Massively Parallel Architectures and Polymer Simulation

B. Ostrovsky,[a] M. A. Smith,[b] M. Biafore,[b]
Y. Bar-Yam,[a] Y. Rabin,[c]
N. Margolus,[b] and T. Toffoli[b]

[a] ECS, 44 Cummington St., Boston University, Boston MA 02215
[b] MIT Laboratory for Computer Science, Cambridge, MA 02139
[c] Department of Physics, Bar-Ilan University, Ramat-Gan 52900, Israel

A new approach to polymer simulation well suited to massively parallel architectures is presented. The approach is based on a novel two-space algorithm that enables 50% of the monomers to be updated in parallel. The simplicity of this algorithm enables implementation and comparison of different platforms. Such comparisons are relevant to a wide variety of scientific applications. We tested this algorithm on three commercially available machines, the MP-1, KSR1, and CM-2; and on a prototype of the CAM-8 architecture. Among the commercial machines we found the MP-1 provided the best performance for highly-parallel fine-grained simulations. Effective utilization of the KSR1 was achieved with attention to synchronization requirements. The small (8 node) CAM-8 prototype, with a kind and cost of hardware comparable to an engineering workstation, achieved a performance within a factor of two of the MP-1 for our application.

I. Introduction

The dynamic properties of natural and man-made polymeric systems are of intense academic and industrial interest. Understanding how these properties arise from underlying microscopics is a central problem of the field. Computer simulations will play a central role since intuitive arguments are difficult and analytic results are virtually impossible. However, the large number of conformations of a long polymer make conventional computer simulations prohibitive. The advent of parallel computation promises to enable a large number of breakthroughs in this field.

In order to improve our ability to simulate the complex behavior of polymers, we have introduced[1,2] a new approach which is well suited for parallel processing. In its most general form, the approach may be used to simulate macromolecules ranging from simple polymers to complex polymers such as proteins. While polymer interactions are non-local along the chain, by recognizing the fact that they are local in space it is possible to develop a general domain-decomposition approach to parallel processing of polymer dynamics.

The concept of space-oriented dynamics is manifest in the very general category of dynamical models known as Cellular Automata. We have developed two general cases of Cellular Automaton dynamics that can simulate abstract models of high molecular weight polymers, grafted polymers, and block-copolymers in a variety of media, including boundaries, obstacles and constrictions. The second class of algorithms that incorporate a novel two-space concept, are particularly efficient for simulation because fully 1/2 of the monomers can be updated in parallel, the long-polymer limit is reached for very few monomers and the prefactor for the dynamical relaxation is small. Using this algorithm, the simulation of polymer dynamics is an ideal candidate for leading the exploitation of the power of parallel architectures.

The two-space algorithm is simple and easy to implement. Due to the simplicity of the algorithm, it is well suited to an evaluation of the performance of diverse architectures on fine-grained parallel problems and coarse-grained parallel performance through task aggregation. Aggregation both by space partitioning as well as polymer partitioning is possible. We have implemented the algorithm on a variety of computer architectures including both serial and parallel machines. This paper describes briefly the implementation of the two-space algorithm

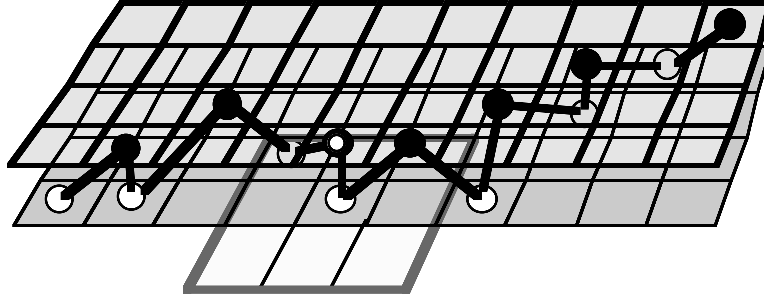

Fig. 1: Schematic illustration of a two-space polymer used in two-space polymer dynamics described in the text. Monomers on the upper plane are shown as filled circles, monomers on the lower plane are shown as open circles. Monomers are attached (bonded) only to monomers in the other plane. The 'bonding-neighborhood' of each monomer is a 3x3 region of cells located in the opposite plane. A lightly shaded region indicates the bonding-neighborhood of the black monomer marked with a white dot. Its two nearest neighbors are located in this bonding-neighborhood. Bonds are indicated by line segments between monomers.

