Hadoop-based MapReduce Study

2016 spring big-data processing homework II

5130309194 Shen Jiyuan

***Abstract*—MapReduce as a programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster, is a extremely significant programming framework.**

***Index Terms*—MapReduce, Hadoop, parallel, distributed, matrix multiplication.**

I. Introduction of MapReduce

A MapReduce program is composed of a Map() procedure (method) that performs filtering and sorting (such as sorting students by first name into queues, one queue for each name) and a Reduce() method that performs a summary operation (such as counting the number of students in each queue, yielding name frequencies).

The "MapReduce System" (also called "infrastructure" or "framework") orchestrates the processing by marshalling the distributed servers, running the various tasks in parallel, managing all communications and data transfers between the various parts of the system, and providing for redundancy and fault tolerance.

MapReduce libraries have been written in many programming languages, with different levels of optimization. A popular open-source implementation that has support for distributed shuffles is part of Apache Hadoop. The name MapReduce originally referred to the proprietary Google technology, but has since been genericized. By 2014, Google was no longer using MapReduce as their primary Big Data processing model, and development on Apache Mahout had moved on to more capable and less disk-oriented mechanisms that incorporated full map and reduce capabilities.

II. Introduction of Apache Hadoop

Apache Hadoop is an open-source software framework written in Java for distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware.

The base Apache Hadoop framework is composed of the following modules: ①Hadoop Common – contains libraries and utilities needed by other Hadoop modules; ②Hadoop Distributed File System (HDFS) – a distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster; ③Hadoop YARN – a resource-management platform responsible for managing computing resources in clusters and using them for scheduling of users' applications; and ④Hadoop MapReduce – an implementation of the MapReduce programming model for large scale data processing.

III. Environment Settings and Preparations

A. Hardware Environment

- Intel(R) Core(TM) i5-4200U CPU @ 1.60GHZ
- Installed RAM: 4.00 GB
- x64 Processor

B. Software Environment

- WMware Virtual Machine
- Ubuntu 14.04
- Hadoop 2.7.2
- Java version "1.7.0_95"
- OpenJDK Runtime Environment (IcedTea 2.6.4)
- OpenJDK Client VM (build 24.95-b01, mixed mode, sharing)

C. More Details about Software Environment Setting

- Notice: Here we simply skip the steps for installing VMware virtual machine and Ubuntu 14.04 in that we believe all of us actually have the experience about the process. The following process are information we get from the internet. You can directly refer to them(may have bugs): http:// blog.csdn.net/hitwengqi/article/details/8008203; http://blog.csd n. net/maojun1986/article/details/38670047.
- Add hadoop user to system group:

```
~$ sudo addgroup hadoop
~$ sudo adduser --ingroup hadoop hadoop
~$ sudo usermod -aG admin hadoop
```

- Ssh

```
~$ sudo apt-get install openssh-server
~$ sudo /etc/init.d/ssh start
~$ ps -e |grep ssh
~$ ssh-keygen -t rsa -P “”
~$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
~$ ssh localhost
~$ exit
```

- Java

```
~$ sudo apt-get install openjdk-7-jdk
~$ java -version
```

- Hadoop

```
~$ sudo tar xzf hadoop-2.7.2.tar.gz
~$ sudo mv hadoop-2.7.2 /usr/local/hadoop
~$ sudo chown -R hadoop:hadoop /usr/local/hadoop
```

[standalone]

(usr/local/hadoop/etc/hadoop/hadoop-env.sh)

```
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk
export HADOOP_HOME=/usr/local/hadoop
export PATH=$PATH:/usr/local/hadoop/bin
~$ source /usr/local/hadoop/etc/hadoop/hadoop-env.sh
```

(Here, ‘standalone’ has been installed successfully!)

