Parallel Programming Exercise 4 - 9

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

The gap between consecutive prime numbers 2 and 3 is only 1, while the gap between consecutive primes 7 and 11 is 4. Write a parallel program to determine, for all integers less than $1,000,000$, the largest gap between a pair of consecutive prime numbers.

Approach

Sequential version: Build a prime table by prime sieving algorithm. In the beginning, we iterate through all integers from 2 to n. Once we face an integer x which is not sieved by any other integer yet, we consider it as a prime and use it to sieve the integers between $x \cdot 2$ and n. Then, we go through all the elements again and find the next prime for each prime. The overall time complexity is $O(n \cdot log\ log\ n)$.

Parallel version: For each processor, build a small prime table with the size equal to $\sqrt{n}$. Then, use the small prime table to sieve the prime table for its own block. The difference is we need to find 500 more integers in its block because we want to let the smaller prime number of the prime pair belong to the current block. In the end, use a similar approach in the sequential version to iterate through all prime pairs in each block and reduce the maximum gap to the first processor.

2 Theoretical Analysis Model

We require all the processors to build a small prime table, so it will cost $\chi(\sqrt{n}\ ln\ ln\sqrt{n})$. For the entire prime table, we build it in parallel. Suppose we can equally separate them, we can do it in $\chi(n\ ln\ ln\ n)/p$. Finally, to get the sum of the answer in each processor, we need to spend $\lambda \lceil log\ p \rceil$ on reduction.

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

$$\chi(\sqrt{n}\ ln\ ln\sqrt{n}) + \chi(n\ ln\ ln\ n)/p + \lambda \lceil log\ p \rceil \tag{1}$$

The time complexity:

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

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

$$\begin{cases} \sigma(n) = \chi(\sqrt{n}\ ln\ ln\sqrt{n}) \\ \phi(n) = \chi(n \cdot ln\ ln\ n) \\ \kappa(n,p) = \lambda \lceil log\ p \rceil \\ \psi(n,p) \le \frac{\sigma(n)+\phi(n)}{\sigma(n)+\phi(n)/p+\kappa(n,p)} \end{cases} \tag{3}$$

3 Performance Benchmark

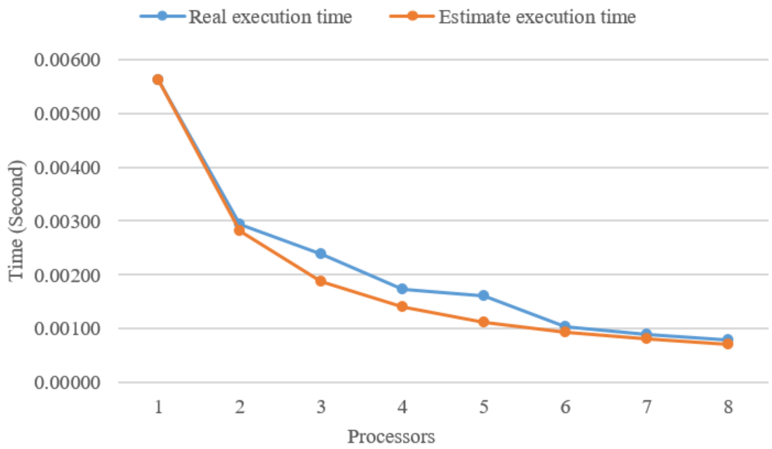
Use Equation 1 and the real execution time of one processor to get the value of χ. λ is approximately equal to 10^{-6} second. Then, use the same equation again to get all the estimation execution time.

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	0.00562	0.00294	0.00239	0.00172	0.00160	0.00104	0.00090	0.00080
Estimate execution time	0.00562	0.00281	0.00188	0.00141	0.00113	0.00094	0.00081	0.00071
Speedup	1.00000	1.91494	2.34807	3.26463	3.50441	5.40908	6.26822	7.04737
Karp-flatt metrics		0.04442	0.13882	0.07508	0.10669	0.02185	0.01946	0.01931

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 we do not need to find 500 more elements for each processor.

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.