Kenneth Lim (kl545), Scott Wu (ssw74), Robert Chiodi (rmc298), Ravi Patel (rgp62)
Final Project

1 Introduction

1.1 Smooth Particle Hydrodynamics (SPH)

1.1.1 Purpose

The purpose of this project is implement a Smoothed Particle Hydrodynamics based on Position Based Fluids (Macklin, Muller 2013). Since the goal of the paper is geared towards real time use, some portions of the algorithm will sacrifice accuracy for speed. Finally, we will be analyzing and parallelizing the algorithm to further improve performance.

1.1.2 Algorithm Overview

In the simulation, our inputs are the initial positions and velocities of particles in a box. Then we take equal time steps, producing the same list of position and velocity outputs at each step.

- At the beginning of each step, we compute candidate velocities and positions by taking an Eulerian step
- We compute the neighbors particles within a certain radius by placing them on a grid
- Candidate velocities and positions are iteratively corrected
  - Per iteration, we attempt to solve the incompressibility constraint
  - Meanwhile maintain the boundary conditions of the box, and apply velocity dampening if necessary
  - Update the candidate positions with those constraints
- Using the new candidate positions, we update the candidate velocities
- We apply vorticity confinement to maintain energy in the system
- We apply viscosity to blur the velocities into a more coherent motion
- Finally we update the positions and velocities with the candidate positions and velocities

1.2 Code Bases

1.2.1 C Code

1.2.2 Fortran Code

The Fortran code is based off of an SPH code used by Professor Bindel in the spring of 2014 for CS 5220: Applications of Parallel Computers. This code was selected because it directly referenced the solution of the relevant equation (Navier-Stokes) and discussed treatments required to have a working code. SPH can be very unstable without sufficient parameter tuning and having the reference code available allowed easier creation of a serial program that could then be parallelized. Several modifications and additions have been made to this code, as well as it being transferred to Fortran, that will be discussed later. The main contributions taken from the reference code were the leap-frog time integration scheme, handling of walls via damping coefficients, and the recalculation of mass during initialization to have particle densities near the particle reference density. The additions and transfer to Fortran for the serial code led to an order of magnitude speed up for the dam break problem created in the C version using the "box indicator".

The initialization of the Fortran code is handled via an initial text file generated with a python script. This text file contains the number of particles, the time step, the size of the particles, domain side length (with the domain being assumed cubic), frequency of visualization file output, final desired simulation time, and initial position and velocity of all particles. This is a very flexible format for initialization that allows easy rerunning of simulation conditions and the creation of a diverse problem set.

1.3 Writing of Visualization Files

The writing of visualization files is handled the same way in each code. At a specified frequency, by a certain number of time steps in the C version or an amount of simulated time in the Fortran version, a text file is written that contains the number of particles along with their position and velocity. The text files are numbered sequentially with integer tags in order to keep the proper time sequence. A python script we wrote, based loosely off of the visualizer script written for the wave-equation assignment, then plots each particle as a point on a 3D scatter plot and encodes the images together with `ffmpeg`.

2 Profiling and Serial Optimization

2.1 C Code

2.1.1 Profiling

2.1.2 Optimizations

2.1.3 Compiler Flags

2.2 Fortran Code

2.2.1 Profiling

2.2.2 Optimizations

2.2.3 Compiler Flags

3 Parallelization

3.1 C Code

3.1.1 OpenMP

Strong Scaling:

Weak Scaling:

3.1.2 Cilk

Strong Scaling:

Weak Scaling:

3.2 Fortran Code

3.2.1 OpenMP

Strong Scaling:

Weak Scaling:

Case	# Particles	LLC (x,y,z)	URC (x,y,z)	# Threads	Time (s)
1	36,000	(0.0, 0.0, 0.0)	(0.5, 0.5, 0.5)	1	182.99
2	36,000	(0.0, 0.0, 0.0)	(0.5, 0.5, 0.5)	2	79.769
3	36,000	(0.0, 0.0, 0.0)	(0.5, 0.5, 0.5)	4	42.479
4	36,000	(0.0, 0.0, 0.0)	(0.5, 0.5, 0.5)	8	24.816
5	36,000	(0.0, 0.0, 0.0)	(0.5, 0.5, 0.5)	12	17.791
6	36,000	(0.0, 0.0, 0.0)	(0.5, 0.5, 0.5)	16	22.772
7	36,000	(0.0, 0.0, 0.0)	(0.5, 0.5, 0.5)	20	18.565
8	36,000	(0.0, 0.0, 0.0)	(0.5, 0.5, 0.5)	24	28.774

Table 1: Configuration of dam break simulations used for strong scaling study. LLC = Lower Left Corner, URC = Upper Right Corner, SS = Strong Scaling, SSE = Strong Scaling Efficiency.