and testing of various parallel architectures. The results of simulations on 2-dimensional high-density melts and the simulation of polymer collapse are described elsewhere.[3,4]

Tests were performed on three commercially available machines, the Thinking Machines CM-2 and MasPar MP-1 and Kendall Square Research KSR1; and on a prototype of the CAM-8 architecture. On the commercial machines vendor-supplied versions of the C-compiler were used. Initial tests performed on a 64-node CM-5 were determined to be inconclusive due to unoptimized virtual processor assignment by the beta-release version of the C-compiler and are not reported here. Since access was not available to a full MP-2, tests on this machine are also not reported. In order to compare the performance of the parallel and serial architectures the following timing tests were performed on the indicated machines:

(1) Time to update a 2-dimensional space containing only one polymer of length two -- one monomer in each space (MP-1, KSR1, CAM-8, all tested serial machines - see Table I).
(2) Communications: Time to transfer 1 byte of data between two processors (CM2, MP-1).
(3) Time to update a 2-dimensional space containing a high-density melt with variable melt-density (CM-2, MP-1, CAM-8, IBM RS/6000 320H).
(4) Time to transfer a lattice between the parallel unit and front-end (MP-1, CAM-8).
(5) Synchronization time (KSR1).

The paper is organized as follows. Section II describes the algorithm. Section III presents the tests on serial architectures. Section IV lists the parallel architectures tested and briefly describes CAM-8. Section V presents the tests on parallel architectures.

II. Algorithm

In the two-space algorithm[1,2] monomers of the polymer alternate between two spaces so that odd-numbered monomers are in one space and even-numbered monomers are in the other. The nearest neighbors along the contour of a monomer are located in the other space. This enables the constraint of maintaining connectivity to be imposed by considering only the positions of monomers in the opposite space. Moreover, the excluded volume constraint is also imposed only through interactions of a monomer with monomers in the opposite space. Even though excluded volume is not explicitly imposed between two monomers in the same space, it arises indirectly through the interactions with the opposite space. Despite the unusual implementation of connectivity and excluded volume, the static and dynamic scaling properties of long chains are preserved.

The advantages of this algorithm include flexible dynamics (fast relaxation times) as well as effective parallelization. The dynamics is flexible because nearest neighbors which are in opposite spaces can be 'on top of each other' so that local expansion and contraction is possible.

More interestingly, it is possible to move all of the monomers in one space in parallel because both connectivity and excluded volume are implemented through interactions with the other space. The possibility of updating half of the monomers in parallel opens a wide range of parallelization schemes.

III. Serial Architectures

Before implementing the algorithm on parallel architectures it has been extensively tested on several serial machines.

(a) Scaling Tests

The two-space polymer dynamics is an abstract polymer model with an unusual implementation of excluded volume. The unusual local interaction does not affect the asymptotic structural and dynamical behavior, which is in the same universality class as other abstract polymer models. This was confirmed by simulating the scaling of known quantities. In Fig. 2 we plot the radius of gyration of a polymer as a function of its contour length and compare with the exact result in two dimensions, $R \sim L^{0.75}$. In Fig. 3 we calculate the longest relaxation time of the chain and compared with the Rouse model prediction, $\tau \sim L^{2.5}$.

