2004

TR-2004005: A Parallel Monte Carlo Simulation of Five-Dimensional Hyperspheres Using a WebComputing Framework

P. A. Whitlock

Marvin Bishop

Dino Klein

A parallel Monte Carlo simulation of five-dimensional hyperspheres using a WebComputing framework *

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

Abstract

The webcomputing software, Small WebComputing (SWC), has been applied to the Monte Carlo simulation of a system of five-dimensional hyperspheres. The software assumes that the user's code has been written in Java. Once the SWC framework is embedded in the application code, the user has the choice of running the calculations as a set of applets, as parallel threads on a symmetric multiprocessor or as independent processes distributed over a network. Numerous comparisons of a C++ serial version of the code and the Java code were performed. It was found that all quantities calculated agreed within the precision of the statistical errors. Moreover,the SWC parallel version was considerably more efficient in both CPU and wall clock time.

Key words: "parallel Monte Carlo simulation", "hyperspheres in five dimensions", "internet computing"

1 Introduction

The Java programming language has provided a tool and the motivation for researchers to develop parallel computing paradigms that allow large computations to be distributed across widely dispersed, heterogeneous networks [5]. Some of these approaches are denoted as WebComputing [3] because theoretically they allow the possibility of a computation to be easily distributed across the Internet. For example, a user wishing to be a volunteer in a large calculation could download a Java applet that contained the code and run it on their own processor(s). The risk in allowing a volunteer computer to participate in such a computation is minimized by the bytecode verification process and by the "sand-box security model" inherent in Java.

One such approach is the Small WebComputing (SWC) framework proposed by Ying and coworkers [23]. SWC provides an easy to use programming interface which is portable to a

*Supported by ONR Grant N00014-96-1-1-1057, by the Brooklyn College computer center and by the Manhattan College computer center.

[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

variety of operating system environments. The user can run their code on a heterogeneous network, distributed on a homogeneous LAN or on a SMP machine with no changes needed in the programming interface. In the application described in this paper, the properties of a system composed of a large number of hard hyperspheres in five-dimensional space are investigated.

The properties of hard spheres in various dimensions has been an active area of research for many years because the high-density behavior of a fluid (a gas or a liquid) is dominated by excluded volume effects associated with the repulsive portion of the interaction potential of the constituent atoms or molecules. While many properties of hard-rod systems in one-dimension were derived exactly more than 60 years ago [20], few exact results for two or higher dimensions are known. Instead, investigators rely upon numerical methods [2], *e.g.* the Percus-Yevick equation, molecular dynamics calculations or the Monte Carlo approach. The present work is a study of hard hypersphere systems using Monte Carlo methods. The particular system of interest is a relatively large number, 3125, of hyperspheres in five-dimensional space. Some of the properties of this system have been computed previously by molecular dynamics [18], by the Weeks-Chandler-Anderson approximation [7] and by using the Percus-Yevick equation [4, 9, 16]. The present computation extends the previous results. Moreover, ours is the first (to our knowledge) calculation of the pair correlation function of the five-dimensional hard hypersphere system. The Monte Carlo calculation can be parallelized in a straightforward manner when there are several thousand hyperspheres in the simulation. If the system were extended to tens of thousands of hyperspheres, a partitioning of the particle coordinate configuration space would be necessary and this would introduce interesting complications to the calculation.

The hypersphere Monte Carlo calculation in five dimensions is an ideal system for use in a WebComputing environment. The calculations can be broken up into a series of independent simulations with a high computation to communication ratio. The large granularity of the calculations gives enhanced performance in a network environment where frequent communication would degrade efficiency. Also, the calculation is hardly effected by permanently lost tasks, though this should never happen in a WebComputing calculation, and even so, can still deliver statistically useful results.

In section 2, the SWC methodology is introduced, the Monte Carlo simulation is described and the application of the SWC software is outlined. Section 3 gives sample results and compares a serial Monte Carlo calculation with a multi-threaded version, a distributed version using remote execution and the SWC applet version. The final section, 4, summarizes the computer experiments and indicates further possible extensions of the calculations.

2 Methodology

2.1 Small Web Computing

The SWC framework is a Master-Worker MIMD parallel programming model implemented in Java. The goal of SWC is to provide an user interface that is as simple to employ as writing multi-threaded code for a SMP machine. In addition, the user should not need to worry about

load-balancing or fault tolerance concerns. In SWC, this is achieved by a two-level, layered organization. There are three different types of processes in the SWC system: master, router and worker. There is only one master; there may be more than one router and there should be more than one worker. The system can be visualized as a tree of depth two where the master is the root, the routers are at depth one and the workers are at depth two.

