Small WebComputing Applied to Distributed Monte Carlo Calculations

P.A. Whitlock[1], Dino Klein[1] and Marvin Bishop[2]

[1] Department of Computer and Information Sciences, Brooklyn College
2900 Bedford Avenue, Brooklyn, NY 11210-2889 whitlock@its.brooklyn.cuny.edu
[2] Department of Mathematics/Computer Science,Manhattan College
Riverdale, New York 10471 marvin.bishop@manhattan.edu

Abstract. The software package, Small WebComputing (SWC), has been applied to the Monte Carlo simulation of a system of hyperspheres in dimensions greater than three. The SWC software was chosen because once the framework is embedded in the application code, the user has the choice of running the distributed computations as a set of applets, as parallel threads on a symmetric multiprocessor or as independent processes distributed over a network. A brief description of the software and a discussion of future directions is given.

Introduction

The properties of hard hyperspherical systems in dimensions greater than three have been studied by Monte Carlo methods[1, 2]. Since the size of the system grows as the dimension is increased (at a minimum several thousand hyperspheres are needed), a parallel or distributed computation was necessary. The Small WebComputing (SWC) protocol developed by Ying and co-workers[3–5] was selected for the project because of several attractive features. Firstly, by simply incorporating several new classes into the Java simulation code, the actual distribution of the code is performed by the SWC software. The user is responsible for the division of the calculation into tasks that can be run in parallel. Secondly, the distributed tasks can be run as Java applets and the only requirement is that the host computer have the appropriate Java libraries, i.e. there is no need to upload the complete SWC software onto every host computer. Thirdly, the distributed tasks can also be run as threads or conventional distributed processes depending on the computers available.

A description of the future changes that could be incorporated in order to increase the effectiveness of the software is given.

The Original SWC Software

The Small WebComputing framework was developed to provide master-worker MIMD parallel programming software for a user[4]. It was meant to be a straightforward-to-use design that separated the programming interface from the underlying

hardware. To achieve its goals, the SWC software was written in Java. The use of Java and Java applets provide inherent security through the "sandbox" security model[6, 7]. A set of extendable classes that sustain the framework for the user's computation are provided. In the original version of SWC, upper level communication, e.g. between the Master process and the server, used the Transmission Control Protocol (TCP) and lower level communication used the User Datagram Protocol (UDP). Eager scheduling[8, 9] was employed to guarantee load balancing and task completion.

The SWC framework envisions a computation as composed of three levels: the Master, the Router, and the Worker components. The user of the software must identify the tasks that the Master and the Worker must complete. The Router component is provided by the software. A computation that is run via the SWC framework contains a single Master, one or more Routers, and one or more Workers. The user programs the Master component so that it divides the computation into smaller tasks, referred to as Work Units. The Workers are responsible for processing the Work Units (completing the task that the Work Unit represents) and returning Result Units to the Master. The purpose of the Router component is to act as a liaison between the Master and the Workers. The Routers pass Work Units and Result Units back and forth; all communication with the Workers is channeled through the Routers which protects the Master from being flooded by connections from Workers. Thus the communication is tiered, with the Master only communicating with the Routers, the Routers communicating with both the Master and the Workers, and the Workers communicating only with the Routers (no inter-Worker transmission of data is allowed). Computations are guaranteed to complete since any incomplete task (no Result Unit has yet been returned for the task) is kept on the task queues by the Routers and reassigned to an idle Worker that requests a Work Unit. The SWC software is completely thread based and therefore can easily take advantage of multiple CPU host computers.

In order to create a computation, the user must extend and implement the following abstract classes and interfaces:

SWCWorkUnit: a class implementing this interface will contain data regarding the task that needs to be done

SWCResultUnit: a class implementing this interface will contain results from the task that was completed

SWCMaster: the implementation of this abstract class must generate Work Units, collect Result Units, and output computation results

SWCWorker: the implementation of this abstract class processes the Work Unit and creates a Result Unit which is returned to the Master.

It is important to note that the concrete class, SWCRouter, which functions as the Router in the system, does not need to be extended since it does not perform any computation specific operations. However, even though the SWCRouter in the original version of SWC is never modified, any specific implementation of it cannot be shared among multiple computations, due to the fact that a Router is connected to only one Master running a single computation.

Small WebComputing 2: An Improved Version of the Software

The new version of the Small Web Computing (SWC2) framework[10] has improved the usability of the software. Computation specificity has been completely removed from the Master and Worker, and now the framework can host multiple, independent, concurrent computations. The Router's functionality is confined to transferring Work/Result Units among the Master and the Workers. This lack of specificity is achieved by incorporating a "pull model" of communication between each level of the framework. A Worker process requests (pulls) a Work Unit from a Router, while the Router requests Work Units from the Master. Computation completion is still guaranteed through eager scheduling.

