[17] Rego, V., and Sunderam, V., Experiments in Concurrent Stochastic Simulation: The EcliPSe Paradigm, *Journal of Parallel and Distributed Computing*, Vol. 14, No. 1, pp. 66–84, January, 1992.

[18] Ripley, B., *Stochastic Simulation,* Wiley, N.Y., 1987.

[19] Stanley, H., E., *Introduction to Phase Transitions and Critical Phenomena* (Oxford, New York, 1971).

[20] Stauffer, D., *Introduction to Percolation Theory* (Taylor and Francis, London, 1985).

[21] Sunderam, V., and Rego, V., EcliPSe: A System for High Performance Concurrent Simulation, *Software, Practice & Experience,* Vol. 21, No. 11, pp. 1189–1219, November 1991.

tions achieved 403 MFLOPS/$Million, as compared to 2.29 MFLOPS/$Million for the CRAY. The tables also indicate that effective scaling to hundreds of processors of similar capabilities is feasible. As far as the authors are aware, this is the first experiment that has utilized, interactively and simultaneously, (a) 128 geographically dispersed workstations; (b) 2 geographically dispersed 64-node hypercubes; and (c) collections of workstations and hardware multiprocessors, as *a unified concurrent machine*.

Acknowledgements

We gratefully acknowledge the cooperation of various friendly machines at the California Institute of Technology, Florida State University, Oak Ridge National Laboratory, Purdue University, and Emory University, and the CRAY Y-MP at the University of Illinois. We thank Ling-Yu Chuang for her help with the CRAY measurements.

References

[1] Bailey, D., "How useful are Today's Parallel Computers?," *Computers in Physics*, Vol. 6, No. 2, Mar/April 1992.

[2] Bell, G., "The Future of High Performance Computers in Science and Engineering," *CACM,* Vol. 32, No. 9, pp. 1091-1101, September, 1989.

[3] de Gennes, P. G., *Scaling Concepts in Polymer Physics* (Cornell University, Ithaca, N. Y., 1979) and references therein.

[4] Flory, P. J., *Statistical Mechanics of Chain Molecules* (Interscience, New York, 1969).

[5] Harris, A., B., J. Phys. C **7**, 1671 (1974), and A. B. Harris, Z. Phys. B **49**, 347 (1983).

[6] Heidelberger, P., "Discrete Event Simulations and Parallel Processing: Statistical Properties," *SIAM J. Statist. Comput.*, Vol. 9, No. 6, pp. 1114-1132, 1988.

[7] Hoshen, J., and Kopelman, R., Phys. Rev. B **14**, 3438 (1976).

[8] des Cloizeaux, J., and Jannink, G., *Polymers in Solution: Their Modelling and Structure* (Oxford, New York, 1990) and references therein.

[9] Kremer, K., Z. Phys. B **45**, 149 (1981).

[10] Lam, P. M., J. Phys. A **23**, L831 (1990); but see also H. Nakanishi and S. B. Lee, J. Phys. A **24**, 1355 (1991).

[11] Lee, S., B., and Nakanishi, H., Phys. Rev. Lett. **61**, 2022 (1988), and S. B. Lee, H. Nakanishi and Y. Kim, Phys. Rev. B **39**, 9561 (1989).

[12] Le Doussal, P., and and Machta, J., J. Stat. Phys. **64**, 541 (1991).

[13] Meir , Y., and Harris, A., B., Phys. Rev. Lett. **63**, 2819 (1989).

[14] B. Mandelbrot, *Fractal Geometry of Nature* (Freeman, New York, 1982).

[15] Nakanishi, H., and Moon, J., (unpublished).

[16] Nakanishi, H., Rego, V., and Sunderam, V., "Superconcurrent Simulation of Polymer Chains on Heterogeneous Networks," Dept. of Computer Sciences, Purdue University, 1992.

