CE-791 Mini Project

Parallezing a serial SPH code for Sod Shock tube

Shridhar S Thakar, Sanket Yannuwar

December 10, 2024

1 Introduction

One of the main issues faced when solving compressible flows numerically are the discontinuities that arise due to shocks. While conventional Computational Fluid Dynamics (CFD) solves this problem by; adding numerical diffusion, 'Higher Order schemes' that can capture direction of information flow, or by solving a local 'Reimann Problem' at the shock through flux splitting. This is often computatinally expensive, and prone to numerical oscillations. A more recent approach to tackle such a problem is the use of a lagrangian approach called Smooth Particle Hydrodynamics. As the goal of this report is to document the parallelization of the code, only a brief description is provided in the next section. For a detailed and thorough explaination, the reader is referred to.

2 Smooth Particle Hydrodynamics

The compressible Euler equation for a fluid packet r_i is given by:

$$\frac{\partial^2 r_i}{\partial t^2} = -\frac{1}{\rho_i}\nabla P \tag{1}$$

where, ρ_i is the density of the packet/particle, while ∇P is the pressure gradient.

The density in SPH is defined by using a normalized smoothing kernel $W(r)$:

$$\int W(r)dr = 1 \tag{2}$$

The smoothed density is now defined as:

$$\rho(r) = \int W(r - r')\rho(r')dr' \tag{3}$$

To close the equation given by Eq. 3 we use Monte Carlo Integration:

$$\rho_i = \sum_j m_j W(r_i - r_j) \tag{4}$$

where r_j are the surrounding particles and m_j is their mass. A keep point to note is that each particle contributes to its own density in the Monte Carlo Integration. The governing equation defined by Eq. 1 is now transformed into:

$$\frac{\partial^2 r_i}{\partial t^2} = -\sum_j m_j \left(\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2} \right) \nabla W \left(r_i - r_j \right) \tag{5}$$

We further close the system by using the ideal gas state for coupling pressure and internal energy:

$$P = e\rho(\gamma - 1) \tag{6}$$

The euler's energy equation for compressible flow is given by:

$$\frac{\partial e}{\partial t} = -\frac{P}{\rho}\frac{\partial v}{\partial x} \tag{7}$$

This gives use the lagrangian equaivalent form for the energy equation as:

$$\frac{\partial e_i}{\partial t} = \sum_j m_j \left(\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2} \right) \left(v_i - v_j \right) \nabla W \left(r_i - r_j \right) \tag{8}$$

2.1 Initialization

A total of 400 particles in the sod shock tube of dimensionless length 1.2, were intialized as follows:

$$x_i = \begin{cases} i(\Delta x/4) - x_0 + \Delta x/4 & \text{if } 0 \leq i \leq 319, \\ (i - 320)\Delta x & \text{if } 320 \leq i \leq 399, \end{cases} \tag{9}$$

where $x_0 = 0.6$ and $\Delta x = x_0/80$. The particles on the left are low density of 0.25, while the particles on the right have a high density 1. The energy is initialized as:

$$e_i = \begin{cases} 2.5 & \text{if } 0 \leq i \leq 319, \\ 1.795 & \text{if } 320 \leq i \leq 399, \end{cases} \tag{10}$$

3 Parallelizing Procedure

The simulation is parallelized using the Message Passing Interface (MPI), where the particles are distributed across multiple processes. The orignal serial code (github repository) is linked in the references. The parallel implementation is submitted as a separatefile. Each process handles a subset of the particles, performs computations on them, and shares necessary information with other processes. As each particle is defined as a struct in the serial code, we need to define an mpi object, as mpi by default only supports primitive types like int, double and float.
The main steps in the parallelization procedure are as follows:

Algorithm 1 Parallelization of SPH Simulation

```
Initialize MPI environment
Get the rank of the process and the total number of processes
Create MPI datatype for the particle structure
Divide particles across processes:
  localParticles ← Subset of particles for each process
Perform particle initialization for each process (local state)
For each timestep:
  record(localParticles, step) {Record initial state of particles}
  start_timer() {Start performance timer}
  For each particle in localParticles:
    Half-kick update on velocity and energy
    Drift update on position
    Handle boundary conditions if applicable
    update(localParticles) {Compute primitive variables}
    If Artificial Viscosity is enabled:
      Compute viscosity force and add it to the acceleration
      Compute power due to viscosity
    Second half-kick update on velocity and energy
  update(localParticles) {Update with final values}
  record(localParticles, step) {write to file final state of particles}
  Gather all particles to rank 0:
    allParticles ← All processes gathered on rank 0
  Calculate performance metrics for the timestep
  stop_timer() {Stop performance timer}
End for each timestep
Gather total performance (FLOPS) across all processes
Display total execution time and performance on rank 0
Finalize MPI environment
```

4 Results and Comparison