**Student: Robert Kigobe**

**Course: CSC 770 Parallel Programming - Project 1**

**Professor: Yumei Huo**

**Date due: 04-04-2021**

1 What to hand in

You are asked to hand in the following:

(1) Unless specially required, all the files (programs, output files and script file for submitting job) should be submitted. Please zip all your files and name your zip file starting with your last name. Submit zip file on blackboard.

(2) For each function explain what it does what the various arguments denote.

In case your programs do not work as specified, you may receive partial credit depending on the documentation supplied (bug list etc).

2 What to implement

Implementations are required for the function described in Part A and B. Part A is worth 300 points. Part B is optional and worth 200 points. Students who don't do it will not be penalized; students who do it can only improve their course grade. A student is not allowed to do Part B only in place of A.

3 Which machine to use

CSI’s supercomputer Andy (Neptune as a interface).

**Part A: There are three tasks in this part.**

**Task 1(50 points): Run the example “Hello world”**

Please copy all the steps you have run on zeus to a text file and submit it to me. Make sure to include all the commands you have typed and the corresponding results.

Name your text file starting with your last name.

**Please follow the following steps:**

In this lab, we will have the first MPI program and will submit job through script file to PBS Batch Queueing System (**You will only use this method to run your MPI program**). Please follow the following steps:

1. log in HPC computer using your account (**Note that if you try to access hpc computer from offcampus, please connect to chizen.csi.cuny.edu, and then ssh to username@mhn**).
2. Type the command of **module load openmpi** to load the mpi.

Similar output will be as the following:

[yumei.huo@login-0-1 ~]$ module load openmpi

Java module loaded - the system's JAVA is replaced by JDK 1.8.0\_211

1. Type the command of **module list** to check the packages were loaded successfully.

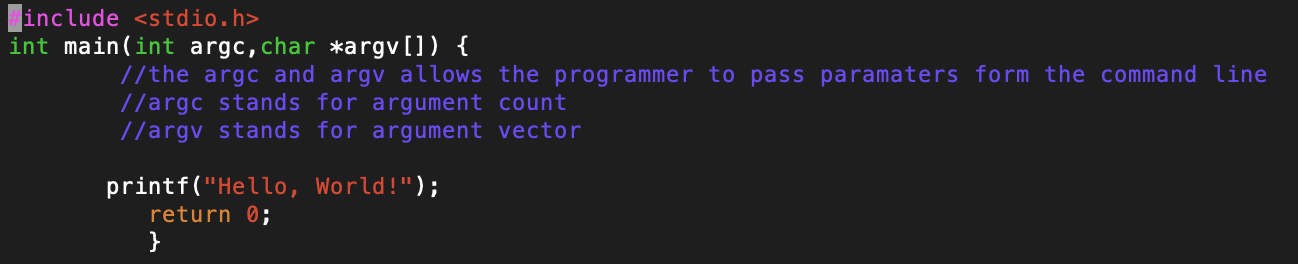
Similar output will be as the following:

[yumei.huo@login-0-1 ~]$ module list

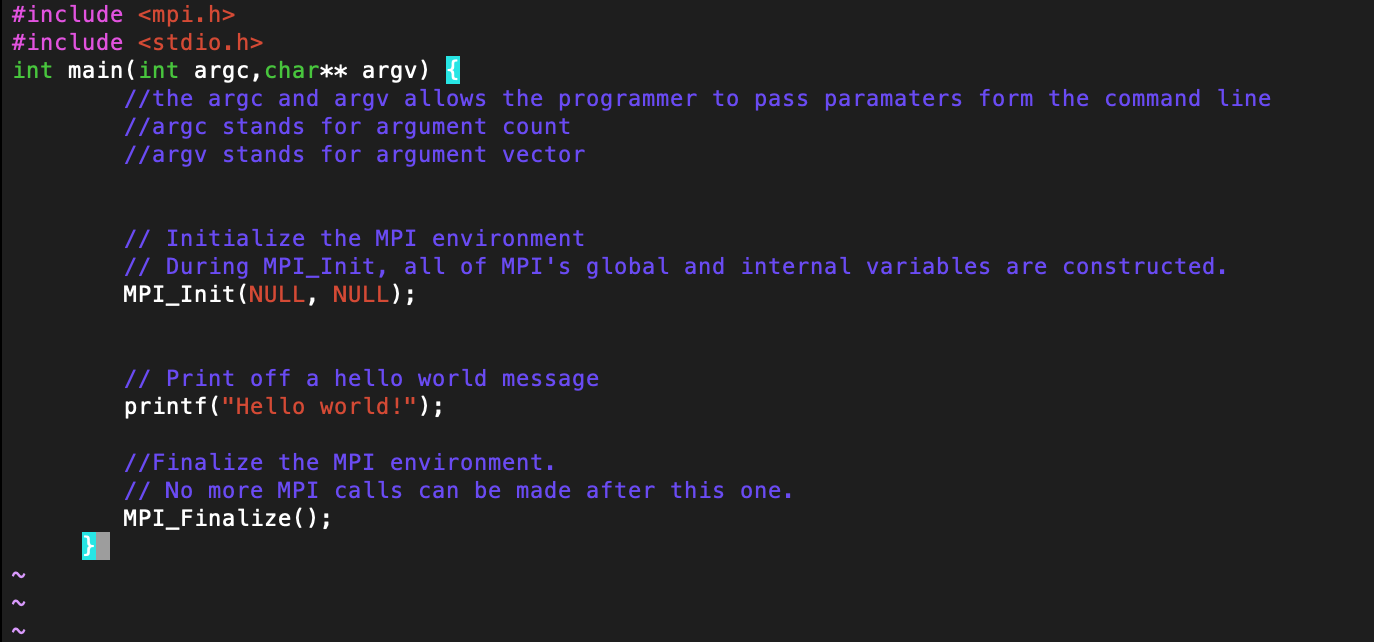
Currently Loaded Modules:

1) cuda/10.1 2) java/1.8.211 3) binutils/2.28 4) gcc/7.3.0 5) openmpi/4.0.1\_gnu

1. Create a new directory with the name of *lab11* and then get into the directory. What will be the command?
2. Create a file with the name of *hello.c*. Use vi editor or other editors you are familiar with(emacs, vim…) to type a program which will print out “Hello World!”. Make sure that the program has two parameters *argc* and *argv* as we discussed in the class. What will be the program?



1. Now revise program *hello.c* such that *MPI\_Init* and *MPI\_Finalize* are included. (In order to use these two functions, what header file do you need? ) What will be the program now?



