

Particle simulation with Lucid

John Plaice

Département d'informatique, Université Laval

Ste-Foy, Québec, Canada

plaice@iad.ift.ulaval.ca

1 Introduction

One of the original purposes for the Lucid language was to demonstrate that numerical problems could best be programmed and computed using the dataflow paradigm. However, most of the recent work on Lucid has dealt with the extension of the intensional programming paradigm, perhaps to the detriment of the original numeric problems.

In physics, however, the concept of dataflow is perfectly natural. Electromagnetic fields, gravitational fields and hydrodynamic flows are all examples of continuous flows that are simulated using assumptions that are quite familiar to people who understand dataflow.

TRISTAN is a proven FORTRAN program for particle in-cell simulation. Designed by the late Oscar Buneman of Stanford University, it has been used to simulate plasmas of different sizes, and is one of the first computer tools to offer a plausible explanation for galaxy formation.

However, the TRISTAN code is far more complex than are the Maxwell–Hertz–Heaviside equations upon which it is based. In fact, most of the FORTRAN code pertains to manipulating five-dimensional streams using assignments to FORTRAN arrays.

In the subsequent sections, an outline of how to translate TRISTAN into Lucid is presented. The two major results are that the Lucid code closely resembles the original differential equations and the level of parallelization increases from a fixed number of 27 to the number of cells, often more than a million, while increasing the level of locality.

2 Particle in-cell simulation

The basis for the TRISTAN code are the Maxwell–Hertz–Heaviside equations for electromagnetic fields:

$$\begin{aligned}\frac{\partial \mathbf{B}}{\partial t} &= -\nabla \times \mathbf{E}, \\ \frac{\partial \mathbf{D}}{\partial t} &= \nabla \times \mathbf{H} - \mathbf{j}, \\ \nabla \cdot \mathbf{D} &= \rho, \\ \nabla \cdot \mathbf{B} &= 0;\end{aligned}$$

and the Lorentz equations for motion:

$$\begin{aligned}\frac{d}{dt}(m\mathbf{v}) &= q(\mathbf{E} + \mathbf{v} \times \mathbf{B}), \\ \frac{d\mathbf{r}}{dt} &= \mathbf{v}.\end{aligned}$$

These differential equations are in a form that is well-suited to programming computers, and this was done by Oscar Buneman in a FORTRAN program called TRISTAN, using standard interpolation techniques for generating finite difference equations.

In the TRISTAN program, electricity (**E**) (resp. magnetism (**B**)) is stored as three arrays **ex**, **ey** and **ez** (resp. **bx**, **by** and **bz**) that vary in the three dimensions x , y and z . At each time iteration, these arrays are re-computed.

To ensure that the discrete computations accurately reflect the continuous differential equations, the values in these cells are staggered as follows:

$$\begin{aligned}
 \text{ex}(i, j, k) &\equiv \text{value of ex at } x = i + .5 & y = j & z = k \\
 \text{ey}(i, j, k) &\equiv \text{value of ey at } x = i & y = j + .5 & z = k \\
 \text{ez}(i, j, k) &\equiv \text{value of ez at } x = i & y = j & z = k + .5 \\
 \\
 \text{bx}(i, j, k) &\equiv \text{value of bx at } x = i & y = j + .5 & z = k + .5 \\
 \text{by}(i, j, k) &\equiv \text{value of by at } x = i + .5 & y = j & z = k + .5 \\
 \text{bz}(i, j, k) &\equiv \text{value of bz at } x = i + .5 & y = j + .5 & z = k
 \end{aligned}$$

which corresponds to the diagram in Figure 1.

In these cells, charged particles, both ions and electrons, navigate. Their direction and velocity are affected by the surrounding electromagnetic forces, and their charge in turn affects those same forces. In the TRISTAN program, there are six arrays for the particles: (x, y, z) for their location (essentially an index into the field arrays) and (u, v, w) for their velocity.

