Parallel Programming Exercise 5 - 11

Author	張凱捷 (r08922054@ntu.edu.tw)
Student ID	R08922054
Department	資訊工程所

(If you and your team member contribute equally, you can use (co-first author), after each name.)

1 Problem and Proposed Approach

Problem

Do harmonic progression and benchmark the program computing $S_{1,000,000}$ to 100 digits of precision, using various numbers of processors.

Approach

Parallelly compute the different denominators on the different processors, then reduce the result. Since the required precision is very high, we need to implement our high precision floating point number with an array. This is a well-known method to handle precision problem. I decide to let each cell of the array record 8 decimal digits. In order to handle the carry digits, I wrote my own MPI operator to reduce the arrays.

Details can be found in MPI_Op_create.

I also use the Decimal class in python to check the correctness of my program. The output od my problem will be similar as follow:

```
Python Decimal:  14.392726722865723631381127493188587676644800013744311653418433045812958507517 9
Our:             14.392726722865723631381127493188587676644800013744311653418433045812958507517 9
2 processes:     0.682317905 seconds
```

2 Theoretical Analysis Model

We require all the processors to build a small interval, but all elements are independent. Thus, it will cost $\chi\ n/p$ per processor. After that, we just need to get the sum of the answer in each processor, we need to spend $\lambda\ \lceil log\ p \rceil$ on reduction. The size of the digit array is $d/8$ which is small enough to be ignored related to n.

Hence, the expected execution time of the parallel algorithm is approximately:

$$\chi\ n/p + \lambda\ \lceil log\ p \rceil + 0.5 * d/\beta \tag{1}$$

The time complexity:

$$O(n/p + log\ p + d) \tag{2}$$

We can further formalize the speedup related to processors by the isoefficiency relation:

$$\begin{cases} \sigma(n) = 0 \\ \phi(n) = \chi\ n \\ \kappa(n,p) = \lambda\ \lceil log\ p \rceil + 0.5 * d/\beta \\ \psi(n,p) \leq \frac{\sigma(n)+\phi(n)}{\sigma(n)+\phi(n)/p+\kappa(n,p)} \end{cases} \tag{3}$$

3 Performance Benchmark

Use Equation 1 and the real execution time of one processor to get the value of χ. λ is approximately equal to 10^{-6} second. Then, use the same equation again to get all the estimation execution time.

I wrote a script to automatically submit the jobs with $p = [1, 8]$. For each job, I ran the main program for 20 times, and eliminate the smallest / largest 5 record. The value in the table is the average of the rest ten records.

Table 1: The execution time (in second)

Processors	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Real execution time	1.34238	0.67521	0.47398	0.35781	0.28761	0.23954	0.20902	0.18297
Estimate execution time	1.34238	0.67119	0.44746	0.33560	0.26848	0.22373	0.19177	0.16780
Speedup	1.00000	1.98810	2.83218	3.75164	4.66737	5.60411	6.42221	7.33650
Karp-flatt metrics		0.00598	0.02963	0.02207	0.01782	0.01413	0.01499	0.01292

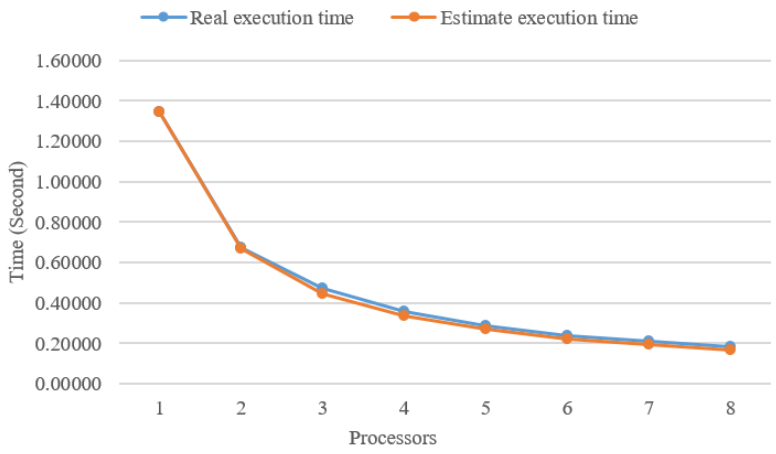

Figure 1: The performance diagram.

4 Conclusion and Discussion

4.1 What is the speedup respect to the number of processors used?

The speedup is less than p since there exist some overhead while doing parallel programming.

4.2 How can you improve your program furthermore?

Consider some cache effects. Maybe there are some libraries dealing with the high precision floating point number.

4.3 How do the communication and cache affect the performance of your program?

In this problem, the only part that requires communication is a reduction in an integer, so it does not affect our outcome too much.

4.4 How do the Karp-Flatt metrics and Iso-efficiency metrics reveal?

The Karp-Flatt metrics values fluctuate because the experiment uncertainty could lead to some small numeric error. The value is small because the sequential part is pretty small.