1. Compile your program, using the following command:
2. create a job script named *job1*, type command of *vi* *job1*, then type the following:

#!/bin/bash

#SBATCH -J PartProdHello

#SBATCH --ntasks 40

#SBATCH --nodes 5

#SBATCH --tasks-per-node 8

#SBATCH --mem-per-cpu 2GB

#SBATCH --partition partition

#SBATCH --account=students

cd $SLURM\_SUBMIT\_DIR

module load openmpi

mpirun --mca opal\_warn\_on\_missing\_libcuda 0 -np 40 ./hello

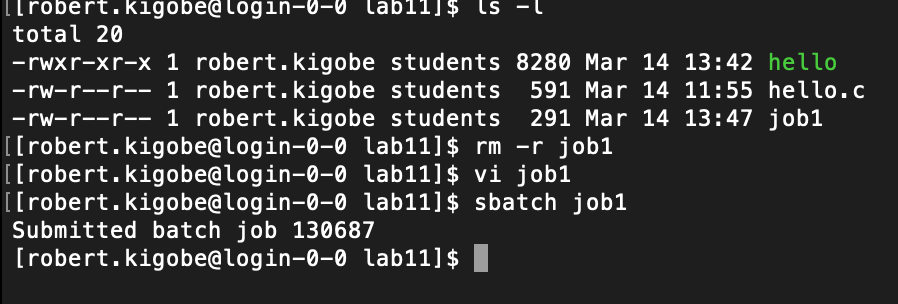
1. Then run command:

*sbatch job1*

Similar output will be as follows:

*[yumei.huo@login-0-0 lab1]$ sbatch job1*

*Submitted batch job 125597*

**

1. run the following command to check the status of your program:

*qstat*

Similar output will be as follows:

*[yumei.huo@login-0-0 lab1]$ qstat*

*Job id Name Username Time Use S Queue*

*------------------- ---------------- --------------- -------- - ---------------*

*123826 dispersion jeffrey.xu 02:13:07 R partsequential*

*123837 P2500bar ali.eltareb 02:13:07 R partproduction*

*123838 P2000bar ali.eltareb 03:15:13 R partproduction*

*123841 P500bar ali.eltareb 03:15:13 R partproduction*

*123842 P-500bar ali.eltareb 03:15:13 R partproduction*

*123843 P-1000bar ali.eltareb 02:13:07 R partproduction*

*123847 D2O\_PHASE ali.eltareb 02:13:07 R partproduction*

*123848 D2O\_PHASE ali.eltareb 02:13:07 R partproduction*

*123887 sc\_e2j2 aleix.boucomas 09:23:26 R partfat1*

*125597 jobhello yumei.huo 00:00:00 C partedu*

1. When your program is finished, check in the current directory whether you have file named like *slurm-xxxxx.out*. Then open the file *slurm-xxxxx.out* and check the content. You should have the following content:

compute-0-96

Java module loaded - the system's JAVA is replaced by JDK 1.8.0\_211

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

Hello World

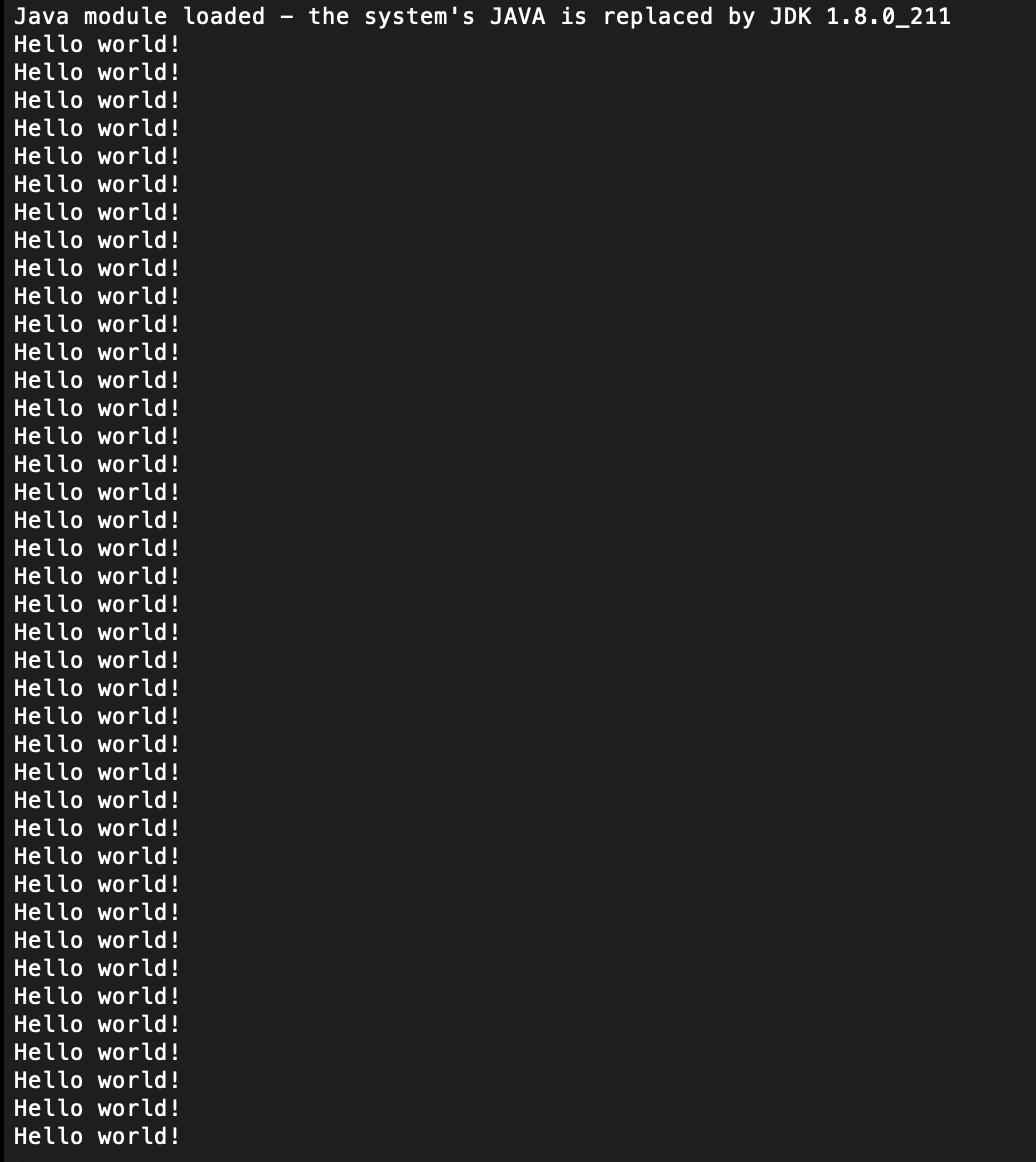
Hello World

Hello World

Hello World

Hello World

Hello World

"slurm-125594.out" 42L, 561C

1. Now copy your program *hello.c* to a new file named *hello\_node.c*. Revise program *hello\_node.c* such that two variables *nprocs* and *mypid* are declared and will hold the value of total number of processes and identifier of calling process respectively. Then print out “Hello world from process xxx of total xxx!” What functions should you use to find the value for *nprocs* and *mypid*? What will be the program now?
2. Compile your program, using the following command:

*mpicc hello\_node.c -o hello\_node*

1. Copy your job script *job1* to a new script file named *job2*, type command of *vi job2*, then change the script file so that you will request 8 tasks (4 nodes for example, then 2 cores of each node) and change the executable file to *hello\_node*.
2. Then run command:

*sbatch job2*

1. Run the following command to check the status of your program:

*qstat*

1. When your program is finished, check in the current directory whether you have the related output file named like *slurm-xxxxx.out*. Then open the file *slurm-xxxxx.out* and check the content. You should have the following content:

Java module loaded - the system's JAVA is replaced by JDK 1.8.0\_211

Hello World from process 0 of total 8!

Hello World from process 2 of total 8!

Hello World from process 1 of total 8!