Configuration (Experiment)	CRAY MIPS	EcliPSe MIPS	CRAY MIPS/$M
15 Sparc	5.74	11.12	2.29
15 RS6000	5.74	45.67	2.29
15 i860	5.74	7.92	2.29
31 Sparc	5.76	26.19	2.30
31 RS6000	5.76	69.80	2.30
31 RX/i860	5.76	16.37	2.30
63 Sparc +RS6000	5.79	26.00	2.31
63 i860	5.79	16.27	2.31
127 Sparc +RS6000	5.72	161.08	2.28
127 i860	5.72	67.63	2.28
191 All	5.74	139.60	2.29

Table 6: EcliPSe MFLOPS vs. CRAY MFLOPS

Configuration (Experiment)	EcliPSe MIPS/$M	EcliPSe/CRAY Effectiveness Ratio
15 Sparc	139.13	62
15 RS6000	403.73	180
15 i860	21.12	9
31 Sparc	148.94	64
31 RS6000	308.52	133
31 RX/i860	21.82	9
63 Sparc +RS6000	64.66	28
63 i860	10.84	4
127 Sparc +RS6000	154.54	69
127 i860	22.54	10
191 All	34.53	15

Table 7: EcliPSe MFLOPS vs. CRAY MFLOPS

are obtainable at the same prices. For example, the cost of a Sparcstation SLC used in our calculations is $4995; as of April 29, 1992, SUN-microsystems announced a price reduction on ELC, IPC, IPX and SS2 workstations, so that a 22 MIP ELC costs as little as $2400. Consequently, using $4995 today, we can purchase systems with double the CPU speed of the Sparcstation SLC.

Tables 4-5, and 6-7 show the MIPS and MFLOPS ratings for each of the experiments, or equivalently, each of the machine configurations used. As previously mentioned, two important factors should be considered when interpreting the significance of these results. First, the algorithm is inherently non-parallel, and highly non-local in terms of memory references. Therefore, the overall rate of instructions executed per second is (not surprisingly) low, since high rates usually require very high cache-hit percentages, as well as the potential for instruction pipelining. The second point to be noted is that for this application, integer operations outnumber floating point operations by a factor of 10:1. Further, the floating point operations are unevenly and sparsely distributed among the integer computations. The combination of these factors leads to low MFLOP rates, especially notable in the timings on architectures such as the IBM RS 6000 and Intel i860. However, we reiterate that *given the nature of the problem and the above constraints*, the *comparative* timings of this application under EcliPSe do indeed exhibit very high performance.

The MIPS and MFLOPS entries in Tables 4-5 and 6-7 were obtained by first executing the equivalent (i.e., identical) runs on the CRAY, and obtaining instruction counts for the execution path of each sampler. These counts were divided by the elapsed times measured in the EcliPSe environments to obtain MIPS and MFLOPS ratings for these configurations. Tables 4-5 show that the MIPS ratings achievable under EcliPSe are far superior to corresponding CRAY Y-MP measurements. The 127-processor configuration, solely using networks of workstations, achieved 1255 MIPS; nearly thirty times the CRAY rating. More significantly, a 16-processor network of Sparcstations could achieve 86 MIPS, twice the figure for a CRAY. The price-performance ratios are even more remarkable. Tables 4-5 show that the best price-performance was achieved by a collection of 16 RS6000/320 workstations that obtained 3.14 GIPS per $Million. The last column is a "cost-effectiveness ratio", obtained by dividing MIPS/$Million for an EcliPSe configuration by the corresponding figure for the CRAY. We assume that the nominal cost of a CRAY Y-MP with 8 processors is $20 Million, with an equivalent per processor cost of $2.5 Million. For example, the second row of Tables 4-5 implies that a network of 16 RS6000/320 workstations is 175 times more cost effective for the integer portions of this application than a CRAY.

