**DSCC 201/401 Homework Assignment #4**

**Due: October 13, 2021 at 9 a.m. EDT**

**Answers to these questions should be submitted via Blackboard. Questions 1-8 should be answered by all students (DSCC 201 and 401) and Questions 9 and 10 should be answered by students registered in DSCC 401. Make sure to answer all parts of the questions to receive credit. Please upload a file containing your answers and explanations to Blackboard (Homework #4: Software Parallelization Models and Techniques) as a Word document (docx) or PDF.**

**1. Create a new directory in your home directory called parallel. Copy the file**

**/public/bmort/parallel/matrix\_mult.c to this directory. Examine the source code of**

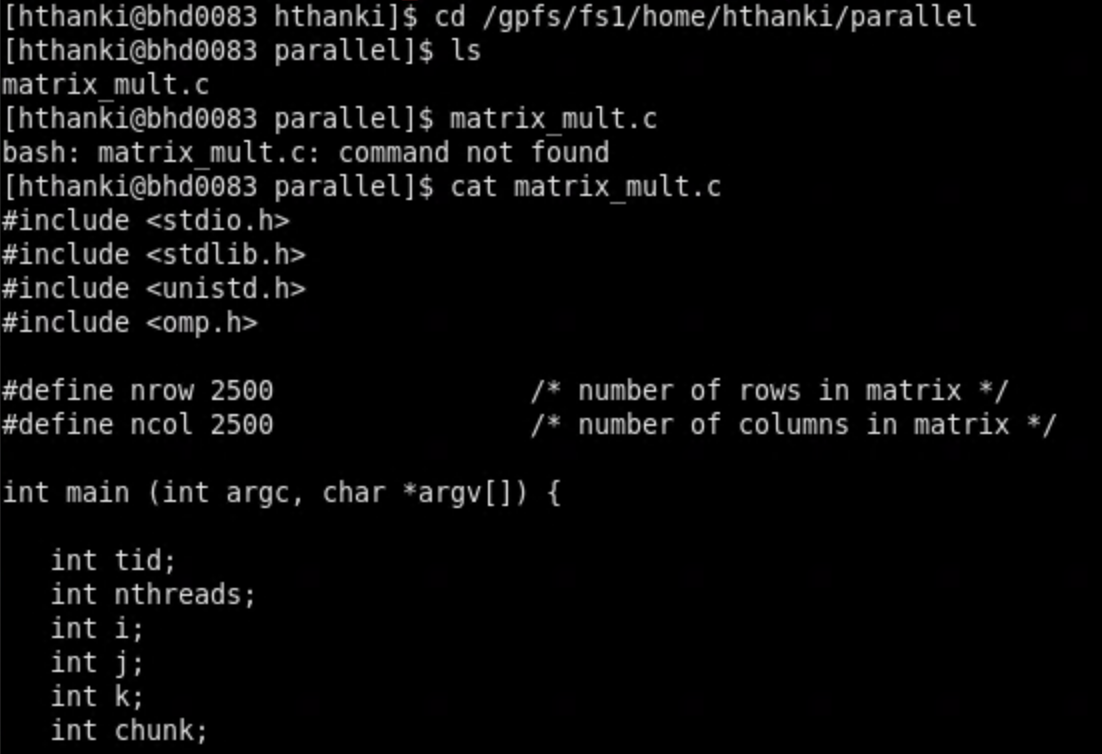
**matrix\_mult.c. Which parallel computing library does this program use?**

Commands**:**

mkdir parallel

cp /public/bmort/parallel/matrix\_mult.c /gpfs/fs1/home/hthanki/parallel

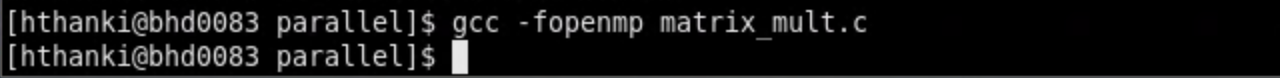
The OpenMP (OMP) library is used for parallel computing.