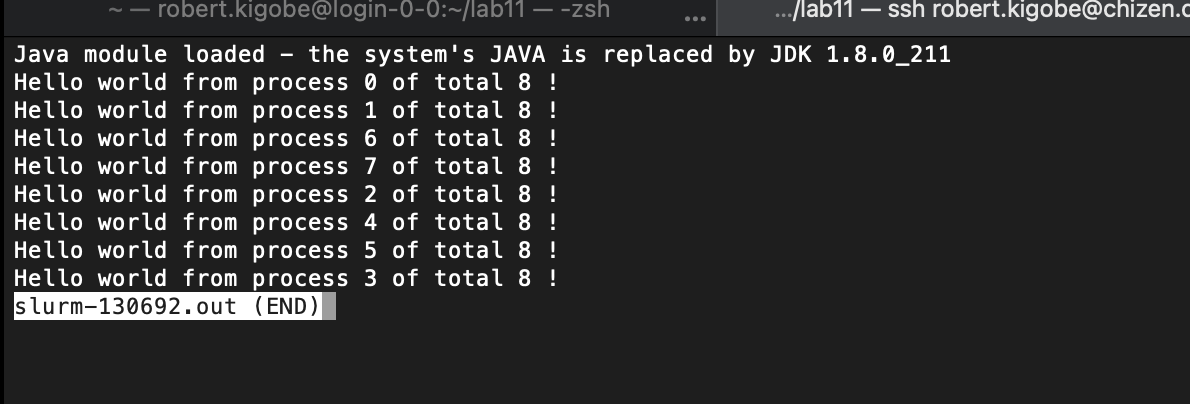
Hello World from process 3 of total 8!

Hello World from process 4 of total 8!

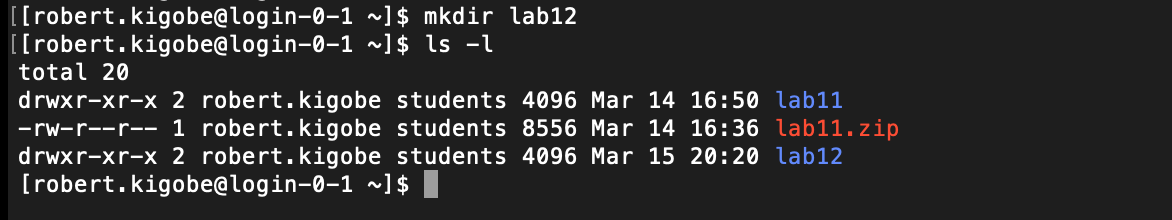
Hello World from process 5 of total 8!

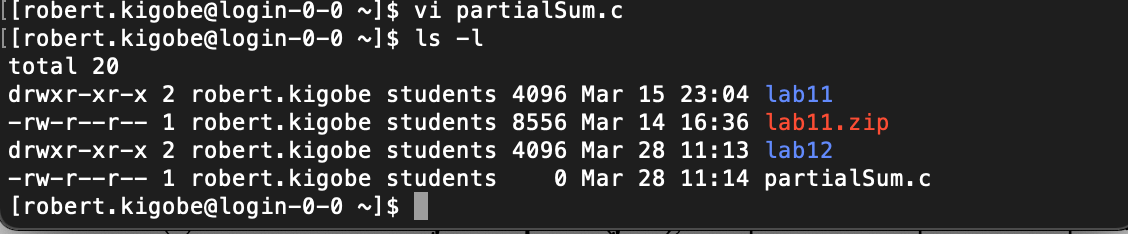
Hello World from process 6 of total 8!

