HPC and Big Data
Assignment - Cluster Computing

Florian Wolf - 12393339 (UvA)
flocwolf@gmail.com

Lotte Felius - 12368032 (UvA)
lotte.felius@student.uva.nl

Veerle Dinghs - 11426020 (UvA)
veerledinghs@live.nl

January 30, 2021

1 Introduction

Part of the course High-Performance Computing and big Data (HPC) at the University of Amsterdam (UvA) is the workshop Cluster Computing. This report tackles the assignment proposed in the context of the workshop. Goal is to visualize and interpret the impact of parallel computing on processing time. Herefore we use the cluster computer facilities of SURF [1] and the programming language C [2]. For the implementation of parallel processes, we use the message passing library OpenMP [3].

2 Experiments

The experiments of this assignment revolve around testing the behaviour and limits of parallel, multi-thread computing. To compare the performance between different setups, we measure the time needed for the calculation of π. While π is an irrational number, meaning it can be defined up to infinite decimals and does not settle to repeat a pattern, it can be calculated with an exact equation. This is given by the Leibniz formula [4]

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots = \frac{\pi}{4}. \tag{1}$$

The setup of the experiments is threefold. First we analyze the the impact of the number of parallel threads on the computation time. Further we explore the relation between computation time and the count of numbers in the Leibniz series. All experiments, are executed on LISA, which is part of the SURF cluster computer facilities. Here jobs containing computation tasks can be submitted, by requesting a specific hardware requirements and run-time. While LISA has nodes with both Graphic Processing Units (GPU) and Central Processing Unit (CPU), we will run the experiments solely on CPU nodes. In average a node on LISA has 16 CPUs. Each CPU can process one task at a time, thus it is reasonable to parallelize tasks. Here threads come into play. The workload of the computation of a script can be split into numerous threads, and if not interdependent, they can run in parallel, each on a CPU of its own. If a CPU gets assigned more than one thread, it will execute them sequentially, with the exception of a method called hyperthreading. Note that CPUs or processors are physical hardware components and threads are a virtual work division, which can be arbitrary adjusted [5].

2.1 Maximum Threads

In this experiment, we report the change in computation time dependent on the number of threads chosen. The equation 1 is implemented in a C script and the number of threads for the computation of π is set as hyperparameter. The script is cloned and build on LISA. A batch script is submitted requesting a unshared node with maximum number of CPUs for one task. The task is to iterate over the number of threads $n_{threads} = 1, \ldots, 24$ and log the time required to estimate π for a fixed number of Leibniz series. The results in figure 1 show a logarithmic drop in computation time, which converges to about 3 seconds. After 16 threads the improvement becomes insignificant. This aligns with the average number of CPUs per node on LISA. A higher number of threads than CPUs does not lead to an accelerated computation since the maximum number of parallel threads is determined by the number of processors, thus additional threads will be executed sequentially.

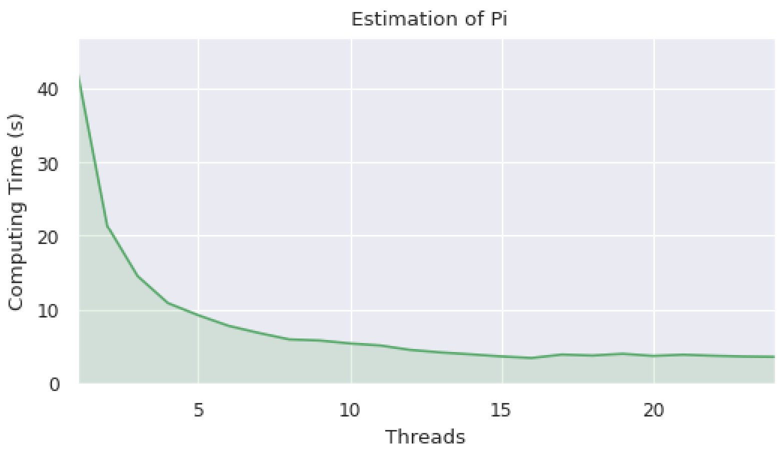

Figure 1: Computing time of π over number of threads.

2.2 Leibniz-time

We pick up the setup from the previous experiment, yet, now we fix the number of threads to 24 and iterate over the number of Leibniz series $n_{Leibniz} = [31.25, 62.5, 125, 250, 500, 1000, 2000] * 10^6$ and log the required computation time. Figure 2 shows a linear relation between the $n_{Leibniz}$ and the time needed to compute it, given the same setup of threads and CPU cores are used. This can be proven by looking at the iterative implementation to calculate π

```
pi = pi + pow(-1, i) * (4 / (2*((double) i)+1))
```

which iterates for $n_{Leibniz}$ times, hence has a linear complexity of $\mathcal{O}(n_{Leibniz})$.

