Assignment 1

group 14
Name: Houwan Liu, Linjing Shen

1. Formulas for block partitioning

- when p/n:
  divide n into p blocks, block number = k, block size = n / p.
  first indice(k) = k * (n / p); last indice(k) = (k + 1) * (n / p) - 1 = first indice(k + 1) - 1.

- when not p/n:
  block size = [n / p], where [n/p] is round down, which means all of fractional part is discarded, remainder = n - p * [n / p].
  Thus first indice = k * [n / p] + min(k, remainder),
  The last index is one less than the first index of the next block.
  So last indice = first indice(k + 1) - 1 = ((k + 1) * [n/p] + min(k+1, remainder)) - 1.

2. Tree-structured global sum

- Conclusion:
  1, Based on the time result and plot, we can know that this code performs well both on strong scalability and weak scalability. Regarding strong scalability, we can see that when the number of processes grows, we can keep the efficiency constant without increasing the problem size. Regarding weak scalability, we can see we can keep the efficiency constant by increasing the problem size at the same rate as we increase the number of cores.
  2, when the process number grows, the time will be reduced at the same rate, which follows my initial expectation.

- Algorithm:
  Conclusion:
  The algorithm works by reducing the number of active processing units by half in each iteration of the loop. When the process is equal to 1, and can't satisfy “i < num_procs”, this part will be passed directly.

  Initialization:
  Each processing unit k starts with its own local number.

  Summation Phase:
  Repeat the following steps until k is 0;
  Find the partner for unit k in the current round, which could be determined by the pattern observed in the tree structure. The pattern often involves pairing units based on whether their IDs can be divided by 2*k in each round.
  If k is supposed to receive in this round, it uses receive to get the sum from its partner and adds it to its local sum.

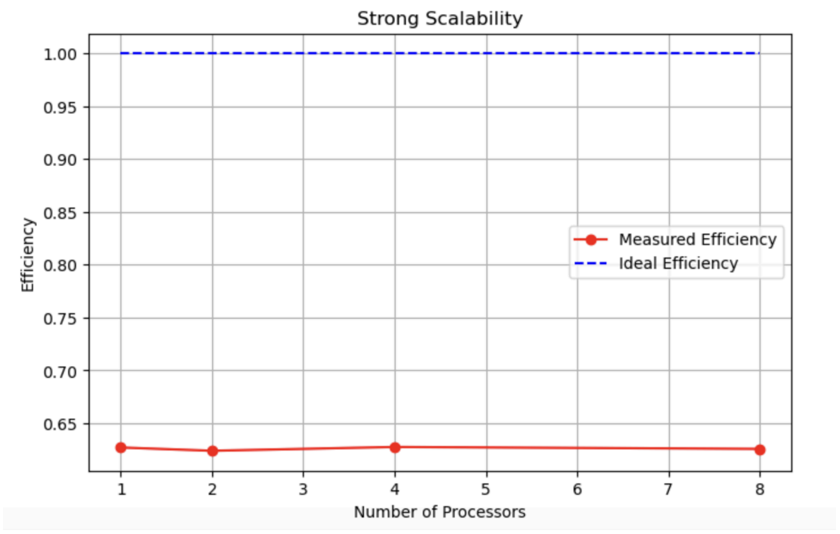
If k is supposed to send, it uses send to pass its local sum to the partner and then waits for the next round.
The algorithm reduces the number of active processing units by half in each round, only even-indexed(0, 2, 4…) units move on to the next round.