Tables 6-7 show similar factors of improvements for EcliPSe over the CRAY, somewhat attenuated in a few instances. Again, collections of workstations exhibit the best performance per processor, as well as price-performance ratios far better than either the CRAY or a hardware multiprocessor, the Intel i860 hypercube. For this application, 16 RS6000/320 worksta-

Configuration (Experiment)	CRAY MIPS	EcliPSe MIPS	CRAY MIPS/$M
15 Sparc	44.75	86.72	17.90
15 RS6000	44.75	356.02	17.90
15 i860	44.75	61.76	17.90
31 Sparc	44.91	204.14	17.96
31 RS6000	44.91	544.10	17.96
31 RX/i860	44.91	127.66	17.96
63 Sparc +RS6000	45.16	202.69	18.06
63 i860	45.16	126.82	18.06
127 Sparc +RS6000	44.62	1255.56	17.84
127 i860	44.62	527.17	17.84
191 All	44.79	1088.07	17.91

Table 4: EcliPSe MIPS vs. CRAY MIPS

Configuration (Experiment)	EcliPSe MIPS/$M	EcliPSe/CRAY Effectiveness Ratio
15 Sparc	1085.08	61
15 RS6000	3147.27	176
15 i860	164.69	9
31 Sparc	1160.94	65
31 RS6000	2404.96	134
31 RX/i860	170.21	9
63 Sparc +RS6000	504.10	28
63 i860	84.54	5
127 Sparc +RS6000	1204.65	68
127 i860	175.72	10
191 All	269.17	15

Table 5: EcliPSe MIPS vs. CRAY MIPS

hypercube) completed in 574.06 seconds, compared to 11493.5 seconds on the CRAY (191 sequential runs, 2 batches each) as shown in the last row of the table. As indicated in Section 2.1, the 8K byte data-cache of the i860 node increased the Intel timings because of severe nonlocal referencing, mainly in the depth-first search phase. The 64K byte cache allowed the Sparc to perform impressively.

This table illustrates the tremendous gains in overall execution times that are obtainable by utilizing a geographically dispersed collection of systems, including both general purpose workstations and hardware multiprocessors. A run requiring 4 hours on a CRAY processor could be completed in a roughly twelve minutes on a network-based massively parallel system. Furthermore, it should be noted that the workstations that were used in these experiments *were not dedicated to our runs; rather, computing resources from these workstations were "scavenged" when they were not in use for other purposes.* This, in effect, is tantamount to constructive use of otherwise wasted cycles, and therefore, demonstrates an important (albeit subjective, and perhaps moot) point. Nevertheless, it is a constructive solution to the "interactive supercomputer" issue raised by Bailey [1].

The raw performance numbers for the experiments are reported in the following discussion. However, as explained earlier in Section 2.1, the problem is an inherently poor candidate for parallelization - as a result, spectacular MFLOP ratings are not observed. What is indeed remarkable are the factors by which the EcliPSe experiments outperform traditional supercomputers like the CRAY Y-MP even in terms of raw performance obtainable. In this section, we describe the hardware platforms on which the EcliPSe experiments were conducted, and lay the foundation for the most significant of our results - excellent price-performance ratios.

Table 3 shows the virtual-machine configurations that were used in each of the experiments, with details of the exact models used. The first column shows the number and type of processors; each group of "15", "31", "63", "127", and "191" configurations corresponds to one set of run parameters. The second column gives the exact configuration of the virtual machine, indicating by model, the composition of each of the clusters. Finally, the third column gives total list prices of these configurations, obtained from vendor announcements. Individual machine prices are based on minimum configurations (memory, disk etc.) required to actually execute this application.

