- Portable Lattice-Boltzmann in Java.
- Rubén Jesús García Hernández.
- Edinburgh Parallel Computing Centre
- Tuesday 28^{th} August 2001, 13:30

- The behaviour of the liquids is governed by the Boltzmann equation

$$\frac{\partial}{\partial t}\Phi(x,t) = -v \cdot \Phi(x,t) + v \cdot \int_{-\infty}^{\infty} K(x,s) \cdot \Phi(s,t)ds \quad (1)$$

- This equation can be discretized. Then the changes in the liquid can be further divided into four parts
  - Set boundary conditions
  - Propagation.
  - Collision.
  - Bounce Back.

- The simulation consists of a rectangular prisma.
- The rest of the universe interacts with the prisma in the surface
- The behaviour in the surface of the prisma has to be modelized so that we don’t have to take into account what happens outside.
- We fix the parameters in the boundary from the beginning of the simulation.

Propagation

- The particles moving in the prisma have a current position and velocity.
- We use particle density and discretize the speeds so that we don't have to keep track of all the atoms in the liquid.

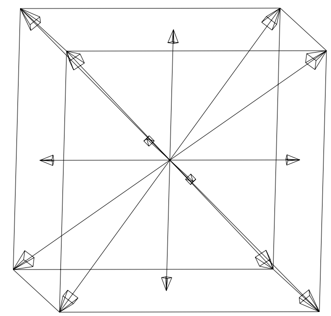

Figure 1 :

- D3Q15 Model: 15 velocities, one with speed zero (a rest particle), six with $speed^2 = 1$ (to nearest neighbours), and eight with $speed^2 = 3$ (to next next nearest neighbours).

- Once the propagation has ended, we have particles from neighbouring sites crashing in our site.
- This stage calculates the movement of the particles after the crash.

- We have added solid objects floating in the liquid.
- The propagation algorithm does not take these into account.
- That means we have some particles which have “entered” the solids.
- This stage takes the particles that have invaded a solid and fixes them.
- The final result is equivalent to the particle having bounced back from the solid.

Here is one of the simulated results using Ludwig:

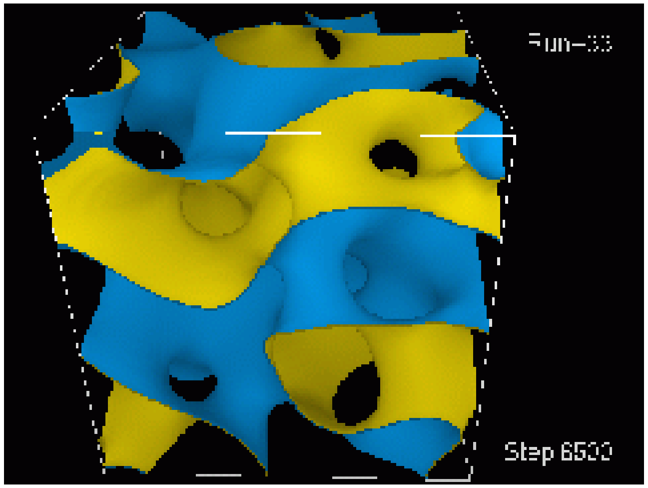

Figure 2 : Evolution of the fluid-fluid interface

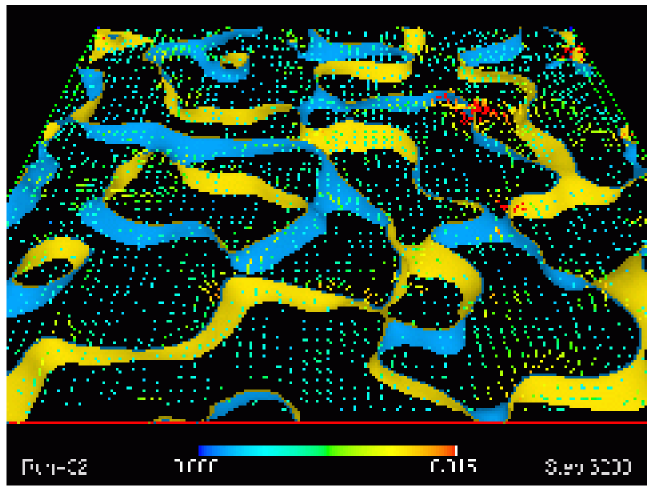

Figure 3 : Time-resolved velocity maps (cropped for clarity to a thin section)

- The behaviour of the Java version is quite similar to the C one.
- The Java version is a bit slower still.