**2. Compile the matrix\_mult.c source code using the gcc compiler. Make sure to include the option for the correct parallel library when compiling your code. What command did**

**you use to compile the source code?**

Command: gcc -fopenmp matrix\_mult.c



**3. Create a Slurm script to run a batch job for the binary produced from the compilation**

**step in question 2. You should request exactly 4 cores and 2 GB of RAM to run your**

**compiled binary. Set the partition to debug. Print out your complete Slurm script.**

#!/bin/bash

#SBATCH --job-name=parallel\_job

#SBATCH --nodes=1

#SBATCH --ntasks=1

#SBATCH --cpus-per-task=4

#SBATCH --mem=2gb

#SBATCH --partition=debug

echo "Running matrix multiplication program on $SLURM\_CPUS\_ON\_NODE CPU cores"

./a.out INPUT

srun a.out

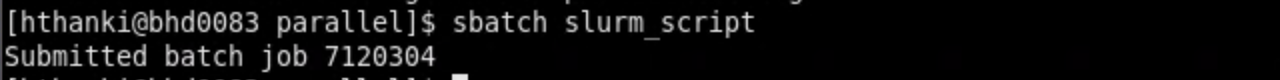
****

**4. Submit your batch job to run in the debug partition on the BlueHive cluster. What**

**command did you use to submit the job? How do you know the job has finished? Print**

**the output from the job.**

Command: sbatch slurm\_script

