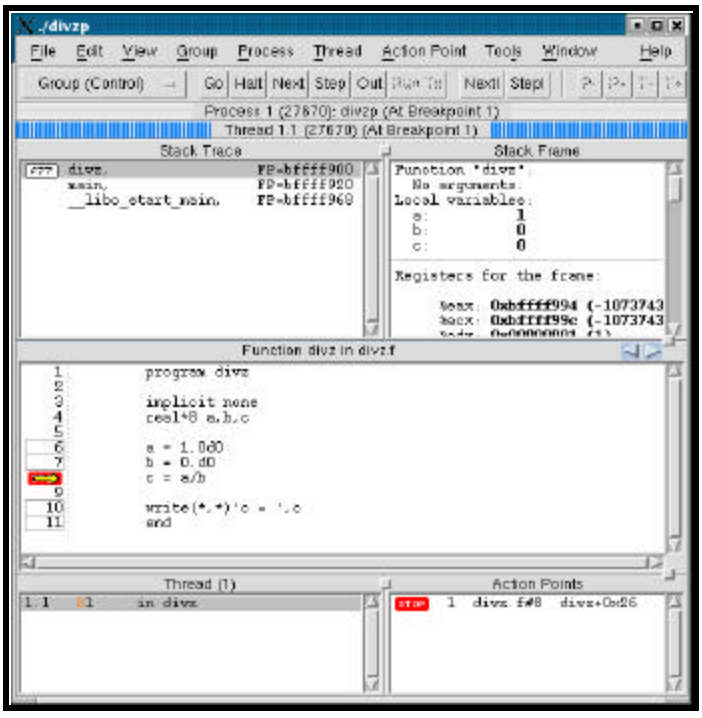


Figure 3. A screenshot of Totalview

References:

[1] Tips and tricks for programming on Icebox and Sierra part 1. Getting the program to run fast, CHPC Newsletter Summer 2002

[2] Tips and tricks for programming on Icebox and Sierra part 2. Tools for timing user programs and smart ways to code for speed, CHPC Newsletter Winter 2003

[3] Tru64 compilers documentation, C: http://h30097.www3.hp.com/compaq_c/documentation.htm, Fortran: http://h18009.www1.hp.com/fortran/docs/index.html

[4] GNU compilers documentation, http://gcc.gnu.org/onlinedocs/

[5] PGI compilers documentation, http://www.pgroup.com/doc/

[6] DDD webpage, http://www.gnu.org/software/ddd/

[7] Totalview support page, http://www.etnus.com/Support/docs/

[8] CHPC's Totalview help webpage,http://www.chpc.utah.edu/index.php?currentNumber=3.2.360

NEW SUPERCOMPUTER DUE AT THE UNIVERSITY -- 1,000 COMPUTER 'METACLUSTER' TO TACKLE TOUGH BIOMEDICAL PROBLEMS

Aug. 13, 2003 - Construction of a $2 million supercomputer comprised of 1,000 smaller computers will begin in September at the University of Utah, where researchers will use the powerful machine to tackle complex problems in biomedical research.

"This will be by far the largest computer in the state of Utah for scientific research," says physicist Julio Facelli, director of the University's Center for High Performance Computing.

When the so-called "metacluster" supercomputer is assembled sometime in the winter or spring of 2004 and tests are performed that show where it ranks in computing power, Facelli says he expects "it will be among the 20 to 30 most powerful computers in the world," excluding classified military and government computers that are not ranked.

The Center for High Performance Computing received a $1,531,008 grant last year from the National Center for Research Resources at the National Institutes of Health. It will combine that money with $500,000 in University of Utah funds to pay for the $2 million supercomputer, which will be named Arches after Utah's famed Arches National Park.

It is called a metacluster because it will be built from five clusters, each of which in turn contains many individual computers. Facelli says it is "like 1,000 desktop computers, all connected together."

"This is a significant computing resource of national caliber that will allow our researchers to tackle some of the most challenging biomedical problems," says Facelli, an adjunct professor of physics, chemistry and medical informatics. "We are very interested in using this system to perform more in-depth analysis of the vast amount of biomedical data at the University of Utah, and couple that data analysis with advanced simulations that will allow us to more precisely understand biological processes."

After a bidding process, the center recently chose Angstrom Microsystems, Inc. of Boston to provide the components and assemble the "metacluster" supercomputer, which will include 1,000 individual Opteron processors or computers made by AMD or Advanced Micro Devices, Inc., of Sunnyvale, Calif. Each AMD Opteron is a 1.4-gigahertz processor with at least one gigabyte (one billion bytes) of memory. The Opteron processors in the supercomputer together will have more than 1,000 gigabytes - or one terabyte - of memory.

"What is interesting is that the Opteron is brand new," says computer scientist Guy Adams, assistant director for systems at the Center for High Performance Computing. "Very few computer centers in the world are using these processors."

