Journal of Computational Physics 231 (2012) 98–108

Contents lists available at SciVerse ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

A new parallel simulation technique

Jose J. Blanco-Pillado, Ken D. Olum [*], Benjamin Shlaer

Institute of Cosmology, Department of Physics and Astronomy, Tufts University, Medford, MA 02155, USA

A R T I C L E I N F O

Article history:
Received 17 November 2010
Received in revised form 21 July 2011
Accepted 30 August 2011
Available online 12 September 2011

Keywords:
Simulation
Parallel processing
Cosmic strings

A B S T R A C T

We develop a “semi-parallel” simulation technique suggested by Pretorius and Lehner, in which the simulation spacetime volume is divided into a large number of small 4-volumes that have only initial and final surfaces. Thus there is no two-way communication between processors, and the 4-volumes can be simulated independently and potentially at different times. This technique allows us to simulate much larger volumes than we otherwise could, because we are not limited by total memory size. No processor time is lost waiting for other processors.

We compare a cosmic string simulation we developed using the semi-parallel technique with our previous MPI-based code for several test cases and find a factor of 2.6 improvement in the total amount of processor time required to accomplish the same job for strings evolving in the matter-dominated era.

1. Introduction

A common task in computational physics is the simulation of some large physical system, described by hyperbolic differential equations. If the system is too large to be represented on a single computer, or the resulting simulation would be very slow, one simulates it using a number of processors (cores) working in parallel. Typically the spatial volume to be simulated is divided into regions, and each processor handles one region. At the boundaries of the regions the processors must communicate using some protocol such as MPI.

This technique has been used successfully for many simulations, but it is not without drawbacks. First, the processors must all be available simultaneously. In certain cases the number of processors must have special properties, such as being a perfect cube. In any case, the largest domain that can be simulated is limited by the number of available processors multiplied by the capability of each. If one processor crashes, due either to hardware failure or problems with the code, the entire simulation must terminate.

Furthermore, for certain simulations there is a serious problem of load balancing. This poses no difficulty if the processors are identical and the amount of work that each must do is the same. For example, to evolve equations of motion on a uniform grid one performs the same operations on each point, so if all processors handle the same number of points, there is no load balancing problem. But for other cases, the work is very different. In the cosmic string example to be discussed below, at late times many volumes are free of string and incur no simulation overhead, while others have densely-packed string points and require much work. When the load is not balanced in a conventional parallel simulation, all processors wait for the slowest processor, and perhaps only a tiny fraction of the total processing power can be utilized.

We describe here a different division of simulation work among processors. The idea is originally due to Pretorius and Lehner [1, Section 4.3], although we discovered it independently and did not learn of their work until later. The fundamental difference is that instead of dividing the spatial volume to be simulated into as many regions as we have processors, we

* Corresponding author. Tel.: +1 617 627 2753.
E-mail addresses: jose@cosmos.phy.tufts.edu (J.J. Blanco-Pillado), kdo@cosmos.phy.tufts.edu (K.D. Olum), shlaer@cosmos.phy.tufts.edu (B. Shlaer).

0021-9991/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2011.08.029

divide the spacetime 4-volume to be simulated into a much larger number of 4-dimensional regions. Each available processor will simulate many of these regions. We construct the regions in such a way that they take in information through some initial surfaces and produce information that is transmitted through some final surfaces, but there are no surfaces with a two-way flow of information [1]. Such regions are known as *globally hyperbolic*.

Since there is no two-way information flow, there is no need for multiple processors to be running simultaneously. Instead, the simulation of each region is a well-defined task that one processor can perform alone, given only that it has the initial-surface information provided by predecessor regions. Thus even a single processor could perform a simulation of arbitrary size without concern about memory usage, by simulating the 4-dimensional regions one by one. The total simulation volume is thus not limited by the available number of processors, although of course large simulations run on few processors will take a great deal of time to complete.

Furthermore, there is no issue of load balancing. The processors run independently, so no processor ever sits idle waiting for others. Suppose one region takes much longer than others. Its successors must wait for it to finish, but other regions can be run simultaneously, even if they are at later times but distant in space. Eventually all regions that are not successors (directly or indirectly) of the slow region will be completed. In this case the simulation is running on a single processor only, but no other processors are tied up waiting for the slow region to complete. They can be used to run other simulations or unrelated tasks.

In order for this technique to work, it must be possible to construct regions with the proper causal structure. In the case of hyperbolic differential equations with a maximum speed of propagation in flat space, this can be done. For example, in a simulation of causal physics in spacetime, the propagation of information is limited by the speed of light. A null 3-surface divides spacetime into a future and a past side; it is not possible for any information to propagate across it in the reverse direction. Thus if the regions are divided by null surfaces, the necessary conditions apply. In other areas of research, the propagation speed might be the speed of sound or some other limiting speed. As long as such a speed exists, the necessary division of spacetime can be made.[1]

This technique is not applicable to situations where conditions in one part of the simulation have an immediate global influence, because then there is no limit to the propagation speed. It also does not apply in cases where the propagation speed cannot be known in advance, for example situations involving rapidly flowing fluids where the propagation speed of sound waves could be significantly affected by the underlying flow.

As presented here, our technique does not work in general curved spacetimes. The problem is that a division of spacetime by null surfaces generically does not remain well behaved into the future[2]. A null surface might develop caustics, where nearby null generators intersect. Beyond such points, the surface no longer divides spacetime into past and future subregions. See Ref. [2] for a discussion of such issues. Furthermore, different "parallel" null surfaces could in fact intersect. However, conformally flat spacetimes (which include the expanding universe cases discussed below), will be free from such pathologies. In more general cases, it may be possible to address caustics and related issues by allowing our surfaces to become spacelike, which need not affect the general structure of the algorithm.