****

**5. Notice that the wall time for the calculation is printed in the output. You can benchmark**

**this code by varying the number of cores for the task. Start with 1 core and increase the**

**values by powers of two up to the maximum value for a BlueHive node in the debug**

**partition (i.e. 1, 2, 4, 8, 16). Include the output from each run of your benchmark and**

**take note in the output of the hostname of the node where your benchmark runs. A)**

**Would it be possible to get better performance of the binary compiled from the**

**matrix\_mult.c source code if you ran it on more than one node? Why or why not? B)**

**Would the performance of matrix\_mult program improve if you requested a GPU with**

**the Slurm option --gres=gpu:1 and submitted the program to the gpu-debug partition?**

**Why or why not?**

1 core - 149.777583

2 cores - 79.855199

4 cores - 40.117706

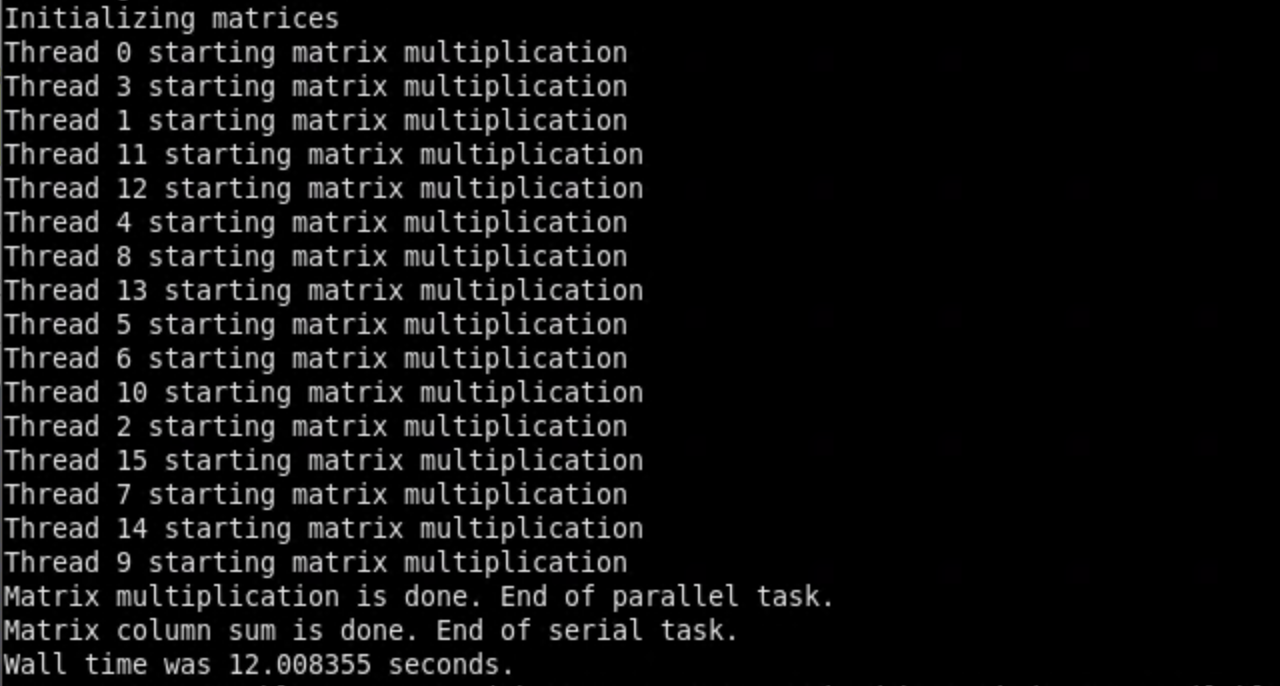
8 cores - 20.345086