Most of the known matter in the universe is in plasma form. Therefore, TRISTAN can be used to simulate phenomena at quite divergent scales, such as galaxy formation, van Allen belt radiation and fusion experiments. As far as simulation is concerned, it is the initial and boundary conditions that differ. Since we are presenting Lucid, we will suppose that people who better understand the physics can deal with these problems.

Once the initial conditions have been defined and checked to ensure that they meet the constraints expressed in the above equations, the program iterates for as many times as the user wishes. Each iteration consists of:

1. half-advancing the magnetic field equations;
2. moving the particles;
3. half-advancing the magnetic field equations;
4. advancing the electric field equations;
5. adjusting the electric fields according to the particle motion.

Each of these aspects is covered in a separate section.

3 Field-updates

The basic FORTRAN sequence for field-updates is as follows: The simple differential equation

$$\frac{\partial \mathbf{B}}{\partial t} = -\nabla \times \mathbf{E}$$

becomes, in FORTRAN:

```

6   do 7 i=1,mx-1
      do 7 j=1,my-1
        do 7 k=1,mz-1
          bx(i,j,k)=bx(i,j,k) + (.5*c) *
&          (ey(i,j,k+1)-ey(i,j,k)-ez(i,j+1,k)+ez(i,j,k))

```

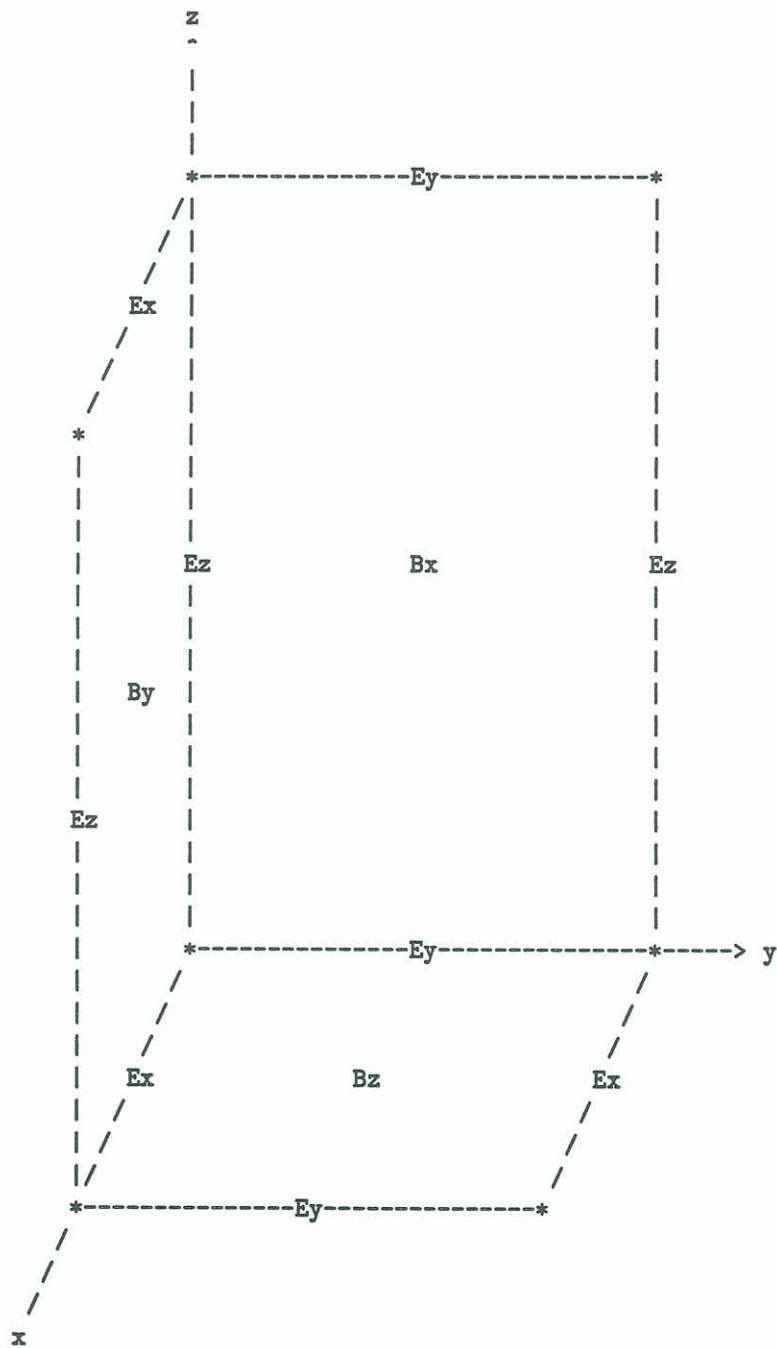


Figure 1: Staggering the entries for elctrical and magnetic fluxes.


```

      by(i,j,k)=by(i,j,k) + (.5*c) *
&      (ez(i+1,j,k)-ez(i,j,k)-ex(i,j,k+1)+ex(i,j,k))
7  bz(i,j,k)=bz(i,j,k) + (.5*c) *
&      (ex(i,j+1,k)-ex(i,j,k)-ey(i+1,j,k)+ey(i,j,k))

```

In Lucid, both the electrical and magnetic fluxes are considered to be five-dimensional streams:

$$\mathbf{B}, \mathbf{E} : \mathbb{Z}^3 \times \{0, 1, 2\} \times \mathbb{Z} \rightarrow \mathbb{R}.$$

The dimensions are called dx , dy , dz , dd and dt . The first three dimensions are space coordinates, the fourth is used for accessing values in triples and the last dimensions refers to time. We assume that the dimensions are numbered from 0 to 4.

The above FORTRAN lines become, in Lucid:

```