Pretorius and Lehner [1] described the division of a simulation with 2 spacetime dimensions. They also discussed simulations with additional spatial dimensions, but their division was done only in one spatial dimension and in time. In those cases the division can be made along null lines or surfaces, and the geometry is straightforward. Here we extend their work to simulations with 4 spacetime dimensions, all of which participate in the division of labor. The additional dimensions require more complicated geometry, and, as we discuss below, the division surfaces must be spacelike, rather than null.

The rest of this paper is organized as follows. First we discuss the "semi-parallel" simulation technique in some detail. We feel that this technique may have applicability to other problems, and so attempt to make it accessible to other practitioners. In Section 2 we discuss the geometrical issues involving division of the simulation 4-volume. Because of the difficulty of representing 4-dimensional objects in figures, we will sometimes illustrate the case of 3 spacetime dimensions instead. We discuss the choice of region size in Section 3 and the management of a simulation consisting of very many regions in Section 4.

Having described the basic idea, we then discuss our cosmic string simulation as an example of the semi-parallel technique. In Section 5 we give an introduction to cosmic strings and their simulation and discuss how we parallelized a cosmic string simulation code. In Section 6 we compare the performance of this technique with that of conventional parallelism. We conclude with a summary and discussion in Section 7.

2. Geometry

In [1], Pretorius and Lehner considered the two-dimensional case and used square regions oriented with the time direction diagonal, so that the edges would be null lines, as shown in Fig. 1. We generalize this to four dimensions and use 4-cubes, oriented so that the time direction lies along a main diagonal. This does not give null 3-surfaces as the boundaries, but we can fix the problem by rescaling the time coordinate.

Let us work in units where the speed of light (or the maximum speed of information flow) is 1. We choose our regions to have unit edges and consider a region whose past vertex is at the origin and whose future vertex is thus at $(x,y,z,t) =$

[1] Of course if the speed of information is too fast, the resulting regions may be too small for efficient simulation.

[2] We thank the referee for calling our attention to this issue.

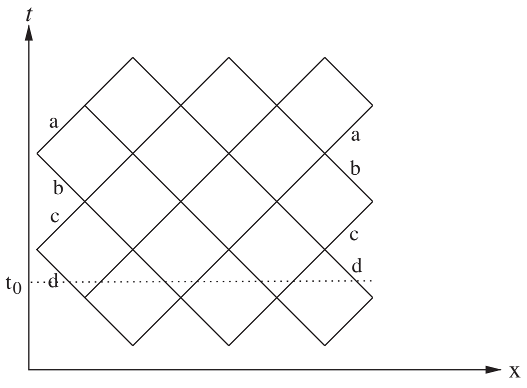

Fig. 1. A 2-dimensional spacetime volume divided into 15 regions by null lines. Each square represents a region that can be simulated separately, using only conditions read in from predecessor regions. If the edges with matching letters are identified, the volume becomes periodic in the one spatial dimension. The dotted line represents a possible initial time for a simulation using these regions.

$(0,0,0,2)$. The simulation regions make up a 4-cubical lattice. We can choose the four future-directed generators of this lattice to be $(1/2,1/2,1/2,1/2)$, $(1/2,-1/2,-1/2,1/2)$, $(-1/2,1/2,-1/2,1/2)$, and $(-1/2,-1/2,1/2,1/2)$. Then the four initial surfaces of our region are 3-cubes that end at time 3/2 at spatial positions $(-1/2,-1/2,-1/2)$, $(-1/2,1/2,1/2)$, $(1/2,-1/2,1/2)$, and $(1/2,1/2,-1/2)$. The main diagonal of such a 3-cube has spatial length $\sqrt{3}/2$ and temporal length 3/2. To make it null, we can shrink our time coordinate by factor $\sqrt{3}$.

If we follow a generator into the future and then another generator into the past, we get a new region whose starting time is the same as that of the original region. Thus the regions can be arranged into "layers" that share a common starting time. The analogous division of a 3-dimensional spacetime volume is shown in Fig. 2.

2.1. Initial conditions

We will start our simulation at some particular initial time t_0. The initial time surface will cut through the 4-lattice of simulation regions. The 3-dimensional analogue is shown in Fig. 3, and the 2-dimensional analogue in Fig. 1. There will be some cubes whose final vertex lies after t_0, but all of whose other vertices lie at or before t_0. Such a cube has no predecessors. We can generate initial conditions inside this cube, and then evolve them until our ending point, writing out information on the four final surfaces to be used by our successors. All such cubes can be done independently; they have no need to communicate with each other.

There will be another family of cubes that are cut further in their pasts by the initial surface. These require some conditions on the $t = t_0$ hypersurface but also input from the first layer of initial regions. In all there will be four different families of cubes that require initial conditions, one of which may be trivial if the cubes are chosen to have vertices at t_0. After these initial regions, all subsequent regions will take their starting conditions only from their predecessors.

2.2. Boundary conditions

Often, as in our cosmic string simulation below, one wants to simulate a finite volume with periodic boundary conditions. Such conditions can be implemented without additional effort merely by adjusting the connections between simulation regions and their successors. First, suppose that we want only one region in each layer. The four successors of this region will all then be the unique region in the subsequent layer, and so on. When we finish simulating a region, we write out four files

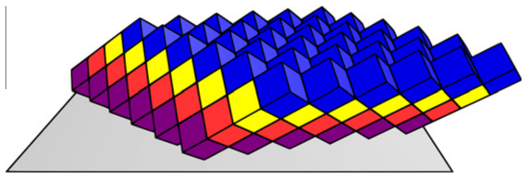

Fig. 2. The division of a 3-dimensional spacetime volume into cubical simulation regions. Regions shown in the same color (shading) have the same starting time and touch each other along their edges. A cube of one layer touches three cubes of the next layer along three of its faces. The top points of the cubes of the bottom layer shown are the same as the bottom points of the cubes of the top layer shown. In four dimensions, the volumes are 4-cubes. A 4-cube of one layer touches four 4-cubes of the next layer along four of its eight 3-cubical faces. The top points of the cubes of one layer are the same as the bottom points of the cubes four layers later.