The user implementation involves the use of two abstract classes and two interfaces. The latter are named *Computation* and *Worker*. The *Computation* interface is implemented by a user class which generates the initial set of tasks, gathers the results returned by workers and defines new tasks. This class is instantiated by the master process. The *Worker* interface is implemented by a user class which is instantiated on a volunteer host processor, either as an user process or a Java applet. This is where the actual calculations of the task are performed. The abstract classes named *WorkUnit* and *ResultUnit* are base classes which the user must extend for their specific application.

SWC is organized using multiple routers which maintain the set of task definitions and it uses "eager scheduling" [1, 6, 8] to assign work to the volunteer workers. In eager scheduling, a task is assigned repeatedly to multiple worker processes until one process completes its execution. Duplicate tasks which finish later are discarded. The eager scheduling algorithm is distributed over all the available routers [24]. This guarantees some load balancing and fault tolerance. When a volunteer worker process completes a task, it returns the results and requests a new task from the router. A worker process stays in existence until explicitly removed and can execute multiple tasks based on different input data.

While the SWC system is intended to be used as Web-based collaborative software, it can also be utilized to run a multi-threaded process on a SMP machine or to run distributed, independent processes on separate machines. The latter approach requires no additional effort on the part of the programmer. For example, in the distributed case, the master process can be on node A, router_0 can be on node B, router_1 can be on node C, and the workers can be on these or other nodes on a LAN. The master communicates with all of the routers. A router communicates with the master and the workers associated with it. The workers communicate only with the routers.

The SWC software is in active development and major changes occurred during the current project. Of particular importance, the original version of SWC was built upon UDP communication [24] and the latest version (2.0) uses TCP communication [21]. Overall, efficient completion of the calculations have been observed with both versions of the SWC software.

2.2 Monte Carlo Method

The Metropolis [19] method was developed to assist in the Monte Carlo evaluation of integrals associated with complex physical systems. It guarantees the asymptotically correct sampling of an integrand, interpreted as a probability distribution function, by performing random walks in the many dimensional configuration space of the problem. The behavior of the multi-dimensional hyperspheres is governed by the Boltzmann distribution function, $f(\mathbf{R})$:

$$f(\mathbf{R}) = \frac{exp[-\sum \phi(r_{ij})/k_bT]}{\int exp[-\sum \phi(r_{ij})/k_bT]d\mathbf{R}} \quad (1)$$

where $\mathbf{R}$ is the d-dimensional vector, $\mathbf{r}_i = (x_{i1}, x_{i2}, \cdots, x_{id})$, $i = 1, \cdots, M$ of the d*M coordinates of the centers of mass of the M hyperspheres in the simulation box, k_b is Boltzmann's constant and T is the absolute temperature of the system. The pair potential, $\phi(r_{ij})$ represents the interaction between two hyperspheres; $r_{ij} = \mid \mathbf{r_i} - \mathbf{r_j} \mid$ and

$$\phi_{hs}(r_{ij}) = \begin{cases} \infty, & \text{when } r_{ij} < \sigma \\ 0, & \text{when } r_{ij} \geq \sigma \end{cases} \tag{2}$$

and where σ represents the contact distance between the two hyperspheres or the hypersphere diameter. The hyperspheres are initially placed in a simulation box on a lattice. The side lengths of this box are determined by the number of hyperspheres and the number density, ρ, the number of particles per unit volume.

Random walks are generated by proposing a move from the current position of a hypersphere, $\mathbf{X}$, to a new position, $\mathbf{X}'$. The new position is chosen from a probability distribution function, $H(\mathbf{X}' \mid \mathbf{X})$. In the present calculation, the new position is randomly chosen from a hyperbox surrounding the current position of the center of mass of the hypersphere. The proposed new position is then accepted or rejected based upon the probability $p(\mathbf{X}' \mid \mathbf{X})$ defined as:

$$p(\mathbf{X}' \mid \mathbf{X}) = min(1, q(\mathbf{X}' \mid \mathbf{X}) \tag{3}$$

where

$$q(\mathbf{X}' \mid \mathbf{X}) = \frac{H(\mathbf{X} \mid \mathbf{X}')f(\mathbf{X}')}{H(\mathbf{X}' \mid \mathbf{X})f(\mathbf{X})} \tag{4}$$

If the new position is not accepted, the hypersphere remains at its current location. The acceptance ratio, the number of accepted moves divided by the number of total moves, is monitored. At higher densities, it is necessary to use much smaller trial moves in order to have a "reasonable" configuration change. Each pass of the random walk consists of one attempted move for each of the M hyperspheres. The move may or may not be accepted but is always counted in the averaging. As the random walk proceeds, a recursive relationship develops between the phenomenological distribution functions, $f_n(\mathbf{R})$, represented by each step of the random walk. As long as the system is ergodic and obeys detailed balance [13, 19, 22], $f_n(\mathbf{R}) \Rightarrow f(\mathbf{R})$ is guaranteed to be true as n, the number of passes, becomes large.