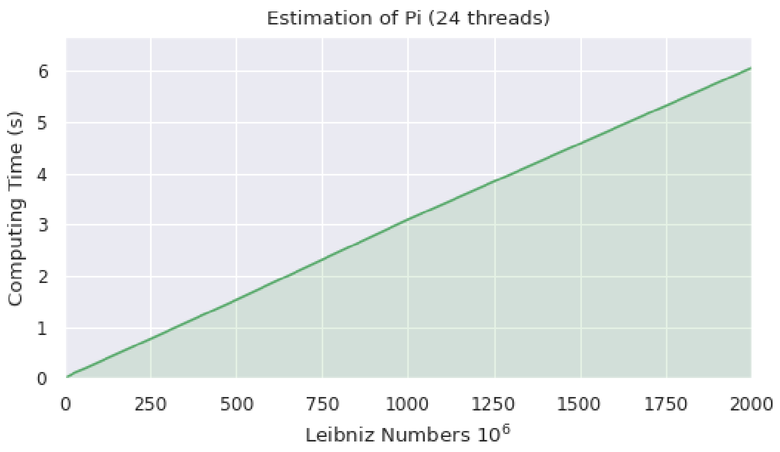

Figure 2: Computing time of π over number of Leibniz series.

2.3 Weak Scaling

In this last experiment, we emphasise the limits of adding more threads in terms of speed-up of the computation. The computation task has now a fixed $n_{Leibniz} = 2000 * 10^6$ and again we iterate over $n_{threads} = 1, \ldots, 24$. The speed-up is measured in multiples of the baseline time of a run with only one thread. Figure 3 shows a logarithmic rise with a convergence to 11 times speedup after 16 threads. The convergence has the same reason as the in the first experiment, namely the number of threads exceeds the available number of CPUs. Further we observe that the speed-up rate is < 1, meaning the speed-up scales weakly to the number of used threads, even for a single CPU per thread.

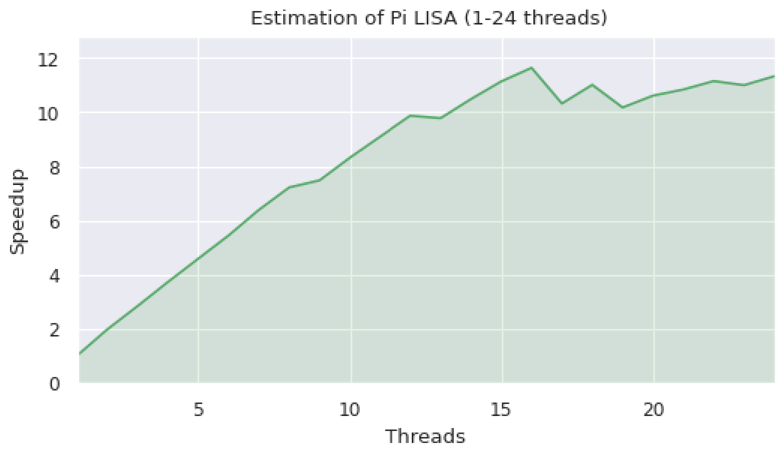

Figure 3: Computation speedup per number of threads.

3 Discussion

Concluding, we have gained 3 major insights from our experiments. Firstly and most important, that a computation time can be significantly reduced by using multi-threading, yet, this is effective number of threads is limited by the number of available processors. Further we have seen that the computation time has a linear correlation with the complexity of our code. Finally we showed the phenomenon of week scaling in times of speed-up compared to the number of included processors.

References

[1] Song Feng, ETH Torsten Hoefler, Switzerland Jessy Li, Damian Podareanu, Qifan Pu, Judy Qiu, Vikram Saletore, Mikhail E Smorkalov, and Jordi Torres. Valeriu codreanu, surfsara, netherlands ian foster, uchicago & anl, usa zhao zhang, tacc, usa.

[2] Brian W Kernighan and Dennis M Ritchie. *The C programming language.* 2006.

[3] OpenMP Architecture Review Board. OpenMP application program interface version 3.0, May 2008.

[4] Jonathan M Borwein, David H Bailey, and Roland Girgensohn. *Experimentation in mathematics: Computational paths to discovery.* CRC Press, 2004.

[5] Anthony Ralston, Edwin D. Reilly, and David Hemmendinger. *Encyclopedia of Computer Science.* John Wiley and Sons Ltd., GBR, 2003.