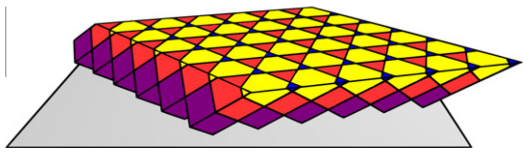

Fig. 3. A slice through the simulation volumes shown in Fig. 2 at a time t_0. The large red (medium gray) triangles are near the latest points of cubes with no predecessors before t_0. Initial conditions there can be generated immediately. The hexagonal yellow (light) regions are in the next layer of cubes and the small triangular blue (dark gray) regions are in the final layer shown in Fig. 2, which is the last for which initial conditions are necessary. In four dimensions there are four layers of cubes that require initial conditions.

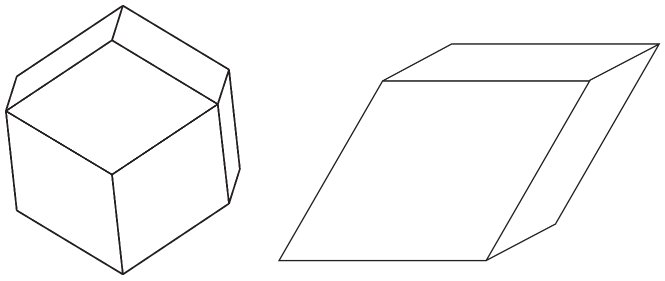

Fig. 4. In 3-dimensional space, the volume closer to one point than its images is a rhombic dodecahedron, shown on the left. But for most purposes it is easier to consider it a rhombohedron made of 60° rhombuses, shown on the right.

corresponding to the four future surfaces. When we go on to the next region, we read these four files and consider them the communication from our four virtual predecessors.

A generic surface of constant time then intersects four regions, one from each of the four layers active at that time. The periodicity vectors of the compactification can be found by going from a region to a successor in one of the four future directions and then to a predecessor in a different past direction. Thus there are 12 periodicity vectors. With the generators above, these are the 3-vectors having one component 0 and the others each ±1. A point and its images form a face-centered cubic lattice. If we take as the representative of each point the one which is closest to the origin, we see that the simulation volume is a rhombic dodecahedron. But it is easier to understand if we reorganize the volume into a rhombohedron made of rhombuses whose acute angles are 60°. Both regions are shown in Fig. 4, and their 2-dimensional analogues are shown in Fig. 5 With the coordinates we have been using, the edges of the rhombohedron have length 1, and the spatial volume of the simulation region is $1/\sqrt{2}$.

Suppose information travels at some speed v through a periodic simulation volume. The volume influenced by the initial conditions at a given point at time t fills a sphere of radius vt. When the sphere around a point touches that around an image of the point, then the existence of periodicity might affect the simulation results. We would like to delay this time as much as possible for a given simulation volume, and thus we would like the spheres around a point and its images to be close-packed. Conveniently, the f.c.c. lattice is a close-packed system, so we get this advantage for free.

To simulate larger volumes, the easiest plan is to combine unit rhombohedral volumes into a 3-dimensional array of such volumes, analogous to a cubic lattice. One could choose the three dimensions of this array independently, but in our case we chose them to be the same, so the overall volume is again a rhombohedron, and the close-packing property above is preserved. Thus each layer consists of N^3 regions, for some "split factor" N. Our largest simulations to date were done with $N = 50$.

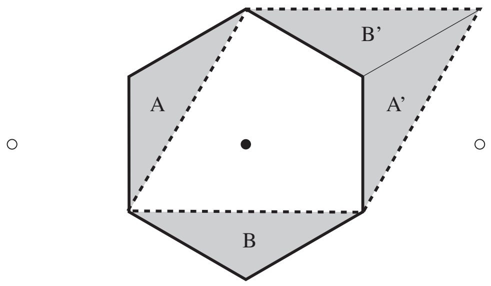

Fig. 5. In 2-dimensional space, the area closer to one point (filled circle) than its images (open circles) is a hexagon. This area can be rearranged into a rhombus by replacing the areas marked A and B by their images A' and B'.

3. Tuning

How should we choose the size of our regions? Clearly, the smaller the regions the larger the ratio of their 3-surface area to their interior 4-volume, and thus the more time will be spent reading information from predecessors and sending to successors as compared to evolution. So, all other things being equal, larger regions will be more efficient. However, there are two factors that argue for smaller regions.

First, the available memory per processor (core) may limit the amount of simulation volume that can be handled by a single processor. By making the simulation volumes smaller than this limit, we buy the opportunity to trade running time for space usage in the simulation code. The cost is an increase in the relative communication time, but in our case we found that optimizing time at the expense of space in our code was much more important than the increase in the small percentage of time spent on communication.

Second, there is more opportunity for parallelism if one splits up the simulation volume into more pieces. The maximum number of processors that can be running simultaneously is given by the number of regions in each layer. With perfect load-balancing the processors will all advance simultaneously to the next layer, but when that cannot be achieved, the realistic parallelism will be significantly less. So, for example, if you have one thousand processors available, you should split up each layer into several thousand regions.

Even finer splitting may be desirable if some regions are more difficult to simulate than others. In this case, when the simulation reaches the difficult regions, the number of processors that can be used simultaneously will decrease. (Other processors are not tied up, but still the available computational resources are not being used to complete the simulation in the shortest possible time.) A finer split will give significantly more simultaneously-runnable regions than there are processors, so if the possible parallelism is reduced by difficult regions, more of the available processors will nevertheless be kept busy.