The Monte Carlo averaging procedure is performed to compute quantities of interest, $< A >$:

$$< A >= \frac{\int A(\mathbf{R})f(\mathbf{R})d\mathbf{R}}{\int f(\mathbf{R})d\mathbf{R}} \tag{5}$$

where $A(\mathbf{R})$, for example, can represent the total energy or the pair correlation function. Since the successive positions of the hyperspheres are not independent, it takes many passes to converge from the initial state to an equilibrated one sampled from $f(\mathbf{R})$. Thus some number of passes in the random walk must be discarded. These discarded passes are referred to as pre-equilibrium passes. Typically, on the order of $10^3 - 10^4$ passes are needed to reach the equilibrated state. Once the asymptotic distribution function is sampled, there is still serial correlation between each step in the random walk and this will affect the determination of the statistical error of the results. One method of dealing with these statistical correlations is to divide the random walks into blocks and use the block average in the error analysis. Another

approach is to run totally independent sets of random walks and average the individual results together. The latter method is easily implemented using the SWC software and is described below.

2.3 Application of SWC to the Monte Carlo Calculation

The Monte Carlo calculation was originally written as a serial code for the hard disk (two-dimensional) problem in the C++ language. This code was rewritten in Java using the SWC extensions. Both the serial and Java codes were then extended to five dimensions. A random walk of 3125 hyperspheres was initiated from a lattice in the five-dimensional space. The order parameter, O, was monitored to decide when the system had reached equilibrium.

$$O = \frac{1}{5 * M} \sum_{j=1}^{5} \sum_{i=1}^{M} cos[4\pi x_{ij} \rho^{1/5}] \tag{6}$$

Here x_{ij} is the j^{th} component of the i^{th} hypersphere. The order parameter has a value of 1 for a completely ordered lattice and randomly oscillates about 0 when the system has equilibrated. At this point all pre-equilibrium passes were discarded and the accumulators for the estimators of interest were re-initialized. The random walk was continued until the statistical error of the result was satisfactory.

Monte Carlo codes are usually straightforward to parallelize and this is true in the current situation for up to several thousand hyperspheres. For example, if 100,000 passes were needed in the random walk to achieve an acceptable standard deviation, the passes can be distributed over K processes, each process performing J = 100,000/K passes. J must be large enough to give statistical meaningful answers since the results from the processes will be averaged together. One major issue is how to handle the pre-equilibrium passes. One process could carry out all the pre-equilibrium passes and a converged configuration of the hyperspheres, determined from monitoring the order parameter, could then be distributed to all the other processes. Each of the processes would use a different sequence of pseudorandom numbers to continue the J passes of the random walk. An alternative is to have each separate process carry out both the pre-equilibrium passes and the additional J production passes. While this method seems to waste computational resources because of all the repetitions of convergence to equilibrium, the results that are averaged together are statistically independent and one significant source of variance, serial correlation, will have been removed. This latter method was used in the present calculation.

It is quite straightforward to use SWC. The code included below demonstrates the implementation of the *Computation* interface for the present calculation. The object WorkQueueAccess is the mechanism of communicating data with the master process.

```
public class ParticleComputation implements Computation
{
//define the WorkQueueAccess object and needed variables
        WorkQueueAccess  wqa;
        double density;
```

```
        int dimensions;
        double[] g;
        int workGotBack = 0;
        int workDistributed = 0;
. . .
}
```

In the method loadConfiguration(), the user class receives the parameters for the calculation through the Map object named config, receives the object WorkQueueAccess and receives an output stream where it can record various information.

```
public void loadConfiguration (Map config, WorkQueueAccess wqa, OutputStream os)
throws Exception
{
        this.wqa = wqa;
        this.config = config;
. . .
        // read in the number of dimensions
        dimensions = getInt("dimensions");
        // read in the density of the system
        density = getDouble("density");
. . .
}
```

A task in this case consists of the pre-equilibrium passes involving all M hyperspheres in addition to the J passes after equilibrium. The method run() instantiates the class Particle-WorkUnit which extends WorkUnit, sends out the work units in addWorkUnit() and receives back the completed result units in getResultUnit().

```
public void run()
{
        for (i =0; i < numTasks; i++)  // numTasks is the number of work units
        {
            ParticleWorkUnit wu = new ParticleWorkUnit();

                wu.WorkUnitID = workDistributed;
                wu.WorkerClassName = "particle2.ParticleWorker";
                wu.dimensions = dimensions;
                wu.density = density;
. . .
                try
                {
                        wqa.addWorkUnit(wu);
                }
                catch (Exception e)
```

```
. . .
        }
        ParticleResultUnit sru;

        while (workGotBack<workDistributed)
        {
                try
                {
                        sru = (ParticleResultUnit)wqa.getResultUnit();
                }
                catch (Exception e)
. . .
        }
        for (i = 0; i < ngr ; i++)  //ngr is the number of values of g(r)
        {
                final_g[i] = sru.g[i] + final_g[i];
        }
. . .
}
```

The *Computation* is completed when all the result units from the workers are combined into the final averages, the statistical errors are calculated and the results are written out .

