Project 5 - Group 5

Team Members

Group Member	R#
Michael Beebe	R11772231
Diego Salas Noain	R11794236
Bandar Alkhalil	R11836831
Yongjian Zhao	R11915830
Denish Otieno	R11743138
Shiva Kumar Neekishetty	R11842757

Required Software

- MPI implementation (we are using Open MPI)
- C Compiler (such as gcc or clang)
- Make
- Bash

Instructions

Load Proper Modules (if on HPCC's Nocona partition)

```
ml load gcc/10.1.0 openmpi/4.1.4
```

Compile

To change the MPI wrapper to something other than `mpicc` (such as `mpich`), edit line 1 of the Makefile. If you are using OpenMPI on the HPCC's Nocona partition, no changes to the Makefile are needed.

```
make
```

Run

```
./run.sh <desired number of processes>
```

The default number of processes is 4. You can change this by passing a command line argument when executing `run.sh`.

If you get an error saying "permission denied", run

```
chmod +x run.sh
```

then rerun `./run.sh`

Clean Build

```
make clean
```

Code Breakdown

The provided code is designed for parallel processing using MPI (Message Passing Interface) to calculate forces in a particle system. Here's a breakdown of its main components and functionality:

Main Function (`main`)

- **MPI Initialization**: Initializes the MPI environment, necessary for any MPI program.
- **Process Information Retrieval**: Retrieves the total number of processes (`P`) and the rank of the current process (`myRank`).
- **Particle Initialization**: Allocates memory for and initializes an array of particles (`particles`) with unique values, done only by the process with rank 0.
- **Broadcasting Particles**: The master process (rank 0) broadcasts the initialized particle array to all other processes in the MPI world.
- **Barrier Synchronization**: Ensures all processes have received the particle data before proceeding.
- **Force Calculation Call**: Calls the `force_calc` function to compute forces on each particle.
- **MPI Finalization**: Cleans up the MPI environment at the end of the program.

Force Calculation Function (`force_calc`)

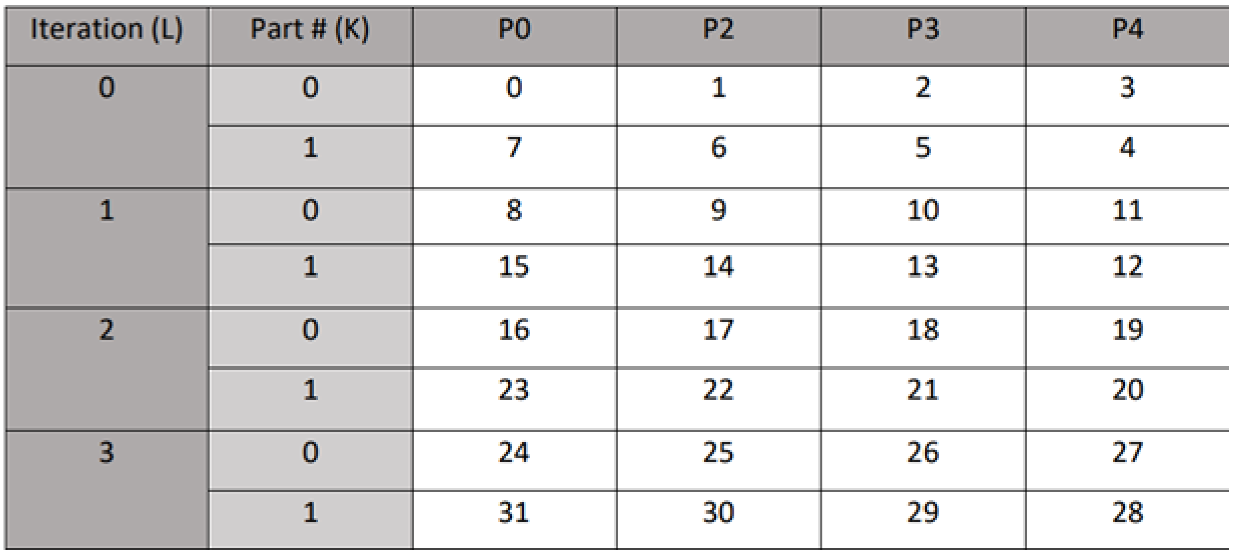
- **Initial Setup**: Allocates a buffer for force values and initializes force array `f` with zeros.
- **Iteration Logic**: Uses a nested loop structure to iterate over particle pairs. The iteration is divided into parts based on the process rank and the number of processes (`P`).
  - **Determining Row Location**: Calculates the row location for force calculation based on the current process rank and iteration.
  - **Force Calculation**: Computes the force exerted on each particle by other particles. The calculation considers the difference in positions (`diff`) and updates forces for each pair.
- **Aggregation of Results**: Uses `MPI_Reduce` to gather computed forces from all processes to the master process.
- **Result Printing**: The master process prints the final aggregated force values.

Achieving Load Balancing

Load balancing achieved by the following equation in order to distribute the rows equally between the processes:

- Part 1:
  - `Row location = (current iteration * (num_processes * 2)) + myRank`
- Part2:
  - `Row location = ((current iteration+1) * (#processes * 2) -1 myRank`

These equations will distribute the rows as the following (if we assume P=4 and N=32), please note the data in the table is the row number:



Iteration (L)	Part # (K)	P0	P2	P3	P4
0	0	0	1	2	3
	1	7	6	5	4
1	0	8	9	10	11
	1	15	14	13	12
2	0	16	17	18	19
	1	23	22	21	20
3	0	24	25	26	27
	1	31	30	29	28

Overall Workflow and Parallelism

- The main function sets up the MPI environment and distributes particle data across all processes.
- The `force_calc` function performs the core computation, leveraging parallel processing to calculate forces. Each process handles a subset of the data, contributing to the overall computation.
- The program demonstrates efficient parallel computation by dividing the workload and using MPI functions for data distribution and result aggregation.