Sustainability of Scalable AI: a Run-time Experiment

Ana Carolina Laurini Malara[1],

[1] Institute of Mathematics and Statistics, University of São Paulo

1 Introduction

Given a C code for approximating the solution of a partial differential equation. This work aims to complete the following tasks:

- Compile and run the program on a computer available in sor.c at [1]. Make sure you get the best possible performance (in terms of runtime).
- What is the expected / theoretical complexity of this algorithm in terms of the two parameters? $O(n.m)$
- Do you think your results would carry over to different hardware?
- What relation between the parameters and the overall energy consumption would you expect?

2 Methods

In order to find an approach of a best performance, it was made an exploratory analysis, performed in two steps. First, the given code sor.c was modified in order to save the outputs *Bandwidth, Compute rate, Relative error, and overall run-time (s)* in a set of indexed files called "*out_n.txt*", such that n is the size of the matrix used for the discretizations. The source file was runned using a bash script called bash.sh, that executes the sor.c 1000 times (from 1 to 1000 and which also represents the maximum number of iterations parameter (m) as input), given a list of k values of n.
Second, a jupyter notebook was created, also available at [1] in order to make the exploratory data analysis between n, m and also between the other 4 variables.

3 Results and Discussions

It was chosen the following list of 8 possible sizes of nxn matrix used for the discretization: 1, 2, 3, 4, 10, 25, 75, 100, 1000.

It is **possible** to increase the size of this list and also the number of iterations using parallelim from the bash script. Therefore, a hardware with multiple cores would perform better.

According to the following pictures 1,2, when $n = 2, 4$, we can see that although the run-time is very low, the relative error is either infinite or equal to 1. Therefore, we can discard matrix with size $2x2$, and $4x4$ and **statistically** conclude that this algorithm have **bad performance** with too small n. This fact was also verified when $n = 1, 3$.

Consider now $n \geq 10$. In this experiment, it is clear that the variables bandwidth and computer rate is 100% linearly correlated as n grows. In fact, table 1 shows the Pearson correlation matrix when n=1000, which can be considered a large n. We can also check in pictures 3,4,6,7 and 8 that as n grows, bandwidth and computer rate are converging to a linear relationship (directly proportional) with m. Hence, we can disregard, for example, bandwidth for the predictive analysis of the best performance. Note that m is also highly correlated with run-time (96.93%) and inversely correlated with relative error (−95.45%)

Moreover, as n and m grow, pictures 3,4,6,7 show that the relative error converges to some constant value (in red). However, picture 8, which have the highest n, shows that its convergency

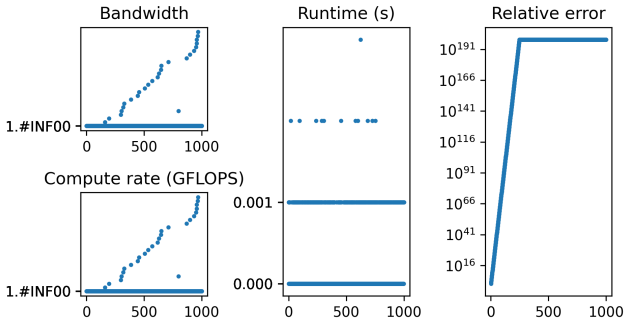

Figure 1. Maximum number of interactions in a grid $2x2$

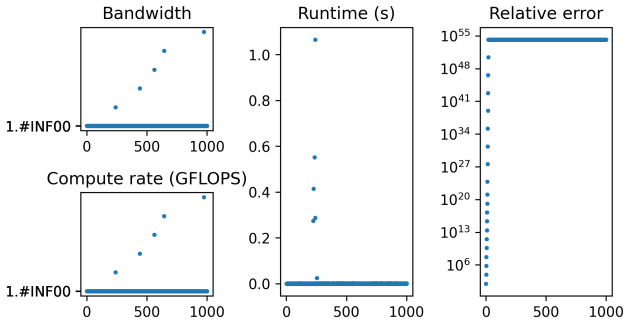

Figure 2. Maximum number of interactions in a grid $4x4$

will probably happen for higher m. It is also possible to see that the run time is linearly increasing (the line angle is getting higher). In fact, it is possible to fit a regression line in each "Runtime picture".

Table 1
Pearson Correlation Matrix when n = 1000

Bandwidth GB/s	**Compute rate** GFlops	**Relative error**	**Runtime** s	**m**
1.000000	1.000000	-0.041016	-0.087348	0.099277
1.000000	1.000000	-0.041016	-0.087348	0.099277
-0.041016	-0.041016	1.000000	-0.954499	-0.977684
-0.087348	-0.087348	-0.954499	1.000000	0.969350
0.099277	0.099277	-0.977684	0.969350	1.000000

4 Conclusion

Therefore, according to this analysis, it is possible to conclude that as n and m get higher:

- The relative error of the algorithm get smaller but it takes more time to converge.
- The runtime increases and it appears to have a linear relationship with m.

