Parallel Implementation of a 2-D LLF Hydrodynamics Code

a CSE 721 final project report by Chris Orban

March 7, 2006

1 Introduction and Objectives

In a scientific computing course at the University of Illinois (U-C) myself and another undergraduate student created a serial implementation of a Local Lax-Friedrichs (LLF) scheme for simulating hydrodynamic systems in 2-D. After quantitative testing of the code it was further developed to include the self gravity of the fluid. My familiarity with this previous project and the relevancy of hydrodynamic simulations to my research interests made continuing the development of this code a very attractive and attainable goal for this project. Unfortunately the pure hydrodynamic (i.e. no gravity) code proved complex enough to make developing the parallel implementation of self-gravity too ambitious for this project. Thus my objectives are somewhat revised.

The primary objective for this project is to create a parallel implementation of a 2-D LLF code using basic MPI commands. Secondarily, optimizations can be considered in attempts to improve the code's performance.

2 Motivation

Hydrodynamics is quite simply the physics of fluids. From supernovae,to hurricanes, to blood vessels, and jet engines, our understanding of these and other phenomena all rest on our foundational knowledge of hydrodynamics. The two most basic equations of hydrodynamics are mass conservation (Eq. 1), perhaps better known as the continuity equation, and momentum conservation (Eq. 2).

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0 \tag{1}$$

$$\frac{\partial \rho \mathbf{v}}{\partial t} + \nabla \cdot (\rho \mathbf{v}\mathbf{v}) = -\nabla P \tag{2}$$

where ρ is the density, $\mathbf{v}$ is the velocity vector field and P is the pressure.

Insight can be gained from analytic consideration of these equations but in many cases of interest the initial conditions (ex. for a jet engine) may be too irregular or asymmetric to solve "by hand". Difficult-to-model turbulence may also be a concern.

With this in mind, it is not hard to see the potential of computers and fast hydrodynamic algorithms to contribute to our scientific understanding and our pursuit of engineering goals. The challenge is that these problems tend to be very computationally intensive. The relevant partial differential equations may be quite complicated and three-dimensional simulations have memory requirements that naturally scale as n^3. (The code and all of the simulations presented here are all two-dimensional as to avoid this trend.) Parallel computing is the obvious route to open up otherwise intractable hydrodynamic problems to investigation.

3 Methodology

Equations (1) and (2), along with an isothermal equation of state $P = c_s^2 \rho$ where c_s is the sound speed, were used as the relevant dynamical equations in these simulations. The local Lax-Friedrichs scheme is a finite-differencing approach to numerically solving these equations which updates each cell based on the cell values and slopes around it. With an isothermal equation of state there remains only three variables to be solved, ρ, v_x, and v_y where v_x and v_y are the velocities in the x and y directions. At each step these variables are updated as "fluxes" in the form ρ, ρv_x, and ρv_y with the following equations,

$$u^{n+1} = u^n - \frac{\Delta t}{\Delta x}(F_{j+1/2} - F_{j-1/2}) \tag{3}$$

$$F_{j+1/2} = \frac{1}{2}(F(u^+_{j+1/2}) + F(u^-_{j+1/2}) - v_{max}(u^+_{j+1/2} - u^-_{j+1/2})) \tag{4}$$

$$u^{+}_{j+1/2} = u_{j+1} - s_{j+1}\frac{\Delta x}{2} \tag{5}$$

$$u^{-}_{j+1/2} = u_{j} + s_{j}\frac{\Delta x}{2} \tag{6}$$

where v_{max} is the max velocity in the domain and s_j, s_{j+1} are the slopes in the variable u at those points, and Δx is the distance between cells.

The parallel code uses this algorithm and employs checkerboard partitioning to divide the domain across the processors. This approach requires four communication steps to preclude each timestep so that the edges of each processor's domain can be exchanged with its neighbors.