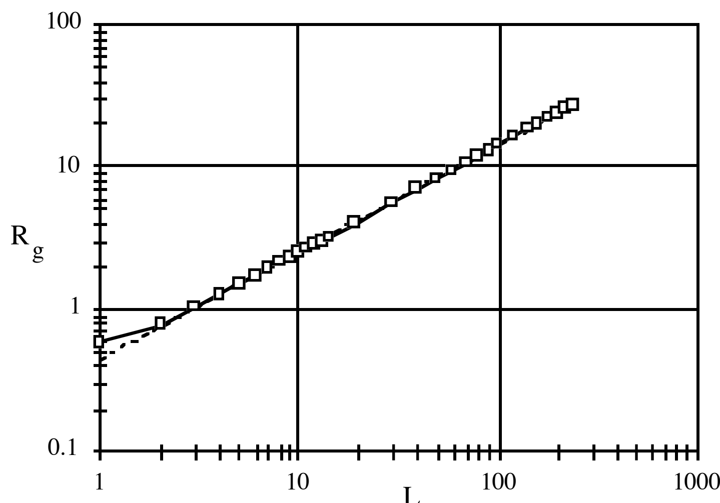

Figure 2: Simulations of the radius of gyration of a polymer R_g, as a function of the polymer length L (one less than the number of monomers), testing the two-space algorithm. An asymptotic fit indicated by the dashed line is $R_g \sim 0.8\ L^{0.751}$ consistent with the exact exponent 0.75. Agreement with the asymptotic values are reached for remarkably small polymers of length L=2.

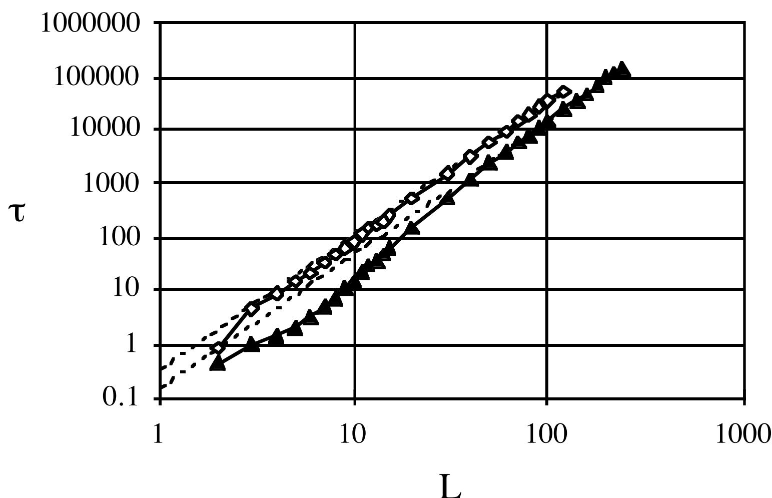

Figure 3: Simulations of the relaxation time τ of a polymer as a function of polymer length L, testing the two-space algorithm (s). Simulations were performed on a RISC station. The asymptotic fit indicated by the dashed line was obtained as $\tau \sim 0.12\ L^{2.55}$ consistent with the Rouse model exponent of 2.5. The small prefactor indicates the efficiency of the two-space algorithm. Polymers of length 30-50 already approach the asymptotic behavior. For comparison the bond-fluctuation algorithm is shown (Δ) with a relaxation time 3 times larger. The single step speed is also 3 times slower (see text).

(b) Comparison with bond-fluctuation method

One of the simplest algorithms for effective simulation of abstract polymers is the bond-fluctuation method[5] which allows bond lengths to vary so as to allow more flexible polymer motion on a single

lattice. The bond-fluctuation method is not especially well suited for parallelization. It is nevertheless of interest to compare the efficiency of the two algorithms on a serial computer. A comparison of relaxation times reveals that the two-space algorithm is more flexible reducing the relaxation time for equal length polymers by a factor of three - see Fig. 3. Furthermore, the intrinsic monomer update speed of the bond-fluctuation method is also slower by a factor of 3. In part, this arises from the necessity of randomly selecting a monomer. The inherent parallelizability of the two-space algorithm enables the monomers to be updated systematically in a space, only the space must be selected at random. The combination of relaxation time and update speed leads to an overall speed up of a factor of 10 of the two-space algorithm from the bond-fluctuation algorithm for polymers of equal numbers of monomers.

(c) Timings on serial platforms

Timings were performed on IBM, DEC and SUN workstations. The algorithm does not require any floating point operations, thus the speed should roughly follow the MIPS rating. The results are shown on Table I in order of decreasing speed. In order to make an update a processor scans through a list of the monomers, rather than scanning the lattice. For serial architectures list processing gives much better times than space processing (see also the section on coarse-graining). The IBM RS/6000 gives the best performance, even when comparing medium rated model 320H with highest end of DEC5000 family. In all tests vendor supplied versions of the Kernighan&Ritchie compiler were used. About 15% better performance was achieved on the SPARCs with the gcc 2.2.2 compiler.

