Home Assignment 6

Yuxuan Jing

April 2020

Question 6.1

It is **possible** to reorganize the calculations so that in each iteration we did all of the calculations for each particle before proceeding to the next particle.

We can use the following pseudocode to implement the project:

```
for each timestep
    for each particle {
        Compute total force on particle;
        Find position and velocity of particle;
        Print position and velocity of particle;
    }
```

Besides the for loop orders, we do **NOT** need any other modification of the solver. The reason is that:

- The updates of position and velocity of each particle do not need any information from other particles. The only value needed is the updated total force.

Question 6.6

Using the pesudocode:

```
# pragma omp parallel
    for each timestep {
        if (timestep output) {
            # pragma omp single
            Print positions and velocities of particles;
        }
        # pragma omp for
        for each particle q
            Compute total force on q;
        # pragma omp for
        for each particle q
            Compute position and velocity of q;
```

```
}
```

- It will **NOT** cause any problem for the first *for* loop with *nowait* clause. The content inside the second loop do not need any communication between threads so each thread do not need to wait for other thread's finishing.
- It will **DO** cause problems for the second *for* loop with *nowait* clause. The print operation need to wait for the finishing and updating of all the threads for the second loop.

Question 6.10

```
int n, thread count, i, chunksize;
double x[n], y[n], a;
. . .
# pragma omp parallel num threads(thread count) \
default(none) private(i) \
shared(x, y, a, n, thread count, chunksize)
{
# pragma omp for schedule(static, n/thread count)
    for (i = 0; i < n; i++) {
        x[i] = f(i); // f is a function
        y[i] = g(i); // g is a function
}
# pragma omp for schedule(static, chunksize)
    for (i = 0; i < n; i++)
        y[i] += a * x[i];
    }// omp parallel
```

We have $n = 64$, $thread\_count = 2$, cache line size is 8 doubles, and each core has an L2 cache that can store 131,072 doubles. Since the scale L2 cache is much larger than the scale of the array. There is no rewrite of L2 cache happens in this problem.

After the first loop the value in each L2 cache is like this:

```
Thread 0
x[] ||00-07||08-15||16-23||24-31||
y[] ||00-07||08-15||16-23||24-31||

Thread 1
x[] ||32-39||40-47||48-53||54-59||
y[] ||32-39||40-47||48-53||54-59||
```

$chunksize = n/thread\_count$

The values needed for each thread for the second loop is:

```
Thread 0
x[] ||00-07||08-15||16-23||24-31||
y[] ||00-07||08-15||16-23||24-31||
Thread 1
x[] ||32-39||40-47||48-53||54-59||
y[] ||32-39||40-47||48-53||54-59||
```

After the second loop the value in each L2 cache is as following, the same as before:

```
Thread 0
x[] ||00-07||08-15||16-23||24-31||
y[] ||00-07||08-15||16-23||24-31||

Thread 1
x[] ||32-39||40-47||48-53||54-59||
y[] ||32-39||40-47||48-53||54-59||
```

So **0 L2 cache miss** happens.

$chunksize = 8$

The values needed for each thread for the second loop is:

```
Thread 0
x[] ||00-07||16-23||32-39||48-53||
y[] ||00-07||16-23||32-39||48-53||
Thread 1
x[] ||08-15||24-31||40-47||54-59||
y[] ||08-15||24-31||40-47||54-59||
```

After the second loop the value in each L2 cache is as before following:

```
Thread 0
x[] ||00-07||08-15||16-23||24-31|| + ||32-39||48-53||
y[] ||00-07||08-15||16-23||24-31|| + ||32-39||48-53||

Thread 1
x[] ||32-39||40-47||48-53||54-59|| + ||08-15||24-31||
y[] ||32-39||40-47||48-53||54-59|| + ||08-15||24-31||
```

So **8 L2 cache misses** happen , which happen on reading $x[32]$, $x[48]$, $y[32]$, $y[48]$ for thread 0 and $x[08]$, $x[24]$, $y[08]$, $y[24]$ for thread 1.

Question 6.13

Using Figure 6.6 as a guide, sketch the communications that would be needed in an "obvious" MPI implementation of the reduced n-body solver if there were three processes, six particles, and the solver used a cyclic distribution of the particles.

