A Spatial Lattice Simulation Kernel

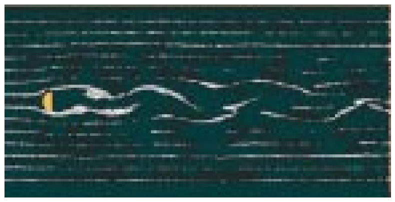
Andrew Parker & Norm Margolus

Artificial Intelligence Laboratory
Massachusetts Institue Of Technology
Cambridge, Massachusetts 02139

http://www.ai.mit.edu

The Problem: A new technique for mapping spatial lattice data into a hierarchical memory system has been developed. This technique makes possible high-performance lattice-computation hardware. Can the same technique also be used as the basis of efficient software simulators?

Motivation: Many interesting computations operate on uniform spatial lattices, or on a coupled hierarchy of such lattices. These include finite difference calculations, image processing and rendering algorithms, functional and physical logic simulation, and cellular automata simulations of a wide variety of physical systems and virtual-reality systems (see Figure 1). The kinds of processing used in these simulations includes both numeric and symbolic operations performed in a uniform manner on regularly spaced sets of lattice sites[2, 3].



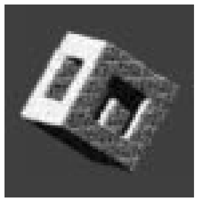


Figure 1: *Left:* A lattice-gas hydrodynamic simulation. Discrete particles conserve momentum as they flow past a half-cylinder. A smoke-fluid (white) is also simulated, to visualize the flow. *Right:* A 3D bit map is rotated on a lattice using discrete shift operations, and rendered using discrete lattice light-particles.

Previous Work: A recently proposed spatial lattice architecture[3] introduces a new technique for mapping lattice data into a memory hierarchy with a granular structure (words, rows, etc.). This technique enables large and efficient lattice simulations on specialized hardware. This technique also makes it possible to avoid much of the overhead that is normally seen in performing non-numerical lattice computations on conventional computers.

Approach: We provide a simulation kernel which embodies the inner-loop of the lattice simulation. By making this kernel into a library routine within Java, a variety of simulators and tools can conveniently be built on top of it.

Figure 2 illustrates the data-layout/processing technique for a simple 1D computation. In this example we consider a lattice consisting of sixteen lattice sites with two bits at each site, which we'll call A and B. We process lattice sites four at a time, which is related to an assumed four-bit memory word size. In the left diagram, we illustrate the order in which lattice sites are processed: groups of sites that are processed simultaneously are shown as shaded ovals. In four processing steps, the data at all sixteen lattice sites are updated. Each group of four A or B bits that are processed simultaneously reside in a single word of memory. The groups of four bits that are used in the first processing step are labeled $A[0]$ and $B[0]$, then we process $A[1]$ and $B[1]$, in the next step, etc. In the diagram, for each four-bit word of memory $A[n]$ or $B[n]$, we show the positions of the bits in the lattice using dark rectangles.

The unusual feature of our data layout is that the bits stored in each word of memory are spread evenly across the entire lattice. This makes it easy to process groups of lattice sites using shifted lattice data, as is illustrated in the right hand diagram. Here we process the same groups of lattice sites as in the left hand diagram, but using data in which the A bits have all been uniformly shifted two positions left on the lattice. Bits shifted past the left edge of the lattice wrap around to the right edge. In each processing step, the A shift is accomplished by simply changing which $A[n]$ is used, since the various $A[n]$'s turn into each other under shifts. In some cases, the bits within an $A[n]$ wrap around, but this

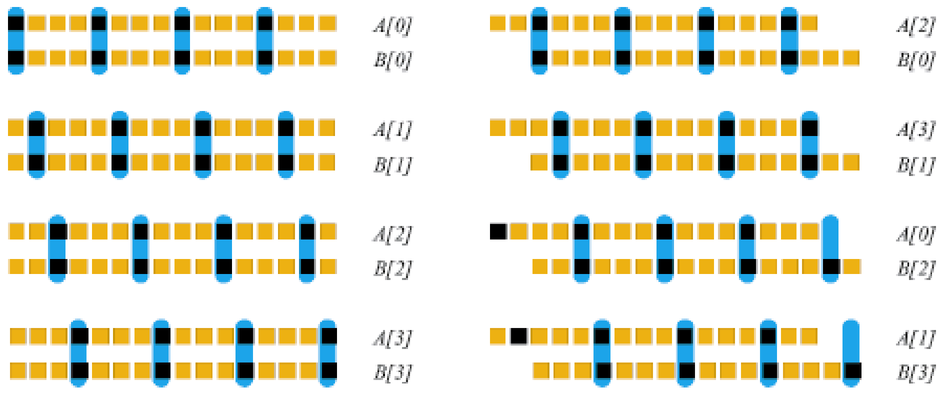


Figure 2: *Left:* Processing 1D lattice without data shift. *Right:* Processing with a shift of top *bit-field*.

just involves rotating the bit-order within a single memory word after it is read. Similarly, by changing the order of the $B[n]$'s and rotating within memory words, we could simultaneously shift both the A's and B's as part of the read operation which brings together the data for a group of lattice sites for processing.

By grouping together data in memory that is spread evenly across the lattice, we essentially turn data shifts into memory addressing operations. This same idea works with more bits at each lattice site, with larger lattices, and with more dimensions. This technique also allows us to deal with multiple levels of granularity in the memory system, by simply making sure that at each level, the bits that are used together are evenly spread out across the lattice.

Note that since lattice sites are processed in small groups, it is possible to do complex processing involving many processing steps in order to update the data at a group of lattice sites, before moving on to another group. This approach has the advantage that temporary variables used in processing one set of lattice sites can be reused for the next set.

Difficulty: On a normal microprocessor, there is an advantage in processing groups of memory words that all reside within the machine's cache before moving on to other groups of memory words, and so the cache defines a level of granularity in the memory hierarchy that our technique can deal with. One issue here is how to take best advantage of the cache in a multi-process operating system environment.

Another challenge involves performing the processing steps efficiently. The choice of when to use word-wide logical operations, versus bringing together lattice data for individual sites into machine registers for other kinds of processing, involves some subtlety.

Impact: Having a good software simulator for lattice calculations will encourage experimentation in this domain. It will also make it easier to disseminate lattice algorithms. Furthermore, the prospect of future availability of efficient hardware optimized for these kinds of algorithms should encourage the development of lattice-based techniques and models using this simulation kernel.

Future Work: A variety of simulator interfaces, lattice-computation languages, graphical interfaces and analysis tools can be built on top of this simulation kernel.

Research Support: This research is supported in part by the Defense Advanced Research Projects Agency under Contract Number F30602-98-1-0172.

References:

[1] N. Margolus. CAM-8: A computer architecture based on cellular automata. In A. Lawniczak and R. Kapral, editors, *Pattern Formation and Lattice-Gas Automata*, pages 167–187. American Mathematical Society, 1996.

[2] N. Margolus. Crystalline computation. In A. Hey, editor, *Feynman and Computation*, chapter 18. Perseus Books, 1998.

[3] N. Margolus. An embedded DRAM architecture for large-scale spatial-lattice computations. In *The 27th Annual International Symposium on Computer Architecture*, pages 149–160. IEEE Computer Society, 2000.