[pseudo distributed mode]

(usr/local/hadoop)

```
~$ mkdir tmp
~$ mkdir hdfs
~$ mkdir hdfs/name
~$ mkdir hdfs/data
```

(usr/local/hadoop/etc/hadoop/core-site.xml)

```
<configuration>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://localhost:9000</value>
  </property>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/usr/local/hadoop/tmp</value>
  </property>
</configuration>
```

(usr/local/hadoop/etc/hadoop/hdfs-site.xml)

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
  <property>
    <name>dfs.name.dir</name>
    <value>/usr/local/hadoop/hdfs/name</value>
  </property>
  <property>
    <name>dfs.data.dir</name>
    <value>/usr/local/hadoop/hdfs/data</value>
  </property>
</configuration>
```

(usr/local/hadoop/etc/hadoop/mapred-site.xml)

```
<configuration>
  <property>
    <name>mapred.job.tracker</name>
    <value>localhost:9001</value>
  </property>
</configuration>
```

(continue)

```
~$ source /usr/local/hadooop/conf/hadoop-env.sh
~$ hadoop namenode -format
```

(usr/local/hadoop/bin)

```
~$ start-all.sh
```

(check for install)(usr/local/hadoop)

```
~$ jps
```

IV. Problem Descriptions

This problem introduces simple mathematical multiplication on matrix. And the computational program is required to designed by map-reduce.

V. Naive Computing Algorithms

A. Main Point for HDFS

When dealing with large scale of matrix, although theoretically it is possible to store such a scale one in file system, we have to optimize it in reality. Notice that normally the matrix is a sparse one, and the relations are not as many. Then we could only store those non-zero values. More concretely speaking, the structure for each record in matrix files is (i,j,A[i,j]). ‘i’ stands for lines; ‘j’ stands for columns; ‘A[i,j]’ stands for the content value. And each record exhibits one line in the hdfs file.

B. Main Point for Computation

- basic matrix multiplication review

  Let $A=(a_{ij})_{mn}$, $B=(b_{jk})_{nl}$, then
  $C=AB=(c_{ik})_{ml}$
  $=(a_{i1}b_{1j}+a_{i2}b_{2j}+...+a_{in}b_{nj})ml$
- look for independent computations

  We can easily observe that computations of each element in the matrix C are exactly independent of each other. That means, take c_{11} for example, we can firstly map all the needed data in matrix A and B to one key. And we get all these elements according to the key and do the simple multiplications and additions for c_{11}.

  Tip: take a_{11} and b_{11} for instance: a_{11} will be used by c_{11}, c_{12}, ..., c_{1l}, and b_{11} will be used by c_{11}, c_{21}, ..., c_{m1}. Generally speaking, any element in matrix A will be stored as l <key,value> (l different keys), and any element in matrix B will be stored as m <key,value> (m different keys).

C. Process View

- MAP

For each a_{ij} in A_{mn}, labeled as l <key, value>, where key=(i, k), k=1,2,...,l; value=('a', j, a_{ij}); For each b_{jk} in B_{nl}, labeled as m <key, value>, where key= (i, k), k=1,2,...,m; value=('b', j, b_{jk}); Then, the key can tell us about the direction of related data; and the value can tell us about the concrete location and value.

- SHUFFLE

Values with a same key will be added into a same list, and output <key, list(value)> pairs. This step is done automatically by Hadoop.

- REDUCE (based on <key, list(value)>)

From key, we can know which element in C are we computing for. From list(value), thanks to our structure for value in stage map, we can then differentiate elements in A and that in B, which is a ground for dot multiplications.

D. One Example

- Input : $A = \begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 0 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{matrix}, B = \begin{matrix} 10 & 15 \\ 0 & 2 \\ 11 & 9 \end{matrix}$

- HDFS form:

A:

```
1 1 1
1 2 2
1 3 3
2 1 4
2 2 5
2 3 0
3 1 7
3 2 8
3 3 9
4 1 10
4 2 11
4 3 12
```

B:

```
1 1 10
1 2 15
2 1 0
2 2 2
3 1 11
3 2 9
```

- MAP

A:

key	value	key	value
(1, 1)	('a', 1, 1)	(3, 2)	('a', 1, 7)
(1, 2)	('a', 1, 1)	(3, 1)	('a', 2, 8)
(1, 1)	('a', 2, 2)	(3, 2)	('a', 2, 8)
(1, 2)	('a', 2, 2)	(3, 1)	('a', 3, 9)
(1, 1)	('a', 3, 3)	(3, 2)	('a', 3, 9)
(1, 2)	('a', 3, 3)	(4, 1)	('a', 1, 10)
(2, 1)	('a', 1, 4)	(4, 2)	('a', 1, 10)
(2, 2)	('a', 1, 4)	(4, 1)	('a', 2, 11)
(2, 1)	('a', 2, 5)	(4, 2)	('a', 2, 11)
(2, 2)	('a', 2, 5)	(4, 1)	('a', 3, 12)
(3, 1)	('a', 1, 7)	(4, 2)	('a', 3, 12)

B:

key	value	key	value
(1, 1)	('b', 1, 10)	(3, 2)	('b', 2, 2)
(2, 1)	('b', 1, 10)	(4, 2)	('b', 2, 2)
(3, 1)	('b', 1, 10)	(1, 1)	('b', 3, 11)
(4, 1)	('b', 1, 10)	(2, 1)	('b', 3, 11)
(1, 2)	('b', 1, 15)	(3, 1)	('b', 3, 11)
(2, 2)	('b', 1, 15)	(4, 1)	('b', 3, 11)
(3, 2)	('b', 1, 15)	(1, 2)	('b', 3, 9)
(4, 2)	('b', 1, 15)	(2, 2)	('b', 3, 9)
(1, 2)	('b', 2, 2)	(3, 2)	('b', 3, 9)
(2, 2)	('b', 2, 2)	(4, 2)	('b', 3, 9)

- SHUFFLE

key	value-list	
(1, 1)	('a', 1, 1)	('b', 1, 10)
	('a', 2, 2)	('b', 3, 11)
	('a', 3, 3)	
(1, 2)	('a', 1, 1)	('b', 1, 15)
	('a', 2, 2)	('b', 2, 2)
	('a', 3, 3)	('b', 3, 9)
(2, 1)	('a', 1, 4)	('b', 1, 10)
	('a', 2, 5)	('b', 3, 11)
(2, 2)	('a', 1, 4)	('b', 1, 15)
	('a', 2, 5)	('b', 2, 2)
		('b', 3, 9)
(3, 1)	('a', 1, 7)	('b', 1, 10)
	('a', 2, 8)	('b', 3, 11)
	('a', 3, 9)	
(3, 2)	('a', 1, 7)	('b', 1, 15)
	('a', 2, 8)	('b', 2, 2)
	('a', 3, 9)	('b', 3, 9)
(4, 1)	('a', 1, 10)	('b', 1, 10)
	('a', 2, 11)	('b', 3, 11)
	('a', 3, 12)	
(4, 2)	('a', 1, 10)	('b', 1, 15)
	('a', 2, 11)	('b', 2, 2)
	('a', 3, 12)	('b', 3, 9)

- REDUCE

key	value
(1, 1)	43
(1, 2)	46
(2, 1)	40

(2, 2)	70
(3, 1)	169
(3, 2)	202
(4, 1)	232
(4, 2)	280

VI. Block Matrix Multiplication Algorithm

A. Definition on Block

Let $A \in R^{a\cdot b}$, $B \in R^{b\cdot c}$, $C \in R^{a\cdot c}$, if $C = A\cdot B$, Then one block method is <m, n, k>, and A, B, C are:

$$A = \begin{pmatrix} A_{1,1} & \cdots & A_{1,n} \\ \vdots & \ddots & \vdots \\ A_{m,1} & \cdots & A_{m,n} \end{pmatrix},\ B = \begin{pmatrix} B_{1,1} & \cdots & B_{1,k} \\ \vdots & \ddots & \vdots \\ B_{n,1} & \cdots & B_{n,k} \end{pmatrix},$$

$$C = A\cdot B = \begin{pmatrix} C_{1,1} & \cdots & C_{1,k} \\ \vdots & \ddots & \vdots \\ C_{m,1} & \cdots & C_{m,k} \end{pmatrix}.$$

Each block: $C_{\alpha,\beta} = \sum_{\gamma=1}^{n} A_{\alpha,\gamma} B_{\gamma,\beta}$

B. Hadoop-based implementation idea

Different from dot multiplication on every element, here block multiplication needs two map-reduce process: first one is to make a block action on inputs and output; second one is to make multiplication computations on blocks and collect all data. The process can be expressed as follows:

- Input:

$$A = \begin{pmatrix} A_{1,1} & \cdots & A_{1,n} \\ \vdots & \ddots & \vdots \\ A_{m,1} & \cdots & A_{m,n} \end{pmatrix},\ B = \begin{pmatrix} B_{1,1} & \cdots & B_{1,k} \\ \vdots & \ddots & \vdots \\ B_{n,1} & \cdots & B_{n,k} \end{pmatrix},$$

- MAP

	α=1, β=1	α=1, β=2	…	α=m, β=k
γ=1	$A_{11}\times B_{11}$	$A_{11}\times B_{12}$	…	$A_{m1}\times B_{1k}$
γ=2	$A_{12}\times B_{21}$	$A_{12}\times B_{22}$	…	$A_{m2}\times B_{2k}$
…	…	…	…	…
γ=n	$A_{1n}\times B_{n1}$	$A_{1n}\times B_{n2}$	…	$A_{mm}\times B_{nk}$

- REDUCE

α=1	α=2	…	α=m
$\sum_{\beta=1}^{k}\sum_{\gamma=1}^{n} A_{1\gamma}\cdot B_{\gamma\beta}$	$\sum_{\beta=1}^{k}\sum_{\gamma=1}^{n} A_{2\gamma}\cdot B_{\gamma\beta}$	…	$\sum_{\beta=1}^{k}\sum_{\gamma=1}^{n} A_{m\gamma}\cdot B_{\gamma\beta}$

C. Matrix Generating

Representation symbols: δ stands for sparse ratio, which is the ratio in which non-zero elements account for all elements. By default, we set $m=n=2^{13}$, $\delta=2^{-7}$. Then we can generate our matrix by using MapReduce and the following algorithms:

- Alg1