Two points are worthy of note, with regard to Table 2. First, the specific machines in each composition are those that were actually available and used in the experiments. However, less expensive machines could have been used, had they been available, without significant impact on the timings obtained. For example, in the "127 Sparc+RS6000" configuration, the 9 RS6000/550 systems could have been substituted with 10-12 RS6000/320 systems, reducing the total configuration cost by nearly $400,000! Secondly, the prices used in the table reflect costs at the time of purchase, i.e. 1-2 years earlier. In the meantime, some prices have fallen, while in other cases, much faster machines

Nprocs	Sun	RS6000	Sun+RS6000
15	7101.54	1992.87	-
31	3194.36	1366.06	-
63	-	-	1015.47
127	-	-	492.21
191	-	-	-

Table 1: Timing comparisons of EcliPSe vs. CRAY Y-MP (seconds)

i860	Sun+RS6000+i860	Average	Equiv CRAY
9226.09	-	6106.83	12735.5
4661.96	-	3074.12	13250.6
2108.07	-	1516.77	11552.5
893.55	-	692.88	10526.4
-	574.06	574.06	11493.5

Table 2: Timing comparisons of EcliPSe vs. CRAY Y-MP (seconds)

4 Performance results

We performed exactly one simulation experiment in each of the eleven computing configurations described. Each of these runs used the same input parameters. As mentioned in Section 4.1, the number entered in the tables for the "number of processors used" excludes the central monitor, and hence is always one less than a power of two. To avoid order-statistic related bias [6], the central monitor requested an equal number of batches (sample vectors) from each processor. The total number of batches obtained was an integer multiple of the number of samplers used. Specifically, in runs using 15, 31, 63, 127, and 191 processors, 495, 496, 441, 381, and 384 batches were computed respectively. Thus, for example, in the 15-processor experiments, each processor computed 33 batches, using a single pair of random number seeds to initiate the entire trajectory. Similarly, 7 batches per processor (each using a single pair of seeds) were computed in all 63-processor experiments.

The yardstick against which the results are compared are *equivalent timings on a single processor* of the CRAY Y-MP system at Illinois. For each of the eleven configuration-dependent simulation experiments we conducted, we made an identical run of optimized code on the CRAY Y-MP. Thus for example, to get a CRAY run equivalent to the 191 processor run of Configuration #11, the CRAY was required to utilize the 191 distinct pairs of seeds used by the 191 processors of this configuration. The results of the CRAY run were identical to the results given by the processors of Configuration #11. These equivalent CRAY runs were made for each of the eleven experiments performed so that we could make a detailed comparison against the CRAY.

Tables 1 and 2 shows the execution times, in seconds, for each of the eleven experimental runs that were conducted. Once again, we point out that each row of the table (i.e. each experiment with different numbers of processors) is slightly different from the others, owing to the different total number of batches computed by each processor configuration of a given size. In other words, the workload for each row is different, as explained at the beginning of this section.

Configuration	Composition	Cost
15 Sparc	16SLC Sparcstations	$79920
15 RS6000	16 RS6000/320 Systems	$113120
15 i860	16 i860 nodes	$375000
31 Sparc	16 SLC Sparcstations + 16 SS1 Sparcstations	$175840
31 RS6000	32 RS6000/320 Systems	$226240
31 i860	32 i860 nodes	$750000
63 Sparc + RS6000	16 SLC Sparcstations + 16 SS1 Sparcstations + 32 RS6000/320 Systems	$402080
63 i860	64 i860 nodes	$1500000
127 Sparc + RS6000	36 SLC Sparcstations + 45 SS1 Sparcstations 38 RS6000/320 Systems 9 RS6000/550 Systems	$1042255
127 i860	128 i860 nodes	$3000000
191 All	36 SLC Sparcstations + 45 SS1 Sparcstations 38 RS6000/320 Systems 9 RS6000/550 Systems 128 i860 nodes	$4042255

