Performance Difference Between Serial Computing and Parallel Computing of GEMM - Based on OpenMP and OpenBLAS

Jiajun Li 2020011422

April 2022

Contents

1 Introduction

1.1 Background

Modern science and technology such as weather forecasting, oil exploration, and nuclear physics mostly rely on computer simulation, and the core of simulation

calculation is matrix calculation representing state transition. At the same time, computer graphics processing and the recent rise of deep learning are also highly related to matrix multiplication. Enough to see the importance of matrix multiplication.

However, matrix multiplication consumes a lot of computing resources. In addition to the continuous update of computer architecture, there is also a lot of research work in software optimization.

1.2 Content overview

First of all, this paper introduces the mathematical method of GEMM(General Matrix Multiplication) and the time complexity and space complexity of the conventional algorithm for calculating GEMM.

Secondly, this paper compares the calculation time difference of changing the loop level of the traditional algorithm in the case of using serial computing, and then compares the running time difference between serial computing and parallel computing under the premise of adopting the optimal loop level.

Then, this paper will focus on parallel computing, take the running time curve and acceleration curve as the judgment criteria, and focus on comparing the impact of OpenMP instructions under different loop levels.

Finally, the runtime curve and speedup curve of OpenBLAS will be presented.

At the same time, this paper will use 1x4 and 4x4 methods to optimize GEMM from the aspect of algorithm, and then compare the operating efficiency with the traditional loop nesting algorithm.

1.3 Program operating environment

- OS:Ubuntu-20.04-LTS(WSL2)
- CPU model:Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz
- Number of cores:4x2=8
- Frequency:2.50GHz
- Memory capacity:16GB

2 Mathematical Principles and Analysis of Traditional Algorithm of GEMM

2.1 Mathematical principles

Mathematically, an $m * n$ matrix is a rectangular array with m rows and n columns. Matrices are a common mathematical tool in advanced algebra, as well as applied mathematics such as statistical analysis.

Generic matrix multiplication (GEMM) is generally defined as:

$$C = AB$$

$$C_{m,k} = \sum_{i=1}^{n} A_{m,i} B_{i,k}$$

This is the most naive algorithm of GEMM.

2.2 Analysis of algorithms

The time complexity of this algorithm is $O(n^3)$.And the space complexity is $O(mk)$.

3 The Results of Changing The Loop Order

In the following, I will use strings such as mnk, nmk, etc. to represent the order of three levels of nested loops.For example,mkn means that for matrices $A_{m,k}$ and $B_{k,n}$, the number of loops in the outermost loop is m, the loop number in the second loop is k, and the loop number in the innermost loop is n.

According to the knowledge of permutation and combination, there are 6 different permutations of m, n, and k. Therefore, according to the experimental data measured in six different orders in 20 groups, the tables and figures are as follows:

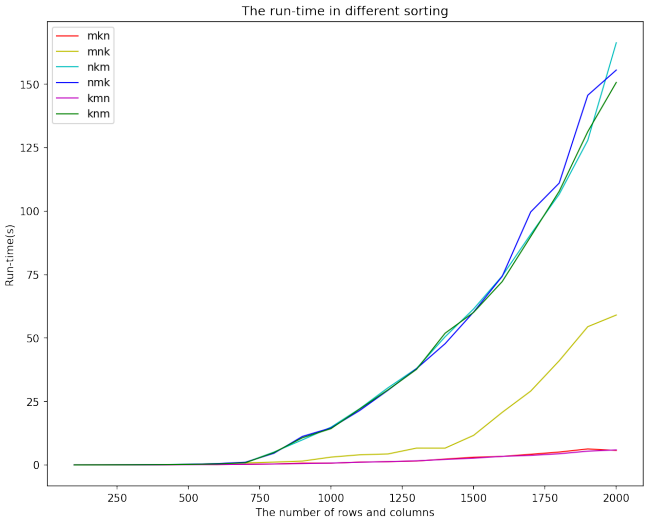

Figure 1: The run-time in different sorting

It can be seen from Table 1 and Figure 1 that the running time is the shortest and the efficiency is the highest in the order of mkn and kmn. In the case of

