CS 470 Slurm and Performance Analysis Lab

Name(s): Brennan Krutis, Josh Derrow

This lab will introduce you to running programs on our cluster as well as the basics of performance analysis.

1. Copy the `nas-benchmarks` folder from `/shared/cs470` to your home directory and examine the resulting files. Make sure you **read the first two sections of the `README.md` file ("Introduction" and "Interactive Submission") in detail**, as they contain an explanation of the provided program file names and examples of how to run the programs on the cluster.

2. Choose a benchmark to work on. *Benchmarks* are well-defined, standardized programs that are generally accepted to be representative of "real" workloads. You have been provided two types of benchmarks: OpenMP and MPI. OpenMP - benchmarks run on a single node and scale up to 32 or 64 threads (on our system) while MPI benchmarks run on multiple nodes with larger workloads and scale up to 128 processes (8 nodes). **CLEARLY MARK** the benchmark you've chosen below by highlighting it below. Note that EP is not a valid choice for this lab because it is naturally parallel and because it is the target of the provided code samples in the readme. Also, the OpenMP benchmarks are only eligible for a total of 9/10 points on this lab because they require less work to run than the MPI benchmarks.

- BT: Block Tri-diagonal numerical linear system solver (OpenMP and MPI)
- CG: Conjugate Gradient approximation of smallest eigenvalue (MPI only)
- FT: Fast Fourier Transform-based differential equation solver (OpenMP only)
- LU: Lower-Upper Gauss-Seidel numerical linear system solver (OpenMP and MPI)
- MG: Multi-Grid mesh solution to the discrete Poisson problem (OpenMP only)
- **SP: Scalar Penta-diagonal solver (OpenMP only) (we are using this one)**
- UA: Unstructured Adaptive mesh-based heat transfer simulation (OpenMP only)

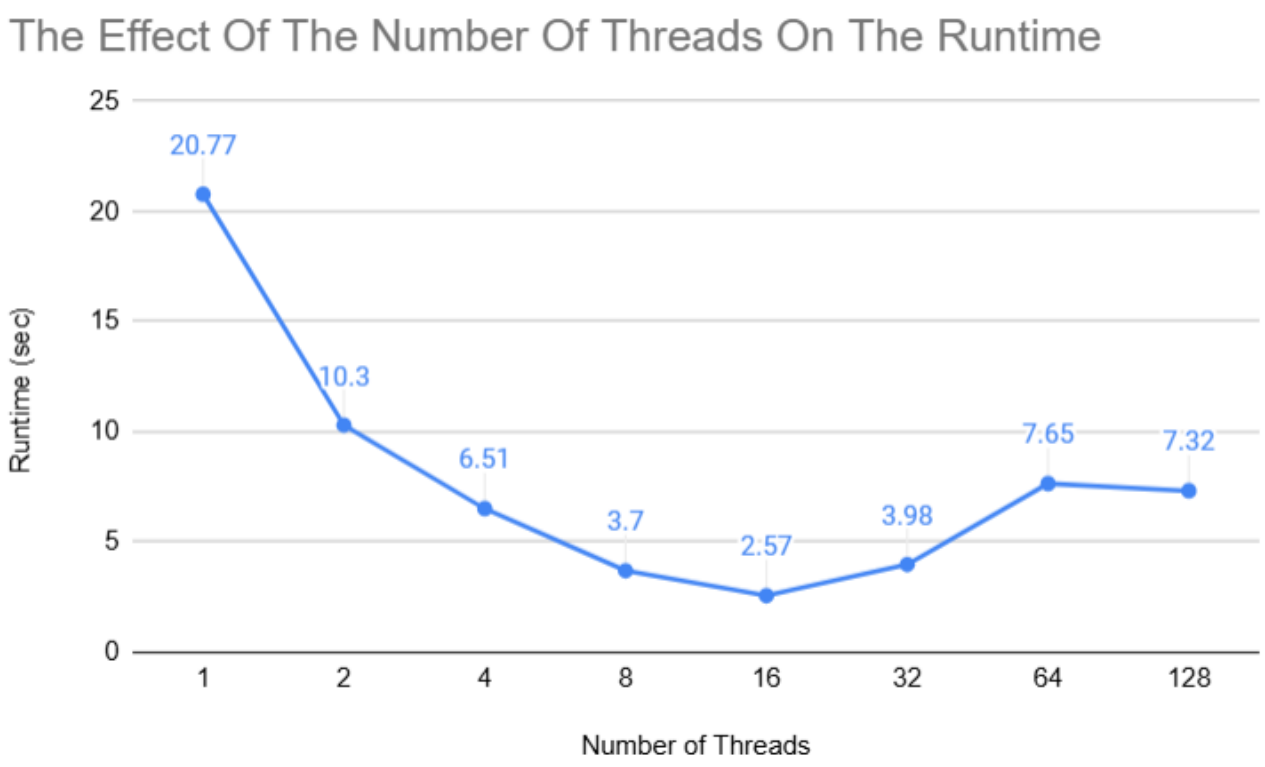
3. (Optional) Create a Slurm batch submission script to run your chosen benchmark. It is recommended that you start with one of the sample scripts on the cluster website or in the provided lab files (see the last section of the `README.md` file called "Batch Submission"). You may skip this step if you wish to use interactive job submission (e.g., `srun`) exclusively.

