CS 7080 High Performance Computing
Homework 1
Matrix Multiplication

Haoxiang Zhang
PowPrint: hzny2

Due: September 9, 2021

1 Homework Description

This homework let us use Boost library's matrix data type and STL to implement matrix multiplication. Given basic code without matrix multiplication part has analysis function to get some information from the algorithm that you wrote.

2 Details

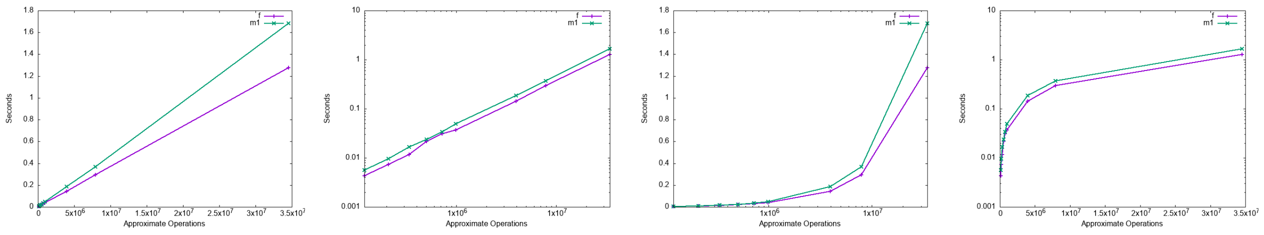
2.1 Method 1 - Basic way

This basic method using three nested for-loop to implement. First and second loop control the position of the value in output matrix. The third loop controls the number when do dot-product.

```
/*
    Use simple three nested loop to implement
*/
int size {lhs.size2()};        // get the size for multiplication

for (int r = 0; r < result.size1(); r++) {
        for (int c = 0; c < result.size2(); c ++) {
                float val {0};

                // Start to compute
                for (int s = 0; s < size; s++) {
                        val += (lhs(r, s) * rhs(s, c));
                }
                result(r, c) = val;
        }
}
```



(a) result (b) result-logxy (c) result-logx (d) result-logy

Figure 1: Three nested for loop with larger matrices

The result with smaller matrices: **real** 7m 0.083s, **user** 6m 46.868s, **sys** 0m 7.080s

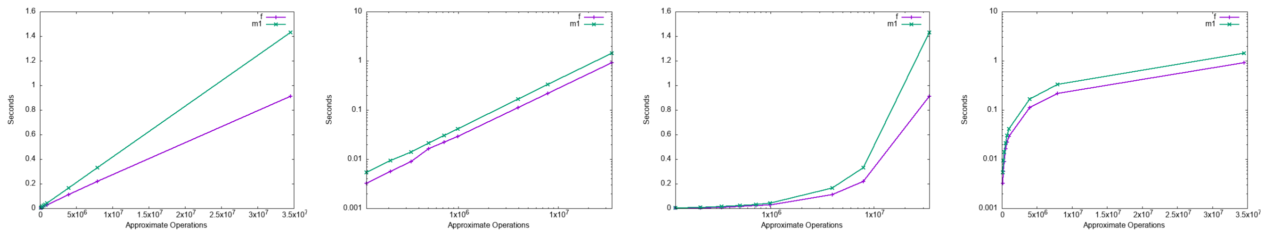
- $MyMethod$
- $100 iterations, (200, 180) * (180, 220) = (200, 220)$ avg: 0.291419 ops: 7920420 computed ele: 44000.
- $100 iterations, (400, 360) * (360, 240) = (400, 240)$ avg: 1.30275 ops: 34560640 computed ele: 96000.
- $TestBuilt - inBoostProd()$
- $100 iterations, (200, 180) * (180, 220) = (200, 220)$ avg: 0.372837 ops: 7920420 computed ele: 44000.
- $100 iterations, (400, 360) * (360, 240) = (400, 240)$ avg: 1.62358 ops: 34560640 computed ele: 96000.

