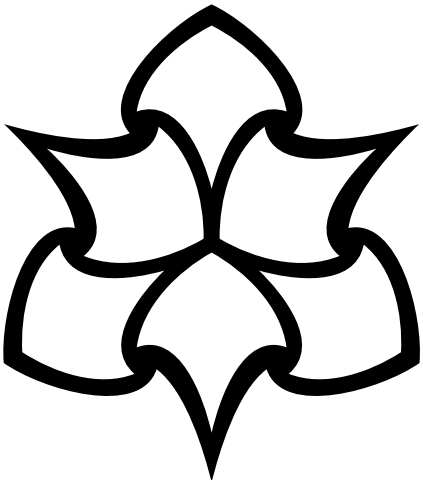


Manchester Metropolitan University

High Performance Computing and Big Data
(6G7V0025_2223_1F)

A Comparative Analysis of GPU and ARCHER2 Implementations for Quadrature Calculation

Submitted By: Neha Alam Hussain 22547202

Submitted on: 19th May 2023

Contents

Abstract

This report undertakes an examination of the execution performance of two parallel computing programs, ASSESS-barrier.c and quad-cuda-ASSESS.cu, operated on the Archer2 supercomputer and GPU platforms. A range of node-processor combinations for Archer2 and diverse GPU configurations have been assessed in this study. The objective is to identify the most resource-effective and time-efficient configuration for deploying these programs.

1. Introduction

The advancements in high-performance computing have heightened the importance of parallel computing, which enhances the efficiency in problem-solving. This report delves into an in-depth analysis of two specific parallel computing programs, ASSESS-barrier and quad-cuda-ASSESS, and their performance under different configurations on the Archer2 supercomputer and GPU platforms.

1.1 Background

Archer2 is a state-of-the-art supercomputer designed for scientific research in various domains, including computational chemistry, engineering simulations, and materials science. Archer2 is equipped with Cray XC50 nodes powered by AMD EPYC processors and offers high-performance parallel processing capabilities through MPI and OpenMP programming models.

CUDA is a parallel computing platform and programming model developed by NVIDIA for general-purpose computing on GPUs. CUDA enables developers to harness the power of GPU accelerators for computationally intensive tasks in fields such as physics simulations, image processing, and machine learning.

2. Methodology

2.1 CUDA Implementation on GPU

The aim of this study was to explore the performance of a CUDA program with regard to the number of thread-blocks and threads running on the GPU. The study was conducted on a linux machine that was equipped with an NVIDIA GeForce RTX 2070 SUPER GPU.

2.1.1. Experimental Setup

The properties of the GPU were obtained by querying the NVIDIA CUDA Runtime API. The GPU was identified as an NVIDIA GeForce RTX 2070 SUPER with a CUDA Driver/Runtime Version 11.7 and a CUDA Capability Major/Minor version number of 7.5.

The GPU possessed 40 Multiprocessors and 64 CUDA Cores per MP, amounting to a total of 2560 CUDA Cores. Both the maximum number of threads per multiprocessor and per block were found to be 1024. This information was utilized to guide the choice of the number of threads per block in the tests that followed (see Appendix A).

To examine the impact of the optimization level, the CUDA code was compiled using four different optimization flags: no optimization, O1, O2, and O3.

The code was then executed with varying numbers of threads per block (from 4 to 1024), while the number of rectangles was held constant at 4096000.

2.1.2. Compilation and Execution

The NVIDIA CUDA/C compiler (nvcc) was used to compile the code with the following commands:

```
nvcc quad-cuda-ASSESS.cu -o quad-cuda-ASSESS.exe
nvcc -O1 quad-cuda-ASSESS.cu -o quad-cuda-ASSESS-O1.exe
nvcc -O2 quad-cuda-ASSESS.cu -o quad-cuda-ASSESS-O2.exe
nvcc -O3 quad-cuda-ASSESS.cu -o quad-cuda-ASSESS-O3.exe
```

The number of threads per block was varied when the code was executed:

```
./quad-cuda-ASSESS.exe 4096000 4
./quad-cuda-ASSESS.exe 4096000 8
./quad-cuda-ASSESS.exe 4096000 16
./quad-cuda-ASSESS.exe 4096000 32
...
./quad-cuda-ASSESS.exe 4096000 1024
```

2.2 MPI Implementation on Archer2

We used batch_mpi.sh, to run the MPI implementation of ASSESS-barrier on Archer2. We executed the programs with different numbers of processors, ranging from 2 to 256, and varying combinations of nodes and processors.