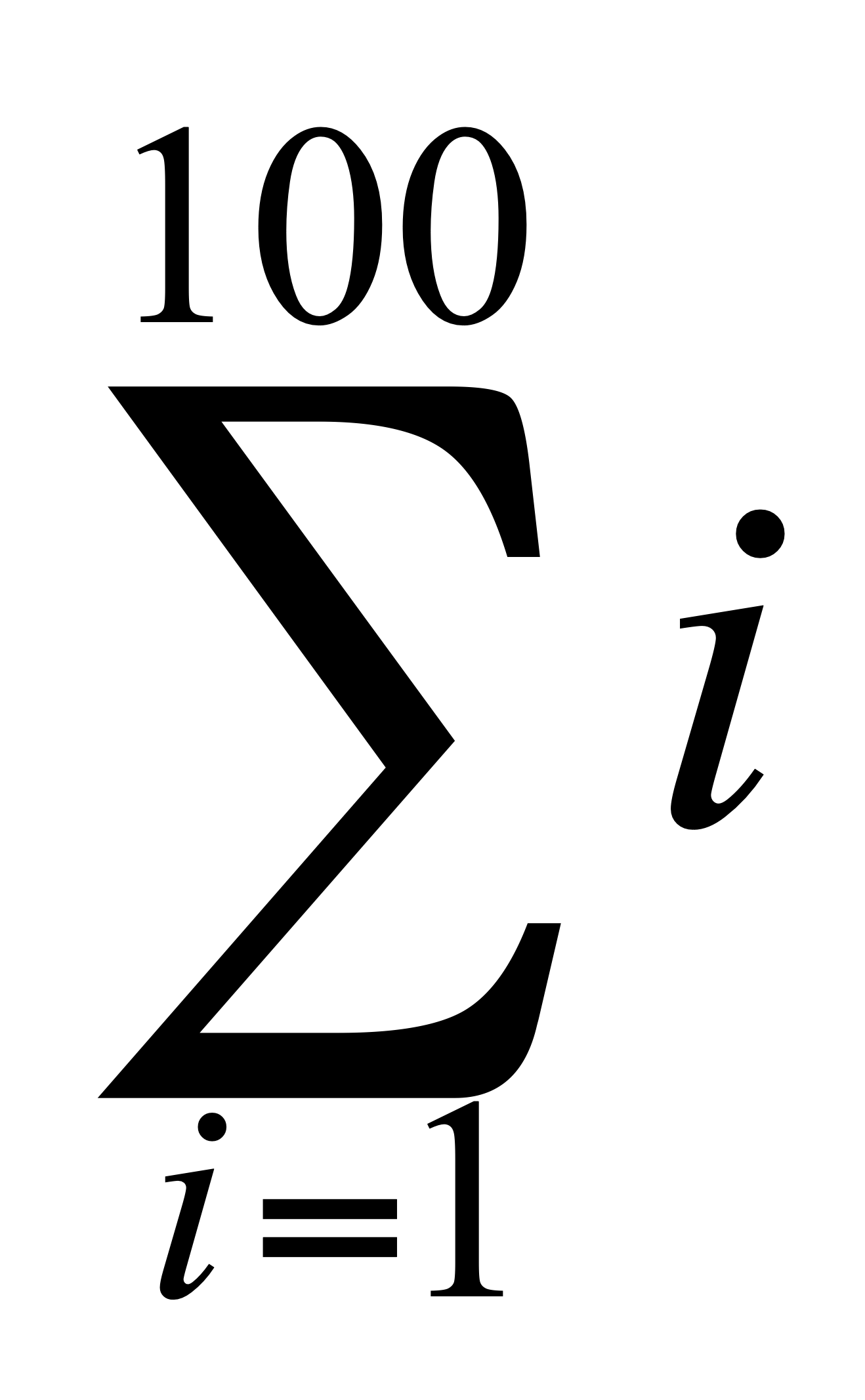
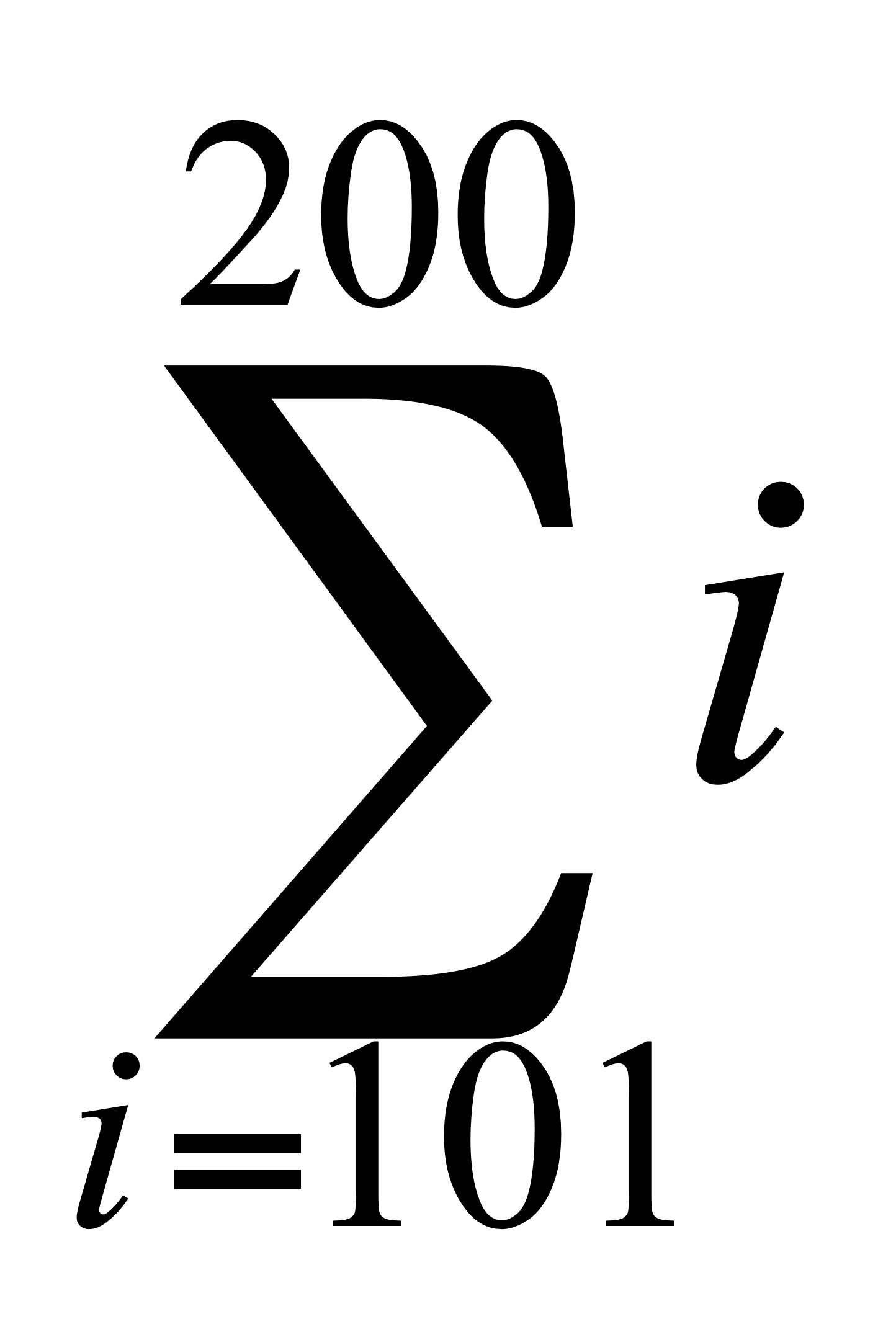
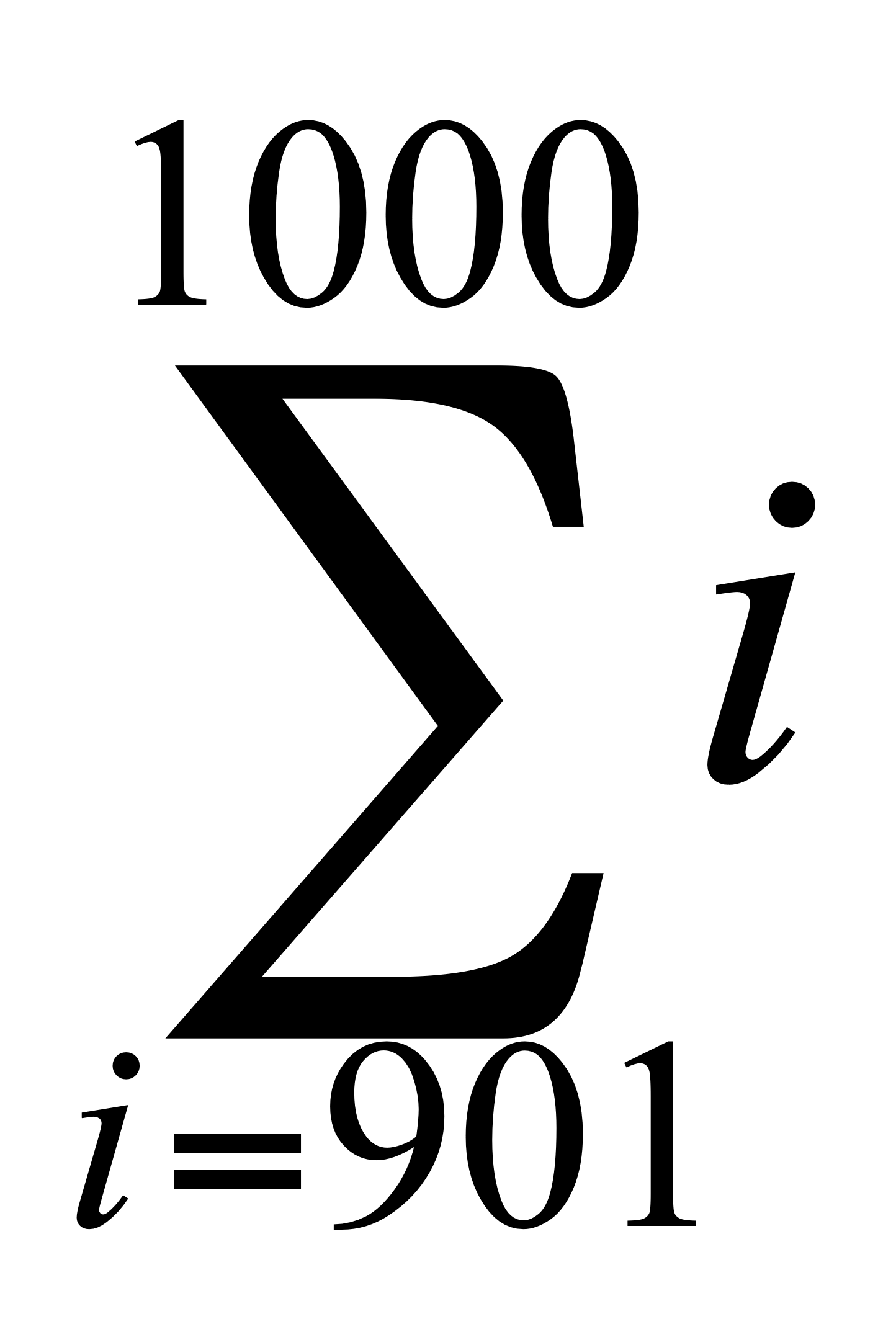
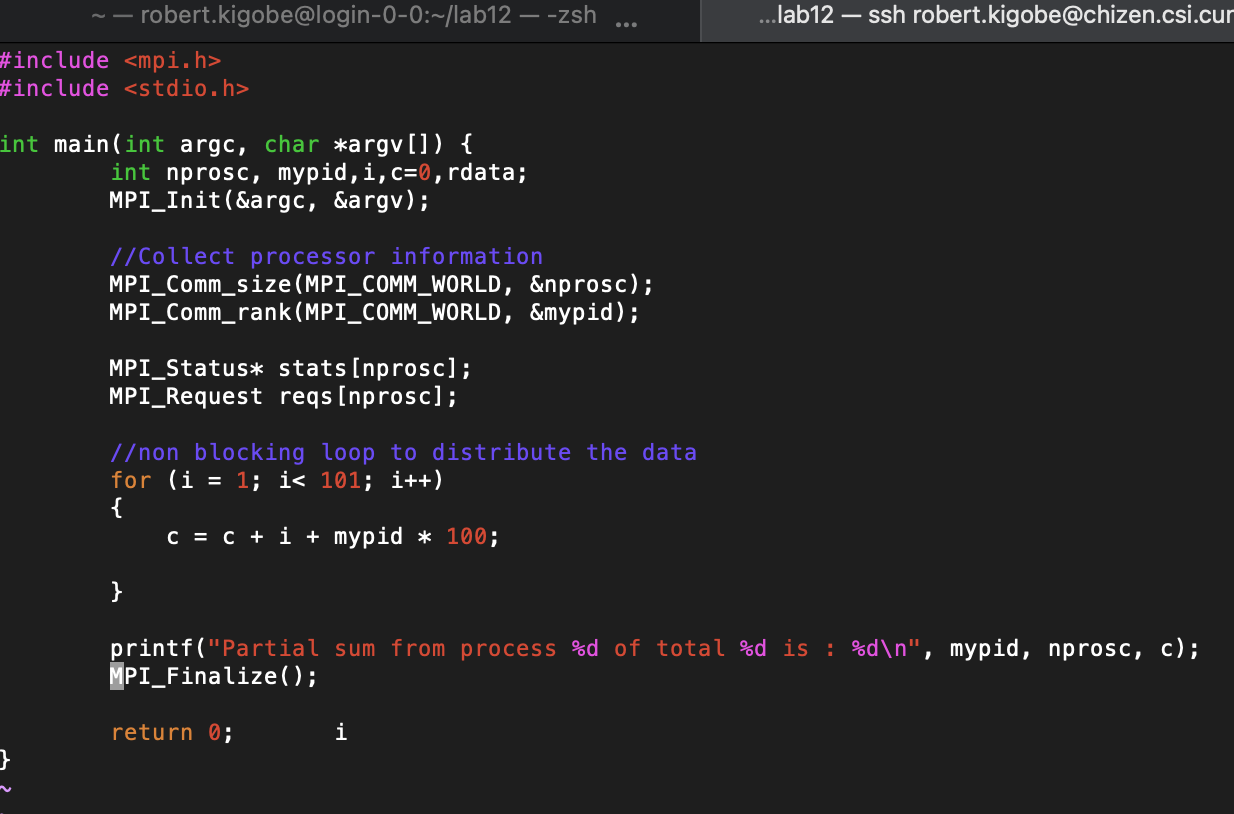
Hello World from process 7 of total 8!