16 cores - 11.869543

16 cores + 1 GPU - 12.008355

16 cores + 1 GPU + 3 nodes - 12.021830

Performance of matrix\_mult after a GPU is requested

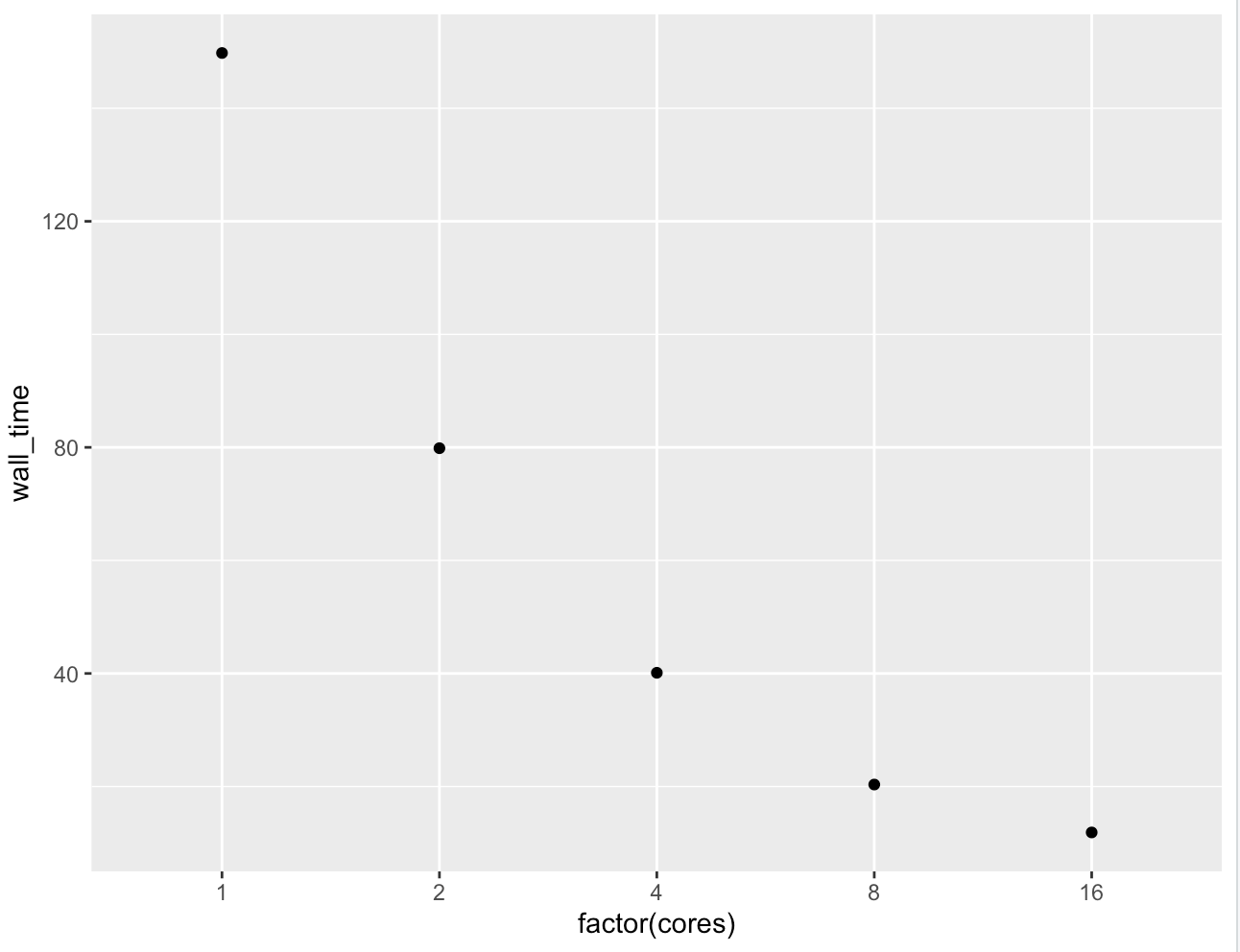
****

The performance does increase when multiple cores are allocated. However, allocating a GPU and increasing the number of nodes does not impact the performance significantly. The performance is not an improvement on multiple nodes since the processes might have been slowed down due to excessive overhead. Moreover, the GPU does not help with the performance either since the process is not large enough to require both the CPUs and the GPU.

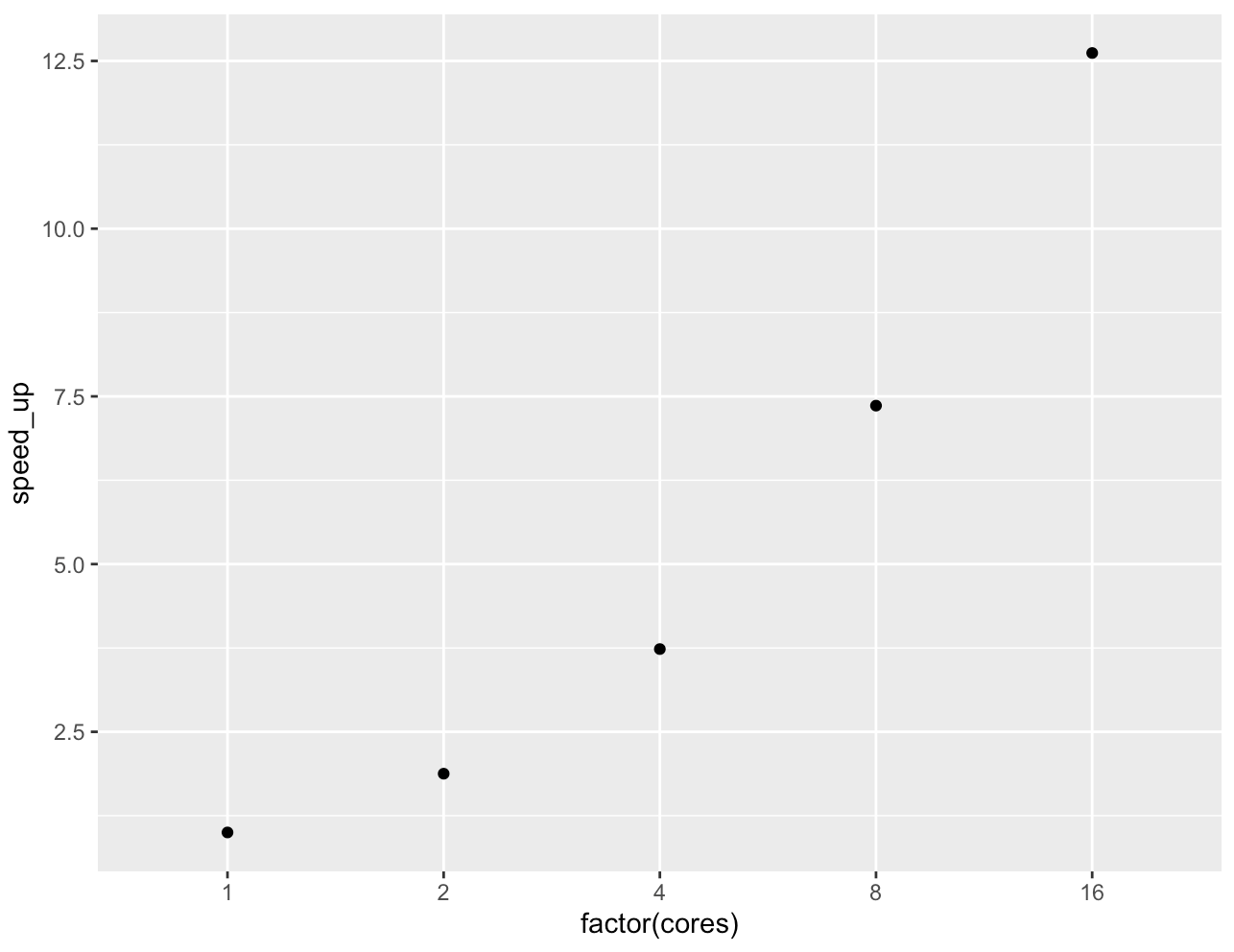
**6. Using a plotting program (e.g. Excel, Python, GNUplot, etc.), construct a graph showing the time it takes for the job to complete (wall time in seconds) vs. the number of cores. Calculate the speedup for each of the data points and create an additional plot that shows speedup (S) vs. cores (N). Embed your graph in the document (e.g. Word**