Facelli and several other University of Utah faculty members are listed as investigators on the federal grant that is paying for most of the supercomputer, so they will have priority for using it. But Facelli says free computing time on Arches also will be available to other faculty members, with research funded by the National Institutes of Health getting higher priority than research funded by other sources.

By working with professors, "both graduate and undergraduate students are going to have access to this first-class resource, and they are going to generate new ideas that will better allow us to understand biomedical systems and come up with new ways to address health problems," Facelli says.

The main investigators on the supercomputer and research they will use it for are:

◆Lisa Cannon-Albright, a professor of medical informatics, is a genetic epidemiologist who uses Utah's extensive family genealogies to identify genes responsible for inherited cancers and other diseases. Existing university computers are inadequate to allow her to analyze all members of a single family at once when looking for disease-causing genes. The new metacluster will allow simultaneous analysis of more family members, and also help her identify the causes of diseases attributed to multiple genes.

◆Greg Voth, a professor of chemistry, uses high-performance computers to simulate the behavior of molecules involved in biological processes, such as the behavior of membranes in living organisms.

◆Jeffrey Weiss, an associate professor of bioengineering, will use the metacluster for studies aimed at improving detection of changes of shape, surface area and size of body tissues. In one study, he will use the supercomputer to compare magnetic resonance images (MRI) of normal mouse brain development with changes caused by Niemann-Pick disease type C, a defect in cholesterol metabolism that kills thousands of children worldwide each year. In another study, he will create computer simulations of a common knee ligament injury with the eventual goal of simulating injuries to other ligaments and entire joints.

◆Facelli, David Grant, a distinguished professor of chemistry, and Ron Pugmire, a professor of chemical and fuels engineering, use nuclear magnetic resonance (NMR) to understand the structure of biologically important molecules. But it takes massive computing power to convert NMR measurements into information about the structure of molecules.

◆Robert Weiss, an associate professor of human genetics, studies and compares the genomes, or genetic blueprints, of humans and other animals, and also is involved in the search for genes that contribute to high blood pressure, addiction and neuromuscular diseases. Weiss now produces more genetic data than can be analyzed efficiently. The new supercomputer will give him the added computing power he needs.

◆Tom Cheatham, an assistant professor of medicinal chemistry, must crunch large amounts of data to gain a detailed picture of the structure and behavior of large, biological molecules to understand how biological processes work. Much of Cheatham's work focuses on the genetic materials DNA and RNA. Cheatham and Facelli also plan to use the new supercomputer to develop an "expert system" that would seek to improve drug treatment of various ailments by accurately predicting how drugs are absorbed, distributed in the body, metabolized and excreted.

The new supercomputer metacluster will include five clusters of Opteron processors. Adams says each cluster has a specific function,
which means the supercomputer is much less expensive than if all of its computers had to have the same capabilities:

◆The parallel computing cluster will contain 512 individual Opteron computers or processors. It will be used for complex calculations that can be run in parallel - divided among numerous individual processors - and that require the processors to communicate with each other rapidly using special networking equipment.

◆The "cycle farm" cluster will include 328 processors. It is for calculations that need much computer time, cannot be divided among as many computers and do not require the processors to communicate rapidly.

◆The data-mining cluster, with 96 individual computers, will be used to look for patterns and relationships in large sets of data, such as genetic information from many members of a single extended family.

◆The visualization cluster will have 18 processors to deal with data that must be shown graphically.

◆The file system cluster will include 34 computers to store the output of calculations from the other clusters.

◆The last 12 processors of the 1,000 in Arches will be used to control the supercomputer.

As part of the supercomputer acquisition, the university is buying 30 terabytes of data-storage equipment from Sun Microsystems, Inc. of Santa Clara, Calif., Facelli says.

Some cluster computers are made of individual personal computers, with relatively bulky boxes arrayed in racks that take a lot of space. The Opteron processors will be installed in smaller units known as "blades" that measure 1.7 inches by 23.5 inches by 30 inches. Sixteen blades fit in a "nest," and nests are then stacked, so the new supercomputer will occupy much less space then a typical cluster supercomputer made of PCs.

Adams says the Center for High Performance Computing's other cluster supercomputers eventually may be connected to Arches and become part of the metacluster.

BIBLIOGRAPHY UPDATES

Archibald, Gregory C. "A Study of the Composition of Ultra High Energy Cosmic Rays Using the Fly's Eye." Doctoral Dissertation, University of Utah, Aug. 2002.

Ayton, G. and G. A. Voth. "Bridging Microscopic and Mesoscopic Simulations of Lipid Bilayers," Biophys. J. 83, 3357-3370 (2002)

Cummings, M. P., S. A. Handley, D. S. Myers, D. L. Reed, A. Rokas and K. Winka. 2003. "Comparing bootstrap and posterior probability values in the four-taxon case." Systematic Biology 52:477-487.