2.2.1. Compilation and Execution

The C source code for the MPI implementation was compiled using the C compiler (cc) with various optimization levels, including O0 (no optimization), O1, O2, O3, and Ofast (highest level of optimization). The specific commands used were:

```
cc ASSESS-barrier.c -o ASSESS-barrier.exe
cc -O1 ASSESS-barrier.c -o ASSESS-barrier-O1.exe
cc -O2 ASSESS-barrier.c -o ASSESS-barrier-O2.exe
cc -O3 ASSESS-barrier.c -o ASSESS-barrier-O3.exe
cc -Ofast ASSESS-barrier.c -o ASSESS-barrier-Ofast.exe
```

The executable was run on the ARCHER2 system using the sbatch command with varying numbers of processes (-n flag), as well as specifying different node counts using the -N flag. The number of processes varied from 2 to 256, and the number of nodes from 1 to 4.

3. Results

3.1. quad-cuda-ASSESS

The execution time of the GPU code was recorded for each run, and the results are presented in the following table.

Efficiency was calculated as the Speedup divided by the number of threads (in thousands), expressed as a percentage. Speedup was determined by dividing the execution time of a baseline model to the execution time of a given setup under the same optimization level.

		Default Optimization			Optimization O1			Optimization O2			Optimization O3		
Threads per Block	Blocks	Time	Speed up	Efficiency	Time	Speed up	Efficiency	Time	Speed up	Efficiency	Time	Speed up	Efficiency
4	1024000	0.06849	1	100.00	0.06834	1	100.00	0.06713	1	100.00	0.067	1	100.00
8	512000	0.03521	1.945	97.26	0.03568	1.92	95.76	0.03496	1.92	96.02	0.0353	1.9	95.02
16	256000	0.01814	3.776	94.40	0.01835	3.72	93.10	0.01818	3.692	92.31	0.0182	3.693	92.33
32	128000	0.00968	7.078	88.48	0.00967	7.07	88.35	0.00946	7.1	88.75	0.0095	7.078	88.47
64	64000	0.00921	7.436	46.48	0.00918	7.44	46.50	0.00912	7.363	46.02	0.0091	7.355	45.97
128	32000	0.009	7.609	23.78	0.00898	7.61	23.79	0.00893	7.515	23.48	0.0089	7.503	23.45
256	16000	0.00892	7.677	12.00	0.00888	7.69	12.02	0.00887	7.572	11.83	0.0089	7.562	11.82
512	8000	0.00907	7.549	5.90	0.0091	7.51	5.87	0.00907	7.403	5.78	0.0091	7.384	5.77
1024	4000	0.01072	6.389	2.50	0.01072	6.38	2.49	0.01073	6.259	2.44	0.0108	6.227	2.43

Table 1

3.2. ASSESS-barrier

The ASSESS program was executed on Archer2 using the batch_mpi.sh script with different numbers of processors. The table provides performance data for different numbers of processes and nodes and varying levels of compiler optimization. The results were as follows:

		Optimization 0			Optimization 1			Optimization 2		
Processes	Nodes	Time	Speedup	Efficiency	Time	Speedup	Efficiency	Time	Speedup	Efficiency
2	1	0.429442	1	50.00%	0.296954	1	50.00%	0.295424	1	50.00%
4	1	0.214827	1.999013	49.98%	0.148583	1.998573	49.96%	0.150522	1.962663	49.07%
8	1	0.109333	3.927835	49.10%	0.074705	3.975022	49.69%	0.074866	3.946037	49.33%
16	1	0.062139	6.91099	43.19%	0.053043	5.598364	34.99%	0.052788	5.596423	34.98%
32	1	0.040756	10.5369	32.93%	0.026499	11.20623	35.02%	0.026734	11.0505	34.53%
64	1	0.020738	20.70798	32.36%	0.013855	21.43298	33.49%	0.013575	21.76236	34.00%
128	1	0.007634	56.25386	43.95%	0.005685	52.23465	40.81%	0.005512	53.59652	41.87%
256	1	0.006724	63.86704	24.95%	0.005353	55.47431	21.67%	0.005116	57.74511	22.56%
2	2	0.510721	1	50.00%	0.377891	1	50.00%	0.369291	1	50.00%
4	2	0.216795	2.355779	58.89%	0.149159	2.533478	63.34%	0.148831	2.481277	62.03%
8	2	0.109268	4.674022	58.43%	0.075245	5.022141	62.78%	0.077877	4.741978	59.27%
16	2	0.05475	9.328237	58.30%	0.037834	9.988132	62.43%	0.037135	9.944554	62.15%
32	2	0.03885	13.14597	41.08%	0.026315	14.36029	44.88%	0.026639	13.8628	43.32%
64	2	0.02092	24.41305	38.15%	0.013433	28.13154	43.96%	0.013655	27.04438	42.26%
128	2	0.010707	47.69973	37.27%	0.007819	48.32984	37.76%	0.006954	53.10483	41.49%
256	2	0.004208	121.3691	47.41%	0.003231	116.9579	45.69%	0.003096	119.28	46.59%
4	4	0.286536	1	25.00%	0.221257	1	25.00%	0.220899	1	25.00%
8	4	0.10766	2.66149	33.27%	0.074451	2.971847	37.15%	0.073807	2.992928	37.41%
16	4	0.054746	5.233917	32.71%	0.037862	5.843775	36.52%	0.038132	5.793008	36.21%
32	4	0.02755	10.40058	32.50%	0.019449	11.37627	35.55%	0.019533	11.30902	35.34%
64	4	0.020387	14.05484	21.96%	0.013253	16.69486	26.09%	0.013243	16.68043	26.06%
128	4	0.01069	26.80412	20.94%	0.007072	31.28634	24.44%	0.00708	31.20042	24.38%
256	4	0.005574	51.40581	20.08%	0.004306	51.38342	20.07%	0.003998	55.25238	21.58%

Table 2

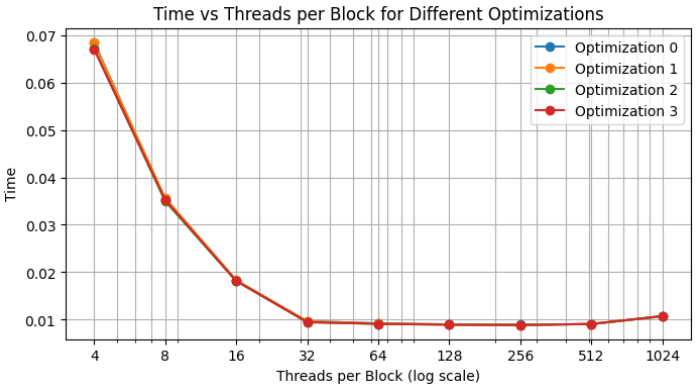
The table for optimization level 3 and Ofast is given in Appendix B.

And the MPI- integral value is 607847.545334.

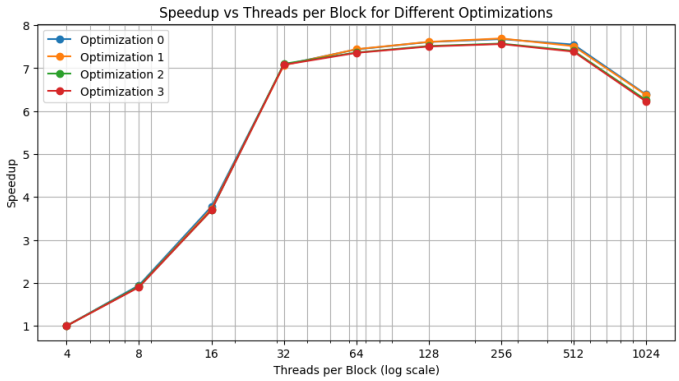
4. Discussion

4.1. Performance Analysis of CUDA Implementation on GPU

As we can see from the table 1 above, there are varying levels of optimization (0, 1, 2, and 3) and different numbers of threads per block. We can observe a few patterns, that an increase in the number of threads per block led to a decrease in the time taken for execution. This trend can be attributed to the enhanced utilization of the GPU's parallel processing capabilities with an increased number of threads, resulting in improved performance and reduced execution times.