Table 1: The run-time in different sorting

Number of rows and columns in matrixs	The run-time in different sorting(s)					
	mkn	*mnk*	nkm	nmk	kmn	knm
100	0.00107	0.00287	0.00179	0.00639	0.00083	0.00289
200	0.01914	0.02544	0.01239	0.01830	0.00944	0.01938
300	0.04027	0.07147	0.05407	0.05004	0.02265	0.04863
400	0.07569	0.12149	0.14166	0.13837	0.04778	0.12464
500	0.22978	0.23264	0.26418	0.23825	0.09321	0.20513
600	0.14355	0.52400	0.47912	0.53141	0.14121	0.39444
700	0.23494	0.80274	0.93603	1.07091	0.25176	0.88175
800	0.38943	1.12345	5.06445	4.59460	0.37440	4.88115
900	0.71937	1.52641	9.89573	11.24396	0.53998	10.72802
1000	0.69839	3.08614	14.86757	14.57811	0.74004	14.29503
1100	1.11751	3.99593	22.10577	21.40002	1.07593	22.00786
1200	1.29852	4.33147	30.41215	29.43476	1.30555	29.58608
1300	1.53475	6.64065	38.03376	37.95410	1.63071	37.67585
1400	2.32229	6.62336	50.49467	47.77264	2.16829	51.90588
1500	3.02814	11.66413	61.47781	60.25421	2.66467	60.12730
1600	3.37208	20.70261	74.43373	74.27745	3.38118	72.22548
1700	4.19900	29.09317	90.86631	99.68398	3.76850	89.97646
1800	5.07991	41.05284	106.65818	111.05730	4.40268	107.83886
1900	6.30134	54.47069	127.88412	145.65938	5.42551	131.31637
2000	5.72157	59.06919	166.35359	155.58979	5.92946	150.72041

mnk followed by. And the running time is the longest under *nkm*, *nmk*, *knm*, which is almost 30 times as long as *mkn* and *kmn*.

4 The Results After Inserting OpenMP Statements at Different Loop Levels

Next, I will use the fastest loop nesting order (*kmn* order) above to test the running time of OpenMP statements at different loop levels when using OpenMP for parallel operations, and compare the differences.

The OpenMP statement I use is:

pragma omp parallel for

5 The Results of OpenBLAS

Next, I will use OpenBLAS to calculate the GEMM.

OpenBLAS is an open source matrix computing library that contains many matrix computing algorithms of precision and form.

Here are the result table and figure:

As can be seen from Table 3 and Figure 3, OpenBLAS runs much faster than serial calculations and simple and practical OpenMP statements. Even

Table 2: The run-time of inserting OpenMP in different loop

Number of rows and columns in matrixs	The run-time in kmn(s)			Number of rows and columns in matrixs	The run-time in kmn(s)		
	k-loop	m-loop	n-loop		k-loop	m-loop	n-loop
100	0.01126	0.09746	0.10777	1100	0.50966	0.90192	2.31689
200	0.01407	0.06606	0.12599	1200	0.77500	1.69329	3.76254
300	0.02166	0.02317	0.25242	1300	0.98313	1.90015	3.94145
400	0.01918	0.02427	0.32449	1400	1.14536	1.86219	4.40344
500	0.04910	0.03962	0.59480	1500	2.13370	2.47146	4.71807
600	0.16468	0.07250	0.61342	1600	2.09476	2.83253	5.62649
700	0.12336	0.14503	0.88443	1700	2.81898	3.24491	6.35896
800	0.18386	0.25131	1.30487	1800	2.97472	3.84590	9.09975
900	0.28858	0.40218	1.45609	1900	3.54334	4.54814	8.64596
1000	0.33949	0.58653	2.00889	2000	4.09172	5.25380	9.94972