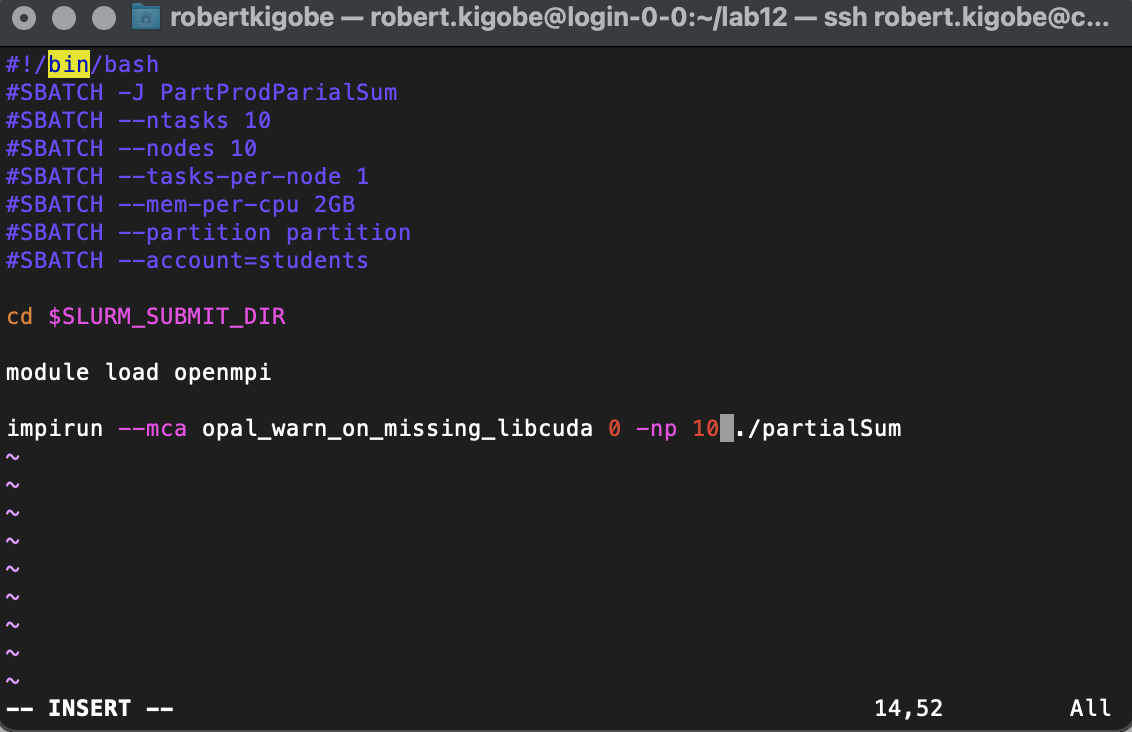
**Task II: find out partial sums (100points)**

1. Create a new directory with the name of *lab12* and then get into this directory.
2. Write a program named partialSum.c such that each process will compute and print out a partial sum of N numbers. Set N=1000 in your program.



1. **Define 10 tasks in job script file (job3)**, then process 0 will print out the partial sum of 1+2+…+100 (that is,), process 1 will print out the partial sum of 101+102+…+200(that is, ), …, process 9 will print out the partial sum of 901+902+…+1000(that is, ),.

(Note: In this program, you can declare a constant N=1000 or use preprocessing statement #define N 1000.)



1. When your program is finished, submit your script file (job3) check in the current directory whether you have the related output file produced. Then open this file and check the content. You should have the following content:

Java module loaded - the system's JAVA is replaced by JDK 1.8.0\_211

Partial sum from process 0 of total 10 is : 5050.

Partial sum from process 6 of total 10 is : 65050.

Partial sum from process 3 of total 10 is : 35050.

Partial sum from process 7 of total 10 is : 75050.

Partial sum from process 2 of total 10 is : 25050.

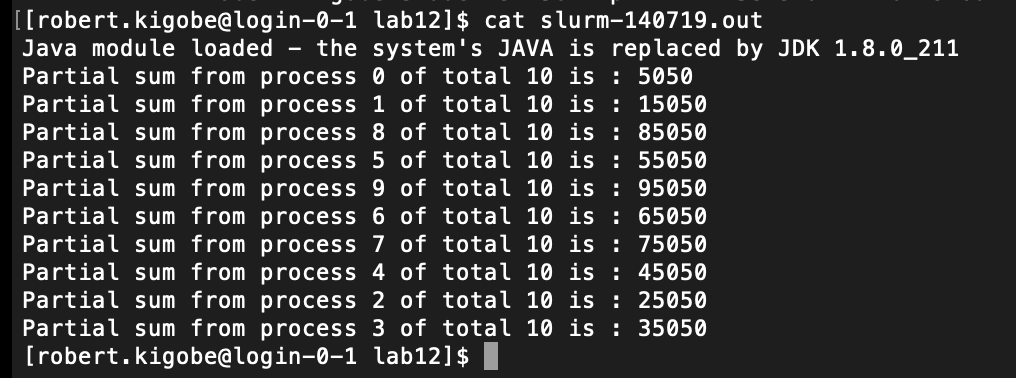
Partial sum from process 9 of total 10 is : 95050.