next.dt B = B + .5*c * curl E
where
  curl E = d right E - d left E
  d dir E = d_space dir (comp dir E)
  comp dir E = E @.dd dir
  d_space dir E = next.dir E
  right = (#.dd+1) mod 3
  left = (#.dd+2) mod 3
  straight = #.dd
  dx = 0
  dy = 1
  dz = 2
  dd = 3
  dt = 4
when
  #.dx>=1 and #.dx<=mx-1 and
  #.dy>=1 and #.dy<=my-1 and
  #.dz>=1 and #.dz<=mz-1

```

Note that the expression

```
d_space dir E = next.dir E
```

is using *dynamic* dimensions. The expression `dir` evaluates to a dimension (an integer), and this value is used as the argument for the `next` operator.

Note also that the subexpression

```

when
  #.dx>=1 and #.dx<=mx-1 and
  #.dy>=1 and #.dy<=my-1 and
  #.dz>=1 and #.dz<=mz-1

```

designates when this expression is valid. For other values of `#.dx`, `#.dy` and `#.dz`, other equations apply.

4 Boundary conditions

The Maxwell-Hertz-Heaviside equations are assumed to apply throughout the (infinite) universe. However, a computer simulation only looks at a finite part thereof. To ensure that simulations are realistic, quite complicated calculations are effected at the boundaries of the simulated space. In the TRISTAN code, The Lindsman boundary methods are used for the surfaces and edges of the space that is being simulated.

The TRISTAN code for calling the boundary routines is as follows:

```

call surface(by,bz,bx,ey,ez,ex,ix,iz,ix,my,mz,mx,1)
call surface(bz,bx,by,ez,ex,ey,iz,ix,iz,my,mz,mx,1)
call surface(bx,by,bz,ex,ey,ez,ix,iz,iz,my,mz,mx,1)
call edge(bx,ix,iz,iz,mx,my,mz,1)
call edge(by,iz,iz,iz,my,mz,mx,1)
call edge(bz,iz,iz,iz,mz,mx,my,1)

```

We will not actually look at the boundary routines, however, we can see that it is not at all clear what cells are affected by these routines. Much simpler is the approach below:

```

next.B = eqn0 when cond0
next.B = eqn1 when cond1
...
next.B = eqnn when condn

```

where the *cond_i* expressions are of the form

```

#.dx>=1 and #.dx<=mx-1 and
#.dy>=1 and #.dy<=my-1 and
#.dz>=1 and #.dz<=mz-1

```

By doing this, it becomes clear what section of the parallelopiped is being defined by what part of the code.