Without the necessity of any user intervention, the router process is the link between the master and the workers. The router will take work from the master and distribute it to the workers and then receive the results and pass it to the master. As mentioned above, the routers perform load balancing to utilize all of the workers.

A separate user class implements the interface *Worker*.

```
public class ParticleWorker implements Worker
{
        double          density;
. . .
        ParticleWorkUnit pwu;
        ParticleResultUnit pru;
. . .
}
```

This class is assigned a work unit in the method setWorkUnit(),

```
public void setWorkUnit (WorkUnit wu)
{
        pwu = (ParticleWorkUnit)wu;
}
```

extracts the task parameters and performs the calculation in the *run* method.

```
public void run()
{
        dimensions = pwu.dimensions;
        density = pwu.density;
. . .
        g = new double[ngr];  //ngr is the number of values of g(r)
. . .
        pru = new ParticleResultUnit(pwu);
        pru.g = g;
. . .
}
```

The ResultUnit is extracted and returned upon completion of the task in the method getResultUnit():

```
public ResultUnit getResultUnit ()
{
        return pru;
}
```

The worker process requests the assignment of more tasks from the router. A worker stays in existence until terminated by the user.

2.4 Parallel Pseudorandom Number Sequences

The first version of the Monte Carlo code written using SWC assigned separate sequences of pseudorandom numbers to each worker by using splitting [15] in conjunction with the built-in Java (Math class) pseudorandom number generator. In the splitting technique, an estimate is made of the total number of random variables that will be needed in a run. The full period of the sequence is divided up into subsequences of this length and the seed values appropriate to each subsequence are obtained. Each worker is given an appropriate seed value to initialize the pseudorandom number generator for its respective subsequence. The Java code employing the default generator and a single worker was found to run about 15 times slower than the serial C++ code.

The Java code was subsequently optimized. One change was to replace the Java default generator with an optimized implementation of the shuffled, nested Weyl generator [11]. Empirical tests have shown that this generator gives reasonable behavior in parallel Monte Carlo calculations [10]. The version of the SWC code with the shuffled, nested Weyl generator, employing just one worker, runs as efficiently as the serial C++ code.

3 Results

Simulations were carried out for a variety of number densities, 0.05 – 0.90, spanning the range from a low density fluid to a nearly frozen solid. As the freezing point density is

approached, the system take much longer to reach equilibrium ("critical slowing down"). At these high densities, the distributed computational approach is very advantageous. Several physical quantities can be obtained from the simulations. One important quantity is the pair correlation function, g(r):

$$g(\mathbf{r}) = (< n(\mathbf{r}) > /V_s)/\rho_p \tag{7}$$

where n($\mathbf{r}$) is number of pairs of hyperspheres in a shell at a distance $\mathbf{r}$ from a reference hypersphere, V_s is the shell volume and ρ_p is the density of pairs in the simulation box. The pair correlation function can be used to obtain several other properties. If $g(\mathbf{r})$ is extrapolated to contact (two hard hyperspheres touching), the equation of state of the system (compressibility vs density) can be determined. Integrating $g(\mathbf{r})$ gives the packing number, which is a measure of how the hyperspheres are arranged in the five-dimensional space. A Fourier transform of $g(\mathbf{r})$ gives the scattering function, $S(\mathbf{k})$, which measures the momentum distributions in the fluid system.

3.1 Consistency and Correctness

First, the serial code was tested at several densities by repeating the calculations using different pseudorandom number sequences for the same input parameters. In addition, a different linear congruential pseudorandom number generator was employed at the same densities. All the computed pair correlation functions at each density were identical within the statistical errors. Moreover, two completely different particle selection schemes[14] – random and sequential – also gave identical results within statistical errors. Then the SWC code was validated by running both the serial C++ code and the distributed SWC code at the same densities. Within the statistical errors, the two versions of the simulation give the same results. Figure 1 shows a comparison of the order parameter at three separate densities. The serial and distributed calculations are in agreement. As expected, higher density states take longer to equilibrate.

One always has some concern that a CPU-bound calculations will run substantially slower when implemented using the Java language. However, for these calculations, the SWC parallel code with only one worker ran as efficiently as the serial C++ code when both codes were run on the same processor. The C++ code was compiled with the gnu C++ compiler at optimization level 2. The Java code was compiled into bytecode using the Sun Java 1.4 compiler. For example, comparing two runs at a density of 0.05 with 3000 total passes, the serial code took 2 hours and 24 minutes, while the Java code took just 2 hours.