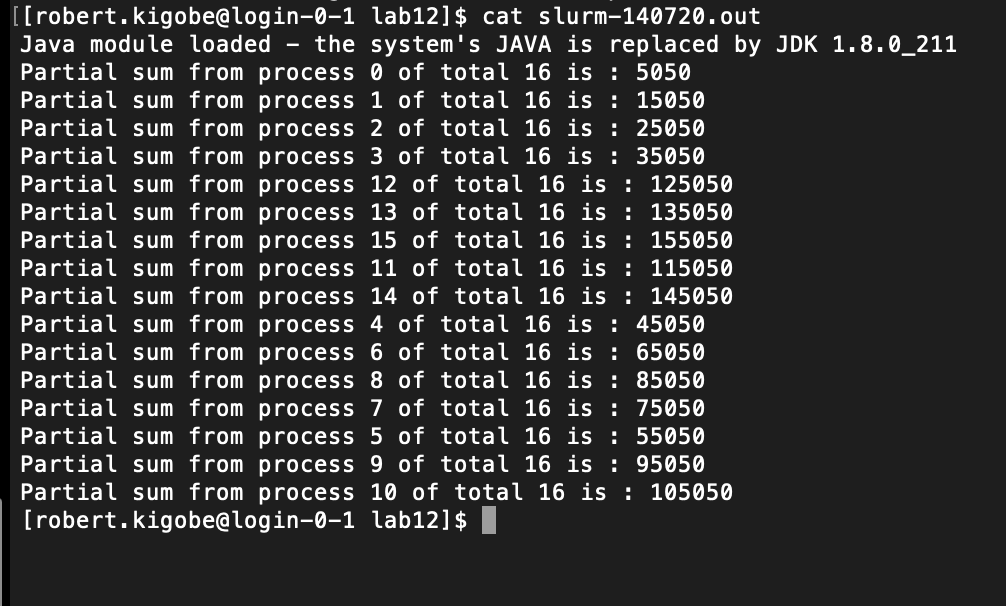
Partial sum from process 5 of total 10 is : 55050.

Partial sum from process 4 of total 10 is : 45050.

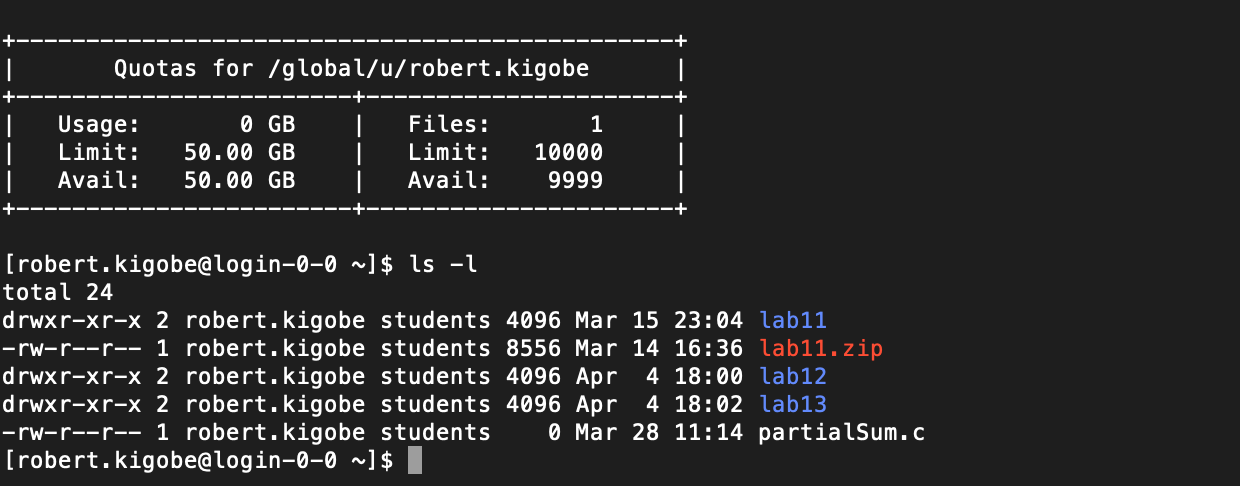
Partial sum from process 8 of total 10 is : 85050.

Partial sum from process 1 of total 10 is : 15050.

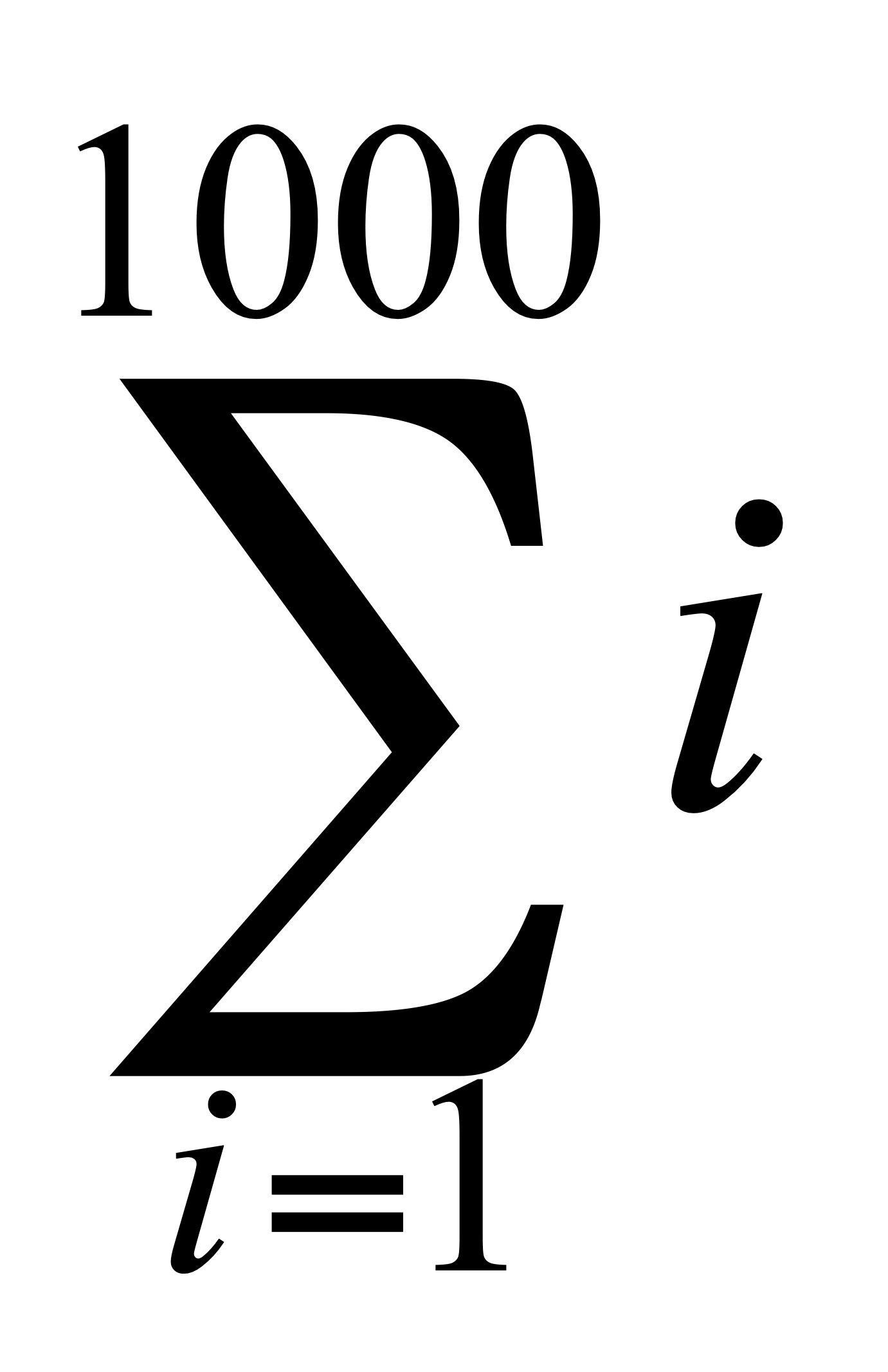


1. **Revise your script file to apply for 16 tasks, and then run your program again. Name this job job4.**