The new version relies on TCP for all communication between the components. The original SWC framework relied on the UDP as the protocol of choice for communicating with Workers. This choice, coupled with the lack of the ability to fragment Work/Result Units at the framework level, limited the size of Work/Result Units to 64 kilobytes. The decision to use UDP may have been due to the implementation of the TCP classes in the JDK current when the SWC framework was originally written. In Table 3 of Ying, Arnow and Clark[3] which gives elapsed times for 100 iterations of transmissions of various data sizes, one notes that TCP failed to transmit 4K and 64K data instances. Another difficulty with the use of UDP was the reliable delivery of Work Units and Result Units. The SWC framework contained code to check periodically for the delivery of UDP packets. This code was the source of timing bottlenecks and bugs which haunted the early usages of the software. Since TCP guarantees delivery, the SWC2 framework does not need to test for delivery and perform retransmission of Work and Result Units. This has simplified the software and lead to greater transparency of the SWC2 source code. There is no longer a restriction on the size of the data transmitted. Exhaustive testing has shown that there is no problem with using TCP for large data sets or frequently sent transmissions in the new version of SWC2.

The Monte Carlo Application

The current application studies the properties of a system of several thousand hard hyperspheres in dimensions greater than three by using the Metropolis[11] Monte Carlo sampling technique [1]. This technique requires repeated random displacements of the hyperspheres which correspond to random walks in the configuration space of the problem. This type of calculation is usually straightforward to parallelize. Each of the random walks can be performed in parallel with all the others and only the final statistics need to be communicated and combined. In our simulation, the restriction of the size of the transmitted data blocks in the original SWC software made frequent communication of large data sets undesirable. Therefore, the application code was programmed in an "embarrassingly parallel" mode.

Ying and co-workers[4] original vision for SWC was that the programmer would divide their computation into small tasks that would take only an hour or two on any individual computer. This would not burden the volunteer host, which would likely be available for this amount of time. A GUI feature present in the software allows the user to program a task completion display that would indicate the amount of time remaining when the applet version of the Worker is used. While the SWC software is robust to losing tasks, i.e. they are just reassigned, a large number of lost tasks does seriously affect the efficiency of the calculation.

In the present case, for lower dimensional systems such as one or two dimensions, a random walk completes in one or two hours. The time for calculating a complete random walk increases dramatically in higher dimensions. A four dimensional system with 4096 hyperspheres typically took six hours and forty one minutes to complete ten parallel instances of a random walk consisting of thirteen thousand passes (each pass represents trying to move all the hyperspheres). An identically sized system in six dimensions, 4096 six-dimensional hyperspheres, took thirty three hours to complete ten parallel instances of a random walk of eleven thousand passes at a similar number density (number of hyperspheres per unit volume). These large times for individual tasks require processors which are available for long periods of time.

When a computationally intensive code is written in Java, there is always a concern that the calculations will run substantially slower than if they were implemented in a different language. Extensive testing was done to compare the Java simulation code using the SWC libraries against a serial C++ code. The serial C++ code and the parallel SWC code were run on the same processor and gave the exact same results. In a comparison of two, five dimensional runs involving 3125 hard hyperspheres with 3000 total passes, the serial code took 2 hours and 24 minutes. Whereas, the Java code with just one Worker took 2 hours. Similar timing results were obtained in all test cases, confirming our decision to use the SWC software.

While some of the calculations of the hard hypersphere systems have been deployed over a heterogeneous set of computers via the Internet, most of the production runs have been carried out on a network of Sun workstations. This occurred because it was easier to harness a large group of networked workstations than to access a large number of unrelated computers. When volunteer hosts are requested to run the applet tasks, the version of the Java Virtual Machine running on each machine can be an issue. However, the applets did run successfully on PCs using Internet Explorer 5 or Netscape 6 running Windows 2000, PCs using Mozilla 1.0.1 running Linux 2.4 and PCs using Netscape 4.7 running Solaris 2. In the setting of separated colleagues collaborating on a calculation and thus able to negotiate the choice of browser and JVM, the applet version of the Worker is a feasible method of performing calculations.

Future Directions

After several years of experience with the software, major improvements are planned. The computations are always run with multiple Routers, so that if one Router is removed, the other Routers accept the returned Result Units and assign new tasks. This is possible because each worker knows about the existence of all the Routers. However, if the Master is removed, all the computations are lost. Parallel Masters as well as parallel Routers are needed, especially, if the user chooses to run multiple concurrent computations.

In the present SWC2 organization, no communication can occur directly between Workers. Each task has to be independent of the other tasks. To extend the hypersphere computation to dimensions greater than seven will require a major change in the way the the random walks are performed. The easiest alteration would be to divide each of the parallel random walks into tasks representing a thousand or so passes and to then serialize the random walk into sets of tasks. This would require the transmission of the intermediate positions of the hyperspheres in the random walk, between the Master and the Worker, to continue the walk. This is easily accomplished with SWC2. An example of organizing a computation to allow for multiple exchanges of data between the Master and the Workers is given by the sample factorial computation in the SWC2 documentation[10]. For very large simulations, a commonly used alternative is to divide the domain space of the hard hypersphere system into subdomains containing sets of neighboring hyperspheres[12, 13]. In the present computation, the domain space is already partitioned into hypercells in order to efficiently detect hypersphere overlap. In a domain decomposition paradigm, the subdomains would represent distributed tasks and could be run on different Workers. If a hypersphere moved sufficiently far, it would need to be transferred to a different subdomain. This transfer would require a type of Worker to Worker communication. The exact form of the communication within the SWC2 framework is not at all clear, since the "sandbox" security model should not be violated. One possibility would be to have a Worker periodically query the Router if there were any communications awaiting delivery. Since the originating Worker would not know the location of the receiving Worker, a broadcast to all Workers might be necessary. Or a barrier may be necessary to ensure that the Workers are approximately all at the same step in the random walk. They would need to check for a communication at every move[14]. This is a well-known problem in parallel simulations[15] and can lead to great inefficiencies in the simulation[16].