Table 3: Configurations of virtual tree-machines

In Tables 1 and 2, the execution time, in seconds, is shown for each virtual machine configuration. For example, a 496-batch (i.e., 31 processors, each sampling 16 batches) run took 3194.36 seconds on a network of Sparcstations, 1366.06 seconds on a network of IBM RS6000 workstations, and 4661.96 seconds on 31 nodes of an Intel i860 hypercube, resulting in an average time of 3074.12 seconds across all EcliPSe runs. By contrast, exactly the same results are obtained on the CRAY Y-MP by running the simulation for 31 pairs of seeds in sequential fashion, but requiring a total time of 13250.6 seconds. Similarly, a 384-batch run under EcliPSe using 191 processors (a combination of Sparcstations, IBM RS6000's and a 64-node

- **Configuration 1: Sixteen nodes**
  16 SUN Sparc workstations (Purdue)
- **Configuration 2: Sixteen nodes**
  16 IBM RS6000 workstations (Florida State University)
- **Configuration 3: Sixteen nodes**
  16 Intel i860 nodes (Oak Ridge National Labs, ORNL)
- **Configuration 4: Thirty-two nodes**
  32 SUN Sparc workstations (Emory University)
- **Configuration 5: Thirty-two nodes**
  32 IBM RS6000 workstations (28-Florida + 4-Emory)
- **Configuration 6: Thirty-two nodes**
  32 Intel i860 nodes (ORNL)
- **Configuration 7: Sixty-four nodes**
  32 Sparcs + 32 RS6000 (32-Emory Sparcs + 28-Florida RS6000 + 4-Emory RS6000)
- **Configuration 8: Sixty-four nodes**
  64 Intel i860 nodes (ORNL)
- **Configuration 9: One hundred and twenty-eight nodes**
  48 RS6000 + 80 Sparcs (28-Florida RS6000 + 5-Emory RS6000 + 15-ORNL RS6000 + 20-Florida Sparcs + 40-Emory Sparcs + 20-ORNL Sparcs)
- **Configuration 10: One hundred and twenty-eight nodes**
  128 Intel i860 nodes (64-ORNL i860 nodes + 64-Caltech i860 nodes)
- **Configuration 11: One hundred and ninety-two nodes**
  48 RS6000 + 80 Sparcs + 64 Intel i860 nodes (28-Florida RS6000 + 5-Emory RS6000 + 15-ORNL RS6000 + 20-Florida Sparcs + 40-Emory Sparcs + 20-ORNL Sparcs + 64-ORNL i860 nodes)

3.1 Fault tolerant tree-structured computations

Most nontrivial stochastic simulations, even massively parallel ones, tend to be long-running compute-intensive applications. For simulations that execute for days, or even weeks, the ongoing parallel computation must be protected against nodal faults or subnetwork outages. We achieve this by arranging processor configurations in tree-structures, where a processor is a node in the tree. Computation and control information is piped bi-directionally along the branches.

Two examples of fault-tolerant tree-machines are shown in Figure 2 (see [16]). Recall that the nodes are arbitrary, geographically distributed machines. A simple tree construction algorithm takes as input the processor configuration file and sets up the tree structure using prescribed limits on the number of children a node can have. In Figure 2 (see [16]), M represents a statistical *monitor*, and S represents a *sampler*. The root node is the central monitor at which computation is initiated. Each sampling processor computes along an independent trajectory and reports results to its parent. Each parent node performs sampling as well as monitoring, combining its children's results with its own, before passing the combined results to its own parent in the tree. Stream connections between the machines maintain flow-control. *In all of the experiments reported here, the central monitor did not perform sampling.* Hence, in reporting results based on a configuration with n processors, we use the number $(n-1)$ in the table to describe the number of processors involved in the computation (e.g, see the first column in Table 1).

In the tree at the top of Figure 2 (see [16]), the rectangles represent processors that are both monitors as well as samplers. Only the leaves are strictly samplers. If a leaf processor is to fail, it is simply ignored by the parent monitor. However, if an interior node is to fail, its parent takes over the responsibility of monitoring the computations of children of the failed node. We use a time-out mechanism to effect this, with a simple reconfiguration protocol which appears to work well. In the worst case, if all interior nodes fail, the central monitor takes over monitoring of the entire computation. The nodes in this figure may be arbitrary processors (though we only used Sparcs, IBM RS6000s, and i860 nodes).