**document or PDF) that you will submit for this homework assignment.**

Plot of performance with 1, 2, 4, 8 and 16 cores:

****

Plot of speed-up v/s cores:

****

**7. What fraction of the work (p) in the matrix\_mult program is parallel? Show and explain**

**how you derived your answer.**

Amdahl's Law states that in parallelization, if P is the proportion of a system or program that can be made parallel, and 1-P is the proportion that remains serial, then the maximum speedup S(N) that can be achieved using N processors is -

S=1/((1-P)+(P/N))

Hence,

P <- (N\*(1-S))/(S\*(1-N))

Substituting values for the number of cores and speed up in the formula above, we get:

Parallelization for 2 cores = 0.9336829

> N <- 2

> S <- 149.777583/79.855199

> P <- (N\*(1-S))/(S\*(1-N))

> P

[1] 0.9336829

Parallelization for 4 cores = 0.976202

> N <- 4

> S <- 149.777583/40.117706

> P <- (N\*(1-S))/(S\*(1-N))

> P

[1] 0.976202

Parallelization for 8 cores = 0.9876168

> N <- 8

> S <- 149.777583/20.345086

> P <- (N\*(1-S))/(S\*(1-N))

> P

[1] 0.9876168

Parallelization for 16 cores =

> N <- 16

> S <- 149.777583/11.869543

> P <- (N\*(1-S))/(S\*(1-N))

> P

[1] 0.9821357

**8. What is linear scaling? Does the matrix\_mult program exhibit linear scaling? Why or why not?**

Linear scaling would indicate that number of cores and performance have a linear relationship and y = ax+b can be used to define the relationship. However, this is not the case. Hence, matrix\_mult does not exhibit linear scaling.

**9. (DSCC 401 Only) You are now going to run the matrix\_mult.c program directly on one of the hosts in a Default FastX session on BlueHive. After logging into a Default desktop**