Table I: Timings (in seconds) of serial workstations performing one million space updates of one polymer of length 2 on a 128x128 lattice.

IBM RS/6000 550	IBM RS/6000 320H	DECstation 5000/240	DECstation 5000/133	SUN SPARC 2	SUN SPARC 1^+	SUN SPARC 1
2.6	3.8	6.0	6.4	6.4	10.3	12.6

IV. Parallel Architectures

(a) Machines tested

Performance tests on various parallel (Table II) and serial architectures were made possible by grants of time from the MasPar Computer Corporation (MP-1), Kendall Square Research (KSR1), Cornell Theory Center (KSR1 and IBM RS/6000 550), Boston University Center for Computational Science (CM-2), and the Boston University Polymer Center (DECstations). Tests of the CAM-8 architecture (see below) and SPARCs were performed at the MIT Laboratory for Computer Science.

Table II: Parallel architectures used in performance tests. Note: (1) On the CM-2 and MP-1 nearest neighbor communications are the fastest (called NEWS and xnet correspondingly) and were used for the testing. (2) On the KSR1 memory is physically distributed but logically shared among the processors. Each processor stores a segment of memory in its cache. Cache consistency is ensured by dedicated hardware.

	processors		communication	list price
	number	type		
MP-1 1208	8192	4-bit custom made with floating point	2-D mesh 128x64	$530,000
KSR1	64	64-bit custom made	Shared Memory	unpublished
CM-2	8192	1-bit custom made	32-dimensional Hypercube	$540,000
CAM-8 (8 node)	8	16-bit (lookup table)	3-D mesh	est.<$30K

(b) CAM-8

CAM-8 is an experimental Cellular Automata machine[6] developed at the MIT Laboratory for Computer Science. It is an indefinitely scalable multiprocessor optimized for spatially fine-grained, discrete modeling of physical systems -- such as lattice-gas simulations of fluid flows. CAM-8 implements a uniform 3-dimensional space of cells, each of which can be thought of as a simple processor connected to nearby neighbors.

In CAM-8, uniform spatial calculations are divided up among a 3-dimensional array of hardware modules that are locally interconnected. Each module simulates a volume of fine-grained processors in a sequential fashion, allowing updating and intermodule communication resources to be timeshared, and allowing considerable flexibility in data analysis and display capabilities.

CAM-8 machines are composed entirely of memory chips 'glued' together into an indefinitely extensible bit-sliced mesh. There are no conventional processors: other memory chips are used as lookup-table processors. Thus the machine is composed entirely of commodity memory chips plus a custom DRAM controller chip arranged in a novel architecture.

An initial prototype has been in operation since last summer: it is this small-scale (8 module) machine whose performance is discussed in this manuscript. With an amount and kind of hardware comparable to that in its SPARCstation front-end (64 Megabytes of conventional DRAM and 2 Megabytes of cache-grade SRAM, all running with a 25 MHz clock), this small prototype has already achieved performance on a number of interesting physical simulations comparable to or exceeding that of commercially available machines (only results for polymer simulations are reported here). Machines up to three orders of magnitude bigger and correspondingly more powerful than this prototype can be built immediately, using the existing architecture.

V. Parallel Architectures

(a) Parallel Implementations

There are two approaches to the parallelization of polymer simulations: polymer partitioning, and space partitioning. When partitioning polymers, a processor is assigned to a set of monomers. When partitioning space, each processor is assigned to a number of lattice sites. In this manuscript our focus is on space partitioning.

Within each of the parallelization schemes processor assignment may take advantage of the full parallelization (fine-graining) or tasks may be aggregated (coarse-graining). In fine-grained space-partitioning each processor is assigned to one double-space lattice site. This method is well suited for dense systems where the fraction of occupied sites is high. Parallel architectures suited for fine-grained simulations require fast intersite communication and large numbers of processors. Virtual process assignment is one mechanism for automatic coarse-graining on some architectures. Depending on the implementation this may improve performance, and was not tested here.