**Task III (150 points): Get the sum of n intergers (You are going to use 8 nodes to do this task.) All nodes except processing node 0 send their partial sums to node 0 and node 0 add up all the partial sums.**

1. Create a new directory with the name of *lab13* and then get into this directory. What will be the command?
2. Write a program named Sum.c such that
   1. Each process will compute and print out a partial sum of N numbers.

(Note: This is same as task II. In this program, you can declare a constant N=1000 or use preprocessing statement #define N 1000.)

* 1. all processes except process 0 send their partial sum to process 0 and print this message
  2. after process 0 receive the partial sums from other processes, process 0 will print out the confirmation that it received the value, add up all the partial sums including its own partial sum, and print out the total sum, that is, the value of 1+2+3+… +1000(that is, ).

1. Compile your program, using the following command:

*mpicc Sum.c –o Sum*

1. create a job script named ***job5*** and make sure 10 nodes will be applied for your program. You can copy the job script file given in lab0 and then modify it.
2. Then run command:

*sbatch job5*

1. run the following command to check the status of your program:

*qstat*

1. When your program is finished, check in the current directory whether you have the related output file produced. Then open this file and check the content. You should have the following content:

Java module loaded - the system's JAVA is replaced by JDK 1.8.0\_211

Partial sum from process 0 of total 10 is : 5050.

Receive 15050 from process 1

Receive 25050 from process 2

Receive 35050 from process 3

Receive 45050 from process 4

Receive 55050 from process 5

Receive 65050 from process 6

Partial sum from process 2 of total 10 is : 25050.

Send 25050 to process 0 by process 2

Partial sum from process 1 of total 10 is : 15050.

Partial sum from process 6 of total 10 is : 65050.

Partial sum from process 4 of total 10 is : 45050.

Send 45050 to process 0 by process 4

Partial sum from process 5 of total 10 is : 55050.

Send 55050 to process 0 by process 5

Partial sum from process 3 of total 10 is : 35050.

Send 15050 to process 0 by process 1

Send 65050 to process 0 by process 6

Send 35050 to process 0 by process 3

Receive 75050 from process 7

Partial sum from process 7 of total 10 is : 75050.

Send 75050 to process 0 by process 7

Partial sum from process 9 of total 10 is : 95050.

Send 95050 to process 0 by process 9

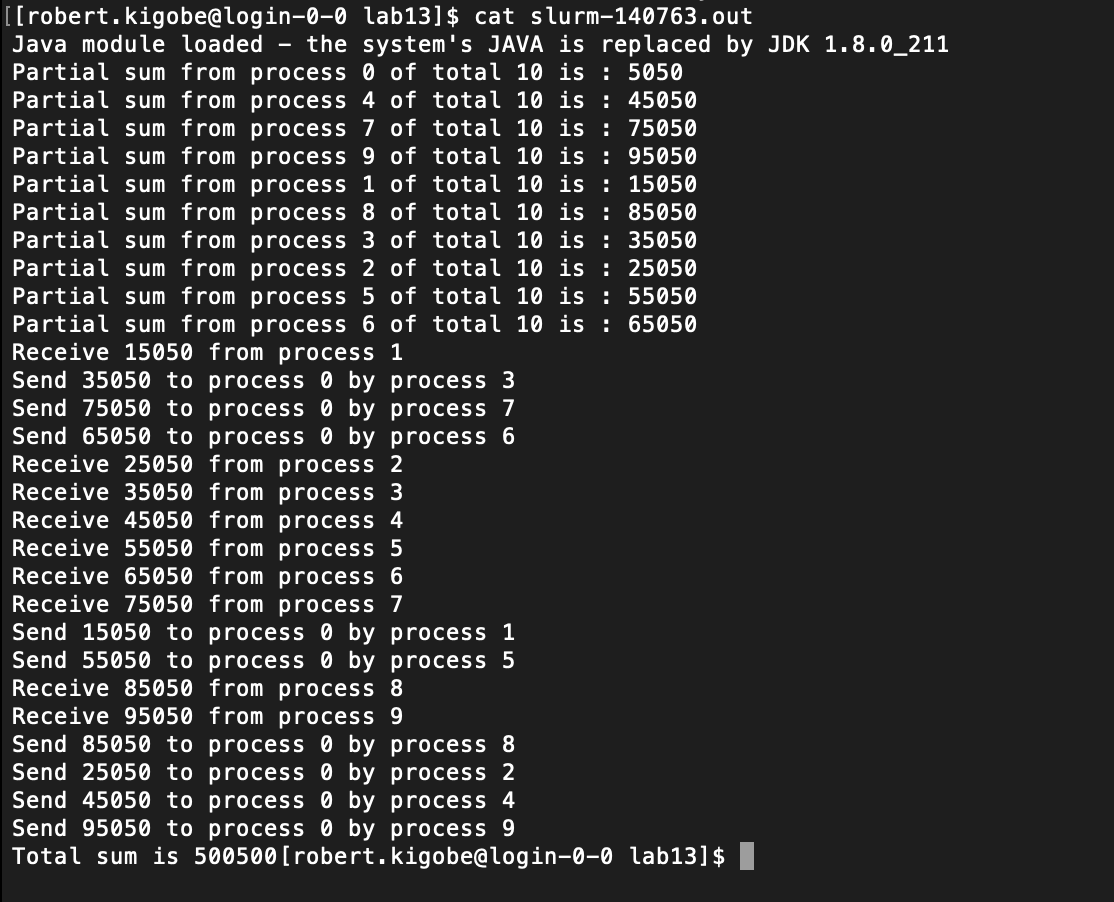
Partial sum from process 8 of total 10 is : 85050.