Figure 2 compares the pair correlation function at the same three densities as Figure 1. The results are identical within the precision shown by the graph. This was found to be true at all densities studied. However, the statistical error of the estimator is a factor of $\sqrt{K}$ smaller in the SWC calculation than in the serial version where K is the number of independent tasks. Figure 3 exhibits the standard deviation of the averaged g(r) from a SWC calculation at 0.9, the highest density studied where critical slowing down is observed, when 10 independent tasks are used. The errors are very small, having a maximum value of 0.004 when the pair correlation function has a value about 2.4.

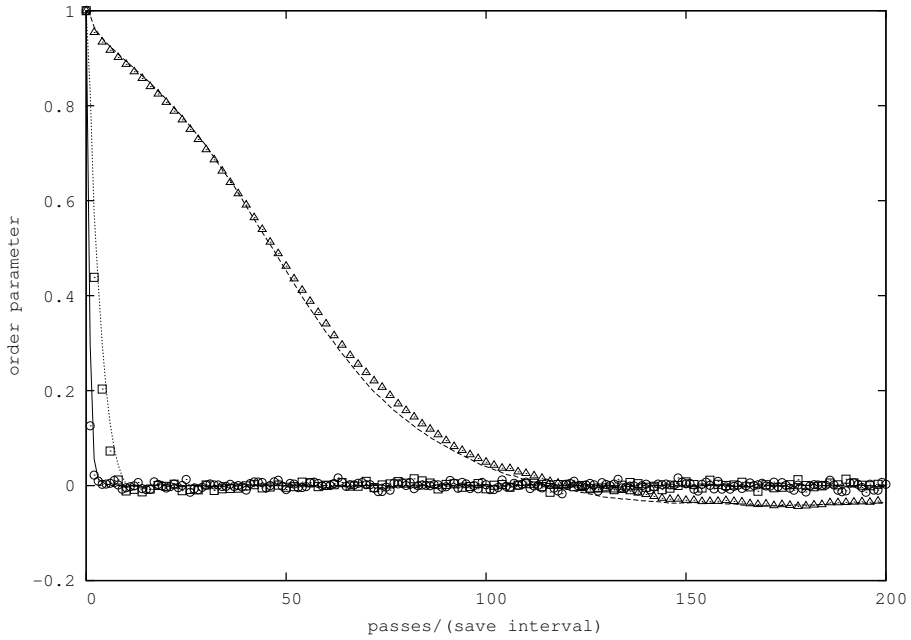

Figure 1: The order parameter from the five-dimensional simulation. The results from the SWC parallel simulation are shown for densities 0.1 (solid line), 0.5 (dotted line) and 0.8 (dashed line). The results from the serial calculation are shown for densities 0.1 (open circles), 0.5 (open squares) and 0.8 (open triangles). The save interval is the spacing in passes between writing out the order parameter. It was 2 for densities 0.1 and 0.5 and 10 for density 0.8.

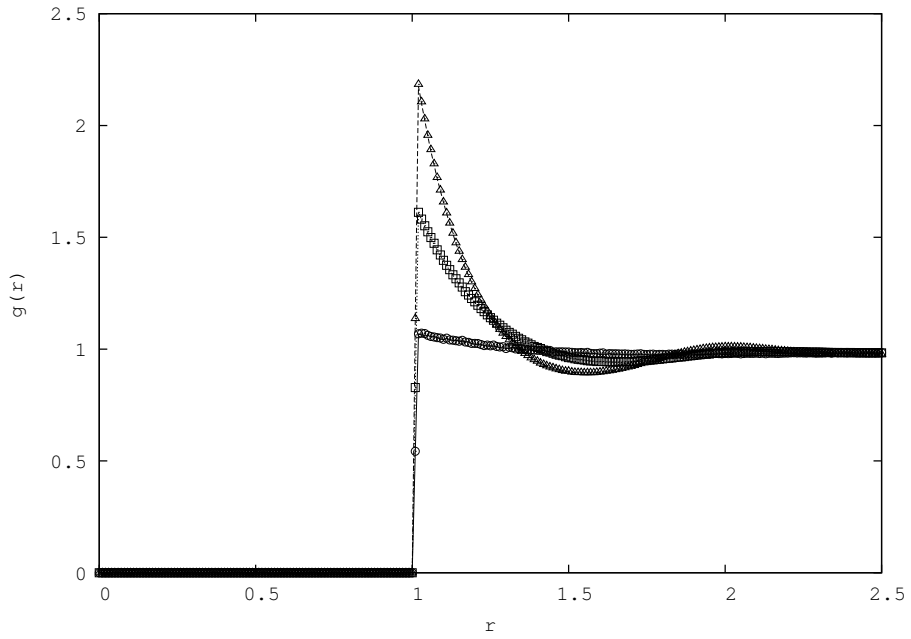

Figure 2: The pair correlation function, g(r), from the five-dimensional simulation. The SWC parallel results are shown for densities 0.1 (solid line), 0.5 (dotted line) and 0.8 (dashed line). The serial results are shown for densities 0.1 (open circles), 0.5 (open squares) and 0.8 (open triangles).