In this article we compare the performance of four architectures: CM-2, KSR1, MP-1, and CAM-8. Most of the tests consider fine-grained performance: processors are assigned to individual sites and are synchronized after each site update. For fine-grained simulations we find that the algorithm is communication-bound. Thus a critical characteristics in performance is data transfer rate from one processor to its neighbor. This maximally parallel implementation is an appropriate first test of the performance of parallel architectures. Since the CAM-8 hardware directly simulates fine-grained spaces and performs sites updates synchronously it is included under fine-grained architectures below. The architecture of the KSR1 enables immediate coarse-graining, thus coarse-grained results on KSR1 are also reported.

Among traditional architectures MP-1 has emerged with the best timing and thus has become our choice for the fine-grained implementation of the algorithm. The CM-2 is significantly slower than the MP-1. The KSR1 requires a special discussion, given below, that does not emphasize interprocessor communication but focuses instead on synchronization. The small (8-node) CAM-8 prototype achieved a performance within a factor of two of the MP-1.

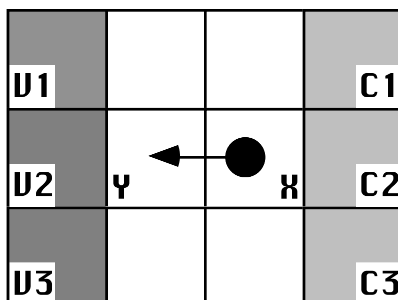

Fig 4. In fine-grained simulation one processor is assigned to a site. Assume the processor assigned to X has a monomer in its odd plane that has elected to move westward. In order to make the step, processor X has to determine that there is nothing in the even plane of processors C1, C2 and C3 (to preserve connectivity) and nothing in the even plane of processors V1, V2 and V3 (to preserve excluded volume).

(b) Fine-grained simulations

Let us consider fine-graining when one lattice site is assigned per processor (or in the case of CAM-8 per virtual processor). This kind of partitioning should allow the use of an architecture with rather high number of relatively slow processors. This approach is well suited for SIMD (Single Instruction Multiple Data) machines. In these machines an instruction is synchronously broadcasted to all processors and executed on data spread over those processors. In this class are the CM-2 and MP-1.

The system is updated in three steps:

a) pick the even or odd plane to update at random;
b) each site that contains a monomer chooses one of four compass directions at random;
c) make a Monte-Carlo step subject to two constraints in the *other* plane: connectivity and excluded volume (Fig. 4);

We see that for each step a processor must communicate with 6 other processors. The total number of communications required depends on the connectivity of the processors. For example, assuming single step communication with eight nearest neighbors the total number of communications is 9. If the same operations are applied to all processors the total number of operations to enable movement of monomers in all four directions is 36. We decrease the number of communications by gathering the information in two stages. We first combine data from C1, C2 and C3 into C2 and combine V1, V2 and V3 into V2. This takes 4 communications per plane update. Then the information may be transferred in 3 steps to X. Additional optimization is realized by noting that the same information may be used either at X for moving west or at Y for moving east. Thus data is collected in vertical and horizontal strips. The result of vertical (horizontal) data gathering is shifted left and right (up and down). The total number of communications per plane update is 4 + 4*3 = 16, compared to the previous 36. Note that with this optimization only NESW communications are needed. Finally, four more communications are needed to move the monomers. We emphasize the number of communications because the communication to computation ratio is large in fine-grained implementation. Computation requires only random number generation, evaluation of a compound logical expression using communicated data, and clearing a variable when a monomer 'leaves' the site. CAM-8 also uses a similar data-gathering technique but multiple data-gathering operations occur in a single cycle and communication is combined with the cell update as a single pipelined operation.

Table III: Communication rates between processing elements (PEs). Measured using 1,000 communications of 1 byte between all processors of a 128x64 array at indicated distance using NEWS and xnet. Using library calls on the CM-2 gave identical results to those in the table. Within the virtual processor environment of the 8-node CAM-8 prototype the corresponding number would be 41μs independent of separation.