2.2 Method 2 - Using data() and transpose

When using at(), it will check the range every time which costs a lot of time. Therefore, using data() with an index to find the value in a matrix will be much faster. In C++, when read data from a matrix, it will read data linearly in horizontal way, but it will read slower in vertical way. Transposing the right hand side will solve this problem.

```cpp
/*
    Transposed right hand side matrix, and using data() to access values
*/
scottgs::FloatMatrix rhs_trans = boost::numeric::ublas::trans(rhs);

for (int i = 0; i < lhs.size1()*rhs.size2(); i++) {
        float val = 0;
        for (int s = 0; s < lhs.size2(); s++) {
                val += (lhs.data()[(i/rhs.size2())*lhs.size2() + s] * rhs_trans.data()[(i%rhs.size2())*lhs.size2() + s]);
        }
        result.data()[i] = val;
}
```



(a) result (b) result-logxy (c) result-logx (d) result-logy

Figure 2: Read value using data() and transpose the rhs matrix with larger matrices test data

The result with smaller matrices: **real** 5m 37.980s, **user** 5m 34.271s, **sys** 0m 1.655s

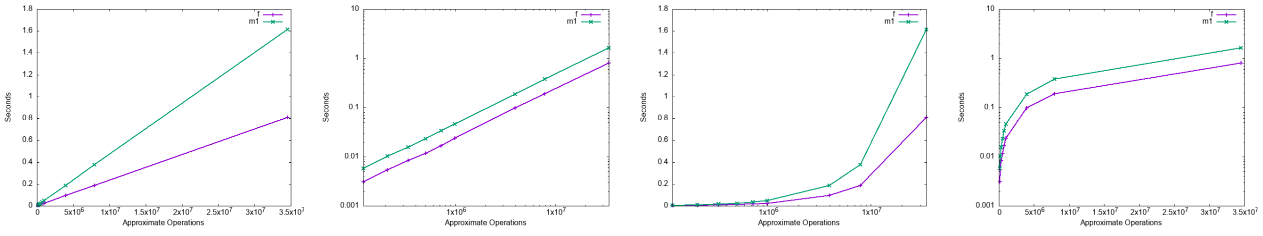
- $MyMethod$
- $100 iterations, (200, 180) * (180, 220) = (200, 220)$ avg: 0.219034 ops: 7920420 computed ele: 44000.
- $100 iterations, (400, 360) * (360, 240) = (400, 240)$ avg: 0.929477 ops: 34560640 computed ele: 96000.
- $TestBuilt - inBoostProd()$
- $100 iterations, (200, 180) * (180, 220) = (200, 220)$ avg: 0.344405 ops: 7920420 computed ele: 44000.
- $100 iterations, (400, 360) * (360, 240) = (400, 240)$ avg: 1.45838 ops: 34560640 computed ele: 96000.

2.3 Method 3 - based on method 2 store size function to variables

Beside, the values of the col and row of a matrix stored in variables, which save the time to call size1() or size2

```cpp
/*
    Based on method 2 store size function to variables
*/
int r = lhs.size1();
int c = rhs.size2();
int size = lhs.size2();        // get the size for multiplication
scottgs::FloatMatrix rhs_trans = boost::numeric::ublas::trans(rhs);

for (int i = 0; i < r*c; i++) {
        float val = 0;
        for (int s = 0; s < size; s++) {
                val += (lhs.data()[(i/c)*size + s] * rhs_trans.data()[(i%c)*size + s]);
        }
        result.data()[i] = val;
}
```



(a) result (b) result-logxy (c) result-logx (d) result-logy

Figure 3: Read value using data() and transpose the rhs matrix with larger matrices test data

The result with smaller matrices: **real** 5m 48.016s, **user** 5m 38.155s, **sys** 0m 5.407s

- $MyMethod$
- $100 iterations, (200, 180) * (180, 220) = (200, 220)$ avg: 0.180454 ops: 7920420 computed ele: 44000.
- $100 iterations, (400, 360) * (360, 240) = (400, 240)$ avg: 0.792425 ops: 34560640 computed ele: 96000.
- $TestBuilt - inBoostProd()$
- $100 iterations, (200, 180) * (180, 220) = (200, 220)$ avg: 0.385367 ops: 7920420 computed ele: 44000.
- $100 iterations, (400, 360) * (360, 240) = (400, 240)$ avg: 1.61242 ops: 34560640 computed ele: 96000.

3 Conclusion

This homework I used two methods to implement the matrix multiplication. Compare to basic method, the second method considered about the linearly reading data and the time cost of the at() function. For the results, compare **Method1**, **Method2**, **Method3**, and **built-in prod()**:

- Method 1 is about **20%** faster than built-in function
- Method 2 is about **36%** faster than built-in function
- Method 3 is about **53%** faster than built-in function
- Method 2 is about **26%** faster than Method 1
- Method 3 is about **16%** faster than Method 2
- Method 3 is about **38%** faster than Method 1

4 Lesson Learned

From this assignment and related lectures, I have learned that we need to know how does each function or data structure work, such as read data in horizontal direction is faster than vertical direction. Also every time call a function it causes time. If store the value that need to call the function many times, it's better to store it to a variable. Besides, operating the data from memory directly is much faster.