The problem of difficult regions could also be solved by dynamical splitting [1]. A single region could be subdivided into several regions to be handled separately, and the initial data for the original region parceled out to different processors. Some of these subregions could be simulated in parallel with others, so the difficult job is spread across several processors. Once the difficult region is past, the final data from several regions could be combined into initial data for a subsequent, larger region. However, we have not implemented such a technique.

4. Simulation management

A typical parallel simulation involves a fixed number of processors, each of which has a particular task, typically the simulation of a specified region. Using our method, there is a pool of a large number of regions to be simulated, some of which are ready to go but most of which are waiting for predecessors to be done. To simulate these regions we have a pool of available processors. In principle one could merely submit one job for each region to a batch-job scheduling system, with a set of dependency conditions. The scheduler would then schedule each region on the next available processor. However, we have found that the individual regions often execute in a much shorter period of time (e.g., 1 s) than the time it takes for batch systems to schedule a new job (e.g., 30 s) , so this procedure is quite inefficient.

Instead, we have found it useful to have a "manager" process that assigns the regions to a variable-size pool of "worker" processes. When a worker completes a region, it informs the manager that region has been completed and the manager replies with a request giving the worker the next region on which to work. If a worker fails, the manager shuts down the entire simulation once the currently running regions are finished. We can repair the problem that caused the failure and continue the simulation to do the remaining regions. Each transaction between the worker and the manager is a single exchange of network packets, so network overhead is minimal.

When some regions are more difficult than others, we can find ourselves in the situation where there are more available workers than regions that can be simulated in parallel. When this occurs, the manager first tells idle workers to sleep for a time comparable to the time it takes to schedule a batch job. If, during this time, another worker finishes a region which makes more than one new region ready to simulate, the manager wakes sleeping workers to simulate the newly ready regions. But if the situation persists, the manager tells the sleeping workers to exit. If at a later time there are persistently more ready regions than running workers, the manager submits new batch jobs to increase the number of workers.

The manager, like the simulation described below, was written in LISP.

5. Semi-parallel cosmic string simulation

5.1. Cosmic strings

Cosmic strings are astrophysically long, microphysically thin objects which may have been formed early in the universe through a phase transition [3] or at the end of inflation driven by superstring theory [4–6]. See [7] for further information. In usual models, cosmic strings cannot end, so strings form a "network" of infinite strings and closed loops. The string network could potentially be observed in many ways such as cosmic microwave background variations [8], gravitational lensing [9,10], cosmic rays [11–13], gravitational waves [14,15] or early reionization [16].

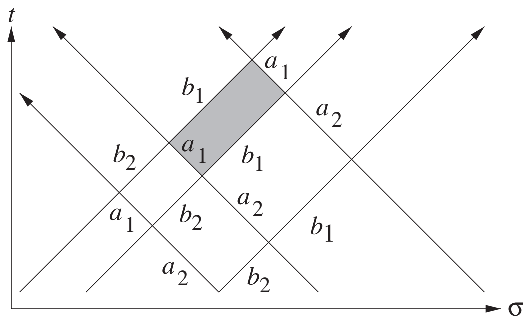

Fig. 6. A piece of the world sheet of a simulated string. The function **a** is composed of linear pieces $\mathbf{a}_1$,$\mathbf{a}_2$,... and similarly for **b**. The shaded section shows the piece of the world sheet where $\mathbf{a}_1$ is added to $\mathbf{b}_1$ to give $\mathbf{x}(\sigma,t)$.

To compute any of these observational effects quantitatively requires a knowledge of the total amount of string, the distribution of loops and the spectrum of excitations on the strings. The evolution of the string network has so far resisted a complete analytic description[3]. This situation has prompted, over the years, several different numerical simulations of the string network aimed to determine the parameters needed to predict observations [24–29]. In this paper we describe the application of the "semi-parallel" technique to the cosmic string simulation problem.

Because the thickness of a cosmic string is so much less than the sizes of structures that we expect to find on it, we can treat it as an infinitely thin relativistic string. We do not attempt here to simulate gravitational effects on the string network, and so the mass density of the string does not affect its evolution. Thus we do not need different simulations for different possible string energy scales.

When two strings cross, they switch partners with some probability p. In cosmic strings arising from superstring theory, p can be as low as 10^{-3} [30], but for the simulations describe here we will take $p = 1$. To perform a simulation one must track the positions and motions of a large network of strings, detect the intersections, and perform the switching of partners.

5.2. Cosmic string simulation algorithm: flat space

The 3-vector position of a cosmic string over time can be written $\mathbf{x}(\sigma,t)$, where σ parameterizes the position on the string and t is the usual time variable. We will first discuss the simulation of strings in a static, flat universe. Including the expansion of the universe will be discussed below.

In the absence of reconnections, the string obeys the Nambu-Goto equations of motion [7], which can be written

$$\ddot{\mathbf{x}}(\sigma,t) = \mathbf{x}''(\sigma,t), \tag{1}$$

where a prime denotes differentiation with respect to σ and a dot differentiation with respect to t. This is just the wave equation, so the solution can be given in terms of left- and right-moving waves,

$$\mathbf{x}(\sigma,t) = (1/2)[\mathbf{a}(t-\sigma) + \mathbf{b}(t+\sigma)]. \tag{2}$$

Given the functions **a** and **b**, which can easily be found from the initial position and velocity of the string, we can immediately write down the position of the string at any time.

In the present work, we use functions **a** and **b** composed of a very large number of linear pieces. To describe the evolution of the string over time, we can use a 2-dimensional "world sheet" embedded in 4-dimensional spacetime. This surface is composed of diamond-shaped regions where a given piece of **a** is combined with a given piece of **b**, as shown in Figs. 6 and 7.