Answer:

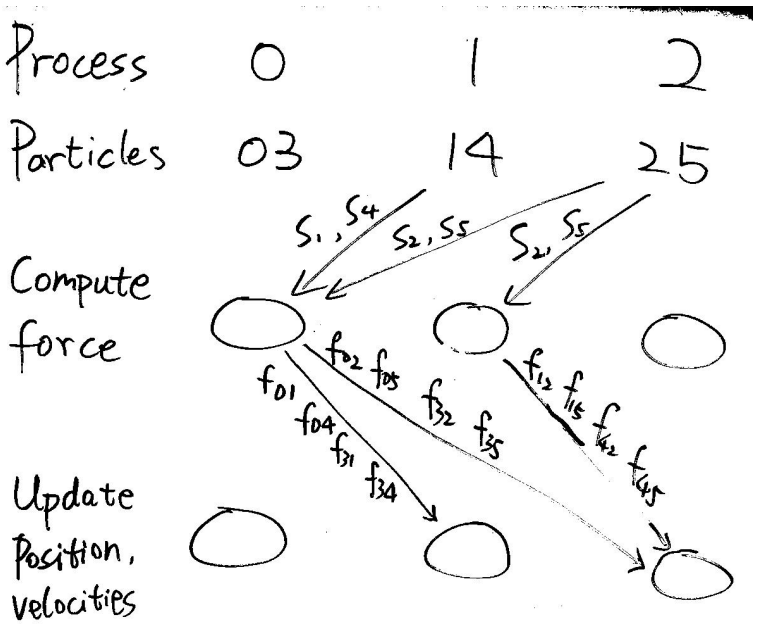

Figure 1: Communication in a possible MPI implementation of the reduced n-body solver with cyclic distribution.

Question 6.15

Assume we have p processors and the array length is N. The global index is i_{global} and the local index is i_{local}. Assuming that the number of processes evenly divides the number of elements in the global array.

6.15.a

The formula for a global index from a local index if the array has a block distribution.

$$i_{global} = local\_rank \times \frac{N}{p} + i_{local} \tag{1}$$

6.15.b

Then formula for a local index from a global index if the array has a block distribution

$$i_{local} = i_{global} - local\_rank \times \frac{N}{p} \tag{2}$$

6.15.c

Then formula for a global index from a local index if the array has a cyclic distribution

$$i_{global} = p \times i_{local} + local\_rank \tag{3}$$

6.15.d

The formula for a local index from a global index if the array has cyclic distribution

$$i_{local} = \frac{i_{global} - local\_rank}{p} \tag{4}$$

Question 6.17

6.17.a

The stack when maximum number of records happens (multiple possibilities):

```
4-3-2-NO_CITY-4-3-NO_CITY-4-NO_CITY-4
```

Maximum number of records is **10**.

6.17.b

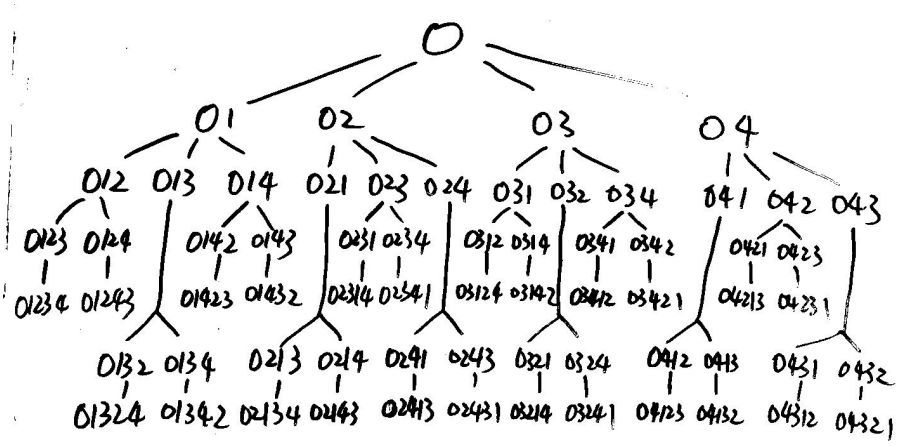

Figure 2: Search tree for five-city TSP.

6.17.c

The stack when maximum number of records happens (multiple possibilities):

```
5-4-3-2-NO_CITY-5-4-3-NO_CITY-5-4-NO_CITY-5-4-NO_CITY-5
```

Maximum number of records is **15**.

6.17.d

For a n-city TSP:

$$\text{Maximum number of records} = 1 + 2 + ... + n = \frac{n \times (n+1)}{2}. \tag{5}$$