```
Matrix Generator(Mapper):
Require: matrix height m
for i←1 to m do
emit(i, {})
end for
Require: matrix height m
for i←1 to m do
emit(i, {})
end for
```

- Alg2

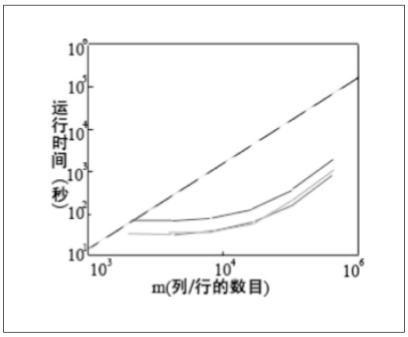
```
Matrix Generator(Reducer):
Require: row index i, sparseness δ, matrix width n
row←{}
for 1 to n do
if random() < δ then
row←row U {random()}
end if
end for
emit(i, row)
```

VII. Results and Analysis

In this experiment, we focus on the correlations between time efficiency and input matrix scale, between time efficiency and sparse ratio, and between time efficiency and block strategy.

A. Matrix Order vs. Complexity

Theoretically, the complexity of two m-order matrix multiplication is $O(m^3)$. However in the case of sparseness, even though the order can be very large, sparseness can usually be high. Then in experiments, real complexity is lower than theoretical one. Here, we generate six test groups, and matrix in each test group are randomly-generated m-order sparse matrix(default parameters mentioned).



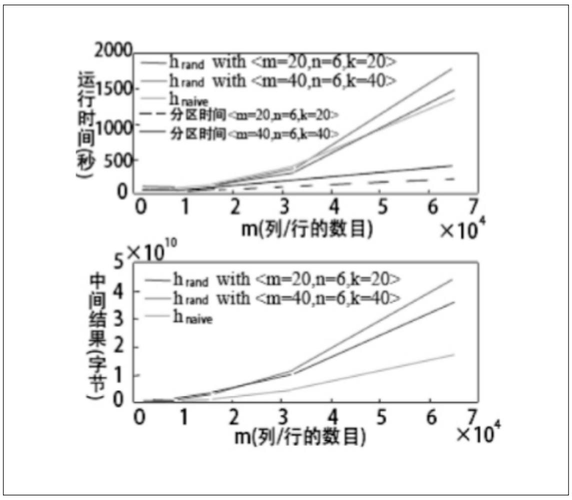
From the figure, three curves respectively stand for computational time, blocking time and multiplication time. And we can explicitly conclude that time complexity will increase non-linearly responding to the increase of matrix order.

However, another observation is that the slope is smaller than expected. This can lies in the sparseness of input matrix, and a large amount of zero units are not stored and calculated.

B. Blocking Function vs. Complexity

Let the direction for each block is directed by function h. Originally, $h_{naive}(\alpha,\beta,\gamma)=\alpha$ mod p, where α=0,..., m-1; β=0,..., n-1; p is the computational node number. Other considerations, $h_{rand}(\alpha,\beta,\gamma)$=hash($\alpha,\beta,\gamma$) mod p. Actually it is much more difficult now for us to draw a solution for this one by coding, therefore we here just directly test some constants.

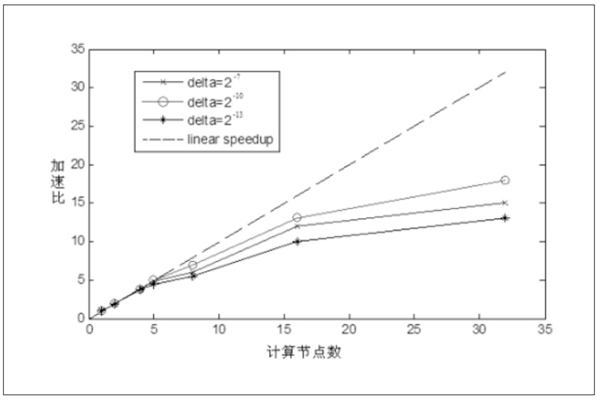
Considering ①h_{rand} = <m=20, n=6, k=20>;