Our algorithm proceeds by generating the diamonds one at a time in chronological order. After each diamond is generated, we check whether this diamond intersects with any other presently-existing diamond. If we find an intersection, we put it in a priority queue to be processed when its time arrives. This step is needed because an intersection discovered later but which takes place sooner could invalidate the one already discovered.

When the time comes to process an intersection, we reconnect the strings as follows. Let $a_1^{(1)}, a_2^{(1)}, a_3^{(1)}$ denote the linear pieces of the function **a** for one of the strings that intersected, and $a_1^{(2)}, a_2^{(2)}, a_3^{(2)}$ those for the other string. (These could be different parts of the same sequence if a single string intersects itself.) Suppose that the intersection took place in the parts of the strings given by $a_2^{(1)}$ and $a_2^{(2)}$. We then split up $a_2^{(1)}$ at the intersection point into two colinear segments $a_2^{(1)}$ and $a_{2'}^{(1)}$, and similarly for and $a_2^{(2)}$. We then rearrange the segments of **a** as $a_1^{(1)}, a_2^{(1)}, a_{2'}^{(2)}, a_3^{(2)}$ and $a_1^{(2)}, a_2^{(2)}, a_{2'}^{(1)}, a_3^{(1)}$. The treatment of **b** is completely analogous.

This procedure has no "simulation resolution", in the sense that it is able to simulate arbitrarily small segments of string. In flat space, it simulates the evolution exactly (to the limits of computer arithmetic), given the piecewise-linear initial conditions. For further details on the algorithm see [31,32,29].

[3] Although some progress has been made in this regard. See [17–23]

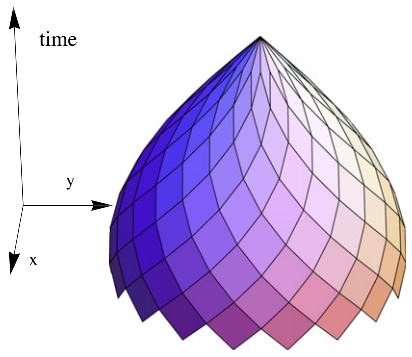

Fig. 7. The world sheet of a collapsing circle of cosmic string (approximated by a piecewise-linear shape) shown embedded in 3-dimensional spacetime.

5.3. Cosmic string simulation algorithm: expanding universe

To implement expansion of the universe, we proceed as follows. We consider only universes whose spatial sections are flat. To describe the position of strings, we use *comoving coordinates*, so that objects which move along with the expansion do not change their coordinates. We also use a conformal time coordinate, which means that light travels one unit of space per one unit of time. In these coordinates, expansion acts as a frictional term that slows down moving strings.

The evolution in expanding universe will take a piecewise-linear string and render it curved. In our simulations, however, we use an approximation which is first-order in the size of the linear segment divided by the cosmological horizon distance, and which keeps the strings piecewise-linear. Each diamond is then a curved surface in 4-space. This makes it more complicated to find intersections between strings, but the basic structure of the algorithm is unchanged.

5.4. Semi-parallel algorithm

We have parallelized our algorithm according to the procedure described above. The simulation 4-volume is divided into a large number of regions in the shape of squished 4-cubes standing on one corner. Each diamond is created during the simulation of the region that contains its earliest point. At the end of simulating a region, the processor passes on to its successors the information about all world sheet diamonds that overlap each final surface.

Because diamonds are extended objects, they can stretch through several simulation regions. In fact, a diamond created in one region may need to go to several successors. In such a case, we deliver the information about the diamond to each of these, along with a notation of which portions of the diamond each successor should handle. Each successor will pass the diamond on to its successors, until eventually it reaches a single common successor, whereupon the pieces from the predecessors are reassembled into a single diamond.

As the string network evolves, cosmic strings can intersect with themselves, producing loops. These loops may self-intersect further, but eventually one finds some non-self-intersecting loops in periodic trajectories. These loops are the main source of potentially observable effects today, so we are interested in the numbers of loops of different sizes. Small, non-self-intersecting loops will only rarely rejoin the long-string network, so once such a loop is formed, its further evolution is not usually of interest. To speed the simulation we record and remove such loops.

As a result of the removal of small loops, the total amount of string in the network declines with time. The typical distance between the remaining strings at time t grows as roughly $0.15t$. In a large simulation, this distance eventually becomes larger than the size of each simulation region, and therefore most regions contain no string at all. When we find a region with no strings in its initial conditions, we do not need to simulate it at all: clearly there will be no string on its output surfaces, and we must merely take note of this fact.

6. Results

The results of our simulation from the point of view of cosmic string physics is presented elsewhere [33]. Here we discuss this simulation from a computational point of view.

The largest simulations we have done involved periodicity length 2000 in units of the initial average distance between strings. The total time simulated was 3500 in the same units[4]. We divided the simulation volume according to the procedure of Section 2.2 with $N = 50$, so that each region has size 40 and volume $40^3/\sqrt{2}$. ("Size" here is the edge length of the rhombohedral region.) The $N = 50$ division gives 125,000 regions in each layer, and about 54 million regions in all. The job was done on

[4] Information in a string network flows only along the strings, resulting in a net speed of information flow through the volume about half the speed of light [29]. This makes possible a simulation of duration 2000 in a volume with periodicity distance 2000. But we run for longer than that to monitor the evolution of loops to see if they are in non-self-intersecting trajectories.

400 processors in elapsed time about 66 hours. Of the 54 million regions, only about 7 million contained string to be simulated. The others were empty.

In such a simulation the number of individual pieces of string grows at early times because segments are divided by reconnections, but then shrinks at late times because loops are removed. The peak number of segments in our size-2000 simulation reached about 14 billion. That makes this among the largest simulations ever performed if one counts the largest total amount of data existing at one moment of simulation time. For comparison, the Millennium Simulation [34] used just over 10 billion particles. In the course of our simulation, about 1 trillion diamonds were created and about 10 billion loops of cosmic string produced.