In the tree at the bottom of Figure 2 (see [?]), we used an IBM RS6000 as the central monitor, with two 64-node i860 hypercubes as subtrees, one at Caltech and one at Oak Ridge (Configuration #10); both hypercubes sampled simultaneously and reported results to monitors. Each of the subcubes devotes one node to the monitoring function. We refrained from building fault-tolerance within subcubes. It has been our experience that if a single i860 node goes down, the entire subcube fails. In our tree, this would affect one-half of the computation, but the simulation would eventually complete, requiring twice as much time as it would have required had the subcube failure not occurred.

2.1 Inherent sequential nature of underlying algorithms

Through the following description, it will become readily apparent that this class of polymer based random-walk problems in three dimensions are indeed a challenge for fast scalar or vector processors. Through a detailed study of the code, we determined that the algorithms exhibit a performance hampering behavior due to a high non-locality of data referencing caused by the cluster generation and search phases in the large random structures (approximately 700,000 to 1 million integers, and 15,000 double precision numbers). In addition, both the clustering algorithm and the depth-first search algorithms described below are inherently sequential, and primarily utilize integer arithmetic for cluster generation and SAW enumeration. The two major subroutines used in the sequential algorithm are *make3* (which generates clusters and checks for connectivity) and *wsaw* (which performs a compute-intensive depth-first type search algorithm). *Both the routines used are based on well-known algorithms that have been used for several years with great success.*

First, subroutine *make3* uses a Monte Carlo scheme to generate a percolation cluster that is a maximal connected component of occupied sites. This cluster must span the cubic grid in all three directions simultaneously. This is done using an algorithm due to Hoshen and Kopelman [7]. Thus, we decide whether each site of the grid is occupied or not by comparing a $U(0,1)$ (i.e., uniformly distributed between 0 and 1) random number to an input parameter p which represents the fill fraction. This is done one at at time, column by column, and layer by layer. The nontrivial part of the algorithm lies in maintaining connectivity information. Initially, an occupied site (i,j,k) is labeled by a cluster number $latt(i,j,k)$ (which labels each connected component). However, when a newly occupied site joins two or more pre-existing clusters, a problem arises. While a simple scheme suggests relabelling all sites by a common cluster number, this is prohibitively time-consuming. Thus instead of relabelling sites that were already labeled, we keep a separate *label tree* which records information about which clusters are in fact connected to which other clusters. In this way, the *tree* keeps the pointers (using a value that is the negative of the cluster number it points to) *and* also the size of the cluster, if the index is the true or the minimum cluster label. That is, if $label(i) = -j$ (with $j > 0$), then the true cluster label of the cluster i is j; if $label(i) = j$ (with $j > 0$), then i is the true label and the cluster size is j.

After the cluster generation, the maximal cluster is identified, and then checked to determine if it spans the cube in all three directions. If so, the periodic boundary conditions are applied in all directions. Finally, a *link* matrix is created from the lattice map of the percolation cluster. This array is two-dimensional with the first index being the site number and the second the number indicating the nearest neighbor. The value of $link(i,j)$ is the site number of the j-th neighbor of the i-th site of the cluster. If a particular neighbor is empty, then the corresponding entry is given a value equal to its own site number. This information is used in the next step when self-avoiding walks (SAWs) are generated on the cluster.