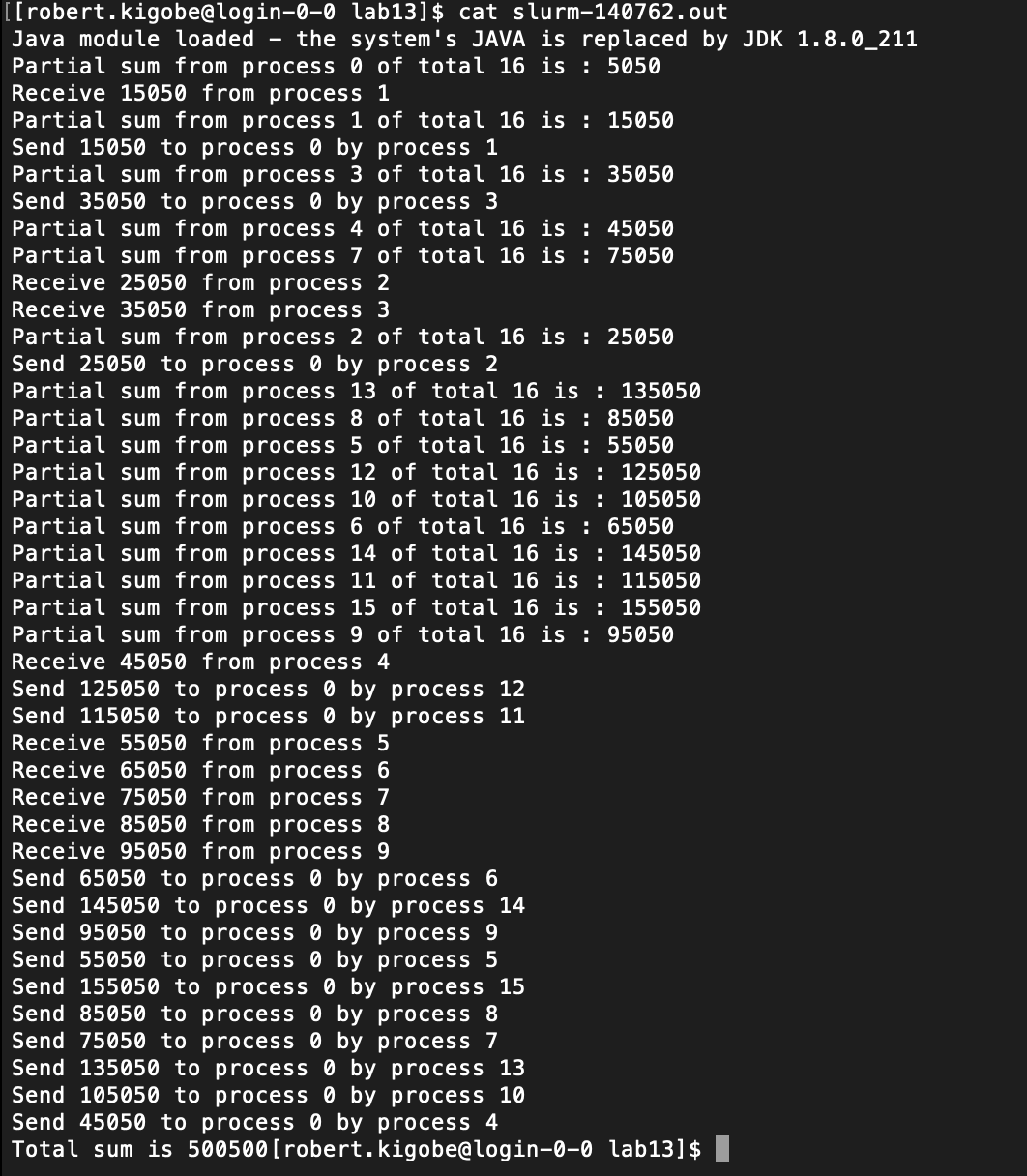
Receive 85050 from process 8

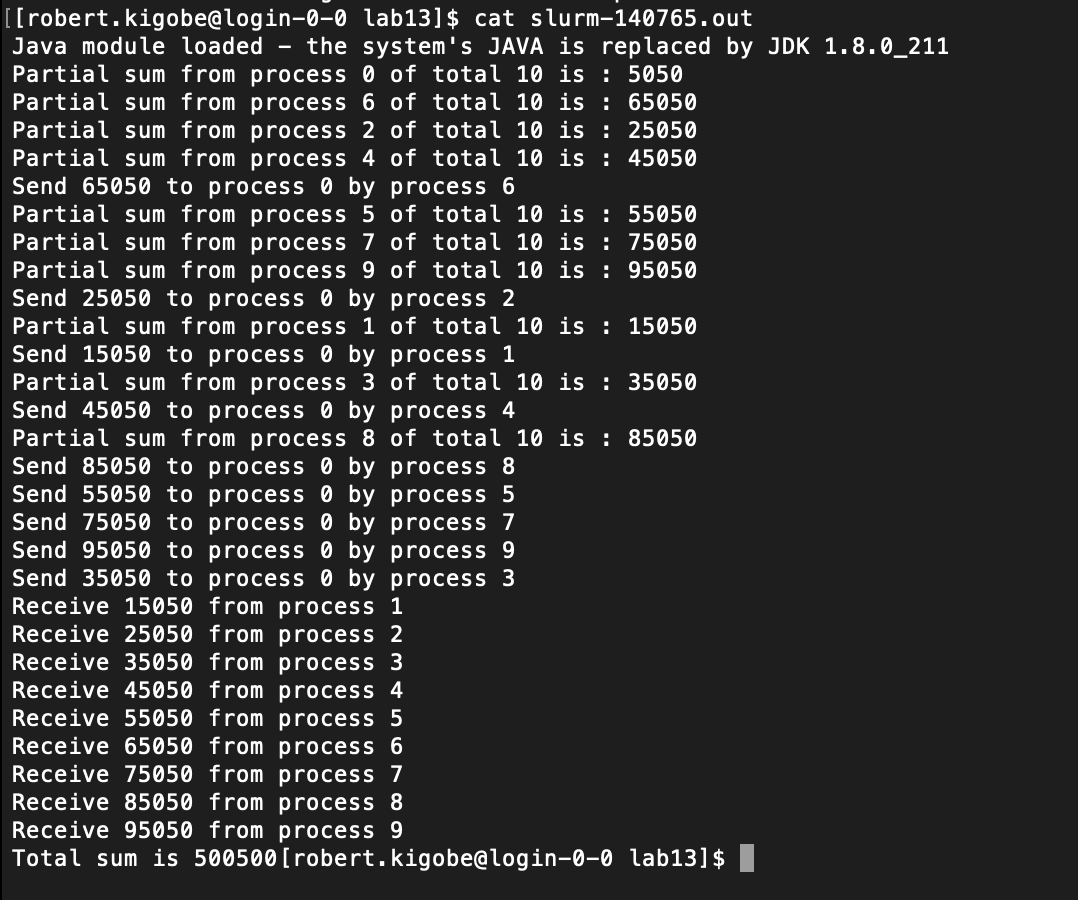
Receive 95050 from process 9

Total sum is 500500

Send 85050 to process 0 by process 8



1. **Revise your program and script file appropriately so that your program is working for 16 nodes, and then run your program. Name this job *job6*.**
2. **Revise your program such that blocking send and blocking receive are replaced with nonblocking send and nonblocking receive. Change your script file to run this program. Name this job *job7*.**



**Part B (optional): Broadcasting by k-ary replication**

You are required to implement the broadcasting algorithm that uses k-ary replication(In the class, we discussed 2-ary(binary) tree broadcasting). You are asked to write a C function with the following syntax and behavior.

void mpi\_mbroadcast(int fromp, int multi, int degree, char \*from, char \*to, int nbytes);

* fromp is the source processor that holds the message to be broadcast. Note that in the algorithm presented in the class it is assumed fromp=0. You are now required to generalize that algorithm and make it work for any value of fromp.
* multi is the length of the data structure that is being copied. (the structure to be broadcast is an array of multi entries, the size of each entry being nbytes long).
* degree is the degree of replication. In other words, each processor, in each superstep, would send the received message to at most degree-1 other processors.
* from is the base address of the first byte that will be broadcast.
* to is the base address of the first position that will receive the broadcast message.
* nbytes is the size in bytes of the elementary data type (array element) that is being copied.