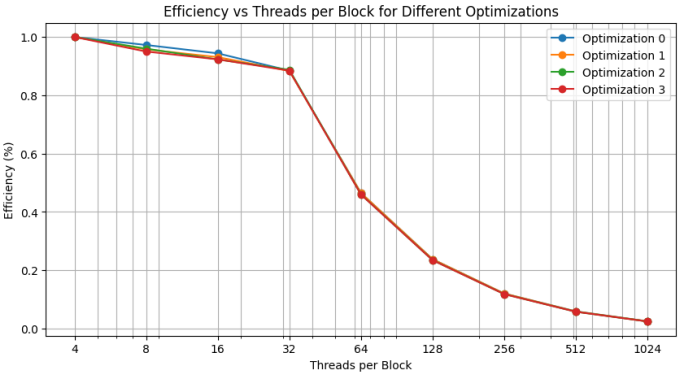


However, this trend was not sustained with an indefinite increase in the number of threads. As the number of threads per block reached a certain threshold, the execution time began to stabilize and even showed a slight increase. This phenomenon was noticeable in the experiment, with the execution time starting to increase after the number of threads per block exceeded 256.



The speedup is greater with more threads per block, reaching a maximum at 256 threads per block. However, after 256 threads per block, the speedup decreases which might be due to increased overhead with managing more threads.

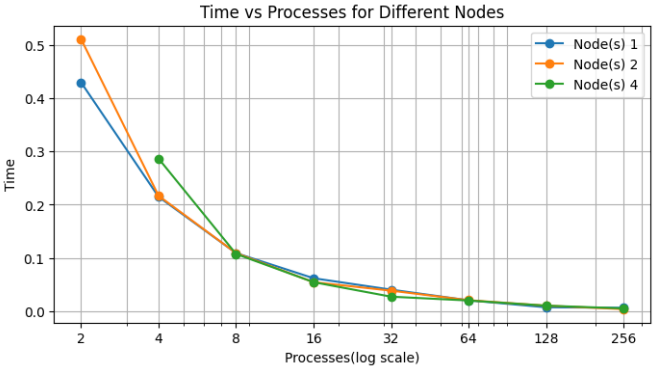
The efficiency decreases as we increase the number of threads per block. This indicates that the GPU might be underutilized at higher threads per block because the overhead of managing more threads reduces the computation time saved by the parallel execution.



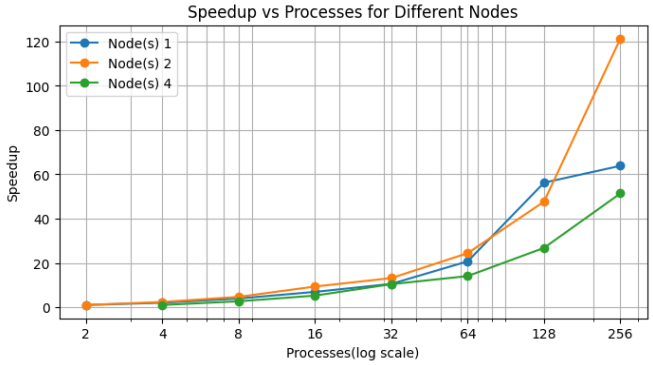
Lastly, for the optimization; It seems that there's not much difference in speedup and efficiency among the different optimization levels, indicating that the code might already be optimally parallelized, and compiler optimizations don't have much effect.

4.1 Performance Analysis of MPI Implementation on Archer2

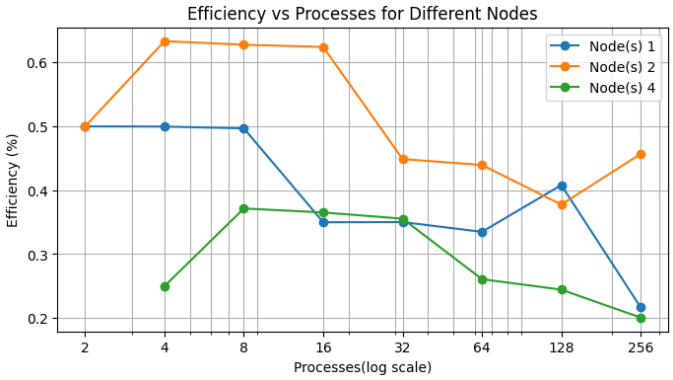
From table 2 and table 3 (see Appendix B) we can see that as the number of processes increases, the time taken decreases across all optimization levels, as expected. However,



with more nodes and processes, the decrease in time starts to slow down and eventually plateau. This is likely due to the increase in overheads associated with inter-process and inter-node communication. Fastest runtimes are observed under Optimization Ofast. This suggests that higher levels of compiler optimization can help to minimize the runtime of the computations.