5 Moving particles

The actual moving of the particles is a very simple process. However, before the particles can be moved, the system must compute the exact electromagnetic forces for each particle. This is done through an interpolation process that is made more complex by the fact that the electrical and magnetic fields are staggered.

The field interpolations are tri-linear (linear in *x*, linear in *y* and linear in *z*), where linear in one direction becomes:

$$f_{i+\delta} = f_i + \delta(f_{i+1} - f_i).$$

However, when the field is staggered, the result becomes:

$$f_{i+\delta} = \frac{f_i + f_{i-1} + \delta(f_{i+1} - f_{i-1})}{2}.$$

So, the FORTRAN code

```

C  E-component interpolations:
    f=ex(1)+ex(1-ix)+dx*(ex(1+ix)-ex(1-ix))
    f=f+dy*(ex(1+iy)+ex(1-ix+iy)+dx*(ex(1+ix+iy)-ex(1-ix+iy))-f)
    g=ex(1+iz)+ex(1-ix+iz)+dx*(ex(1+ix+iz)-ex(1-ix+iz))
    g=g+dy*
    & (ex(1+iy+iz)+ex(1-ix+iy+iz)+dx*(ex(1+ix+iy+iz)-ex(1-ix+iy+iz))-g)
    ex0=(f+dz*(g-f))*(.25*qm)
C  -----
    f=ey(1)+ey(1-iy)+dy*(ey(1+iy)-ey(1-iy))
    f=f+dz*(ey(1+iz)+ey(1-iy+iz)+dy*(ey(1+iy+iz)-ey(1-iy+iz))-f)
    g=ey(1+ix)+ey(1-iy+ix)+dy*(ey(1+iy+ix)-ey(1-iy+ix))
    g=g+dz*
    & (ey(1+iz+ix)+ey(1-iy+iz+ix)+dy*(ey(1+iy+iz+ix)-ey(1-iy+iz+ix))-g)
    ey0=(f+dx*(g-f))*(.25*qm)

```

```

C -----
f=ez(l)+ez(l-iz)+dz*(ez(l+iz)-ez(l-iz))
f=f+dx*(ez(l+ix)+ez(l-iz+ix)+dz*(ez(l+iz+ix)-ez(l-iz+ix))-f)
g=ez(l+iy)+ez(l-iz+iy)+dz*(ez(l+iz+iy)-ez(l-iz+iy))
g=g+dx*
& (ez(l+ix+iy)+ez(l-iz+ix+iy)+dz*(ez(l+iz+ix+iy)-ez(l-iz+ix+iy))-g)
ez0=(f+dy*(g-f))*(.25*qm)

```

becomes, simply

```

E0 = .5 * qm *
      (interpolate left
        (interpolate right
          (half_interpolate straight (cell E d) d) d) d)
      where
        interpolate dir E d =
          E + (d @.dd #.dir) * (next.dir E - E)
        half_interpolate dir E d =
          (E + prev.dir E + (d @.dd #.dir) * (next.dir E - prev.dir E))/2
        cell E d = ((d @.dx (d @.dd dx) @.dy (d @.dd dy)) @.dz (d @.dd dz))

```

where d is the offset within the cell of the particle.

