EX3: Task 3

[December, 2024]

Group Members:

- Marcus Konrath, 01300688
- Uğur Cem Erdem, 12346958
- Katharina Perl, 01426575

Monte Carlo Interation

The Monte Carlo Integration (MCI) is computed as:

$$I \approx \frac{xmax - xmin}{N} \sum_{i=1}^{N} f(x_i)$$

where x_i are random points uniformly sampled in $[xmin, xmax]$, and N is the total number of samples

We expected the following results:

$$\text{SINX} = \int_{-\pi/2}^{\pi/2} \sin(x)\, dx = 0$$

$$\text{COS2XINV} = \int_{-\pi/2}^{\pi/2} \cos^2\left(\frac{1}{x}\right)\, dx \approx 1.21772$$

$$\text{X4M5} = \int_{-\pi/2}^{\pi/2} 5x^4\, dx \approx 19.126$$

To efficiently parallelize the computation, the total number of samples N is distributed evenly across all threads. This ensures balanced workload distribution. In cases where the number of samples is not divisible by the number of threads, the master thread (Thread ID 0) is assigned the remaining samples.

HPC Cluster

Each CPU of the HPC Cluster has 20 cores (40 threads), giving a total of 40 cores/80 threads per node. Each core supports 2 threads: 1 core = 2 threads.

Cores needed = $\frac{\text{Threads desired}}{\text{Threads per core}}$

Threads	Cores Needed
1	1
5	3
10	5
40	20
80	40

We set up the batch file to run all three functions together, so the results were combined into one output file. The only thing we needed to change was the number of cores and threads in the batch file.

We encountered the following runtimes:

Threads	Runtime
1	0.81533
5	0.282098
10	0.198622
20	0.127703
40	0.0704629
80	0.0600694

Table 1: * SINX

Threads	Runtime
1	1.01566
5	0.359375
10	0.242065
20	0.144327
40	0.081681
80	0.0587195

Table 2: * COS2XINV

Threads	Runtime
1	0.596725
5	0.251936
10	0.164719
20	0.109085
40	0.0539882
80	0.0437341

Table 3: * X4M5

(see plot here: Figure 1)

We also had a look at the speedup:

The speedup S is defined as:

$$S = \frac{T_1}{T_p}$$

Where:
- T_1: Runtime with one thread
- T_p: Runtime with p threads

(see plot here: Figure 2)