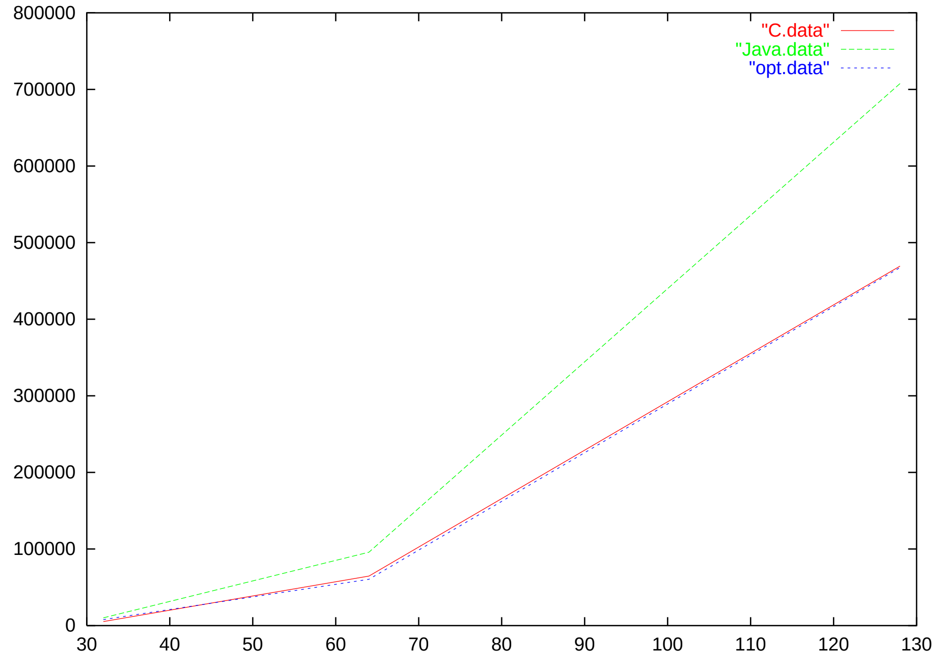

Figure 1 : Graphic comparison C vs Java.

Time as a function of the length of the side of the lattice.

Parallel Java versions

- Java OpenMP version.
- Message Passing Interface for Java version.

- Porting problems.
- Benchmarking results.
  - 1^{st} version. Bad scaling due to the propagate function.
  - 2^{nd} version. Optimizing the first sequencial loop.
  - 3^{rd} version. Using double buffer to avoid copies

|epcc|

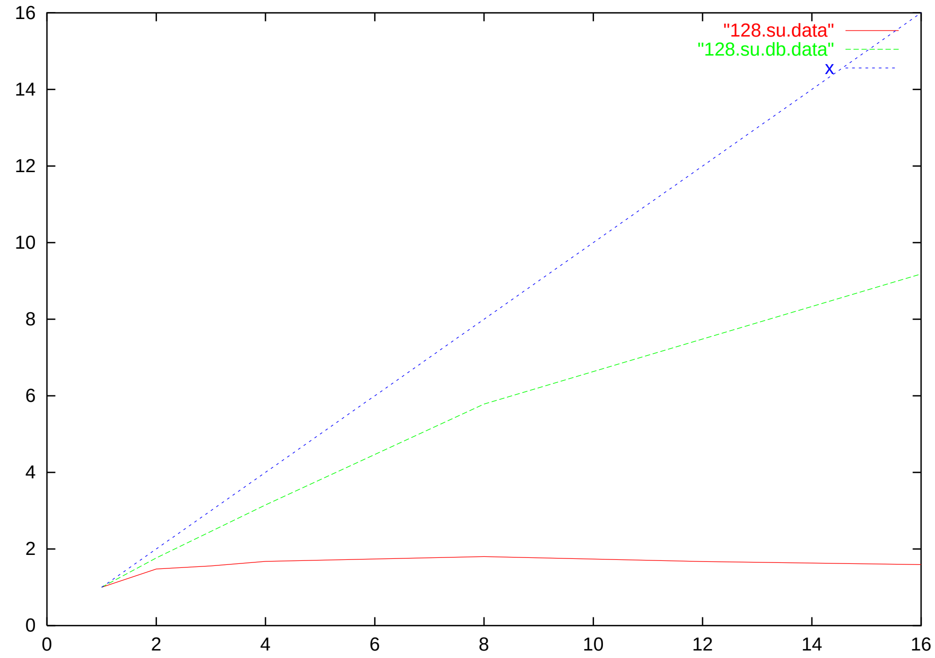

Figure 2 : Comparison between the scaling of the original parallel version and the new double buffer version

- Porting problems.
- Benchmarking results.