Long-Range Lattice-Gas Simulation on the CAM-8 Prototype
Phillips Laboratory Technical Report

J. Yepez
Phillips Laboratory, Hanscom Field, Massachusetts 01731

(14 Dec 1994)

Abstract

Presented is an implementation of a long-range lattice-gas algorithm on the CAM-8 prototype. A minimal interaction liquid-gas model is illustrated.

Contents

1 Introduction

To clearly illustrate the method of long-range lattice-gases, we have provided a computer implementation of this method for the prototype lattice-gas machine, the CAM-8. Further information concerning the CAM-8 prototype has been provided by Margolus [1]. The language we use to implement the model is called "CAMForth" and was developed at the MIT Laboratory for Computer Science as a superset of the Forth language provided on SUN SPARCworkstations. Local computation on the CAM-8 parallel computer is achieved using look-up tables stored in fast static ram chips. Since the CAM-8 data word is 16-bits wide, a look-up table contains $64K$ entries. These tables are usually created off-line and stored on disk as binary encoded files, where the address of an individual entry is the input word, and the value of that entry is the output word. We have provided a couple C-language subroutines that we use to create the look-up tables for our implementation. On the eight module CAM-8 prototype, a 512×512 two-dimensional simulation runs at almost 400 frames per second. The interface dynamics, *i.e.* coalescence of drops, for the type of simple liquid-gas model illustrated here are clearly observable in real-time on the CAM-8's frame-buffer display or within an X-window on the host workstation. The implementation in this report closely follows the our algorithmic description of a long-range lattice-gas previously given [2].

2 Implementing a Lattice-Gas in the CAM-Forth Language

The following is a simple example of a long-range lattice-gas implemented in the CAMForth language. This is an Appert-type minimal lattice-gas [3] with an interaction range of 8. It is implemented here with periodic boundary conditions in two-dimensions on the triangular lattice. The example code given below uses a 512 by 512 lattice size. We use a rhomboidal mapping of the triangular lattice onto the square lattice by using lattice directions: $\pm\hat{x}$, $\pm\hat{y}$, and $\pm(\hat{x}+\hat{y})$. Therefore, on the CAM-8's frame-buffer display, the representation of the lattice-gas on screen is sheared. Although it is possible to avoid this, for simplicity of coding, we have presented this example which is the most direct memory mapping. Our implementation uses a single messenger bit, or "photon", for each particle momentum state. Therefore, our implementation uses a total of 12 bits of local site data: 6 particle momentum states plus 6 photon states. Since we use six photon states the implementation actually does simultaneous updating of all lattice partitions.

All the dynamics are encoded into two 16-bit look-up tables. After the particle bits are kicked, the first look-up table, *lrlg.emit.lut*, actually performs two functions: (1) it does the local collisions; and (2) it then emits photons that are to be used in the long-range interaction step. After the photon bits are kicked, the second look-up table, *lrlg.absorb.lut*, performs local momentum changes by using the kicked photons, and in this way locally computes the long-range interaction. Any interaction range can be chosen, and in our code given below is set in the constant r. If a negative interaction range is set, the model still runs, however the interaction will be repulsive instead of being attractive. Therefore, a kind of parity operation with respect to the interaction range causes a conjugation of the interaction operation that results in flipping the polarity of the interparticle force.

```
\ ************* INITIALIZATION, SITE DATA, AND DATA MOVEMENT
new-experiment  512 by 512 space

0  0 == p0  1  1 == p1     2  2 == p2   3  3 == p3  4  4 == p4     5  5 == p5
6  6 == z0  7  7 == z1     8  8 == z2   9  9 == z3 10 10 == z4    11 11 == z5

: p-kicks kick   p0 field -1 x  0 y
                 p1 field  0 x -1 y
                 p2 field  1 x -1 y
                 p3 field  1 x  0 y
                 p4 field  0 x  1 y
                 p5 field -1 x  1 y
;
8  constant   r
r  negate constant  -r

: z-kicks kick   z0 field -r x  0 y
                 z1 field  0 x -r y
                 z2 field  r x -r y
                 z3 field  r x  0 y
                 z4 field  0 x  r y
                 z5 field -r x  r y
;
\ ************* PREPARE LOOK-UP TABLES AND DEFINE STEP
create-lut   collision.tab "" lrlg.emit.lut collision.tab load-buffer
create-lut interaction.tab "" lrlg.absorb.lut interaction.tab load-buffer

: prepare-tables
    lut-data   collision.tab switch-luts
    lut-data interaction.tab switch-luts
    step
; this is when-starting

define-step stepx
    site-src lut  lut-src site
    p-kicks run new-table z-kicks run new-table
end-step  this is update-step

\ ************* COLORMAP, DISPLAY TABLE, AND INITIAL PATTERN
"" lrlg.pal palette load-buffer palette>display
"" lrlg.dtab display-table load-buffer
"" lrlg.pat file>cam xvds
show-function xvds
```

This CAMForth example code, *lglr.fth*, loads several additional binary encoded files (to setup for frame-buffer visualization, data interaction through look-up table, and initial conditions) to run on the CAM-8 prototype or the CAM-8 simulator. These additional files are the following:

- *lrlg.pal* - Color palette used for display
- *lrlg.dtab* - Look-up table to render the density field
- *lrlg.pat* - Initial random pattern at 30
- *lrlg.emit.lut* - Local collisions and photon emission
- *lrlg.absorb.lut* - Long-range interaction table