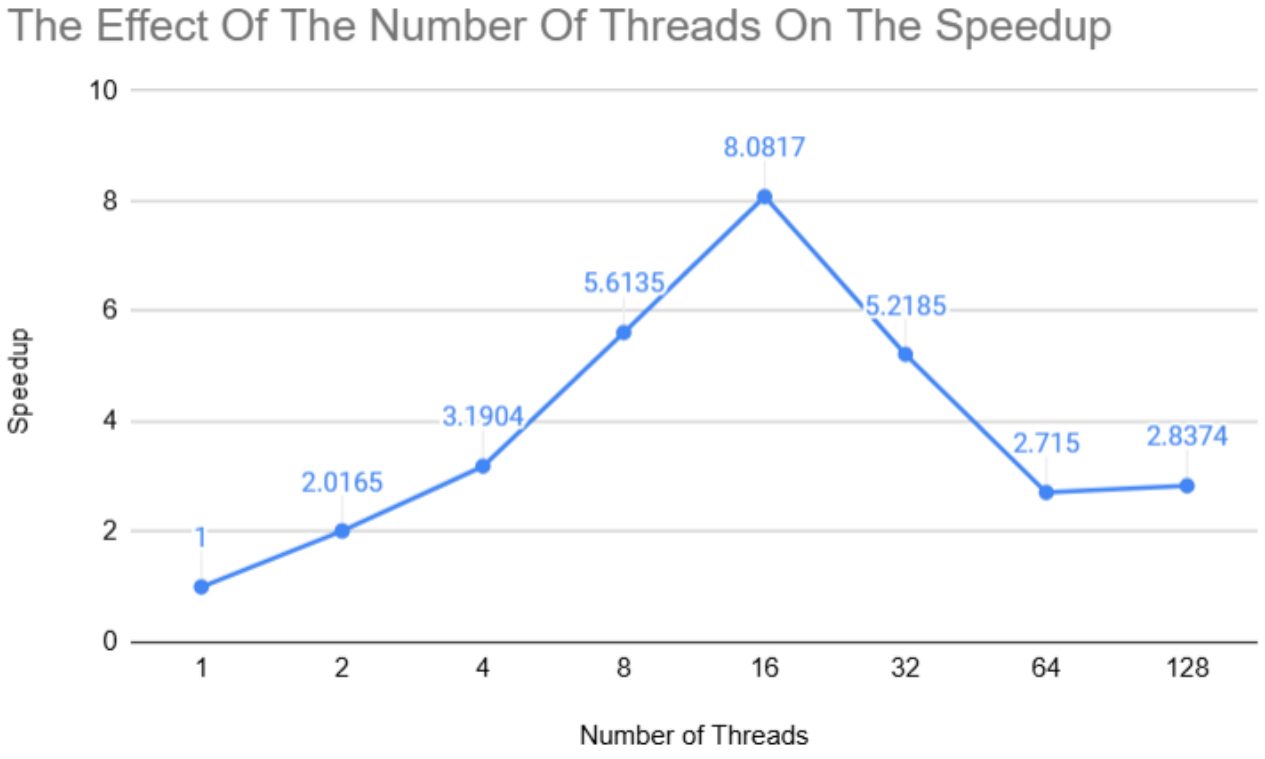
4. Using the Slurm job submission system, run a series of experiments on your chosen benchmark, varying logarithmically the number of threads (for OpenMP benchmarks) or processes (for MPI benchmarks) for each experiment. Use the output of your experiments to fill out the following table (you may not need all of the rows):