As for the FORTRAN code

```

C B-component interpolations:
f=bx(l-iy)+bx(l-iy-iz)+dz*(bx(l-iy+iz)-bx(l-iy-iz))
f=bx(l)+bx(l-iz)+dz*(bx(l+iz)-bx(l-iz))+f+dy*
& (bx(l+iy)+bx(l+iy-iz)+dz*(bx(l+iy+iz)-bx(l+iy-iz))-f)
g=bx(l+ix-iy)+bx(l+ix-iy-iz)+dz*(bx(l+ix-iy+iz)-bx(l+ix-iy-iz))
g=bx(l+ix)+bx(l+ix-iz)+dz*(bx(l+ix+iz)-bx(l+ix-iz))+g+dy*
& (bx(l+ix+iy)+bx(l+ix+iy-iz)+dz*(bx(l+ix+iy+iz)-bx(l+ix+iy-iz))-g)
bx0=(f+dx*(g-f))*(.125*qm/c)

```

```

C -----
f=by(l-iz)+by(l-iz-ix)+dx*(by(l-iz+ix)-by(l-iz-ix))
f=by(l)+by(l-ix)+dx*(by(l+ix)-by(l-ix))+f+dz*
& (by(l+iz)+by(l+iz-ix)+dx*(by(l+iz+ix)-by(l+iz-ix))-f)
g=by(l+iy-iz)+by(l+iy-iz-ix)+dx*(by(l+iy-iz+ix)-by(l+iy-iz-ix))
g=by(l+iy)+by(l+iy-ix)+dx*(by(l+iy+ix)-by(l+iy-ix))+g+dz*
& (by(l+iy+iz)+by(l+iy+iz-ix)+dx*(by(l+iy+iz+ix)-by(l+iy+iz-ix))-g)
by0=(f+dy*(g-f))*(.125*qm/c)

```

```

C -----
f=bz(l-ix)+bz(l-ix-iy)+dy*(bz(l-ix+iy)-bz(l-ix-iy))
f=bz(l)+bz(l-iy)+dy*(bz(l+iy)-bz(l-iy))+f+dx*
& (bz(l+ix)+bz(l+ix-iy)+dy*(bz(l+ix+iy)-bz(l+ix-iy))-f)
g=bz(l+iz-ix)+bz(l+iz-ix-iy)+dy*(bz(l+iz-ix+iy)-bz(l+iz-ix-iy))
g=bz(l+iz)+bz(l+iz-iy)+dy*(bz(l+iz+iy)-bz(l+iz-iy))+g+dx*
& (bz(l+iz+ix)+bz(l+iz+ix-iy)+dy*(bz(l+iz+ix+iy)-bz(l+iz+ix-iy))-g)
bz0=(f+dz*(g-f))*(.125*qm/c)

```

it simply becomes

```

B0 = .5 * qm *
      (half_interpolate left
        (half_interpolate right
          (interpolate straight (cell E d) d) d) d)

```


6 Charge updates

The TRISTAN code modifies the electrical field by passing through the particles, one at a time, and calculating the effects of that particle on the surrounding cells. The equations are set up in such a way that a particle cannot advance more than half a cell-width in one iteration, therefore no particle may travel to a cell that is not adjacent.

Each particle will move during an iteration. Its arrival point may be an adjacent cell, and to get there, it will possibly pass through other (adjacent) cells. For each affected cell, i.e. where a particle is tourist or immigrant, the charge of that particle affects the electrical field of that cell, as well as that of its 26 immediate neighbors. These computations can be run in parallel, thereby yielding the 27-fold parallelism in TRISTAN.

However, in Lucid, the basis for parallelism is the cell. Each cell has a list of the particles that are contained in it, and each cell inquires from its neighbors what are the particles that affect it. By doing this, the particles can be aggregated into a sort of “super-particle”, thereby reducing the number of computations to make, as well as *massively* increasing the level of parallelism.

7 Remarks

This outline paper has shown how an existing FORTRAN program can be studied and analyzed in order to produce a Lucid program that is significantly shorter and clearer, in which the level of parallelism is orders of magnitude greater.

Once the translation into Lucid is finished, then we envisage using a compiler to translate this code into a massively parallel language for existing supercomputers.