Next, we discuss the subroutine *wsaw* which enumerates all SAWs of *nstep* steps starting from a fixed starting point (site *init*) by a depth-first type search algorithm. Thus, each time a step is added, we always try to add an occupied nearest neighbor that is the first unattempted neighbor in the order listed in the array *ndr*. The array *link* created by *make3* tells whether the particular neighbor is occupied or not and if so what the site number of the neighbor is. If this is a success, i.e., if it has not been visited by a previous step, $(mocc(nsite) = 0)$, then it is accepted into the ensemble and various properties are measured, and their moments updated. After that, the next step is immediately attempted to extend the length of the SAW. On the other hand, if the attempt fails (because that neighbor has been visited by the same SAW before), then this branch of the SAW is finished. Thus, the SAW must retract one step, and begin a new branch in the depth-first search with the same preference order of the search directions. When the search is retracted all the way to the starting point, the whole *tree* is complete, and control returns to the calling program.

3 Parallelization with EcliPSe

For this experiment, we ran the parallelized version of the polymer chain simulation on *eleven different heterogeneous computing environments* dispersed across the country. The various machines used to build these network configurations are shown in Figure 1 (see [16]). The simulation was done in parallel across the machines named in each configuration, with results being combined at monitoring nodes in the tree-structured virtual machine. The configurations we used are:

hours, days and even weeks on distributed systems. We achieve fault-tolerance by building a controllable amount of statistical redundance into the tree structure, so that if a node fails, only those nodes in the subtree which have the failed node as a parent must reorganize themselves in the tree. This scheme ensures that the entire computation does not fail, while simultaneously guaranteeing unbiased statistics in sampling-oriented parallel statistical computations [6].

2 Polymer Chains and Scale-Invariant Phenomena

The particular problem studied in this work is one in which some fundamental aspects of the statistical mechanics of polymer solutions [4] are investigated. An especially delicate and controversial problem in this area is that of the conformational properties of polymer chains in disordered media. The question is how the conformation of a macromolecule in solution is affected when the solution is confined to a quenched random medium. The potentially relevant experimental realizations include polymers trapped in porous media (such as a sandstone), transport of polymer through microporous membranes, and gel permeation chromatography. Despite rather intensive theoretical and numerical studies in recent years, the problem remains unsolved, with results which seem to contradict one other.

Besides interest from the materials application point of view, there is also a more fundamental theoretical interest. This is due to the close relationship that exists between the problem of long chain polymers and that of scale-invariant phenomena at the critical point [19] (such as in liquid-gas phase transition or ferromagnet-paramagnet phase transition). This relationship has been established theoretically and demonstrated experimentally in a most convincing manner by various French groups, such as that of de Gennes [3] and that of Jannink [8]. It is the recognition of such links, based on the ideas of *scaling* and *universality*, among seemingly totally unrelated problems, that led to the recent Nobel Prize for de Gennes. In the case of polymers in disordered media, it is possible to have two different critical behaviors competing simultaneously, since the polymer size can be very large and the disorder in the medium can be long-range correlated. Such competition of two large *length scales* (in the jargon of statistical physics) makes this problem a particularly delicate and fertile ground for theoretical investigations using the tools of critical phenomena.

In this work, we focus on a linear chain which has a restricted interaction with the medium; that is, there are forbidden regions (infinite energy barrier), and the chain is confined to other parts of the medium. If the forbidden regions occur randomly and the minimum length scale of these is much smaller than the size of the polymer in homogeneous media, then this problem can be modeled, after suitable graining, by a self-avoiding walk (SAW) on a randomly diluted lattice. Thus, we first create a *percolation cluster* [20] on a finite grid (of size L^3) by randomly diluting the grid (i.e., removing sites as inaccessible to the chain) with probability $1-p$. The remaining sites then form connected components called *clusters*. Above a certain threshold p_c, there exists one cluster that spans the grid from end to end, and this is the cluster of interest. On this *disordered* cluster, a starting point of a SAW is chosen randomly, and then *all* SAWs of a predetermined number of steps N are generated by a depth-first type search algorithm. At each of the N steps, various conformational properties of the chain are measured, such as the moments of the end-to-end distance R_N and radius of gyration S_N and those of the number of chains C_N. These are then averaged over the ensemble of SAWs on the particular disorder configuration. This is repeated for a large number of disorder configurations, and finally both linear and logarithmic means are calculated over the disorder ensemble.