②h_{rand} = <m=40, n=6, k=40>; ③h_{naive} = <m=20, n=6, k=20>;
And we can finally get the following graph:



From the figures, three curves respectively stand for blocking strategies portrayed above. The upper figure focuses on time complexity, and the below one focuses on space complexity. Then we can explicitly observe that h_{naive} is the strategy which introduces better performance both in time complexity and space complexity.

C. Speedup Ratio vs. Computational node number p

We consider three test situations referring to the sparseness δ. And we can get the figure shown below:



From the figure above, we can directly conclude that when computational nodes increase, the speedup ratio will definitely increase. However, none of the three multiplications reach a linear speedup. For instance, we observe that for $\delta = 2^{-7}$, it's speedup is close to 7 when there are 8 nodes.

VIII. TIPS

New sparse matrix data files: (.csv) In your files, please make sure only non-zero values are stored, and are exactly in the format mentioned in part V.

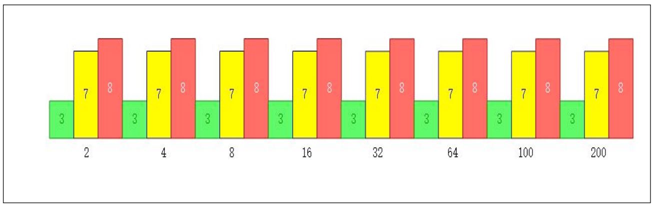
Notice that the installation codes need to be revised properly to adapt to your environment. Please check them carefully.

Our source codes includes two versions:MatrixMultiply.java and SparseMatrixMultiply.java. They are two implementations for two algorithms involved in our report.

Table II. Parallel Computing

thread	add		sub		multiply	
	time	speed	time	speed	time	speed
2	0.0000	3.000	0.0000	1#inf	0.0000	1#inf
4	0.0000	3.000	0.0000	1#inf	0.0000	8.000
8	0.0000	3.000	0.0000	1#inf	0.0000	8.000
16	0.0000	3.000	0.0000	1#inf	0.0000	8.000
32	0.0000	1#inf	0.0000	7.000	0.0000	8.000
64	0.0000	3.000	0.0000	1#inf	0.0000	8.000
100	0.0000	3.000	0.0000	7.000	0.0000	1#inf
200	0.0000	3.000	0.0000	1#inf	0.0000	8.000

Figure I. Speedup-ThreadNum(conclusion)



A. Matrix Dimension = 10

Table III. Serial Computing Time (s)

addition	subtraction	multiplication
0.0000	0.0000	0.0000

Table IV. Parallel Computing

thread	add		sub		multiply	