Case	# Particles	LLC (x,y,z)	URC (x,y,z)	# Threads	Time (s)
1	2,197	(0.0, 0.0, 0.0)	(0.5, 0.5, 0.5)	1	6.3919
2	4,000	(0.0, 0.0, 0.0)	(0.5, 0.5, 0.5)	2	7.2487
3	8,000	(0.0, 0.0, 0.0)	(0.5, 0.5, 0.5)	4	10.717
4	16,000	(0.0, 0.0, 0.0)	(0.5, 0.5, 0.5)	8	11.526
5	24,389	(0.0, 0.0, 0.0)	(0.5, 0.5, 0.5)	12	14.846
6	32,768	(0.0, 0.0, 0.0)	(0.5, 0.5, 0.5)	16	24.198
7	39,304	(0.0, 0.0, 0.0)	(0.5, 0.5, 0.5)	20	24.268
8	46,656	(0.0, 0.0, 0.0)	(0.5, 0.5, 0.5)	24	24.203

Table 2: Configuration of dam break simulations used for weak scaling study. LLC = Lower Left Corner, URC = Upper Right Corner, SS = Strong Scaling, SSE = Strong Scaling Efficiency.

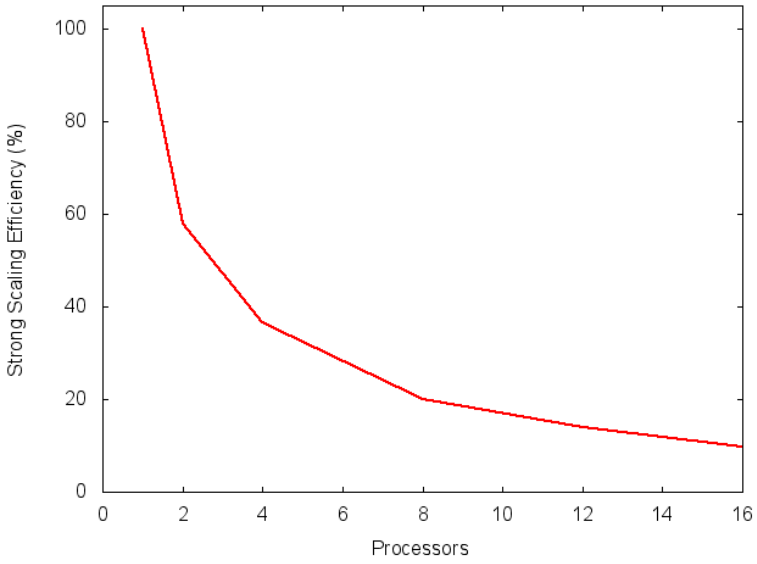

Figure 1: Strong scaling for simulations in Table 1 using the Fortran code parallelized with OpenMP.

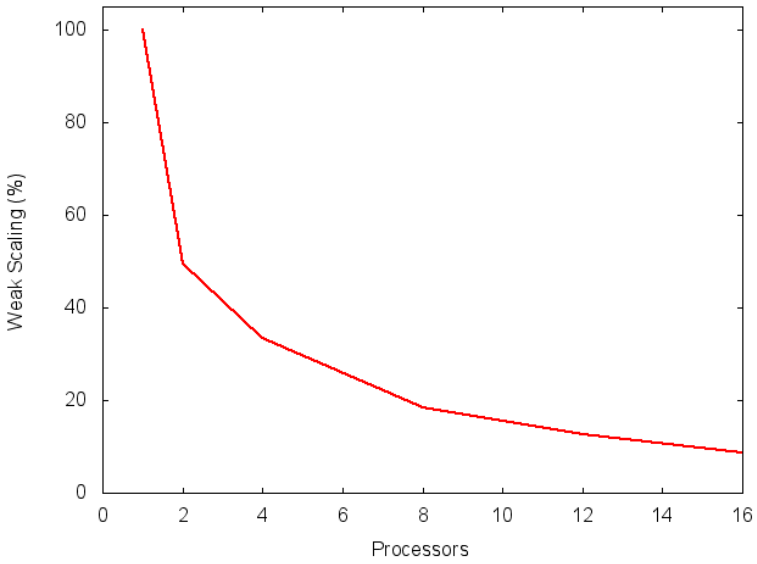

Figure 2: Weak scaling for simulations in Table 2 using the Fortran code parallelized with OpenMP.

3.2.2 MPI

Strong Scaling:

Weak Scaling:

4 Summary and Future Work