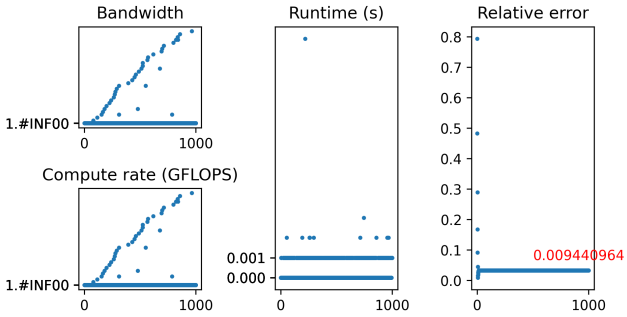

Figure 3. Maximum number of interactions in a grid $10x10$

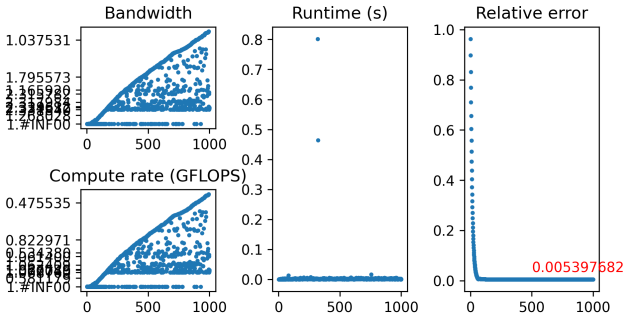

Figure 4. Maximum number of interactions in a grid $25x25$

- Bandwidth and Compute rate increases but it seems to have some upper bound. It would be good to check with more simulations.

Table 2 shows the minimum relative error for each n, and its respect m, compute rate, and run-time.

Table 2

n	m	Compute rate GFlops	Rel. error	Runtime s
10	8	inf	0.009441	0.0
25	155	0.403936	0.005398	0.002
50	158	0.963889	0.000016	0.004
75	931	1.222559	0.000586	0.05
100	389	1.289003	0.000004	0.037
1000	832	0.770096	0.18581	12.093

When $n = 10$, we get an infinite Compute rate, so we can reject this choice. However, a good choice it would be $n = 25$ and $m = 155$, because it has the smallest compute rate of this experiment and an acceptable relative error of .5%. Another good choice would be $n = 50$ and $m = 158$, which has a much smaller relative error of 0.0016%, but a higher compute rate. Note that in terms of energy performance, the first approach is better because it has a significant lower compute rate and therefore a lower bandwidth. It also have a smaller runtime which means a better energy performance.

References

[1] Ana. C. L. Malara. *Simulations.* `https://github.com/carollaurini/Sustainability-of-Scalable-AI`. 2025.

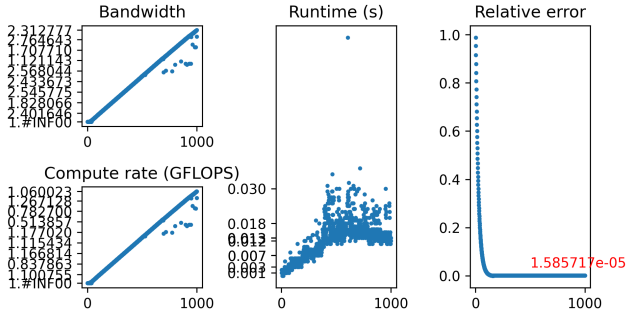

Figure 5. Maximum number of interactions in a grid $50x50$

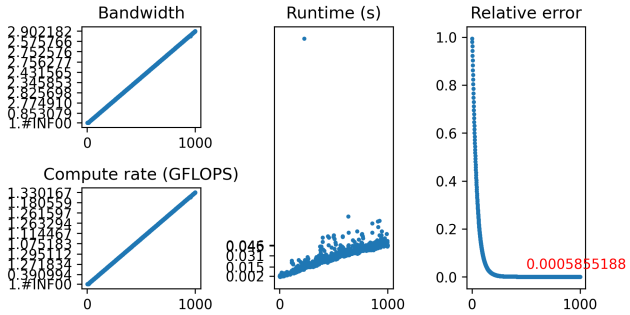

Figure 6. Maximum number of interactions in a grid $75x75$

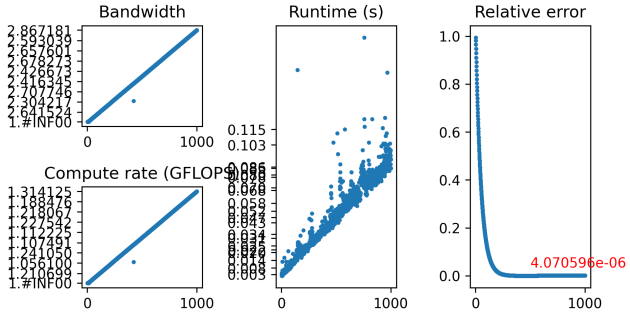

Figure 7. Maximum number of interactions in a grid $100x100$

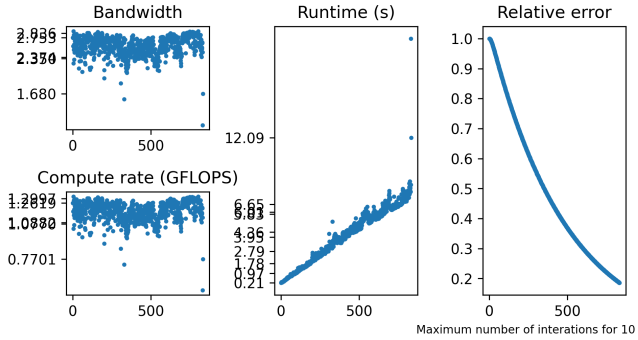

Figure 8. Maximum number of interactions in a grid $1000x1000$