	time	speed	time	speed	time	speed
2	0.0000	10.49	0.0000	23.25	0.0000	2.682
4	0.0000	10.49	0.0000	31.00	0.0000	1.283
8	0.0000	10.49	0.0000	23.25	0.0000	1.553
16	0.0000	10.50	0.0000	15.50	0.0000	2.107
32	0.0000	8.400	0.0000	15.50	0.0000	2.107
64	0.0000	7.000	0.0000	18.60	0.0000	2.185
100	0.0000	8.400	0.0000	18.60	0.0000	2.034
200	0.0000	8.400	0.0000	15.50	0.0000	1.788

Figure II.I Speedup-ThreadNum(add)(2-200)

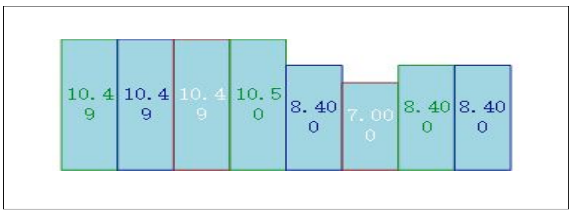


Figure II.II Speedup-ThreadNum(sub)(2-200)

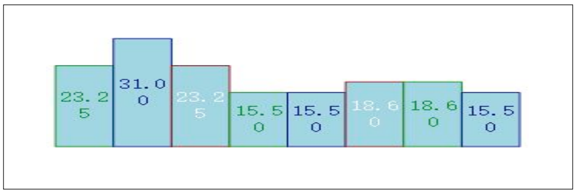
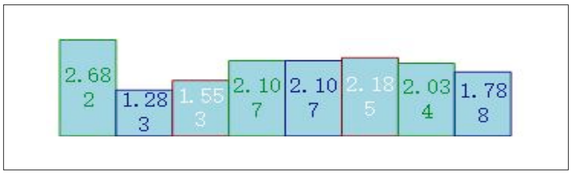


Figure II.III Speedup-ThreadNum(mul)(2-200)



B. Matrix Dimension = 100

Table V. Serial Computing Time (s)

addition	subtraction	multiplication
0.0003	0.0002	0.0060

Table VI. Parallel Computing

thread	add		sub		multiply	
	time	speed	time	speed	time	speed
2	0.0002	2.676	0.0001	1.798	0.0065	0.926
4	0.0001	2.437	0.0001	1.804	0.0067	0.893
8	0.0001	2.712	0.0001	1.946	0.0066	0.904
16	0.0002	1.317	0.0001	1.804	0.0068	0.874
32	0.0003	1.043	0.0001	1.686	0.0070	0.850
64	0.0001	1.893	0.0001	1.640	0.0098	0.610
100	0.0001	2.272	0.0002	1.592	0.0059	1.005
200	0.0001	2.000	0.0001	1.798	0.0059	1.017

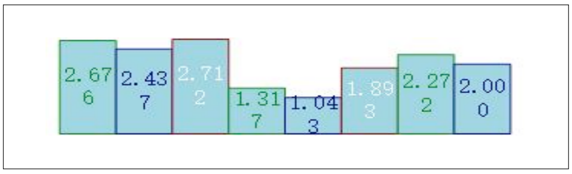


Figure III.I Speedup-ThreadNum(add)(2-200)

Figure III.II Speedup-ThreadNum(sub)(2-200)

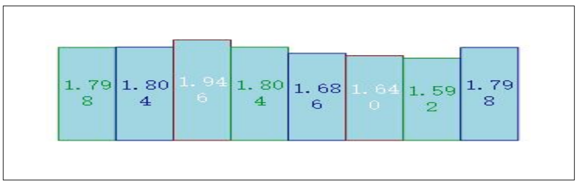
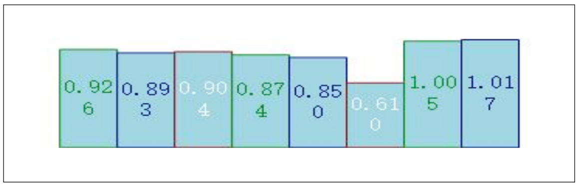


Figure III.III Speedup-ThreadNum(mul)(2-200)



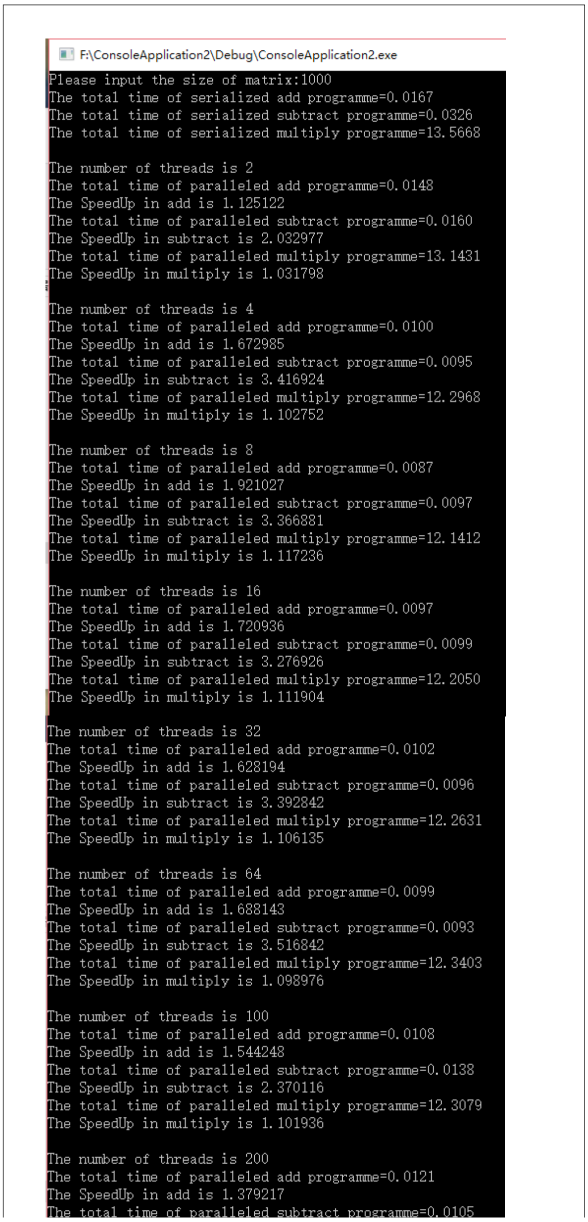
C. Matrix Dimension = 1000

Table VII. Serial Computing Time (s)

addition	subtraction	multiplication
0.0167	0.0326	13.5668

Table VIII. Parallel Computing

thread	add		sub		multiply	
	time	speed	time	speed	time	speed
2	0.0148	1.125	0.0160	2.033	13.143	1.032
4	0.0100	1.673	0.0095	3.417	12.297	1.103
8	0.0087	1.921	0.0097	3.367	12.141	1.117
16	0.0097	1.721	0.0099	3.277	12.205	1.112
32	0.0102	1.628	0.0096	3.393	12.263	1.106
64	0.0099	1.688	0.0093	3.517	12.340	1.099
100	0.0108	1.544	0.0138	2.370	12.308	1.102
200	0.0121	1.379	0.0105	3.095	12.476	1.087