The total comoving length of string in the simulation at a given time does not depend strongly on the cosmological model. However, because strings are (typically) not straight, the stretching of a given segment of string is less than the overall expansion of the universe, and so its length in comoving units decreases. Thus the total number of segments (and consequently the simulation effort) required to simulate a certain comoving length of string at a late time is much higher in expanding-universe simulations. Expansion in the matter era is faster than that in the radiation era, so the matter era simulations feel this issue most strongly.

When the number of regions in each layer is much larger than the number of available processors, and the difference between one region and the next is not too large, the processors can all run simultaneously. The size 2000 run described above was done in flat space. In this case, there were always thousands of regions that were ready to go any given time, except when the simulation was nearly finished. So the 400 processors that we used were always busy.

However, in a matter era run the situation is very different. After the beginning stages of the simulation are past, often only a few processors are running. These are the ones that have most of the string, which is in quite small segments. Meanwhile, all regions that are not in the future of these regions have been completed, so there is nothing more to do. In this case the elapsed time depends on the simulation of regions with the largest number of string segments, but the rest of the available processors are not tied up waiting, so they can be used for other things. In the same simulation with conventional parallelism, most of the CPU time would be wasted as the processors wait for the processor with the highest number of string segments to run.

The difference between cosmological models also affects the choice of the optimal region size. Memory constraints limit us to a region size of order 65 per processor, meaning that total periodicity distance 2000 must be split by a factor of at least 31 (in each direction). The communications overhead in this simulation is generally negligible, so additional splitting has a low cost.

For matter-era simulations, a much more important issue is to split the regions with the largest density of string at late times. Thus in that case we usually split into much smaller pieces whose size is of order 17.[5]

Because we had our simulation based on the above algorithm already running under MPI before we converted it to the semi-parallel technique, we were able to compare the two techniques. Unfortunately, along the way we also did quite a bit of optimization of the code, so the total CPU times for the new code are significantly lower and the comparison is not as direct as one might wish. Some of this improvement was made possible because the new technique avoids memory limitations and so permitted us to use algorithms that were faster (mostly because of additional caching) at the expense of greater memory use. But the majority of the improvement was simply careful optimization of time-intensive portions of the code and is not related to the change of parallelization techniques.

In Table 1, we compare the two techniques for several small simulations done in flat space, in a radiation-dominated universe, and in a matter-dominated universe. The flat space case has the most balanced load, and the matter-dominated case the most unbalanced. All simulations were started on 64 processors and the numbers in the table are the average of 5 runs. We show the total amount of CPU time used by all the processors, the elapsed time from start to finish of the run, and the total amount of processor-time devoted to this task, whether the processor is running or idle. In the case of the MPI-based simulation, all 64 processors are in use simultaneously, so the total real time is just 64 times the elapsed time. But in the semi-parallel simulation, processors with nothing to do are freed, so that the total real time is a smaller multiple of the elapsed time. This multiple is the average number of processors, shown in the table. The CPU utilization percentage is (100 times) the total CPU time divided by the total real time, and the degree of parallelism is the total CPU time divided by the elapsed time, i.e., how many times faster the code ran in the multiprocessing system as compared to a single processor with 100% utilization.

In Table 2, we show the advantage of the new code over the old. Each row shows the ratios of the corresponding quantities in Table 1. One can consider the fundamental quantities in this table to be the CPU time advantage (how much faster the new code is than the old) and the CPU utilization advantage (how much better the new parallelization procedure is at keeping the processors doing useful work). The total real time advantage is the product of the advantages in CPU time and CPU utilization; the elapsed time advantage is the advantage in CPU time multiplied by the improvement in the degree of parallelism. Except for the issue of space-time trade-off described above, the figure of merit of the semi-parallel technique

[5] The initial conditions in our simulation are generated by the algorithm of Vachaspati and Vilenkin [35], which is non-local in that the initial strings in a particular region of spacetime depend on the generation of initial data outside that region. This sets a minimum size for each simulation region. If regions are smaller than $12\sqrt{2} \approx 16.97$ it is not possible for different regions in the first layer to generate consistent initial data, since they are not in communication with each other.

Table 1
Comparison of new and old parallelization techniques for cosmic string simulation.

	MPI-based			Semi-parallel		
	Flat	Rad.	Matter	Flat	Rad.	Matter
Size	250	180	120	250	180	120
Split factor N				10	10	7
Total CPU time (hours)	28.96	47.78	73.33	8.17	32.78	14.23
Elapsed time (hours)	0.887	1.60	3.75	0.155	0.669	0.595
Total real time (hours)	56.74	102.24	240.01	9.75	37.26	17.75
Avg. number of processors	64.00	64.00	64.00	62.83	55.66	29.83
CPU utilization percentage	51.03	46.73	30.55	83.84	87.98	80.18
Degree of parallelism	32.66	29.91	19.56	52.67	48.97	23.92

Table 2
Advantages of the new parallelization technique over the old.

	Flat	Rad.	Matter
Size	250	180	120
CPU time	3.54	1.46	5.15
Elapsed time	5.72	2.39	6.30
Total real time	5.82	2.74	13.52
CPU utilization	1.64	1.88	2.62
Degree of parallelism	1.61	1.64	1.22

Table 3
The effects of the choice of split factor N for a run of size 120 in the matter era.

Split factor	5	6	7
CPU time	15.40	14.63	14.23
Elapsed time	0.852	0.655	0.595
Total real time	19.28	18.12	17.75
Avg. number of processors	22.62	27.65	29.83
CPU utilization percentage	79.91	80.77	80.18
Degree of parallelism	18.08	22.33	23.92

is the improvement in CPU utilization. Dividing spacetime volume into regions with only initial and final surfaces has decreased the amount of processing resources needed to accomplish the most difficult task studied here by a factor of 2.6.

