Parallel Programming Exercise 9 - 10

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

A perfect number is a positive integer whose value is equal to the sum of all its positive factors, excluding itself. The first two perfect numbers are 6 and 28. Write a parallel program to find the first eight perfect numbers.

Approach

Use the Euclid approach. If there exist a n such that $2^n - 1$ is a prime, $(2^n - 1) \cdot 2^{n-1}$ is a perfect number. We simply enumerate n from 1 to 50, and check the above property.

The program consist of 50 rounds. Each process will check the corresponding factor blocks, $[2..\sqrt{2^n - 1}]$, for each n, and reduce if there exist a factor of $2^n - 1$ at the end of each round. Once we discovered a prime $Process_0$ will print it out.

2 Theoretical Analysis Model

For the sequential version, the run time is $\chi n 2^{n/2}$. We can equally divide the factor to all the process, so the run time of the parallel version is simply $\chi n 2^{n/2}/p$. Finally, the communication takes $\lambda \log p$ for each round to reduce the prime property.

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

$$\chi n 2^{n/2}/p + \lambda \log p \tag{1}$$

The time complexity:

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

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

$$\begin{cases} \sigma(n) = 0 \\ \phi(n) = \chi n 2^{n/2} \\ \kappa(n) = \lambda \log p \\ \psi(n, m, p) \leq \frac{\sigma(n)+\phi(n)}{\sigma(n)+\phi(n)/p+\kappa(n,p)} \end{cases} \tag{3}$$

$$n 2^{n/2} \geq \log p \tag{4}$$

This method is highly scalable.

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. We can also do some experiments to get $w = 5 \cdot 10^8$, $r = 10^8$. 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 (second), ($n = 50$)

Processors	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Real execution time	2.15214	1.08565	0.72222	0.53894	0.43461	0.37274	0.31240	0.27757
Estimate execution time	2.15214	1.07607	0.71738	0.53803	0.43043	0.35869	0.30745	0.26902
Speedup	1.00000	1.98235	2.97988	3.99330	4.95183	5.77390	6.88914	7.75339
Karp-flatt metrics		0.00890	0.00338	0.00056	0.00243	0.00783	0.00268	0.00454

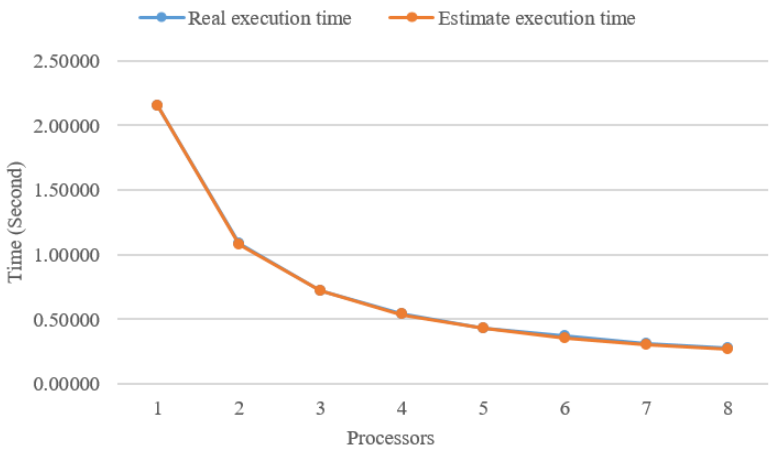

Figure 1: The performance diagram.

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. However, the overhead is small in this problem.

4.2 How can you improve your program furthermore?

While checking the divisibility of each number, we can return the function once we find a factor. It can improve a lot on sequential program, but the run time is difficult to estimate for the parallel version. Thus, we decided not to use this approach.

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

The only communication is to reduce the prime property for each number, so it do not effect out runtime too much.

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

The Karp-Flatt metrics values fluctuate is small because the overhead is pretty small.