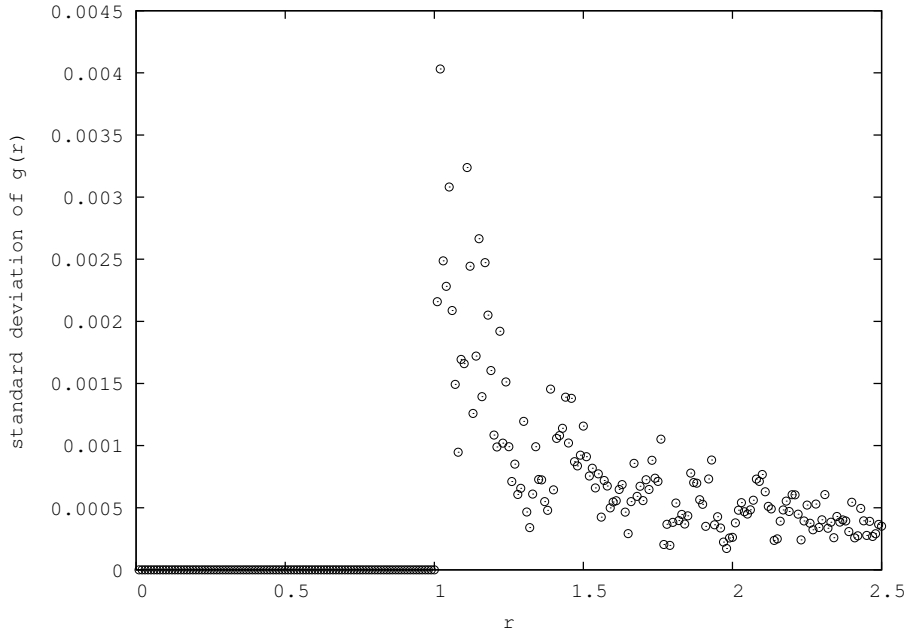

Figure 3: The standard deviation of the averaged pair correlation function from the SWC parallel calculation at a density of 0.9.

3.2 Robustness of the calculation

The SWC software claims to be fault-tolerant and this was inadvertently tested in the present calculation due to a misassignment of some internal parameters when the SWC software was implemented on a Brooklyn College LAN. Intermittently, the router processes refused to acknowledge the existence of some of the worker processes. The problem was not immediately apparent because the simulations produced correct results in a timely fashion. Any delays were on the order of those expected from competition from other (unrelated) processes on the network nodes.

3.3 Efficiency of SWC Calculations

To more fully evaluate the efficiency of the SWC software in the Monte Carlo calculation, a series of lengthy calculations were carried out. The same input parameters were used in a serial (one process) computation on a 14 processor Sun SMP computer, a multi-threaded calculation using 10 independent threads on the same SMP computer, a distributed calculation on 10 Sun workstations whose processor speeds were the same as the SMP computer and a distributed applet calculation. The three latter calculations all used the SWC software.

The total post-equilibrium passes in all the calculations was 20,000. Every task in the multi-threaded and the distributed calculation discarded 1000 pre-equilibrium passes. The serial calculation also discarded 1000 passes. Thus, the serial calculation performed 21,000 passes in all, in 34 hours. Counting all ten independent tasks, the multi-threaded as well as the distributed calculations performed 30,000 passes. Each of the ten tasks independently performed 1,000 equilibration plus 2,000 equilibrium passes. The wall clock times were 9 hours and 26 minutes and 9 hours and 56 minutes for the multi-threaded and distributed calculations, respectively. The SWC runs were approximately 3.5 times faster than the serial calculation. The distributed calculation was somewhat slower than the multi-threaded one due to the communication latency over the network.

The exciting aspect of the SWC software is that the same code can be distributed in multiple ways, depending on the processors that are available. The fourth manner of performing the calculations was to make applets available to volunteer computers. The computers/operating systems/browsers located at multiple sites (in different states) that participated in this version were a PC running Solaris 5.8 with Netscape 4.7, a laptop and a Pentium 3 running Linux 2.4 with Mozilla 1.0.1, and a PC running Windows 2000 with Netscape 6 or Internet Explorer 5.5. The server was on a Pentium 3 running Linux 2.4. It is difficult to compare the timing of this calculation directly with the others because the individual volunteers were all at different processor speeds and joined the computation at different times. However, any client browser that contacted the server and had the necessary Java Virtual Machine (JVM) was able to participate in the calculation.