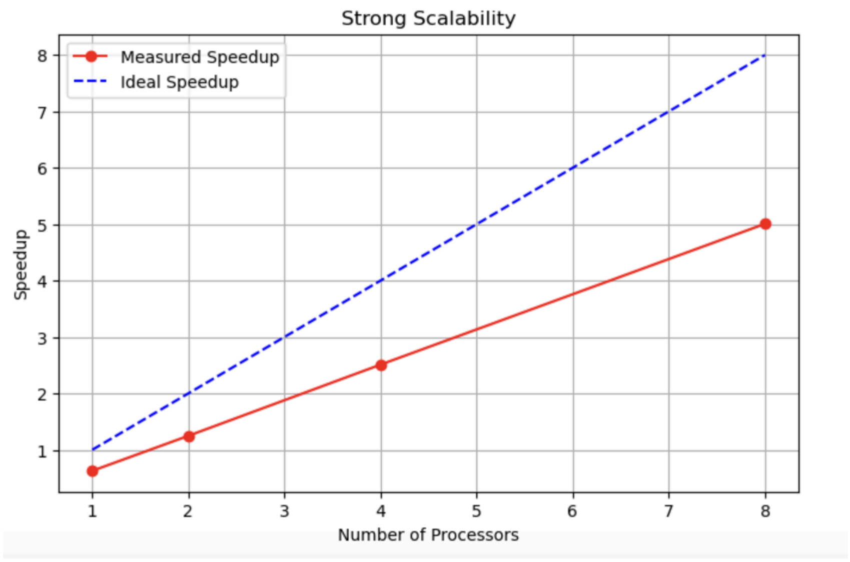
Finalization: After log2(p) rounds, unit 0 ends up with the global sum.

```
for i := 1 to num_procs-1 by i * 2
    if rank mod (2 * i) = 0 then
        if rank + i < num_procs then
            received_sum := 0
            received_sum := receive from process (rank + i)
            local_sum := local_sum + received_sum
        end if
    else
        dest := rank - i
        send local_sum to process dest
        break
    end if
end for
```

- Sequential run time: The best sequential algorithm time.
- Speedup: S = Ts/Tp, where Ts is time to execute the best serial algorithm.
- Efficiency: E = Ts/(p*Tp)
- strong scalability: problem size n = 2 ^22

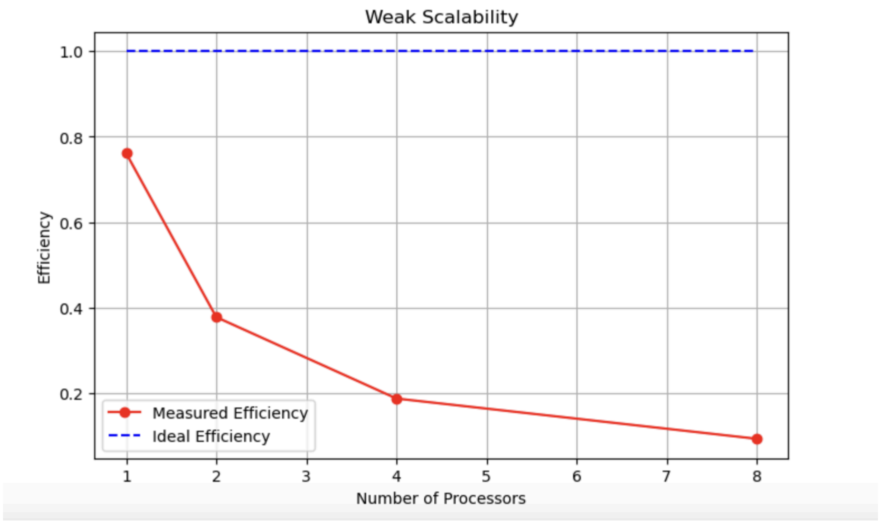
num of proc	1	2	4	8
parallel time	0.1421	0.0714	0.0355	0.0178
sequential time	0.0891	0.0891	0.0891	0.0891
speedup	0.6270	1.2479	2.5099	5.0056
efficiency	0.6270	0.6239	0.6275	0.6257





- weak scalability: problem size per process n = 2 ^ 19
  —when evaluating weak scaling, I choose to modify total problem size to 2^19, 2^20…respectively, to make sure the problem size per process is always the same.

num of proc	1	2	4	8
parallel time	0.0176	0.0177	0.0178	0.0178
sequential time	0.0134	0.0134	0.0134	0.0134
speedup	0.7614	0.7571	0.7528	0.7528
efficiency	0.7614	0.3785	0.1882	0.0941



3. Cost analysis of tree-structured global sum algorithm

The “?” in this function should be (p-1),where p is the number of process, so the final function is T(p) = (p - 1) * r + (p - 1)* a = (p - 1) * (r + a).
The number of receives and additions that processing unit 0 both are (p - 1).
First step, receive half of the processes, so receive p/2, while at the same time send p/2; Second step, receive (p/2)/2; Third time, receive ((p/2)/2)/2……
Totally receive p/2 + (p/2)/2 + ((p/2)/2)/2…. $\approx$ p - 1; At the same time send is also p - 1.