**environment on BlueHive, open the terminal application. Using the hostname**

**command, what is the host name of the node you are on? Now set an appropriate**

**environment variable to control the number of threads for your matrix\_mult.c program.**

**Set the appropriate environment variable to use 1 thread. Run the matrix multiplication**

**program directly on the command line in the terminal session. Take note of the wall**

**time to execute. Change the number of threads to 2 using the environment variable and**

**run the program again, noting the wall time. Repeat for 4 and 8 threads. Plot the timing**

**data as you did in question 6 (wall time vs. number of threads and speedup vs. number**

**of threads). Are there any significant differences between the graphs obtained in**

**question 6 and this question? Why or why not?**

Hostname - bhd0083

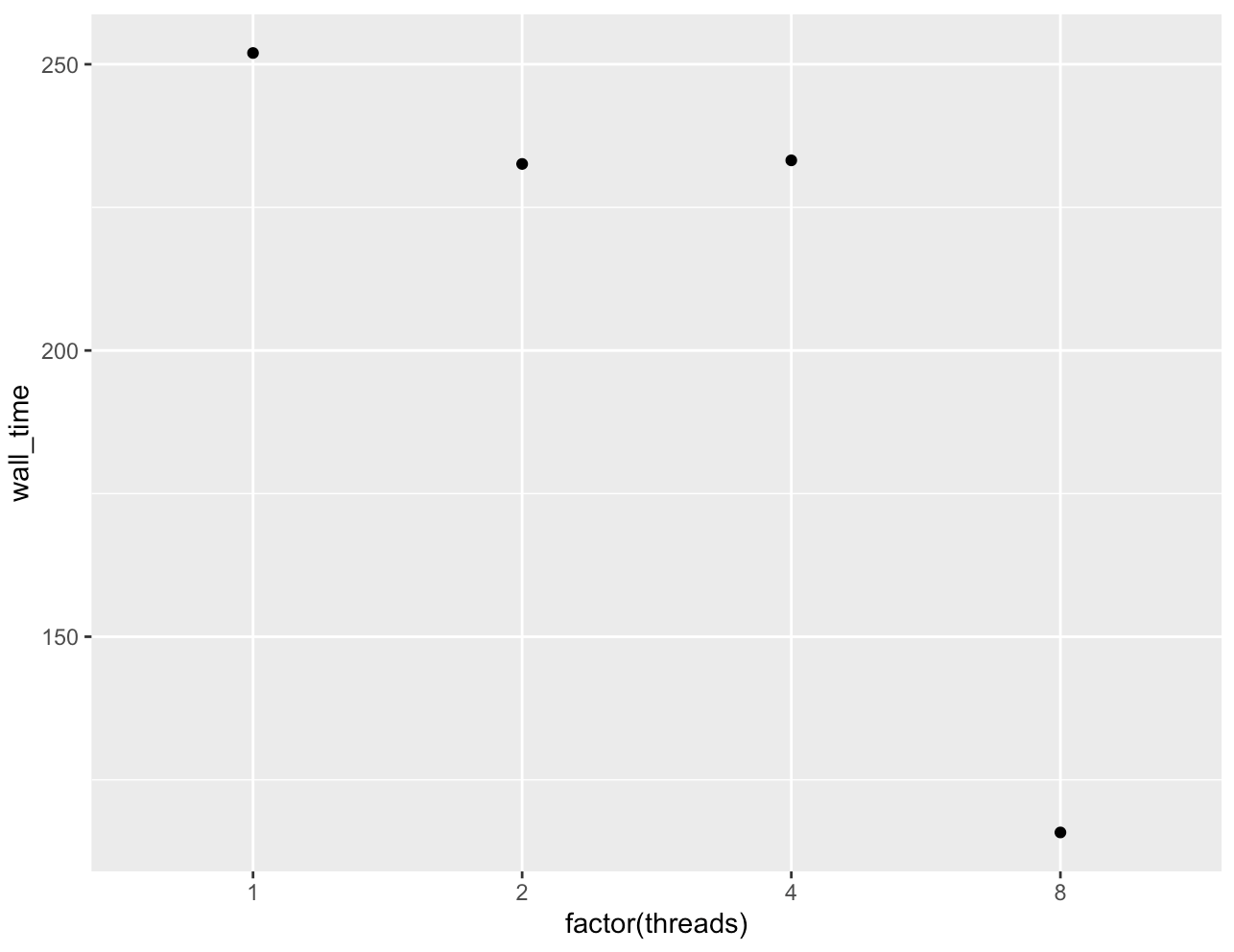
Walltime to execute matrix\_mult.c on 1 cpu and 1 thread per cpu = 251.969715