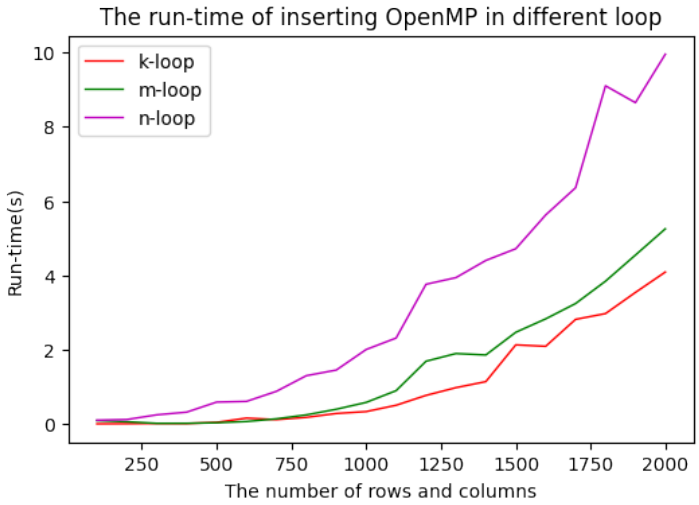

Figure 2: The run-time of inserting OpenMP in different loop

Table 3: The run-time of using OpenBLAS

Number of rows and columns in matrixs	The run-time in OpenBLAS(s)	Number of rows and columns in matrixs	The run-time in OpenBLAS(s)
100	0.02875	1100	0.09880
200	0.02835	1200	0.13433
300	0.01867	1300	0.14617
400	0.00311	1400	0.16313
500	0.05262	1500	0.20487
600	0.06019	1600	0.22121
700	0.02424	1700	0.30849
800	0.08589	1800	0.33321
900	0.05249	1900	0.54454
1000	0.09584	2000	0.43445

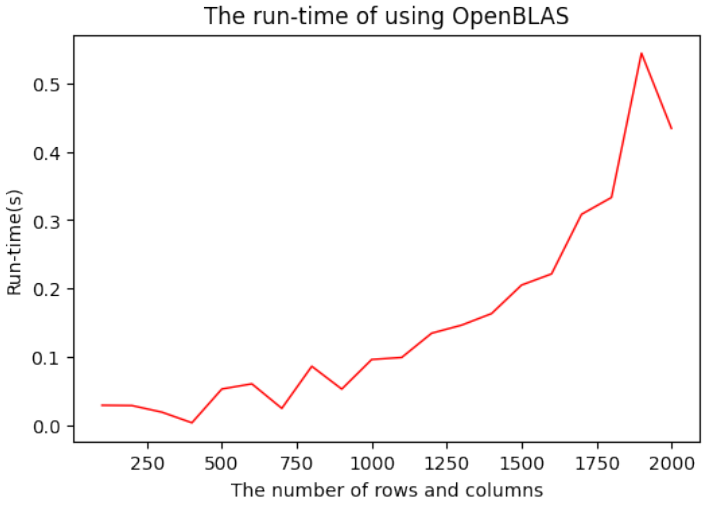

Figure 3: The run-time of inserting OpenMP in different loop

a general matrix-matrix multiplication of 1000x1000 scale can still run in 0.5 seconds.

OpenBLAS, which is used in scientific computing, data analysis, deep learning algorithms, artificial intelligence and other fields, is naturally optimized to the extreme in its internal algorithms and parallel computing processing. We still have a lot to learn and explore in this regard.

6 Other Optimization Methods of Traditional GEMM Algorithm

There are many optimization algorithms for GEMM. Here I choose to use the method of dividing the matrix into several 4x4 small matrices and using OpenMP to perform parallel operations for optimization.

6.1 Principle of 4x4-blocks algorithm

As shown in the figure, any 4x4 small matrix in the target matrix can be expressed as the product of the row and column division of the A matrix and the B matrix.

Due to space limitations (actually I can't use Overleaf to insert code segments correctly), I only introduce the optimized algorithm logic and OpenMP's parallel writing method for nested loops.