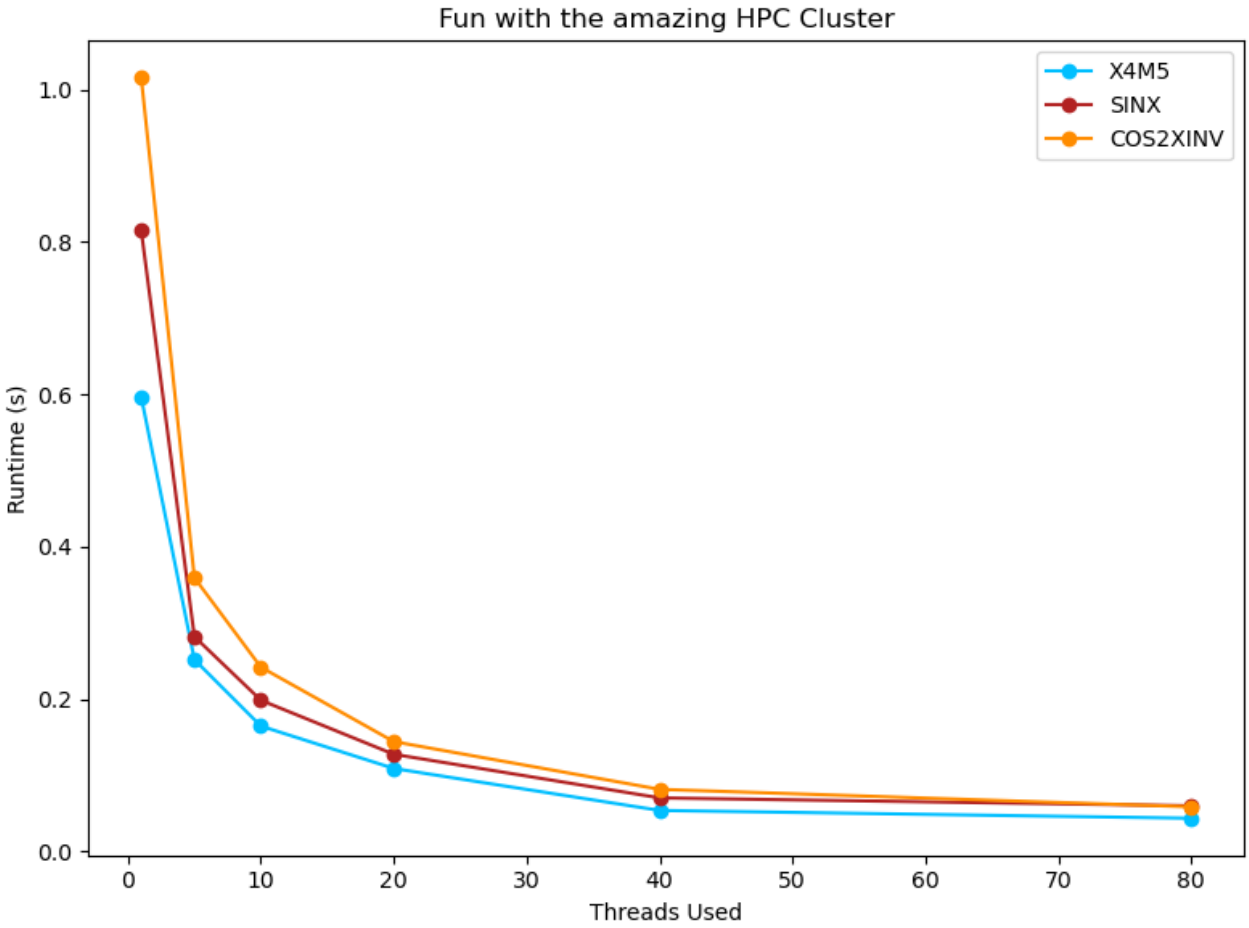

Figure 1: runtime vs. threads

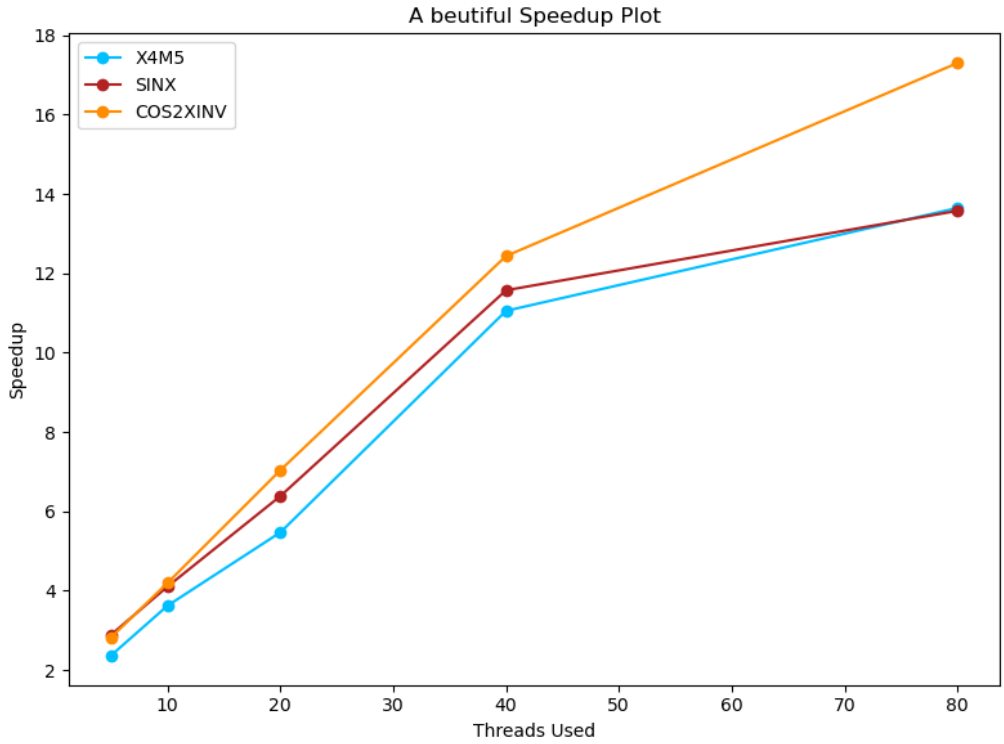

Figure 2: speedup

The plot (Figure 1) shows that runtimes dropped noticeably as we added more threads. For all the functions we tested, adding threads sped things up, with the biggest improvement happening around 40 threads. But when we got to 80 threads, the speedup started to level off because of overhead (also see Figure 2). Overhead basically means the extra work the system has to do to manage all those threads, like keeping them organized, making sure they don't clash, and sharing data between them. At some point, this extra work starts to cancel out the benefits of having more threads, so we don't see as much of a boost.

Conclusio: In this case, threading worked most efficiently with 40 threads.