Walltime to execute matrix\_mult.c on 1 cpu and 2 threads per cpu = 232.588784

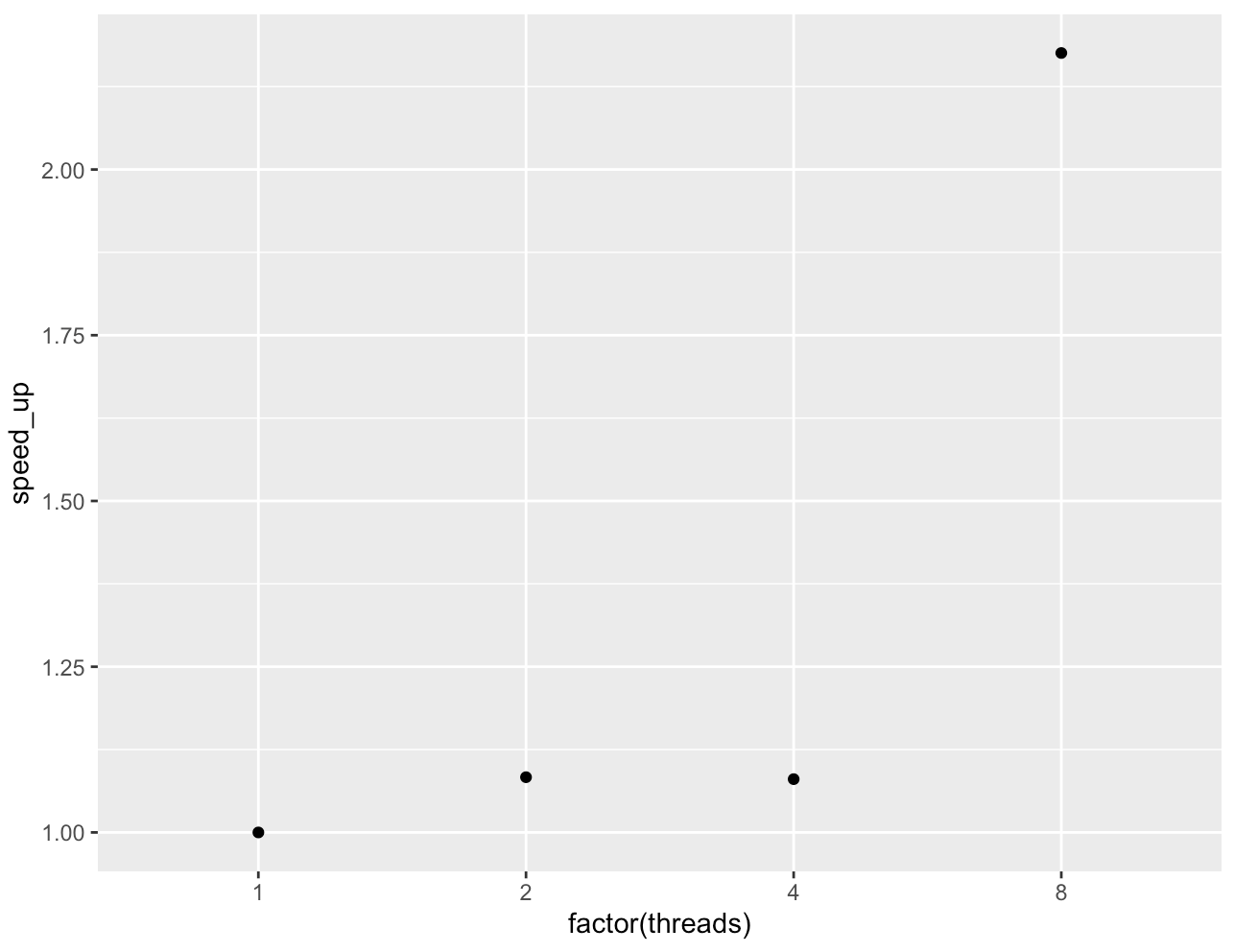
Walltime to execute matrix\_mult.c on 2 cpus and 2 threads per cpu = 233.217386

Walltime to execute matrix\_mult.c on 4 cpus and 2 threads per cpu = 232.588784

wall time vs. number of threads



speedup vs. number of threads

****

There is a significant difference between the graphs obtained in question 6 and question 9.

The speedup does not steadily increase in case of a multithreaded system. This might be due to the fact that multiple threads spread out over multiple cores might lead to a greater overhead in terms of processing.

**10. (DSCC 401 Only) Read the paper, “Deep Learning at 15PF,” by Kurth, et al. A copy of this paper has been uploaded to the Blackboard site and is available under the instructions for this homework assignment (kurth.pdf). Provide detailed answers to the following questions:**

**A. Examine Figure 6. Focus on the "synchronous" approach (blue line). If you were**

**performing an analysis of HEP, would you recommend running the calculation on**

**1024 nodes? Why or why not?**

Figure 6 shows a steady decline in the performance of the HEP network for the synchronous approach when the number of nodes exceeds about 500. Thus, 1024 nodes are not recommended.

**B. Based on the data in Figure 6, provide a rough estimate for the fraction of the**

**The synchronous algorithm that is parallelized.**

99.5% of the algorithm was parallelized if the value of S was approximately 200.

**C. Again, looking at Figure 6, why would the speedup for 1024 nodes be less than the**

**speedup for 512 nodes when using the synchronous approach?**

The synchronous algorithm cannot be scaled for nodes more than 256 according to figure 6a since the performance is inferior for 1024 nodes as compared to 512, hence, the speedup is less.