3 Creating the Look-up Tables in the C-Language

We have written a short C-language code to generate the look-up tables for this simple example. We will use the following notation for the number variables. The particle momentum states, $n(a) \equiv n_a$ and $\_n(a) \equiv 1 - n_a$ and $np(a) \equiv n'_a$ and $\_np(a) \equiv 1 - n'_a$. Similarly for the photon states, $z(a) \equiv z_a$ and $\_z(a) \equiv 1 - z_a$ and $zp(a) \equiv z'_a$ and $\_zp(a) \equiv 1 - z'_a$. The number variables for the particles and photons are then coded as global data in the C-language as follows:

```
#define BIT(x,y) ((y>>x)%2)
#define CAM_WORD unsigned short
#define B 6

char   bit [B] ,  bitp [B] ;
char  zbit [B] , zbitp [B] ;

#define  n(a)  bit  [(a+B)%B]
#define _n(a)  (1-n(a))

#define  np(a)  bitp [(a+B)%B]
#define _np(a)  (1-np(a))

#define  z(a)  zbit  [(a+B)%B]
```

```
#define _z(a)  (1-z(a))

#define  zp(a)  zbitp [(a+B)%B]
#define _zp(a)  (1-zp(a))
```

It is possible to express the long-range lattice-gas dynamics as a logical function. Furthermore, it is possible to write a C-language subroutine as a direct implementation of this logical function that accepts a CAM input word and computes the correct CAM output word returned by the subroutine. The first look-up table, *lrlg.emit.lut*, is generated by calling the following C-language routine for all possible input configurations:

```
CAM_WORD lrlg_emit_lut (CAM_WORD input)
{
  char      a , C ;
  CAM_WORD  output=0  ;

  /*  FHP local collisions with z-emission */

  /* Convert input CAM word to input boolean variables */
  for ( a=0 ; a<B ; a++)  n(a) = BIT(a, input) ;

  for ( a=0 ; a<B ; a++)
    {
      C =  n(a+2) *  n(a+5) * _n(a+0) * _n(a+1) * _n(a+3) * _n(a+4)
        -  n(a+0) *  n(a+3) * _n(a+1) * _n(a+2) * _n(a+4) * _n(a+5)
        +  n(a+1) *  n(a+3) *  n(a+5) * _n(a+0) * _n(a+2) * _n(a+4)
        - _n(a+1) * _n(a+3) * _n(a+5) *  n(a+0) *  n(a+2) *  n(a+4) ;

      np(a) = n(a) + C ;
    }

  /* z-emission */
  for ( a=0 ; a<B ; a++)  z(a) = np(a+B/2) * _np(a) ;

  /* Convert output boolean variables to output CAM word  */
  for ( a=0 ; a<  B ; a++) output += np(a)<<a ;
  for ( a=B ; a<2*B ; a++) output +=  z(a)<<a ;

  return output ;
}
```

The low six bits hold the particle states, bits 0 to 5, and the high six bits hold the photon states, bits 6 to 11. The local collision operator contains the FHP

2-body and 3-body collisions [4]. The second look-up table, *lrlg.emit.lut*, is generated by calling the following C-language routine for all possible input configurations

```
CAM_WORD lrlg_absorb_lut (CAM_WORD input)
{
  int     a ;
  CAM_WORD output=0  ;
  char      C ;


  /* Convert input CAM word to input boolean variables */
  for ( a=0 ; a<B ; a++) np(a) = BIT(a, input) ;

  /* Condition for outgoing photons */
  for ( a=0 ; a<B ; a++) z(a) = np(a+B/2) * _np(a) ;

  /* Streamed incoming photons */
  for ( a=B ; a<2*B ; a++)  zp(a) = BIT(a, input) ;

  /* Nonlocal interactions, photon absorption */
  for ( a=0 ; a<B ; a++)
    {
      C =  zp(a+B/2) * z(a)  -  zp(a) * z(a+B/2) ;
      n(a) = np(a) + C ;
    }

  /* Convert output boolean variables to output CAM word  */
  for ( a=0 ; a<B ; a++) output += n(a)<<a ;

  return output ;
}
```

4 Conclusion

We have provided the algorithm detail necessary for one to understand the essential computational structure of a long-range lattice-gas model. The example presented here of a liquid-gas system was coded in the CAMForth language and the C-language. Code to generate display tables and color palettes was not presented here since that is not essential for understanding our lattice-gas implementation and is CAM-8 specific.

References

[1] Norman Margolus. Cam-8: a computer architecture based on cellular automata. In Ray Kapral and Anna Lawniczak, editors, *Proceedings of the Pattern Formation and Lattice-Gas Automata NATO Advanced Research Workshop*. Fields Institute for Research in Mathematical Sciences, American Mathematical Society, 1993. To appear.

[2] Jeffrey Yepez. Long-range lattice-gas algorithm. Technical report, Phillips Laboratory, PL/GPA Hanscoms AFB, MA 01731, October 1994. To appear.

[3] Cécile Appert and Stéphane Zaleski. Lattice gas with a liquid-gas transition. *Physical Review Letters*, 64:1–4, 1990.

[4] Uriel Frisch, Brosl Hasslacher, and Yves Pomeau. Lattice-gas automata for the navier-stokes equation. *Physical Review Letters*, 56(14):1505–1508, 1986.