A simple observation shows that the elements of the matrix used for the innermost calculation are the same. Therefore, an element of the target matrix can be read into the register, thereby realizing 4 times of data multiplexing.

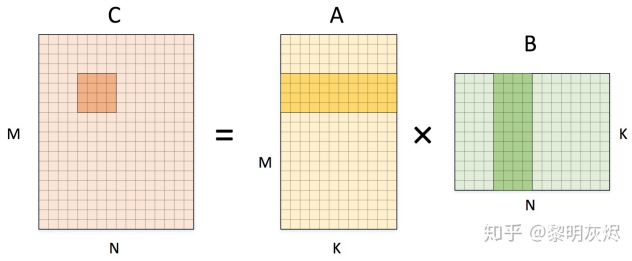

Figure 4: Matrix multiplication calculation 4x4 output

After such an optimization, the number of memory access operations becomes (3+1/4) , where 1/4 is the effect of the optimization.

6.2 The run-time comparison between the original and 4x4-blocks

The actual runtime results are as Table 4 and Figure 5.The data of the original algorithm comes from the data sorted by *mnk* in Table 1 and Figure 1.

Table 4: The comparison between *mnk* sorting and 4x4-blocks parallel running

Number of rows and columns in matrixs	The run-time of original *mnk* (s)	The run-time in 4x4 blocks(s)	Number of rows and columns in matrixs	The run-time of original *mnk* (s)	The run-time in 4x4 blocks(s)
100	0.00287	0.02012	1100	3.99593	0.81765
200	0.02544	0.02043	1200	4.33147	1.40075
300	0.07147	0.01628	1300	6.64065	0.99866
400	0.12149	0.04315	1400	6.62336	1.08468
500	0.23264	0.07015	1500	11.66413	1.38936
600	0.52400	0.10612	1600	20.70261	2.29651
700	0.80274	0.17987	1700	29.09317	2.14740
800	1.12345	0.28216	1800	41.05284	2.86001
900	1.52641	0.61573	1900	54.47069	3.08149
1000	3.08614	0.65739	2000	59.06919	4.24567

7 Conclusion

Through this experiment, I understood the concept of GEMM, related algorithm principles, and realized that changing the order of loops under the traditional three-layer nested loop algorithm changes the running time of the program in serial computing. Then try to use the OpenMP statement for preliminary parallel computing processing, and try to change the position of the OpenMP statement in the nested loop to explore the difference. Finally I learned about OpenBLAS and felt the extraordinary power of OpenBLAS in high performance computing. In the end, my original three-level nesting algorithm was optimized and changed into a 4x4-Blocks algorithm, which greatly improved the program performance.

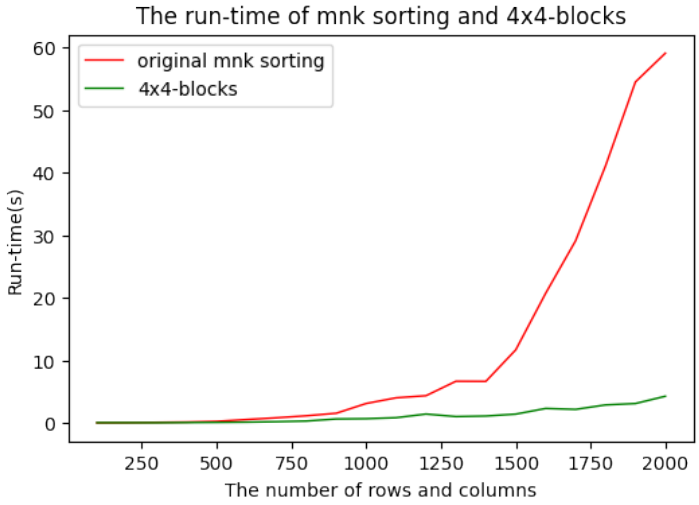

Figure 5: The run-time of *mnk* sorting and 4x4-blocks

To sum up, this is just the first step in my exposure to parallel programming. There is still a lot of knowledge waiting for me to explore and learn, and I am willing to work hard for it.