Communication task	CM2	MP-1
send 1 byte of data 1 PE away	112.3 μs	4.0 μs
get 1 byte of data from 1 PE away	64.5 μs	4.6 μs
send 1 byte of data 2 PE's away	162.4 μs	5.0 μs
get 1 byte of data from 2 PE's away	78.7 μs	5.5 μs

Measurements of data transfer times for 128x64 arrays on CM-2 and MP-1 are shown in Table III. The MP-1 data transfer rate is 12-30 times faster than that of the CM-2. Moreover, MP-1 uses full 4-bit processors while CM-2 has only 1-bit processors, thus we expect overall performance of the MasPar machine to be significantly better. Indeed, simulation results confirm this as shown in Table IV.

Space partitioning indeed gives good results for fairly dense systems. Fig. 5 shows timings on simulating melts of different densities comparing serial and parallel machines: IBM RS/6000 320H and MasPar MP-1. Similar to the simulations in Table I the workstation used list processing of monomer moves. Runs were performed on a 128x64 lattice with increasing number of polymers of length 30. The highest density corresponds to a density of 0.47 monomers per site. Each system made 10^4 updates of a space. For systems with less than 6 chains the single processor outperforms the parallel computer. However, as the density increases the ratio of times increases up to 20.

Table IV. Number of updates of a space per second for parallel architectures. Measurements for CM-2 and MP-1 were done on 128x64 lattice. CAM-8 simulations were performed on a 512x512 lattice and the result was multiplied by 32 to be comparable with the two previous tests. For these systems the number of monomers in the space is irrelevant since the whole space is updated simultaneously. Melt simulations on each architecture yielded equivalent times (See Fig. 5). The advantage of multiple operations in a single cycle is apparent in the performance of CAM-8.

CM-2	MP-1	CAM-8
237.5	6849.3	3382.6

For many parallel architectures, periodic data analysis is often not easily parallelized. Both MP-1 and CAM-8 enable high speed data transfer between the parallel processors and the front-end. Measurements are shown in Table V. For the MP-1 transfer of char data (1 byte) is approximately the same as int data (4 bytes). This can be used for packing data. Without packing the 'round-trip' time to transfer 'two spaces' 128x64 front-end to DPU and back is 17 ms. The update time of one space is about 0.1 ms. Since data transfer is typically required after hundreds or thousands of updates this overhead is small.

Table V. Times (in milliseconds) to transfer array of data round-trip between front-end and parallel unit for different sizes of arrays and different sizes of data. The numbers for MP-1 are 'user times'. Actual time (including system overhead) may be as much as two times higher.

Size of array	MP-1 char (1 byte)	MP-1 int (4 bytes)	CAM-8 short (2 byte)
64x64	5.59	5.68	4.72
64x128	8.55	9.21	5.18
64x128 + 64x128	16.46	17.95	7.28
512x512			78

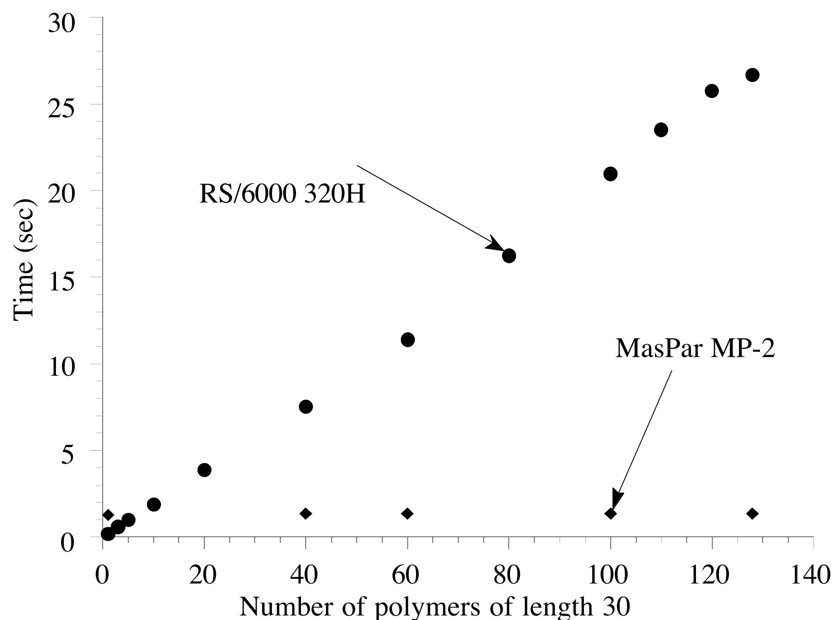

Fig. 5: Comparison of serial (IBM RS/6000 320H) and parallel (MP-1) architectures performing increasing density simulations of 10^4 space updates of a 128x64 lattice. List processing was used on the serial computer, fine-grained space processing on the MP-1.