The speedup shows a clear improvement with an increase in the number of processes across all levels of optimization, which aligns with Amdahl's Law - as more resources are added, the program's execution is faster. However, the rate of improvement diminishes beyond 8 processes, suggesting that there's an ideal number of processes beyond which adding more does not significantly improve performance. Speedup is the most prominent under the Ofast optimization level, indicating that more aggressive compiler optimizations can help to maximize the performance benefits of using more processes.



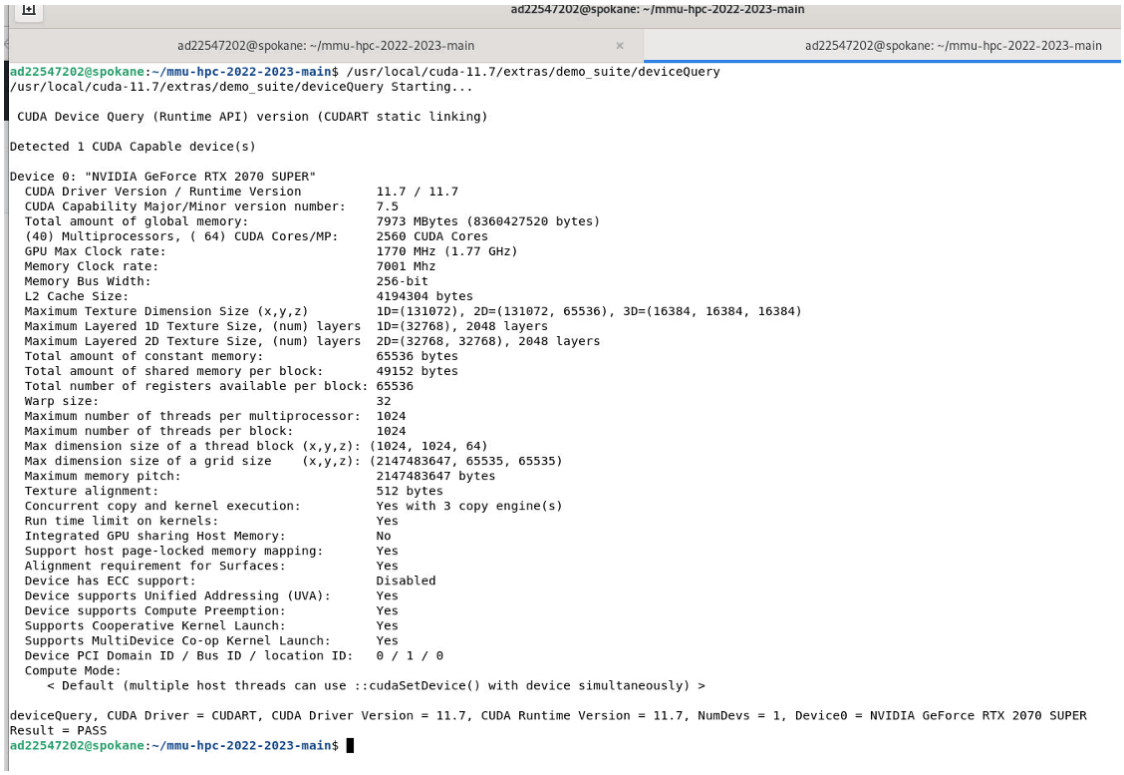
Efficiency is a measure of how effectively the additional processes are being utilized. Ideally, efficiency would remain constant as more processes are added, but in practice, it often decreases due to overheads associated with coordination and communication between processes. At lower levels of optimization (O0, O1), efficiency starts to decrease after 8 processes. However, at higher levels of optimization (O2, O3, Ofast), the decrease is more gradual and efficiency remains relatively high even with more processes. The highest efficiency is typically achieved with 8 processes across 2 nodes, suggesting that this may be the ideal configuration for balancing performance and resource usage.