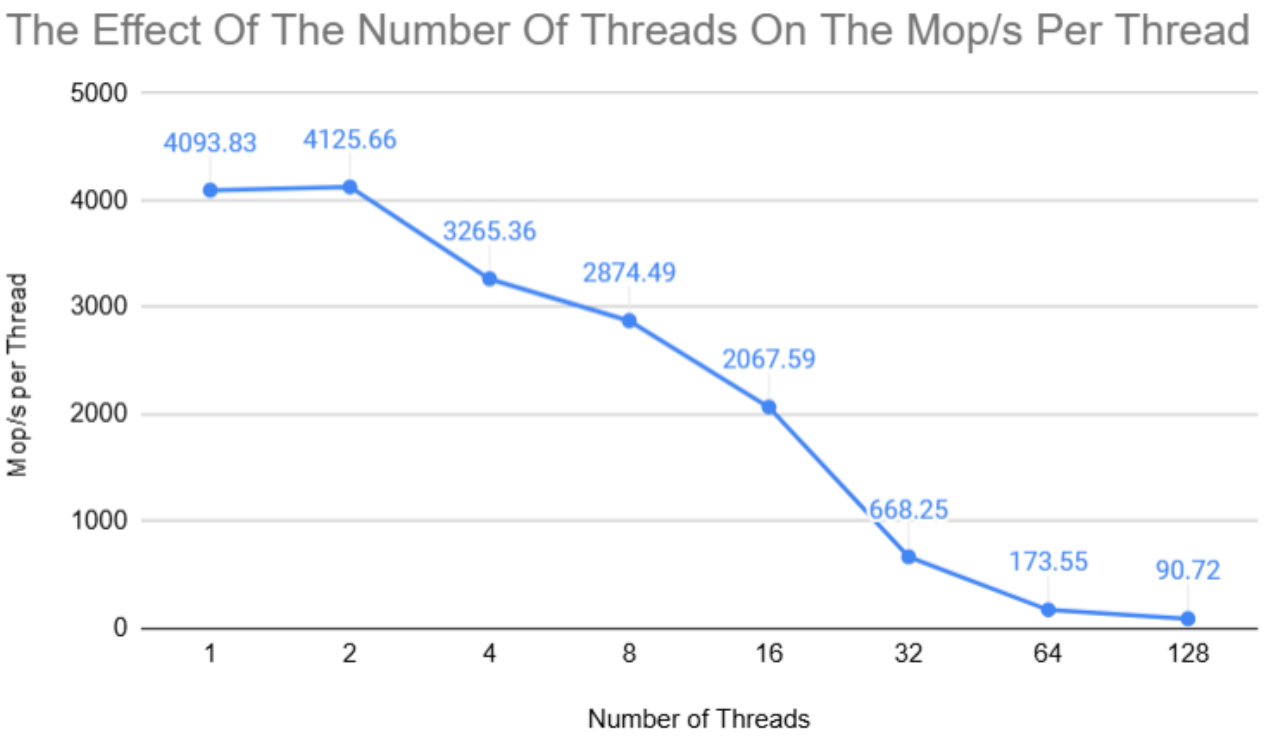
Number of threads/processes	Time in seconds	Speedup (T_s/T_p)	Mop/s total	Mop/s per thread/process
1	20.77 (T_s)	N/A	4093.83	4093.83
2	10.30	S=2.0165	8251.33	4125.66
4	6.51	S=3.1904	13061.44	3265.36
8	3.70	S=5.6135	22995.90	2874.49
16	2.57	S=8.0817	33081.51	2067.59
32	3.98	S=5.2185	21383.88	668.25
64	7.65	S=2.7150	11107.51	173.55
128	7.32	S=2.8374	11611.96	90.72

Note: For this lab, just use the 1-thread/process run time for Ts

5. Create three graphs of this data, each with thread/process count on the x-axis (logarithmic). For the y-axis (not logarithmic), one graph should have **run time**, one should have **speedup**, and one should have **Mop/s per thread/process**. Make sure all graphs, axes, and data sets are clearly and correctly labeled.


