Molecular Dynamics simulation technique using Message Passing Interface (MPI) library

Andrew Pownuk
Osei Kofi Tweneboah
Kamal Nyaupane
Md Al Masum Bhuiyan

Abstract

In this project, we present the parallel programming framework in molecular dynamics of an n-body problem using Message Passing Interface (MPI) library. We measure the performance of our implementation for different number of processes and characterize the method for minimizing computational cost at the expense of communication. Our approach is to use Newton's second law from classical mechanics and the concepts of statistical mechanics.

1 Introduction

Molecular dynamics (MD) is a computer simulation technique that allows one to predict the time evolution of a system of interacting particles (atoms, molecules, granules etc.) If the force between the particles is completely described by adding the forces between all pairs of particles and if the force between each pair acts along the line between them, this is called an n-body central force problem (often just an n-body problem). Such a problem is a good choice for parallelization because it can be described with n items (the particles) but requires $\mathcal{O}(n^2)$ computation (all the pairs of particles). Thus, we can expect good speedup for large problems because the communication between processes will be small relative to the computation.

1.1 Basics of Molecular dynamics

First, for a system of interest, one has to specify:

- Set of initial conditions i.e. initial positions and velocities.
- Secondly, the evolution of the system in time can be followed by solving a set of classical equations of motion for all particles in the system.

Within the framework of classical mechanics, the equations that governs the motion of classical particles are the ones that correspond to the second law of classical mechanics formulated by Sir Isaac Newton. Newton's second law of motion states that force, F, equals to the rate of change of momentum, $\frac{dp}{dt}$, i.e

$$\begin{aligned} F &= \frac{dp}{dt}, \quad p = mv \\ &= \frac{d(mv)}{dt} \\ &= \frac{mdv}{dt} \\ &= ma \end{aligned}$$

where F is the force exerted on the particle, m is mass of the particle and a is the acceleration of the particle.

To calculate the force experienced by each atom in our problem, we use the gravitational force i.e.

$$F = \frac{GM_1M_2}{r^2},$$

where M_1 and M_2 are the masses of the interacting particles, G is the gravitational constant and r is the distance between the two particles. In the next section, we describe the method used to find the velocity and position of every particle.

1.2 Euler forward method

From the knowledge of the force on each atom, we can determine the position and velocity of the particle in the system. Here, we use the explicit Euler's method to get the new parameters i.e position and velocity.

$$\begin{aligned} \overrightarrow{v}_{i+1} &= \overrightarrow{v}_i + \overrightarrow{a}_i \Delta t \\ \overrightarrow{r}_{i+1} &= \overrightarrow{r}_i + \overrightarrow{v}_i \Delta t \\ t_{i+1} &= t_i + \Delta t \end{aligned}$$

2 The n-body problem

In this section, we study the n-body problem using MPI features, including new collective operations, persistent communication request, and new derived data-types. In implementing an n-body code, we first need to divide the particles evenly among the processes. For example, if there are 100 particles and 10 processes, we put the first 10 particles on process 0, the second 10 particles on process 1, and so forth. To compute the forces on the particles, each process must access all the particles on the other processes.

First, we define a particle datatype and send the data to other processors. We could use **MPI_Send**, **MPI_Recv** to send the data. But it does not scale (it takes time proportional to the number of processes), it may deadlock (because the code requires that **MPI_Send** provide buffering), and other problems as well which will be discussed in the presentation. To overcome these problems, we use **MPI_Gather** and **MPI_Bcast**.

It is more convenient and efficient to combine the gather and broadcast operations into a single operation. **MPI_Allgather** does this. If all processes had the same number of particles, then we could use **MPI_Allgather** to get the particles.