The ability of a volunteer computer to join a computation is very dependent on the browser used. Applets are at the mercy of the browser's JVM. Despite the goal of "platform independence" in Java, developers are actually faced with an assortment of different applet platforms. To make matters worse, attempts at detecting the browser are inadequate because the same browser could, depending on configuration, support two or more different JVMs. For example,

on Internet Explorer 5 or 6 on Windows, you can configure the browser to use Microsoft's JVM or Sun's. The latter is more up-to-date but the former more widespread in Internet Explorer. And of course the developer, who controls the server, has no control over which JVM the user has configured.

In the setting of separated colleagues collaborating on a calculation and thus able to negotiate the choice of browser and JVM, or the availability of an accessible LAN, the SWC software shows great promise. It enabled the current investigation to be performed in a timely manner, consuming considerably less CPU time than an equivalent number of serial computations.

4 Conclusion and Future Work

The simulation of 3125 hyperspheres in five-dimensional space was successfully carried out using the SWC constructs in Java to distribute the calculation among threads on a parallel computer, as independent processes over a network and as applets over the internet. Maximum efficiency was achieved with the parallel threads on a SMP computer but the power of the calculation comes from the ability to safely distribute the simulation over a large number of computers. Thus, the production calculations were performed on a college network with dozens of processors. The organization of this simulation was a good fit with the software because the parallel calculations were entirely independent and no communication between the master and workers occurred except at initiation and completion of a set of runs. Thus it was possible to achieve maximum distribution and efficiency.

However, a few thousand particles in five dimensions does not represent a particularly large simulation. From the packing number and the pair correlation function, we observe that at high densities, there are 12 layers around each particle. To better describe the properties of the equilibrium system, it would be desirable to have many more layers around each particle. The Monte Carlo calculation attempts to minimize the affect of the relatively small number of hyperspheres in the simulations by imposing periodic boundary conditions. That is, when a hypersphere moves beyond the face of the five-dimensional simulation box, its mirror image enters the box through the opposite face. Even with the periodic boundary conditions, the size affects are still important. The number of hyperspheres could easily be increased. However, the efficiency of the calculation decreases at each step of the random walk due to the necessity of computing the distances between all pairs of hyperspheres in the system in order to apply Eqs. (2) and (3).

An alternative method of parallelization that addresses the above concerns while allowing for larger systems would be domain decomposition in which the configuration space is divided into a convenient number of subdomains. This is straight forward to envision in two- or three-dimensional space, less so in higher dimensions. A possible decomposition would be to divide the simulation box into subdomains that contain on the order of 1000 hyperspheres each. Each task would involve walking all the hyperspheres in a subdomain until one crosses the boundary to another subdomain.

Load balancing becomes a less critical issue since each worker has approximately the same number of hyperspheres in a single random walk. Communication between the workers,

however, becomes necessary. If each worker proceeds forward with no concern for the other workers, as in a Time Warp scheme[12], when a hypersphere crosses a subdomain boundary a rollback to a previous state may be needed. Check-pointing of the worker's state at the end of each step in the random walk is necessary and the memory usage of the worker increases as well. At present, the SWC software does not support inter-worker communication and so domain decomposition requires extensions to the functionality of the SWC software.

The need for check-pointing and rollbacks can be avoided at a communication and efficiency cost. Dependent on the choice of transition function, $H(\mathbf{X}' \mid \mathbf{X})$, and the subdomain geometry, no hypersphere should move beyond its nearest neighbor subdomain. In two and three dimensions, a "checkerboard" type algorithm[17] has been used to parallelize simulations using the Metropolis method. In this implementation, all the subdomains are divided into two sets, **R** or **B** in such a way that the nearest neighbor subdomains of the i^{th} subdomain are all in a different set. A step in the random walk proceeds by having all the workers in one set, say **R**, begin. If a hypersphere proposes a move to a neighboring subdomain in **B**, the **R** worker communicates to the **B** worker who evaluates the move. The result, acceptance or rejection, is communicated back to the initiating worker. When all the hyperspheres contained in the **R** have completed one step, their workers inform the routers, pause and wait for a message to resume from the routers. The workers in set **B** now complete their step in the random walk. This method also requires direct worker-to- worker communication.

Investigations into multi-dimensional hypersphere systems continues. To study scaling phenomenon, a single SWC code that works for any number of dimensions has been written. Since the server can distribute tasks for diverse sets of input data with reference to the same code, multiple runs in different dimensions and different densities can now be simultaneously submitted. The worker picks up a new task as soon as it finishes its existing task, with no requirement of overlap of input between the two tasks. We have found that organizing, submitting and processing large distributed simulations is straightforward with the SWC code. The only effort required is implementing the Java code with the SWC interfaces and classes incorporated.