Day, T. J. F, A. V. Soudackov, M. Cuma, U. W. Schmitt and G. A. Voth. "A Second Generation Multi-State Empirical Valence Bond Model for Proton Transport in Aqueous Systems," J. Chem. Phys. 117, 5839-5849 (2002)

Liu, Feng, Clayton Williams; Lee Siegel, science news specialist, University of Utah Public Relations. "Observing the 'Wings' of Atoms Study Indicates It Is Possible to See Electrons' Orbital Paths Around Atoms." University of Utah Press Release. June 2003.

Myers, D. S., and M. P. Cummings. 2003. "Necessity is the mother of invention: a simple grid computing system using commodity tools." Journal of Parallel and Distributed Computing 63:578-589.

Oklejas, Vanessa; Sjostrom, Christopher; Harris, Joel M. "Surface-Enhanced Raman Scattering Based Vibrational Stark Effect as a Spatial Probe of Interfacial Electric Fields in the Diffuse Double Layer." Journal of Physical Chemistry B (2003), 107(31), 7788-7794.

Oklejas, Vanessa; Harris, Joel M. "In-Situ Investigation of Binary-Component Self-Assembled Monolayers: A SERS-Based Spectroelectrochemical Study of the Effects of Monolayer Composition on Interfacial Structure." Langmuir (2003), 19(14), 5794-5801.

Izvekov, S. and G. A. Voth, "Carr-Parrinello Molecular Dynamics Simulation of Liquid Water: New Results," J. Chem. Phys. 116, 10372-10376 (2002)

Schlegel, H. B., S. S. Iyengar, X. Li, J. M. Milliam, G. A. Voth, G. E. Scuseria, and M. J. Frisch, "Ab Initio Molecular Dynamics: Propagating the Density Matrix with Gaussian Orbitals. III. Comparison with Born-Oppenheimer Dynamics," J. Chem. Phys. 117, 8694-8704 (2002)

Hart, K. A., W. J. Steenburgh, D. J. Onton, and A. J. Siffert, 2003: "An evaluation of mesoscale model based model output statistics (MOS) during the 2002 Olympic and Paralympic Winter Games." Accepted by Wea. Forecasting.

Horel, J, T. Potter, L. Dunn, W. J. Steenburgh, M. Eubank, M. Splitt, and D. J. Onton, 2002: "Weather support for the 2002 Winter Olympic and Paralympic Games." Bull. Amer. Meteor. Soc., 83, 227-240.

Mass, C. F., and W. J. Steenburgh, 2000: "An observational and numerical study of an orographically trapped wind reversal along the west coast of the U.S." Mon. Wea. Rev., 128, 2363-2396.

Onton, D. J., and W. J. Steenburgh, 2001: "Diagnostic and sensitivity studies of the 7 December 1998 Great Salt Lake-effect snowstorm." Mon. Wea. Rev., 129, 1318-1338.

Onton, Daryl. Ph.D.: An observational and numerical modeling investigation of Great Salt Lake-effect snow. Doctoral Dissertation, University of Utah, 2002.

Schultz, D. M., W. J. Steenburgh, R. J. Trapp, J. Horel, D. E. Kingsmill, L. B. Dunn, W. D. Rust, L. Cheng, A. Bansemer, J. Cox, J. Daugherty, D. P. Jorgensen, J. Meitin, L. Showell, B. F. Smull, K. Tarp, and M. Trainor, 2002: "Understanding Utah winter storms: The Intermountain Precipitation Experiment." Bull. Amer. Meteor. Soc., 83, 189-210.

Siffert, Andy. M.S.: Point-specific MOS forecasts for the 2002 Winter Games. Master's Thesis, University of Utah, 2001.

Steenburgh, W. J., 2002: "Using real-time mesoscale modeling in undergraduate education." Bull. Amer. Meteor. Soc., 83, 1447-1451.

Steenburgh, W. J., and D. J. Onton, 2001: "Multiscale analysis of the 7 December 1998 Great Salt Lake-effect snowstorm." Mon. Wea. Rev., 129, 1296-1317.

Welcome to CHPC News!

If you would like to be added to our mailing list, please fill out this form and return it to:

UNIVERSITY OF UTAH
Center For High Performance Computing
155 S 1452 E ROOM 405
SALT LAKE CITY, UT 84112-0190
FAX: (801)585-5366

(room 405 of the INSCC Building)

Name:
Phone:

Department or Affiliation:
Email:

Address:
(UofU campus or U.S. Mail)

Thank you for using our Systems!

Please help us to continue to provide you with access to cutting edge equipment.

ACKNOWLEDGEMENTS

If you use CHPC computer time or staff resources, we request that you acknowledge this in technical reports, publications, and dissertations. Here is an example of what we ask you to include in your acknowledgements:

"A grant of computer time from the Center for High Performance Computing is gratefully acknowledged."

Please submit copies of dissertations, reports, preprints, and reprints in which the CHPC is acknowledged to: Center for High Performance Computing, 155 South 1452 East, Rm #405, University of Utah, Salt Lake City, Utah 84112-0190

UNIVERSITY OF UTAH
Center for High Performance Computing
155 South 1452 East, RM #405
SALT LAKE CITY UT 84112-0190