The Effect Of The Number Of Threads On The Runtime
Runtime (sec)
25
20
15
10
5
0
20.77
10.3
6.51
3.7
2.57
3.98
7.65
7.32
1
2
4
8
16
32
64
128
Number of Threads


The Effect Of The Number Of Threads On The Speedup
Speedup
10
8
6
4
2
0
1
2.0165
3.1904
5.6135
8.0817
5.2185
2.715
2.8374
1
2
4
8
16
32
64
128
Number of Threads



6. Write a 1-2 paragraph performance analysis of your results. How well does your chosen benchmark scale on our system? Use your data and graphs to support your argument. Be as specific and quantitative as possible. Do your results provide evidence that the program demonstrates strong scaling, weak scaling, or both? Does it appear to scale linearly? What are the limitations of your experiments?

The OpenMP-SP benchmark seems to scale well on our system. The OpenMP-SP benchmark does not exhibit weak scaling because the problem size never increases, just the number of threads. It appears to exhibit linear-strong scaling for a low number of threads ($p \leq 16$), but no scaling for a larger number of threads ($p > 16$). Initially, the runtime roughly halved from 20.77 seconds to 10.3 seconds and the speedup is doubled when increasing from 1 to 2 threads. From 2 to 4 threads, the runtime decreased by about 37% and the speedup increased by about 50%. This trend of decreased runtime and increased speedup continued until running the benchmark with 32 threads, after this point, the runtime slightly increased and the speedup slightly decreased. Finally, once the benchmark was run with 128 threads, the runtime slightly decreased by about 5% and the speedup slightly increased by about 5%. The performance increases/decreases were the smallest between the 64 and 128 thread benchmarks, which seems to suggest that the data is stabilizing around/after this point.

One limitation of our experiment is the maximum number of threads ($p = 128$). The runtime and speedup seem to be stabilizing at this point but further trials with more threads ($p > 128$) could confirm/deny this. Another limitation of our experiment is the fact that each benchmark was only ran once per thread count. If each benchmark was ran 3 or more times per thread count, the data might be more conclusive.

7. Estimate the percentage of the program that cannot be parallelized (i.e., *r* in our equation for Amdahl's Law).

$p = \#\ of\ processors$
$r = serial\ \%\ of\ program$
$Speedup = S$

$$S = \frac{T_s}{\frac{(1-r)T_s}{p} + rT_s}$$

$$S\left(\frac{(1-r)T_s}{p} + rT_s\right) = T_s$$

$$\frac{(1-r)T_s}{p} + rT_s = \frac{T_s}{S}$$
$$(1-r)T_s + prT_s = p\frac{T_s}{S}$$
$$1 - r + pr = \frac{p}{S}$$
$$-r + pr = \frac{p}{S} - 1$$
$$-r(1-p) = \frac{p}{S} - 1$$
$$-r = \frac{\frac{p}{S} - 1}{(1-p)}$$
$$r = \left| -(\frac{\frac{p}{S} - 1}{(1-p)}) \right|$$
$p_2 = 2,\ S_2 = 2.0165 \rightarrow r_2 \approx 0.0082$
$p_4 = 4,\ S_4 = 3.1904 \rightarrow r_4 \approx 0.0846$
$p_8 = 8,\ S_8 = 5.6135 \rightarrow r_8 \approx 0.0607$
$p_{16} = 16,\ S_{16} = 8.0817 \rightarrow r_{16} \approx 0.0653$
$p_{32} = 32,\ S_{32} = 5.2185 \rightarrow r_{32} \approx 0.1655$
$p_{64} = 64,\ S_{64} = 2.715 \rightarrow r_{64} \approx 0.3583$
$p_{128} = 128,\ S_{128} = 2.8374 \rightarrow r_{128} \approx 0.3473$
$$r_{avg} = \frac{(r_2 + r_4 + r_8 + r_{16} + r_{32} + r_{64} + r_{128})}{7} = 0.1557 * 100 = 15.57\%$$

We estimate that around 15.57% of the program cannot be parallelized.

8. Submit this document as a PDF to the appropriate Canvas assignment by the due date. If you worked in a group, please submit ONLY ONE copy per group and make sure the list of names at the top is accurate.