Acknowledgments

The authors wish to thank John Donahue and Sean Yang for writing the original hard disk code using SWC, Alex Etinger for extending the calculation to four and five dimensions, Leonid Teverovskiy for detecting problems with the SWC router code, and Melvin Lasky for writing the five-dimensional serial C++ code. PAW thanks David Arnow for many useful conversations about WebComputing.

References

[1] A.D. Alexandrov, M. Ibel, K.E. Schauser and C.J. Scheiman, SuperWeb: Research Issues in Java-Based Global Computing, in: Concurrency: Practice and Experience, Wiley, New York, 1997, pp. 535–553.

[2] M. P. Allen and D.J. Tildesley, Computer Simulation of Liquids, Clarendon Press, Oxford, 1987.

[3] D. Arnow, G. Weiss, K. Ying and D. Clark, SWC:A Small Framework for Webcomputing, in: Proceedings of the International Conference on Parallel Computing, Delft, Netherlands, August, 1999.

[4] M. Baus and J.L. Colot, Thermodynamics and structure of a fluid of hard rods, disks, spheres, or hyperspheres from rescaled virial expansion, Physical Review A 36 (1987) 3912-3925.

[5] L. Bernadin, A Java framework for massively distributed symbolic computing, Mathematics and Computers in Simulation 45 (1999) 151-160.

[6] A. Baratloo, M. Karaul, Z. Kedem and P. Wyckoff, Charlotte: Metacomputing on the Web, in: Proceedings of the 9th International Conference on Parallel and Distributed Computing Systems, (PDCS-96) 1996.

[7] M. Bishop, A. Masters, and J.H.R. Clarke, Equation of state of hard and Weeks-Chandler-Anderson hyperspheres in four and five dimensions, Journal of Chemical Physics 110 (1999) 11449-11453.

[8] P. Cappello, B. Christiansen, M.R. Ionescu, M.O. Neary, K.E. Schauser and D. Wu, Javelin: Internet-based parallel computing using Java, in: Proceeding of the Sixth ACM SIGPLAN Symposium of Principles and Practices of Parallel Programming, 1997.

[9] B.C. Freasier and D.J. Isbister, A remark on the Percus-Yevick approximation in higher dimensions: Hardcore systems, Molecular Physics 42 (1981) 927–936.

[10] T. Gurov and P.A. Whitlock, Investigation of the sensitivity of the Monte Carlo solution for the Barker-Ferry equation using different sequential and parallel pseudorandom number generators, CUNY Ph.D. Program in Computer Science Technical Report, TR-200208, CUNY Graduate Center, New York, NY, June 2002.

[11] B.L. Holian, O.E. Percus, T.T. Warnock, and P.A. Whitlock, Pseudorandom number generator for massively parallel molecular dynamics simulations, Physical Review E 50 (1994), 1607–1615.

[12] D.R. Jefferson, Virtual Time, ACM Transactions on Programming Languages and Systems 7 (1985) 404–425.

[13] M.H. Kalos and P.A. Whitlock, Monte Carlo Methods, John Wiley and Sons, Inc., New York, 1986, pp. 73–86.

[14] D.P. Landau and K. Binder, A Guide to Monte Carlo Simulations in Statistical Physics, Cambridge University Press, Cambridge, 2000, pp. 71–72.

[15] P. L'Ecuyer and S. Cote, Implementing a Random Number Package with Splitting Facilities, ACM Transactions on Modeling and Computer Simulation 17 (1985) 98–111.

[16] E. Leutheusser, Exact Solution of the Percus-Yevick Equations for a Hard-Core Fluid in Odd Dimensions, Physica 127A (1984) 667–676.

[17] A. Linke, D.W. Heermann, and P. Altevogt, Simulating very large Ising systems for short timescales, Computer Physics Communications 90 (1995) 66-72.

[18] M. Luban and J.P.J Michels, Equation of state of hard D-dimensional hyperspheres, Physical Review A 40 (1990), 6796-6804.

[19] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller and E. Teller, Equations of state calculations by fast computing machines, Journal of Chemical Physics, 21 (1953), 1087–1091.

[20] L. Tonks, The complete equation of state of one, two and three-dimensional gases of hard elastic spheres, Physical Review, 50 (1936) 955–963.

[21] P.A. Whitlock and D. Klein, Extensions and improvements to the SWC Webcomputing software, in preparation, 2004.

[22] W.W. Wood, Monte Carlo studies of simple liquid models, in: H.N.V. Temperley, J.S. Rowlinson and G.S. Rushbrooke, (Eds.), The Physics of Simple Liquids, North-Holland, Amsterdam, 1968, Chapter 5.

[23] K. Ying, D. Arnow 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.

[24] K.M. Ying, WebComputing: Design and Performance, Ph.D. dissertation, Computer Science, City University of New York, 2000.