By virtue of the link between long-chain polymers and scale-invariant phenomena at the critical point, each of the quantities that are calculated will have a *universal* dependence on N for large N, involving power-laws with appropriate characteristic *critical exponents*. Here *universal* means that the values of these exponents do not depend on many details of the system but only on a few fundamental specifications of the system (such as the dimensionality, and whether the chain is linear or branched). The question of main interest is whether these exponents are affected by the presence of disorder, particularly at the critical threshold p_c of the dilution parameter (at which the medium has an infinitely long-ranged correlation [20]). Previous calculations, both analytical [5, 12, 13] and numerical [9, 11, 10], have resulted in a whole range of possible answers to this question. The delicate nature of this issue can be attributed to the wide distributions of the quantities measured, particularly at p_c where the structure of the cluster is intricately *self-similar* [14].

Superconcurrent Simulation of Polymer Chains on Heterogeneous Networks

H. Nakanishi and V. Rego

Departments of Physics and CS
Purdue University
W. Lafayette, IN 47906

V. Sunderam

Department of CS
Emory University
Atlanta, GA 30322

Abstract

We present the results of an experiment on massively parallel stochastic simulation of polymer chains for measuring scale-invariant phenomena at critical temperatures. The parallelization is achieved through the EcliPSe toolkit, and conducted on a flexible, tree-structured virtual machine made up of arbitrary and heterogeneous computing nodes dispersed across the country. These nodes cooperate to perform the simulation and pool results together in real time at a central node which initiates the parallel simulation. The advantage of the tree-structure is that it allows for a fault-resilient, flexible environment for long-runing parallel simulations. We present results on sheer performance, price performance, and toolkit-based parallelization by monitoring the same computations on a CRAY Y-MP and making detailed comparisons. Most significant are the excellent price-performance ratios given by the heterogeneous computing environment.

1 Introduction

Since its use at Los Alamos during World War II, stochastic simulation [18] has gained increasing acceptance as a viable tool for understanding the behavior of complex phenomena. In a recent article on High Performance Computing [2], Gordon Bell describes "Large-Scale Scientific Experiments Based on Simulation," to be one of six key application areas. Nobel Laureate Ken Wilson characterizes [2] computer simulation as the third paradigm of science, supplementing theory and experimentation, a paradigm shift that will transform every facet of science, engineering and mathematics.

In essence, stochastic simulation is a technique that allows an analyst to mimic a stochastic process of interest. Using pseudo-random numbers, a high speed computing machine can be made to yield realizations of a random process several times faster than working directly with a physical process. In many instances speed is not the only factor since the cost of working with a physical process may be prohibitive. Stochastic simulation is therefore an invaluable aid in studying stochastic phenomena, a tool that is likely to grow in importance with the complexity and size of problems which, not yielding to deterministic methods, must be solved via sampling.

Even though the speeds of modern uniprocessors now make possible simulations that were either too time-consuming or simply impossible to conduct a decade ago, there is an ever present need for reducing model execution time. When one considers the amount of computational power that we have distributed across the country, connected by high-speed links, it stands to reason that there must be a way of harnessing massive parallelism in a flexible manner, from arbitrary machines in a network. With the twofold objective of providing a transparent mechanism for harnessing this power for complex data-parallel applications, and simultaneously providing the computational scientist with interactive access to this power without having to learn parallel programming, we embarked on the EcliPSe project [17, 21]. A description of EcliPse can be found in the accompanying publication.

In this report, we provide an executive summary of the performance of the high performance, hierarchical and fault-tolerant concurrent simulation system, known as EcliPSe [17], on an important class of applications in polymer physics. By hierarchy, we imply that computations are performed by arbitrary processors which arrange themselves in a tree-structure. As will be explained later (see Section 3.1), this allows for fault-tolerant computations. Such fault-resilience is necessary for simulation applications that run for