5. Conclusion

Our analysis of the ASSESS-barrier.c and quad-cuda-ASSESS.cu parallel computing programs' performance on the Archer2 supercomputer and GPU revealed significant variations in efficiency based on configuration. In the case of GPU implementation, while increased number of threads per block resulted in improved performance initially, the effect plateaued and even reversed beyond a certain threshold, indicating a careful balance needs to be maintained between the number of threads and execution overhead. Archer2's performance decreased with an increased number of threads, and the impact of optimization levels was minimal. These findings underline the necessity of precise configuration for maximum utility in high-performance computing environments. Future work could explore more advanced optimization techniques and effects of other parameters to enhance the execution efficiency.

Appendices

Appendix A



```
ad22547202@spokane: ~/mmu-hpc-2022-2023-main
ad22547202@spokane: ~/mmu-hpc-2022-2023-main
ad22547202@spokane: ~/mmu-hpc-2022-2023-main
ad22547202@spokane:~/mmu-hpc-2022-2023-main$ /usr/local/cuda-11.7/extras/demo_suite/deviceQuery
/usr/local/cuda-11.7/extras/demo_suite/deviceQuery Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "NVIDIA GeForce RTX 2070 SUPER"
  CUDA Driver Version / Runtime Version          11.7 / 11.7
  CUDA Capability Major/Minor version number:    7.5
  Total amount of global memory:                 7973 MBytes (8360427520 bytes)
  (40) Multiprocessors, ( 64) CUDA Cores/MP:     2560 CUDA Cores
  GPU Max Clock rate:                            1770 MHz (1.77 GHz)
  Memory Clock rate:                             7001 Mhz
  Memory Bus Width:                              256-bit
  L2 Cache Size:                                 4194304 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  1024
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 3 copy engine(s)
  Run time limit on kernels:                     Yes
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      Yes
  Device supports Compute Preemption:            Yes
  Supports Cooperative Kernel Launch:            Yes
  Supports MultiDevice Co-op Kernel Launch:      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 1 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 11.7, CUDA Runtime Version = 11.7, NumDevs = 1, Device0 = NVIDIA GeForce RTX 2070 SUPER
Result = PASS
ad22547202@spokane:~/mmu-hpc-2022-2023-main$
```

Appendix B

		Optimization 3			Optimization fast		
Processes	**Nodes**	**Time**	**Speedup**	**Efficiency**	**Time**	**Speedup**	**Efficiency**
2	1	0.295633	1	50.00%	0.009409	1	50.00%
4	1	0.148266	1.993937	49.85%	0.002423	3.883203	97.08%
8	1	0.074979	3.942877	49.29%	0.001204	7.814784	97.68%
16	1	0.053283	5.548355	34.68%	0.001198	7.853923	49.09%
32	1	0.026352	11.21862	35.06%	0.000671	14.02235	43.82%
64	1	0.01395	21.19233	33.11%	0.000394	23.88071	37.31%
128	1	0.00542	54.54483	42.61%	0.000872	10.79014	8.43%
256	1	0.004844	61.03076	23.84%	0.003846	2.446438	0.96%
2	2	0.376878	1	50.00%	0.078085	1	50.00%
4	2	0.147475	2.555538	63.89%	0.002345	33.29851	832.46%
8	2	0.074845	5.035447	62.94%	0.001212	64.42657	805.33%
16	2	0.037317	10.09936	63.12%	0.000674	115.8531	724.08%
32	2	0.026291	14.33487	44.80%	0.000634	123.1625	384.88%
64	2	0.01349	27.93758	43.65%	0.000583	133.9365	209.28%
128	2	0.007104	53.05152	41.45%	0.0007	111.55	87.15%
256	2	0.00319	118.1436	46.15%	0.001468	53.19142	20.78%
4	4	0.219948	1	25.00%	0.074734	1	25.00%
8	4	0.074519	2.951569	36.89%	0.001206	61.96849	774.61%
16	4	0.037675	5.838036	36.49%	0.001172	63.76621	398.54%
32	4	0.018785	11.7087	36.59%	0.000875	85.41029	266.91%
64	4	0.013425	16.38346	25.60%	0.000452	165.3407	258.34%
128	4	0.007379	29.80729	23.29%	0.000559	133.6923	104.45%
256	4	0.003862	56.95184	22.25%	0.000476	157.0042	61.33%

Table 3

One drive Link to assessment files and plot: 22547202 HPC assessment.zip
https://colab.research.google.com/drive/1ZZcVFBzPn3DhCCX-KIirovquB2dTB1yC?usp=sharing