(c) Coarse-graining

In architectures with powerful individual processors task aggregation by coarse-graining enhances the performance. In this article we consider only coarse-graining in the KSR1 because of its ease of implementation for this shared-memory architecture.

For coarse-grained space partitioning, each processor is assigned to a region of the lattice rather than to a single site. For simplicity we will assume that these subregions are of rectangular shape. The basic structure is shown in Fig. 6.

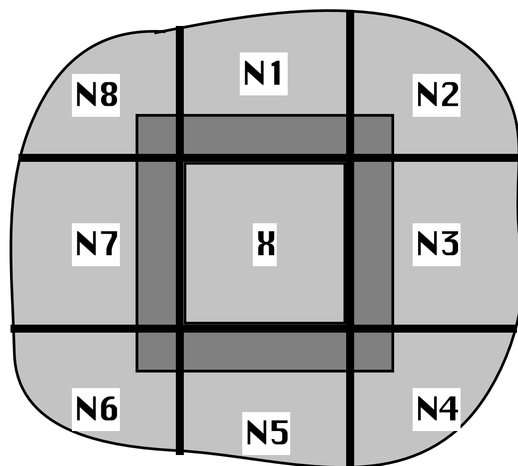

Fig 6: Schematic drawing of the processor assignment for coarse-grained simulations by space partitioning (see text).

Individual processors are assigned to regions X, N1-N8. The following steps have to be made in order to update a space:

1. One of two spaces is chosen.
2. All processors synchronize.
3. Processor X receives information from its 8 neighbors N1-N8 about moves performed on monomers within two lattice units from its boundaries. Shown in gray in Fig. 6.
4. Each processor updates its assigned region. Here we use space scanning to update each partition. However, in dilute systems list processing may be used within each region.

This method does not require synchronization before each *site* update, but rather only before the update of a whole *space*. Thus in a MIMD architecture each processor can work independently during the space update. Since communication between processors only describes the boundary regions the frequency of communication is reduced. Moreover, all of the information transferred between two processors before each space update may be combined into one message. However, adequately fast synchronization is required. When single update synchrony is not maintained, explicit synchronization must be performed between space updates.

In a shared memory architecture explicit programming of the communication is not necessary since each processor addresses the common memory space. We find synchronization to be a key factor in KSR1 performance. Fig. 7 shows time measurements taken with and without synchronization between processors. Without synchronization the simulation is incorrect. However, this test provides information on the effect of synchronization on performance. As the number of processors increases synchronization requires more time.

Timings were performed with different numbers of processors for 10^4 updates of a 128x128 system containing 1 polymer of length 2. However, spatial scanning by each processor is still performed. With synchronization the minimum time is for 32 processors and is about 31 seconds. For 64 processors the synchronization dominates on this trial system. The individual KSR1 processors are slower by a factor of 2 from the IBM RS/6000 320H (178 seconds for the