In Table 3, we compare the performance of a small simulation for split factors $N = 5$, 6 and 7. The CPU time is somewhat larger for coarser division of the volume, because we simulate completely each 4-cube that has any point below the final simulation time. Thus larger 4-cubes lead to a somewhat larger total volume being simulated. The more important effect, however, is that finer division leads to greater opportunities for parallelism, so that the average number of processors simultaneously in use and the degree of parallelism are larger, and consequently the elapsed time is less.

7. Summary and discussion

We have implemented a new "semi-parallel" technique for parallelizing a large simulation, as suggested by Pretorius and Lehner [1], and used it to do large simulations of cosmic strings. We have found this technique to have many advantages, which we summarize briefly here.

First, because the data does not all need to be processed simultaneously, we are able to perform larger simulations than would otherwise be possible. For example, the simulation described in Section 5 used 125,000 regions per layer, and each region used at maximum about 400 MB of memory. But a given layer only occupies 2/3 of the entire volume at most, so the total amount of memory used for a single time slice through the simulation was 75 TB.[6] To our knowledge, only the largest cluster currently available, TACC Ranger, has enough total memory to perform such a simulation using conventional techniques. With our techniques we were able to do this simulation on the Tufts Research Cluster, which has about 1% the memory of Ranger.

Furthermore, because the regions can be simulated independently, there is never any need for processors to wait for each other. Processor utilization can be much higher than in conventional parallelism. For the same reason, there is no need for a

[6] Of course we have made no attempt to optimize memory usage. On the contrary, we have preferred larger memory usage in exchange for a decrease in runtime.

particular number of processors to be available simultaneously. Since the time to simulate a single region can be made quite small by choosing small regions, there is no need to have guaranteed access to processing resources, and the simulation functions well in environments where jobs can be preempted.

In the case where one region requires much more effort than its contemporaries, we may find that all regions not dependent on the difficult region complete, leaving this region and its future yet to be done. In this case, the simulation is, for the moment, no longer parallel, and the elapsed time until completion may be dominated by the time to do the job on one processor at a time. However, it is not tying up any extra processors. This is of particular importance when there are other users of the processor cluster. The remaining processors can also be used for additional simulations, so one can complete many simulations in the same amount of elapsed time that would otherwise be needed for just one.

Since issues of memory size can be solved by finer division of the simulation volume, we are free to speed up our code by making space-time trade-offs in the direction of more space consumption and lower runtime.

The division into small, independent regions also makes debugging and dealing with failures much easier. If a processor crashes, only the work on its current region is lost. That region can be started again from its initial surfaces with little waste of effort. In contrast, a conventional parallel simulation would lose all the work done by all processors since the beginning or since a checkpoint.

Similarly, if the simulation of a region fails due to a bug, that single region can be run to investigate the problem, without the need of any other processors. When the bug is found and corrected, that region can be restarted and the simulation completed.

This simulation technique could in principle lend itself to "@Home" style simulations where different jobs could be dispatched to otherwise idle machines over the Internet, potentially enlarging the number of nodes to be used by the simulation enormously. This method has been used in several projects such as Seti@Home (http://setiathome.berkeley.edu/) and Einstein@Home (http://einstein.phys.uwm.edu/).

On the other hand, implementing such a plan in our case would be somewhat constrained by the fact that actual home computers, while often quite fast, are usually connected to the Internet by fairly slow links, especially for upload (data transfer from home to the Internet). So making use of such computers imposes additional constraints on the ratio of dataset size to runtime. It is possible that idle computers at workplaces may be a better target. We have not attempted to implement this Strings@Home project for our cosmic string simulations.

Acknowledgments

We would like to thank Frans Pretorius, Vitaly Vanchurin and Alex Vilenkin for helpful conversations. This work was supported in part by the National Science Foundation under Grant Numbers 0855447 and 0903889. We are grateful for access to the Nemo computing facility at the University of Wisconsin–Milwaukee where some of these simulations were performed. Nemo is funded by part by the National Science Foundation under Grant 0923409.