```
F:\ConsoleApplication2\Debug\ConsoleApplication2.exe
Please input the size of matrix:1000
The total time of serialized add programme=0.0167
The total time of serialized subtract programme=0.0326
The total time of serialized multiply programme=13.5668

The number of threads is 2
The total time of paralleled add programme=0.0148
The SpeedUp in add is 1.125122
The total time of paralleled subtract programme=0.0160
The SpeedUp in subtract is 2.032977
The total time of paralleled multiply programme=13.1431
The SpeedUp in multiply is 1.031798

The number of threads is 4
The total time of paralleled add programme=0.0100
The SpeedUp in add is 1.672985
The total time of paralleled subtract programme=0.0095
The SpeedUp in subtract is 3.416924
The total time of paralleled multiply programme=12.2968
The SpeedUp in multiply is 1.102752

The number of threads is 8
The total time of paralleled add programme=0.0087
The SpeedUp in add is 1.921027
The total time of paralleled subtract programme=0.0097
The SpeedUp in subtract is 3.366881
The total time of paralleled multiply programme=12.1412
The SpeedUp in multiply is 1.117236

The number of threads is 16
The total time of paralleled add programme=0.0097
The SpeedUp in add is 1.720936
The total time of paralleled subtract programme=0.0099
The SpeedUp in subtract is 3.276926
The total time of paralleled multiply programme=12.2050
The SpeedUp in multiply is 1.111904

The number of threads is 32
The total time of paralleled add programme=0.0102
The SpeedUp in add is 1.628194
The total time of paralleled subtract programme=0.0096
The SpeedUp in subtract is 3.392842
The total time of paralleled multiply programme=12.2631
The SpeedUp in multiply is 1.106135

The number of threads is 64
The total time of paralleled add programme=0.0099
The SpeedUp in add is 1.688143
The total time of paralleled subtract programme=0.0093
The SpeedUp in subtract is 3.516842
The total time of paralleled multiply programme=12.3403
The SpeedUp in multiply is 1.098976

The number of threads is 100
The total time of paralleled add programme=0.0108
The SpeedUp in add is 1.544248
The total time of paralleled subtract programme=0.0138
The SpeedUp in subtract is 2.370116
The total time of paralleled multiply programme=12.3079
The SpeedUp in multiply is 1.101936

The number of threads is 200
The total time of paralleled add programme=0.0121
The SpeedUp in add is 1.379217
The total time of paralleled subtract programme=0.0105
```

Figure IV.I Speedup-ThreadNum(add)(2-200)

Figure IV.II Speedup-ThreadNum(sub)(2-200)

Figure IV.III Speedup-ThreadNum(mul)(2-200)

Figure IV.IV Result Screenshot(for 1000_dim)

D. Key Summary

- speedup = serial_time / parallel_time
- efficiency = speedup / thread_num
- if thread number increase, execution time will decrease. (if thread number double, execution time will decrease to the half)
- if thread number increase, speedup will increase. (if thread number double, speedup will increase to almost double)
- if thread number increase, the efficiency of the program will decrease (relies in the pure time values). However, the efficiency will tend to be stable with the increasing of thread number.