Additional improvements to the SWC2 software are currently being implemented. The creation of a new computation is now done through command line instructions. An improvement to the user interface, a GUI computation creation, is under development. The programming of a GUI management tool has been completed and is undergoing evaluation for ease of use and error checking. In the current version of SWC2, a Worker may continue to calculate tasks that have already been completed by another Worker. This is almost unnoticeable for short tasks. However, when lengthy task are run, the whole computation may be completed and Workers could continue to compute tasks for hours that

were assigned by the Router just prior to the return of a Result Unit. A GUI interface has been designed and implemented that allows the user to keep track of Workers and send a terminate task instruction to a Worker. To achieve this with a pull model of communication requires the Worker to periodically check the Router for waiting messages. This is a useful improvement that needs to be coordinated with the general issue of communication between the Worker and the Router in addition to the transmission of Work and Result Units.

Summary

The Small WebComputing framework has been successfully used in the investigation of large hypersphere systems distributed over the Internet and on local area networks. To move the calculation to the next level of investigation, dimensions higher than seven and tens of thousands of hyperspheres, requires changes to the organization of the computation code and perhaps changes to the SWC2 software that will enable some form of Worker to Worker communication. How well this would work over the Internet with communication latency and a higher frequency of lost tasks needs careful evaluation.

Acknowledgments

We wish to thank the Brooklyn College Computing Center and the Manhattan College Computing Center for their support. One of us, D.K., was partially supported by PSC/CUNY Award # 65410-0034.

References

1. Whitlock, P.A., Klein, D., and Bishop, M.: A parallel Monte Carlo Simulation of a 5-Dimensional hard sphere system using SWC. CUNY Ph.D. Program in Computer Science Technical Report, TR-200405, http://www.cs.gc.cuny.edu/tr/, and submitted to J. Parallel and Dist. Computing
2. Bishop, M., Whitlock, P.A. and Klein, D.: The Structure of Hyperspherical Fluids in Various Dimensions. J. Chem. Phys. (accepted 2004)
3. Ying, K., Arnow, D. and D. Clark: Evaluating Communication Protocols for WebComputing. In Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications, CSREA Press, Las Vegas, June (1999)
4. Arnow, D., Weiss, G., Ying, K. and Clark, D.: SWC:A Small Framework for WebComputing. In Proceedings of the International Conference on Parallel Computing, Delft, Netherlands, August (1999)
5. Ying, K.M.: WebComputing: Design and Performance. Ph.D. dissertation, Computer Science, City University of New York (2000)
6. http://java.sun.com/sfaq/verifier.html
7. McGraw, G. and Felten,E.W.: Securing Java: Getting Down to Business with Mobile Code, 2nd Edition. John Wiley & Sons, New York, (1999) Chapter 2.

8. Baratloo, A., Karaul, M., Kedem, Z. and Wyckoff, P.: Charlotte: Metacomputing on the Web. In Proc. of the 9th International Conference on Parallel and Distributed Computing Systems, September (1996), http://cs.nyu.edu/milan/charlotte/
9. Neary, M.O. and Cappello, P.: Advanced Eager Scheduling for Java-Based Adaptively Parallel Computing. In Proc. Joint ACM Java Grande - ISCOPE Conference (2002), ~www.cs.brandeis.edu/~dilant/ WebPage_TA160/02JavaGrande.pdf
10. ~http://acc6.its.brooklyn.cuny.edu/~dinklein/swc2docs/
11. Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H. and Teller, E.: Equations of state calculations by fast computing machines. J. Chem. Phys. **21** (1953) 1087
12. Barnes, J.E. and Hut, P.: A Hierarchical O(NlogN) Force Calculation Algorithm. Nature **324** (1986) 446–449
13. Greengard, L. and Rokhlin, V.: A Fast Algorithm for Particle Simulations. J. Comp. Phys **73** (1987) 325–348
14. Wilkinson, B. and Allen, M.: Parallel Programming, Second Edition. Pearson Prentice Hall, Upper Saddle River, NJ (2005) Chapter 6
15. Jefferson, D.R.: Virtual Time, ACM Transactions on Programming Languages and Systems **7** (1985) 404–425
16. Jones, K. and Das, S.R.: Combining Optimism Limiting Schemes in Time Warp Based Parallel Simulations. In: D.J. Medeiros, E.F. Watson, J.S. Carson and M.S. Manivannan, eds., Proceedings of the 1998 Winter Simulation Conference