# Project Euler Problem 6: Sum square difference

Maria Cristina Menendez Ortiz

July 8, 2014

# 1 Design

## 1.1 The problem

Find the difference between the sum of the squares of natural numbers

$$s_i = 1^2 + 2^2 + 3^2 + ... + n^2 \tag{1}$$

and the square of the sum.

$$s_j = 1 + 2 + 3 + ... + n \tag{2}$$

$$d_T = s_i - s_j^2 \tag{3}$$

## 1.2 STAPL

### 1.2.1 Pseudocode

Input: n
Output: The difference between the sum of the square of numbers and the square of the sum.
Fill the array with sequential values, from 1 to n.
Compute the inner product of the array with itself.
Compute the sum of the values in the array. Square the sum of the values int the array. Compute the difference between the inner product and the square of the sum.

#### 1.2.2 Components

- stapl::array<int> c(n)
- stapl::array_view<stapl::array<int>> v(c)
- Fills the array with sequentially increasing values, starting with 1.
  stapl::iota(v, 1);
- Compute the inner product of the array with itself. First, make the product of the elements, and then sum up the products.
  stapl::inner_product(v, v, 0);
- Compute the sum of the values in the array.
  stapl::accumulate(v, 0);

# 2 Experiments

The experiments were compiled using GNU g++ 4.7.2 with -O3 optimization level and performed on a Cray XE6m with 24 nodes, each with 16 or 32 cores and 2GB of memory per core. The system has a total of 576 cores. Scalability for $p$ processors is defined as the ratio of the execution time for 1 processor and $p$ processors. In other words, $\frac{T_1}{T_p}$. The expected behavior is a linear trend in the graph of the scalability.

## 2.1 Graphs

### 2.1.1 Strong Scaling

Strong Scaling is when we tested the same input for each processor.

Table 1 shows the time that it takes to execute the program.

| - | Input | | | |
|---|---|---|---|---|
| Processors | 1,000,000 | 10,000,000 | 100,000,000 | 1,000,000,000 |
| 1 | 0.005318 | 0.055198 | 0.541016 | 5.36348 |
| 2 | 0.004455 | 0.040912 | 0.396846 | 3.94037 |
| 4 | 0.002931 | 0.024883 | 0.233062 | 2.30440 |
| 8 | 0.002356 | 0.019600 | 0.175823 | 1.71272 |
| 16 | 0.002671 | 0.015076 | 0.125586 | 1.24923 |
| 32 | 0.002594 | 0.007410 | 0.066258 | 0.629117 |
| 64 | 0.002720 | 0.004987 | 0.035212 | 0.317434 |
| 128 | 0.003000 | 0.006377 | 0.019832 | 0.161215 |
| 256 | 0.003412 | 0.005153 | 0.024853 | 0.228704 |
| 512 | 0.003750 | 0.004466 | 0.013267 | 0.115878 |

Table 1: Relation between the input and the time.

Figure 1 is comparing the time that it takes the processors to finish the process with respect to the number of input.

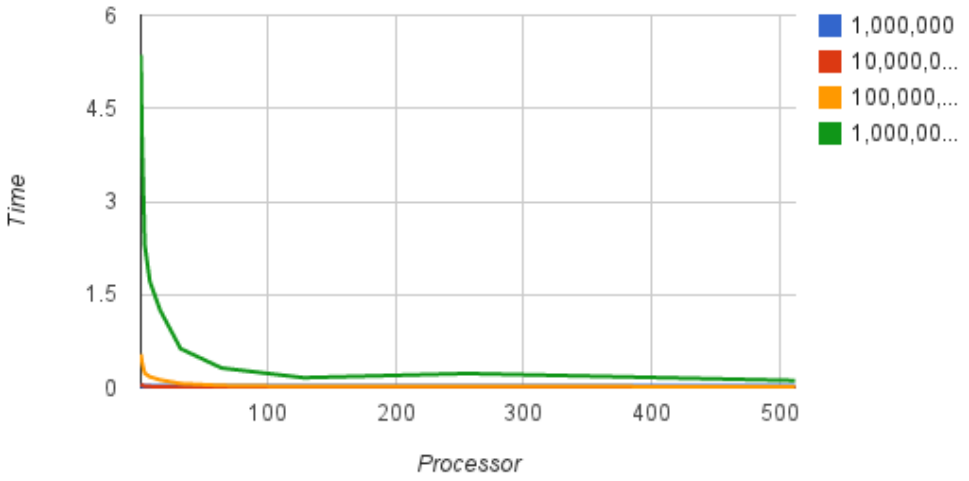
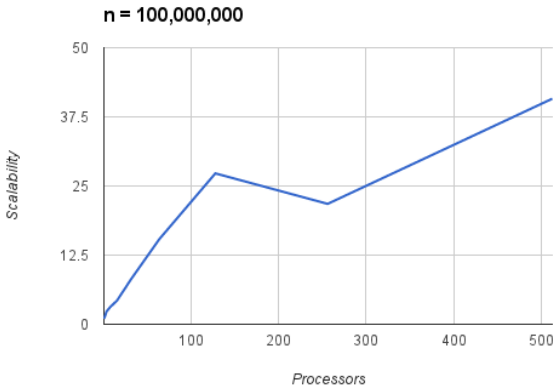


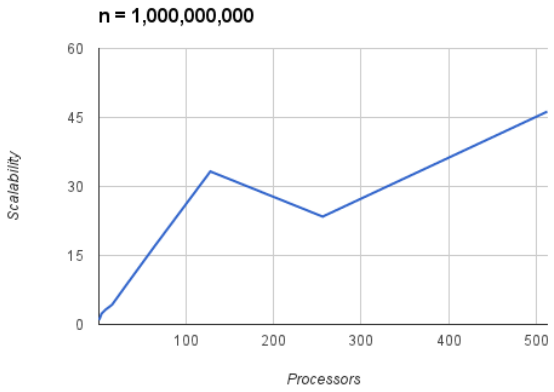

Figure 1: Relation between the input and the time.

Table 2 shows the scalability for n is equals to 100,000,000 and 1,000,000,000.

Scalability

| Processors | 10,000,000 | 1,000,000,000 |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 1.349188502 | 1.361161515 |
| 4 | 2.218301652 | 2.327495227 |
| 8 | 2.816224490 | 3.131556822 |
| 16 | 3.661315999 | 4.293428752 |
| 32 | 7.449122807 | 8.525409423 |
| 64 | 11.06837778 | 16.89636271 |
| 128 | 8.655794261 | 33.26911268 |
| 256 | 10.71181836 | 23.45162306 |
| 512 | 12.35960591 | 46.28557621 |



## 2.2 Analysis

The graph above illustrates the difference of scalability when we have two different inputs. When the input is equal to *10,000,000* the scalability increases 12 times, but when the input is 10 times bigger (*1,000,000,000*) the scalability increase 46 times. In other words, the efficiency raises when the number of inputs and processors increase.
The reason why we do not observe the expected behavior is because the input data is small. For larger input size, the graph is closer to the expected behavior.