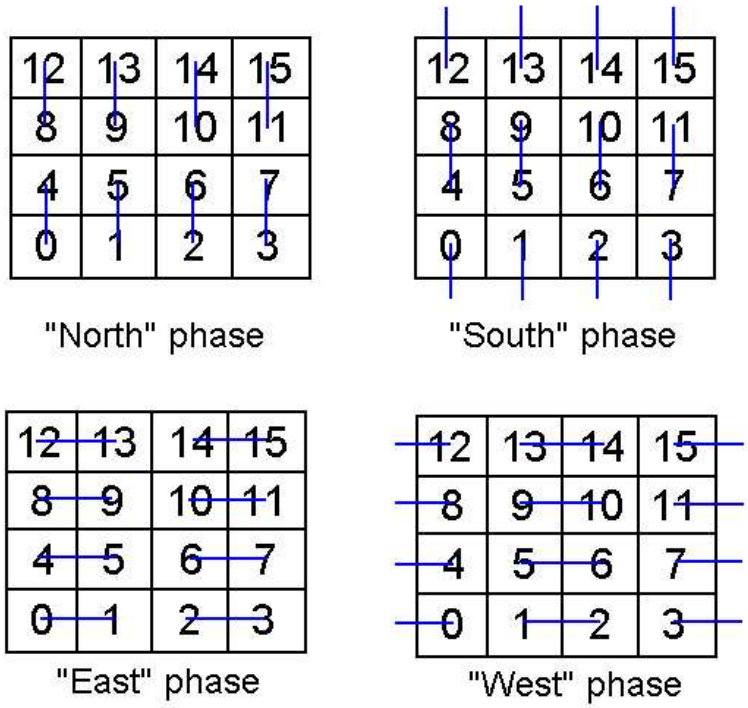

Figure 1: Communication patterns in parallel implementation. Note that the boundary conditions here are repeating or "wrap around" and that the symmetry of these phases require that $p = 4^2, 6^2, 8^2$....

4 Measurements

The most critical test of the parallel code involves propagating a cosine wave through the domain. Both the serial and parallel code should have $O(N^{-1})$ convergence to the analytic solution for the propagating wave. Figure 2 confirms this prediction.

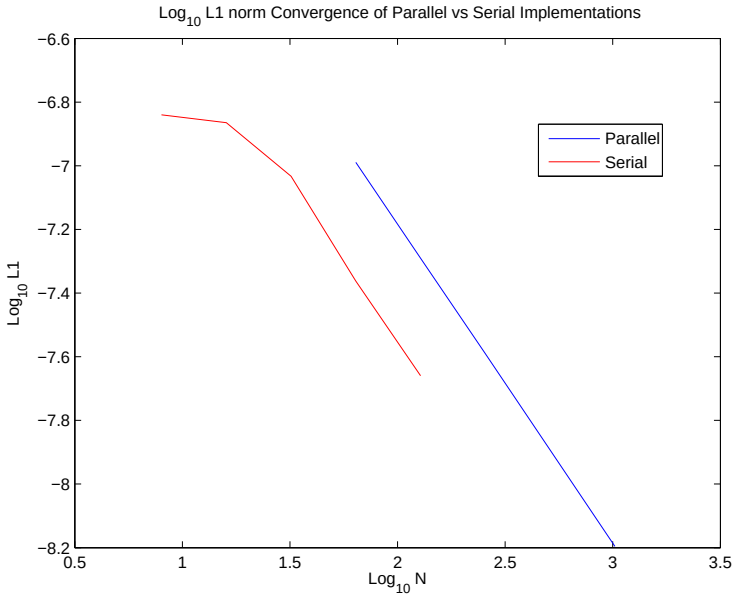

Figure 2: A convergence test comparing the serial algorithm with the parallel code running on 16 processors. Both lines show $O(N^{-1})$ convergence (log-log slope of -1). All numerical experiments in this report were performed on the Itanium 64 cluster at OSC.

As a more qualitiative test a gaussian density spike can be simulated in serial and parallel and compared. Figure 3 presents the cross sections from the serial and parallel code operating at the same total resolution.