References

[1] F. Pretorius, L. Lehner, Adaptive mesh refinement for characteristic codes, J. Comput. Phys. 198 (2004) 10–34, doi:10.1016/j.jcp.2004.01.001. arXiv:gr-qc/0302003.
[2] H. Friedrich, J.M. Stewart, Characteristic initial data and wave front singularities in general relativity, Proc. Roy. Soc. Lond. A385 (1983) 345–371.
[3] T.W.B. Kibble, Topology of cosmic domains and strings, J. Phys. A9 (1976) 1387–1398, doi:10.1088/0305-4470/9/8/029.
[4] S. Sarangi, S.H.H. Tye, Cosmic string production towards the end of brane inflation, Phys. Lett. B536 (2002) 185–192, doi:10.1016/S0370-2693(02)01824-5. arXiv:hep-th/0204074.
[5] E.J. Copeland, R.C. Myers, J. Polchinski, Cosmic F- and D-strings, JHEP 06 (2004) 013, doi:10.1088/1126-6708/2004/06/013. arXiv:hep-th/0312067.
[6] G. Dvali, A. Vilenkin, Formation and evolution of cosmic D-strings, JCAP 0403 (2004) 010, doi:10.1088/1475-7516/2004/03/010. arXiv:hep-th/0312007.
[7] A. Vilenkin, E.P.S. Shellard, Cosmic Strings and other Topological Defects, Cambridge University Press, Cambridge, 2000.
[8] N. Kaiser, A. Stebbins, Microwave Anisotropy Due to Cosmic Strings, Nature 310 (1984) 391–393, doi:10.1038/310391a0.
[9] A. Vilenkin, COSMIC STRINGS AS GRAVITATIONAL LENSES, Astrophys. J. 282 (1984) L51–L53.
[10] J.R. Gott III, Gravitational lensing effects of vacuum strings: Exact solutions, Astrophys. J. 288 (1985) 422–427, doi:10.1086/162808.
[11] P. Bhattacharjee, Cosmic Strings and Ultrahigh-Energy Cosmic Rays, Phys. Rev. D40 (1989) 3968, doi:10.1103/PhysRevD.40.3968.
[12] T. Damour, A. Vilenkin, Cosmic strings and the string dilaton, Phys. Rev. Lett. 78 (1997) 2288–2291, doi:10.1103/PhysRevLett.78.2288. arXiv:gr-qc/9610005.
[13] T. Vachaspati, Cosmic rays from cosmic strings with condensates, Phys. Rev. D81 (2010) 043531, doi:10.1103/PhysRevD.81.043531. arXiv:0911.2655.
[14] T. Vachaspati, A. Vilenkin, Gravitational radiation from cosmic strings, Phys. Rev. D31 (1985) 3052, doi:10.1103/PhysRevD.31.3052.
[15] T. Damour, A. Vilenkin, Gravitational wave bursts from cosmic strings, Phys. Rev. Lett. 85 (2000) 3761–3764, doi:10.1103/PhysRevLett.85.3761. arxiv:gr-qc/0004075.
[16] K.D. Olum, A. Vilenkin, Reionization from cosmic string loops, Phys. Rev. D74 (2006) 063516, doi:10.1103/PhysRevD.74.063516. arXiv:astro-ph/0605465.
[17] A. Vilenkin, Cosmological density fluctuations produced by vacuum strings, Phys. Rev. Lett. 46 (1981) 1169–1172, doi:10.1103/PhysRevLett.46.1169.
[18] T.W.B. Kibble, Evolution of a system of cosmic strings, Nucl. Phys. B252 (1985) 227, doi:10.1016/0550-3213(85)90439-0.
[19] D. Austin, E.J. Copeland, T.W.B. Kibble, Evolution of cosmic string configurations, Phys. Rev. D48 (1993) 5594–5627, doi:10.1103/PhysRevD.48.5594. arXiv:hep-ph/9307325.
[20] J. Polchinski, J.V. Rocha, Analytic study of small scale structure on cosmic strings, Phys. Rev. D74 (2006) 083504, doi:10.1103/PhysRevD.74.083504. arXiv:hep-ph/0606205.
[21] F. Dubath, J. Polchinski, J.V. Rocha, Cosmic String Loops, Large and Small, Phys. Rev. D77 (2008) 123528, doi:10.1103/PhysRevD.77.123528. arXiv:0711.0994.
[22] J. Polchinski, J.V. Rocha, Cosmic string structure at the gravitational radiation scale, Phys. Rev. D75 (2007) 123503, doi:10.1103/PhysRevD.75.123503. arXiv:gr-qc/0702055.

[23] E.J. Copeland, T.W.B. Kibble, Kinks and small-scale structure on cosmic strings, Phys. Rev. D80 (2009) 123523, doi:10.1103/PhysRevD.80.123523. arXiv:0909.1960.
[24] A. Albrecht, N. Turok, Evolution of cosmic strings, Phys. Rev. Lett. 54 (1985) 1868–1871, doi:10.1103/PhysRevLett.54.1868.
[25] D.P. Bennett, The evolution of cosmic strings, Phys. Rev. D33 (1986) 872, doi:10.1103/PhysRevD.33.872.
[26] B. Allen, E.P.S. Shellard, Cosmic string evolution: a numerical simulation, Phys. Rev. Lett. 64 (1990) 119–122, doi:10.1103/PhysRevLett.64.119.
[27] C.J.A.P. Martins, E.P.S. Shellard, Fractal properties and small-scale structure of cosmic string networks, Phys. Rev. D73 (2006) 043515, doi:10.1103/PhysRevD.73.043515. arXiv:astro-ph/0511792.
[28] C. Ringeval, M. Sakellariadou, F. Bouchet, Cosmological evolution of cosmic string loops, JCAP 0702 (2007) 023, doi:10.1088/1475-7516/2007/02/023. arXiv:astro-ph/0511646.
[29] K.D. Olum, V. Vanchurin, Cosmic string loops in the expanding universe, Phys. Rev. D75 (2007) 063521, doi:10.1103/PhysRevD.75.063521. arXiv:astro-ph/0610419.
[30] J. Polchinski, Collision of macroscopic fundamental strings, Phys. Lett. B209 (1988) 252, doi:10.1016/0370-2693(88)90942-2.
[31] V. Vanchurin, K. Olum, A. Vilenkin, Cosmic string scaling in flat space, Phys. Rev. D72 (2005) 063514, doi:10.1103/PhysRevD.72.063514. arXiv:gr-qc/0501040.
[32] V. Vanchurin, K.D. Olum, A. Vilenkin, Scaling of cosmic string loops, Phys. Rev. D74 (2006) 063527, doi:10.1103/PhysRevD.74.063527. arXiv:gr-qc/0511159.
[33] J.J. Blanco-Pillado, K.D. Olum, B. Shlaer, Large parallel cosmic string simulations: New results on loop production, Phys. Rev. D83 (2011) 083514, doi:10.1103/PhysRevD.83.083514. arXiv:1101.5173.
[34] V. Springel et al, galaxies and their large-scale distribution, Nature 435 (2005) 629–636, doi:10.1038/nature03597. arXiv:astro-ph/0504097.
[35] T. Vachaspati, A. Vilenkin, Formation and evolution of cosmic strings, Phys. Rev. D30 (1984) 2036, doi:10.1103/PhysRevD.30.2036.