6. Speedup and efficiency

$$T_s(n) = n^2$$

$$T_p(n,p) = \frac{n^2}{p} + \log_2(p)$$

$$S(n,p) = \frac{T_s(n)}{T_p(n,p)} = \frac{n^2}{\frac{n^2}{p} + \log_2(p)}$$

$$E(n,p) = \frac{S(n,p)}{p} = \frac{\frac{n^2}{\frac{n^2}{p}+\log_2(p)}}{p}$$

Speedup (S)						
p \ n	10	20	40	80	160	320
1	1. 00	1. 00	1. 00	1. 00	1. 00	1. 00
2	1. 96	1. 99	2. 00	2. 00	2. 00	2. 00
4	3. 70	3. 92	3. 98	4. 00	4. 00	4. 00
8	6. 45	7. 55	7. 88	7. 97	7. 99	8. 00
16	9. 76	13. 79	15. 38	15. 84	15. 96	15. 99
32	12. 31	22. 86	29. 09	31. 22	31. 80	31. 95
64	13. 22	32. 65	51. 61	60. 38	63. 05	63. 76
128	12. 85	39. 51	82. 05	112. 28	123. 67	126. 89

Efficiency (E)						
p \ n	10	20	40	80	160	320
1	1. 00	1. 00	1. 00	1. 00	1. 00	1. 00
2	0. 98	1. 00	1. 00	1. 00	1. 00	1. 00
4	0. 93	0. 98	1. 00	1. 00	1. 00	1. 00
8	0. 81	0. 94	0. 99	1. 00	1. 00	1. 00
16	0. 61	0. 86	0. 96	0. 99	1. 00	1. 00
32	0. 38	0. 71	0. 91	0. 98	0. 99	1. 00
64	0. 21	0. 51	0. 81	0. 94	0. 99	1. 00
128	0. 10	0. 31	0. 64	0. 88	0. 97	0. 99

In the case of a fixed problem size, as the number of processing units increases, the speed correspondingly improves because more processing units allow for more parallel distribution of workload, resulting in shorter execution times. However, with the increase in processing units, the efficiency decreases accordingly. This is reasonable because the denominator in the formula grows faster than the numerator with the increase in the number of processing units, likely due to the added overhead of parallelization, such as increased communication costs between processing units and potential load imbalance issues.

In the case of a fixed number of processing units, as the problem size increases, the speed likewise increases as larger problem sizes benefit more from parallelization, requiring more workload distribution among the processing units. However, with the increase in problem size, the efficiency correspondingly decreases. This is also reasonable because as the problem size grows, the parallelization-related overhead increases accordingly.

7. Scalability

Given that $E(n,p) = \frac{Ts(n)}{Tp(n,p)\times p}$.

Based on the question, we increase p to $k \times p$, where $k > 1$. Suppose the new problem size is n', then the new parallel execution time is $Tp(n', k \times p)$.

According to the same efficiency, we get the equation:

$$\frac{Ts(n)}{Tp(n,p) \times p} = \frac{Ts(n')}{Tp(n', k \times p) \times (k \times p)}$$

Substitute the values of $Ts(n)$ and $Tp(n,p)$ into n':

$$\frac{n}{\left(\frac{n}{p} + \log_2(p)\right) \times p} = \frac{n'}{\left(\frac{n'}{k\times p} + \log_2(k \times p)\right) \times (k \times p)}$$

Solving the equation, we get:

$$n' = n \times k \times \left(1 + \frac{\log_2(k)}{\log_2(p)}\right)$$

Therefore, we need to increase n by $k \times \left(1 + \frac{\log_2(k)}{\log_2(p)}\right)$ times to maintain the same efficiency.