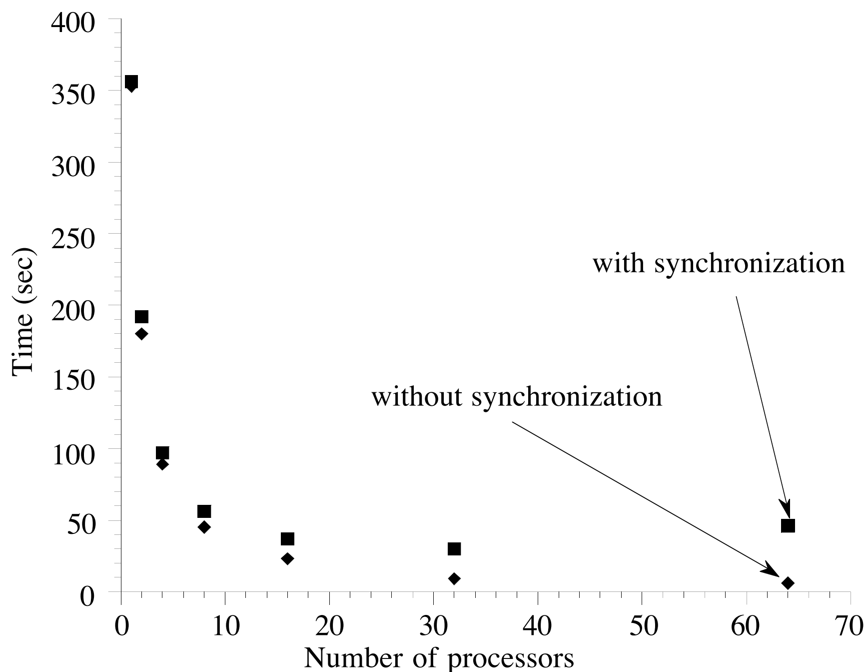

Fig. 7: Comparison of timings on the KSR1 with and without synchronization on 10^4 updates of a 128x128 system containing 1 polymer of length 2. Optimal times of the parallel architecture on this trial system are for 16-32 processors. More processors may be used effectively on larger systems.

same space scanning test as opposed to 360 seconds on a single KSR processor). Despite the lower individual processor speed a total speed-up by a factor of six compared to the IBM RISC is realized. The importance of synchronization will vary between applications. For example, by increasing the size of the space 64 processors will be more effectively utilized.

(d) Random Number generators

Random number generation is an important part of the simulations that have been described. Indeed even on serial computers significant time savings can be achieved by recognizing that in order to choose a space only one random bit is needed, and for selecting a movement direction only 2 random bits are needed. Since typical random numbers are 4 bytes long the individual bits may be used reducing the costly random number calls. The performance improves significantly. For example, for the two-monomer system optimizing only the space selection on IBM RS/6000 320H time decreased from 4.76 to 3.47 for 1 million updates on a 128x128 lattice. Thus special attention must be placed to optimizing random number generation on parallel architectures.

We have encountered significant difficulties in use of random number generators on both MP-1 and KSR1 architectures. Our simulations make use of our own parallel random number generator.

The MP-1 parallel random number generator generates extremely correlated numbers: for example for 12 out of 100 calls we obtain equal last two bits in all processors.

The KSR1 C language `rand()` is not a parallel random number generator. Instead, when more than one processor calls it simultaneously, these calls are serialized. However, there is now available on all KSR machines a function `prng()`, developed at Cornell,[7] that is fully parallel.

The CAM-8 architecture is well suited for random number generation because simultaneously with other calculations random bits may be generated as part of a site update.

We strongly recommend that manufactures pay special attention to the performance of random number generators.

We wish to acknowledge the assistance of B. Wheelock of MasPar Computer Corp., G. Vichniac of Kendall Square Research, R. Putnam of Thinking Machines Corp., and J. Zollweg of the Cornell Theory Center. We thank R. Giles for helpful discussions.

---

[1] Y. Bar-Yam, Y. Rabin, and M. A. Smith, Macromolecules Reprints, 25, 2985-6 (1992)

[2] M. A. Smith, Y. Bar-Yam, Y. Rabin, C. H. Bennett, N. Margolus and T. Toffoli, in *Complex Fluids*, (E. B. Sirota, D. Weitz, T. Witten and J. Israelachvili eds.) MRS Symp. Proc. Vol. 248 (1992) p. 483; Journal of Computational Polymer Science (in press)

[3] M. A. Smith, B. Ostrovsky, Y. Rabin, Y. Bar-Yam (preprint in preparation).

[4] B. Ostrovsky, Y. Bar-Yam (preprint).

[5] I. Carmesin and K. Kremer, Macromolecules 21, 2819 (1988)

[6] For more hardware details, see N. Margolus and T. Toffoli, in *Lattice Gas Methods of Partial Differential Equations* (Doolen et al. eds.), Addison-Wesley Longman Publishing Group Ltd. (1989).

[7] O. E. Percus, and M. H. Kalos, J. of Parallel and Distributed Computing, 477 (1989).