In most cases, however, each process will have different numbers of particles. Then, we can use a variant of **MPI_Allgather** that permits differing sizes of data to be sent from each process. The routine **MPI_Allgatherv** takes the lengths of each item to be received and the displacement relative to the receive buffer (in units of the extent of the receive datatype) where the item will be stored.

```
int MPI_Gather(const void *sendbuf, int sendcount, MPI_Datatype sendtype,
               void *recvbuf, int recvcount, MPI_Datatype recvtype, int root,
               MPI_Comm comm)

int MPI_Allgather(const void *sendbuf, int sendcount, MPI_Datatype sendtype,
               void *recvbuf, int recvcount, MPI_Datatype recvtype,
               MPI_Comm comm)

int MPI_Allgatherv(const void *sendbuf, int sendcount, MPI_Datatype sendtype,
               void *recvbuf, const int *recvcounts, const int *displs,
               MPI_Datatype recvtype, MPI_Comm comm)
```

Figure 1: C bindings for n-body code

3 Cost Analysis

On networks supporting bidirectional, fully connected, single-ported communication the **allgather** operation can be solved in the optimal $\log_2 p$ number of communication steps for any p number of processors. If m is the total amount of data to be gathered on each process, then at every step, each process sends and receives $\frac{m}{p}$ amount of data. Therefore, the time taken by this algorithm is given by,

$$T_{\text{comm}} = t_s \log_2 p + t_w \frac{(p-1)m}{p}. \tag{1}$$

The bandwidth term t_w cannot be reduced further because each process must receive $\frac{m}{p}$ data from $p-1$ other processes.

4 Run-time of the n-body problem

For our n-body problem, the message transmitted across p processors is $m = 4n/p$. The serial runtime for our n-body problem is given as

$$T_{\text{s}} = n^2 \tau t_c. \tag{2}$$

and the parallel runtime of the n-body problem is:

$$T_{\text{p}} = \frac{n^2 \tau t_c}{p} + \left(t_s \log_2 p + 4 t_w \frac{(p-1)n}{p^2} \right) \tau \tag{3}$$

where t_c is the time for an arithmetic operation, τ is the number of time iteration, t_s is the message startup time, and t_w is the transmission time per word.

5 Theoretical Analysis

5.1 Speedup

In our problem, we have the serial runtime,

$$T_{\text{s}} = n^2 \tau t_c$$

and parallel runtime,

$$T_{\text{p}} = \frac{n^2 \tau t_c}{p} + \left(t_s \log_2 p + 4 t_w \frac{(p-1)n}{p^2} \right) \tau.$$

Therefore, speedup is:

$$\begin{aligned}\text{speedup} &= \frac{T_\text{s}}{T_\text{p}} \\ &= \frac{n^2\tau t_c}{\frac{n^2\tau t_c}{p} + \left(t_s \log_2 p + 4t_w \frac{(p-1)n}{p^2}\right)\tau} \\ &= \frac{n^2 t_c}{\frac{n^2 t_c}{p} + \left(t_s \log_2 p + 4t_w \frac{(p-1)n}{p^2}\right)}\end{aligned}$$

5.2 Isoefficiency

The parallel overhead function, $T_0(W,p)$, is given as:

$$\begin{aligned}T_0(W,p) &= pT_p - T_s \\ &= p\left(\frac{n^2\tau t_c}{p} + \left(t_s \log_2 p + 4t_w \frac{(p-1)n}{p^2}\right)\tau\right) - n^2\tau t_c \\ &= n^2\tau t_c + t_s\tau p \log_2 p + 4t_w\tau\frac{(p-1)n}{p} - n^2\tau t_c\end{aligned}$$

Thus,

$$T_0(W,p) = t_s\tau p \log_2 p + 4t_w\tau\frac{(p-1)n}{p}. \tag{4}$$

We know,

$$W = K \cdot T_0(W,p) \quad \text{where,} \quad K = \frac{E}{1-E} \quad \text{where} \quad E = \text{efficiency}$$

Now, $W = \ominus(n^2) \implies n = \ominus(W^{1/2})$ and so (4) becomes:

$$T_0(W,p) = t_s\tau p \log_2 p + 4t_w\tau\frac{(p-1)W^{1/2}}{p}. \tag{5}$$

From (5), either

$$W = Kp \log_2 p \tag{6}$$

or

$$W = K\frac{(p-1)W^{1/2}}{p}$$

$\implies$

$$W = K\frac{(p-1)^2}{p^2} \tag{7}$$

Since $Kp\log_2 p$ has a higher asymptotic rate than $K\frac{(p-1)^2}{p^2}$, the isoefficiency of the parallel system is $W = \ominus(p\log_2 p)$. The constant K is the factor that most limit the scalability of the algorithm.