In terms of performance, Table 1 summarizes the job completion time for 16, 36, and 64 processors with job sizes growing in proportion to the availible processors. The serial completion time, T_s, was inferred from a definite $N^{2.7}$ trend in the completion time at lower N. Since the serial domain has N^2 elements, this is a very reasonable result since

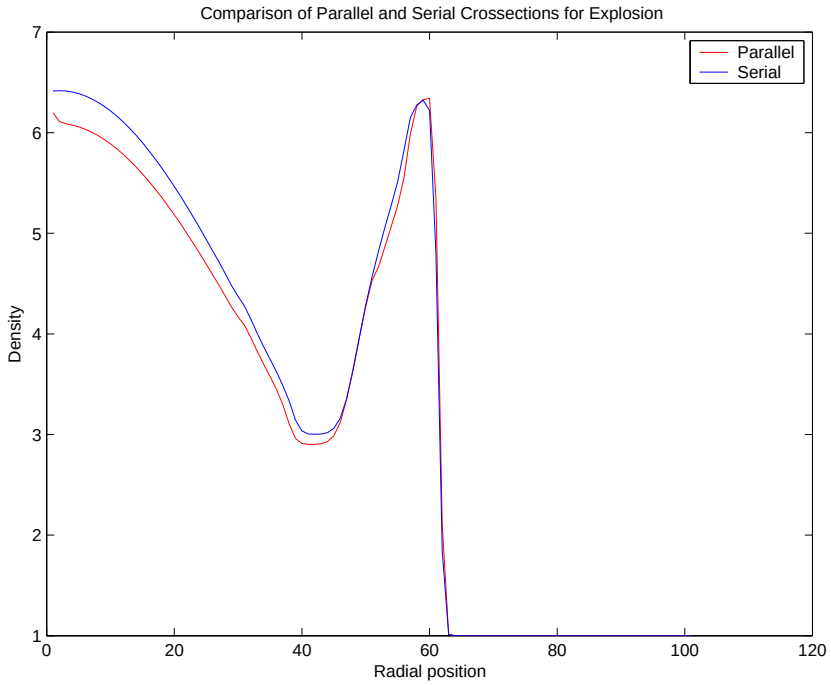

Figure 3: A radial cross section through a propagating explosion. The serial data is at the same resolution as the parallel data. See powerpoint for surface plot.

a T_s scaling of at least N^2 is to be expected.

Table 1. Completion times for p = 16, 36, and 64

Processors	N	Median time (s)	Max time (s)	T_s (s)	T_s/p
16	800	17.6	18.54	689	43.06
36	1200	28.0	32.8	2059	57.2
64	1600	36.5	67.68	4478	69.9

Striped partitioning was considered as a potential optimization. The advantage of this would be to reduce four communications per timestep down to two. This places more of the "burden" of communication on the bandwidth-associated transfer time and improves the latency by a factor of two. Simple analytic estimates of this improvement, however, showed that the advantage is gained only for small problem sizes ($N \sim 100$) and only when using a few processors. The memory scaling was also poorer. For these reasons this optimization scheme was abandoned.

5 Conclusions

On the one hand Fig. 2 is very encouraging since it affirms that the parallel code does have the expected convergence. And additionally, the more qualitiative explosion test appears accurate, but table 1 shows a rather troubling superlinear speedup. Upon further investigation it was learned that the convergence, although near-perfect for 16 processors, contrary to expectations the same is not true of 64 processors.

The best explanation is that something is lowering the precision of the exchanges for $p > 16$. The effect is such that though each processor is successfully communicating with the others, the accuracy only increases as if just processor were working in isolation. In other words, there is a point where adding processors and correspondingly increasing the problem size does not increase the overall precision of the calculation.

The primary evidence for this is in the parallel completion time, which can only be explained by analogy to the serial algorithm working on an $N/\sqrt{p} \cdot N/\sqrt{p}$ domain. T_s was measured to be $T_s = 10^{-5}N^{-2.7}$ so the "isolation problem" would involve completion times of order $T_i = 10^{-5}(N/\sqrt{p})^{2.7}$. For 64 processors and the associated problem size in the table T_i evaluates to 16.322 seconds. The measured completion time T_p is 36.5 seconds, which is much faster than the theoretical minimum of 69.9 seconds.

Though this failure is disappointing, the success of the 16-processor convergence affirms that the code is potentially accurate enough to do science, albeit restricted to that many processors. This project has been my first exposure to MPI programming, and has involved writing by far the largest code I've ever written. To know that it is scientifically viable in some way and to have some understanding of what prevents the code from reaching higher accuracy is enormously satisfying. This project represents a huge leap in my programming skills relative to the beginning of the quarter.