5.3 Cost Optimality

Our parallel system is cost optimal if and only if , $pT_p = \ominus(W)$.
From this we have,

$$\begin{aligned} pT_p &= T_0(W,p) + W \\ \implies \ominus(W) &= T_0(W,p) + W \\ \implies T_0(W,p) &= \mathcal{O}(W) \end{aligned}$$

Thus,

$$t_s \tau p \log_2 p + 4t_w \tau \frac{(p-1)n}{p} = W \tag{8}$$

From our problem size, $W = \ominus(n^2) \implies n = W^{\frac{1}{2}}$ and so (8) becomes,

$$\begin{aligned} t_s \tau p \log_2 p + 4t_w \tau \frac{(p-1)W^{1/2}}{p} &= W; \\ \implies \frac{(p-1)W^{1/2}}{p} &= W \\ \implies \frac{(p-1)}{p} = W^{\frac{1}{2}} &= n \\ \implies \frac{(p-1)}{p} &= n \\ \implies p &= \mathcal{O}(n^2). \end{aligned}$$

5.4 Performance results

Time of the calculations in seconds (Stampede)

n	p =2	p=4	p= 8	p=16
1000	0.259	0.131	0.067	
2000	1.216	0.526	0.263	0.133
4000	4.143017	2.441	1.051	0.525
8000	19.371	8.320	4.911	2.104

6 Conclusion

In this project, we have discussed how molecular dynamics simulations can be performed using MPI library. We theoretically analyzed the speedup, isoefficiency and cost optimality of our problem. The performance of our implementation was done for different number of processes and we characterized the method for minimizing computational cost at the expense of communication using MPI_Allgather.

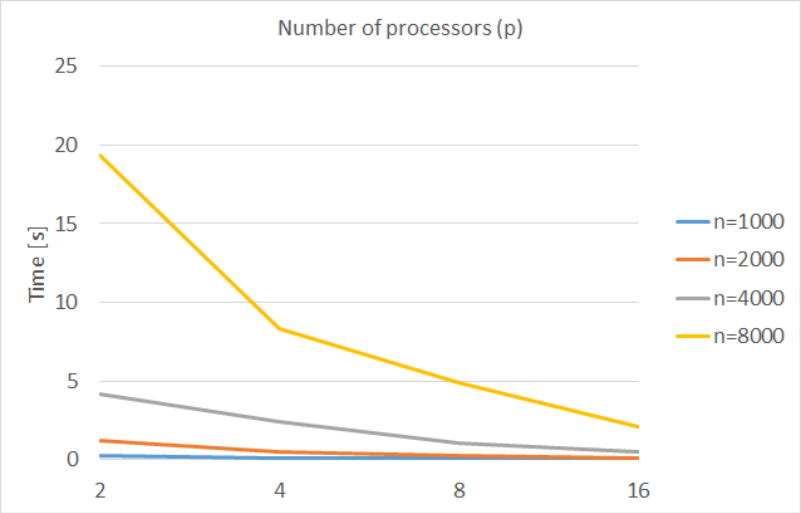

Figure 2: Plot of time for number of particles and processors.

In future, once we use the implementation of OpenMP or GPU as well, it would give us a comprehensive idea about the cost analysis of n-body problems which would be helpful to apply in astrophysics, plasma physics and fluid dynamics as well.

References

[1] "Introduction to Parallel Computing", Second Edition - Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar.

[2] "A priemier for parallel implementation of molecular dynamics simulaitons", David Keffer, 2003.

[3] "Using MPI Portable Parallel Programming with the Message-Passing Interface" - Third Edition - William Gropp, Ewing Lusk, Anthony Skjellum

[4] "High Performance N-body Simulation on Computational Grids" - Derk Jan Groen.

[5] http://www.cis.upenn.edu/ cis110/13sp/hw/hw02/nbody.shtml

[6] n-body problem, *Wikipedia*.