UNIVERSITAT POLITÈCNICA DE CATALUNYA
FACULTAT D'INFORMÀTICA DE BARCELONA

Bachelor Degree in Informatics Engineering
Computer Engineering Specialization
Degree Final Project

Performance analysis and optimization of a combustion simulation

Final Degree Thesis

Author
GUILLEM RAMÍREZ MIRANDA

Director
MARTA GARCIA GASULLA

Co-director
DAVID VICENTE DORCA

Tutor
JULIAN DAVID MORILLO POZO

14th January 2021

Contents

List of Figures

List of Tables

Chapter 1

Background

This chapter introduces preliminary concepts needed to understand the work. We will take a look at:

- High Performance Computing (HPC) environment basics.
- Alya and the combustion use case.
- Application analysis methodology.

At first, we will understand how an HPC environment works, and we will contextualize it within the BSC MareNostrum4 infrastructure, which is the target environment for the application. Therefore, we also will work in the same domain. Then an introductory study of Alya and a hands-on is done to understand how to work with the application. Finally, we will discuss the application analysis methodology giving detail about the steps and the concepts behind.

1.1 HPC enviornment

For completeness concerning the work, I will explain the different concepts of an HPC environment, contextualized in the BSC Marenostrum4 infrastructure[1].

A supercomputer, also known as a cluster, is a set of machines interconnected, and usually coordinated to solve a problem. The machines are named nodes, and each has memory, CPU, storage, and is running an operating system. On the Marenostrum4 platform, each node consists of 2 physical Intel Xeon Platinum 8160 with 24 cores each and 96 GB of DDR4 RAM. There are also particular nodes with more memory available which are the same but with 384 GB of DDR4 RAM.

Each node can send and receive data from the other nodes from the system. Thus there is an interconnection network underlying. Networks are difficult to design as we expect low latency and high bandwidth on communication up to among 150.000 processes concurrently. The MareNostrum4 platform uses Intel Omni-Path technology for interconnection. It has six routers that are responsible for handling messages from a group of nodes, precisely one-sixth of them. Intel Omni-Path allows speeds up to 100Gbps.

To send data, applications, use the Message Passing Interface (MPI) standard. MPI allow programmers to divide the problem among the nodes in the machine and do communications and synchronizations via the interconnection network. HPC industry widely uses MPI, and therefore there are a lot of implementations. As we focus on the MareNostrum4 platform which integrates Intel technologies, we will use Intel MPI implementation and Intel compilers.

We have commented out that we will use different software and implementations. HPC environment use modules for managing the diverse software and implementations that are installed in the clusters. Modules provide the users with an easy way to select which software they want to use, managing the Linux environment variables. Listing 1.1 shows an example of how to load the Intel MPI implementation and the Intel compiler suite version 2017.4 and then change to the version 2018.4.

```
1 module load intel/2017.4 impi/2017.4
2 module unload intel/2017.4 impi/2017.4
3 module load intel/2018.4 impi/2018.4
```

Listing 1.1: Example of module usage.

Another essential element of an HPC environment is the Resource Manager. The Resource Manager establishes the access policies to the resources offered by the HPC infrastructure. In the context of the MareNostrum4 infrastructure, the resource manager used is Slurm[2]. Slurm has a queue system where the users send jobs, wait for their jobs to pass through the queue, and finally, Slurm allocates the resources demanded and runs the job. The priority a user has on the queue varies on the number of resources required, the time demanded with the resources and other factors. Listing 1.2 shows how to allocate a node for ten minutes, query the job identification of the job, and cancel the job.

```
1 salloc -N 1 --time=10:00
2 squeue #look for the JOBID
3 scancel JOBID
```

Listing 1.2: Example of slurm usage.

For a more detailed description of the commands and different options visit the Slurm documentation [3].

1.2 Alya and the combustion use case

Alya is an HPC application for computational mechanics. It has diverse modules that solve a specific problem, which can be coupled to solve a general problem. For example, we can couple together the module to simulate how the temperature evolves in a system and the module that computes the particle motion in the same run.

The code is written in Fortran90 and can use MPI, OpenMP/OmpSs and GPU. The module we are asked to optimize is pure MPI and despite the code is written in Fortran, it uses Cantera library for a critical computation inside the module. Cantera[4] is an open-source suite of tools for problems involving chemical kinetics, thermodynamics, and transport processes written in C++.

The use case we will study consists of a bunsen flame in 2D. It is a simple use case although representative of the industrial real use cases of the application. Figure 1.1 a real picture of a bunsen flame.

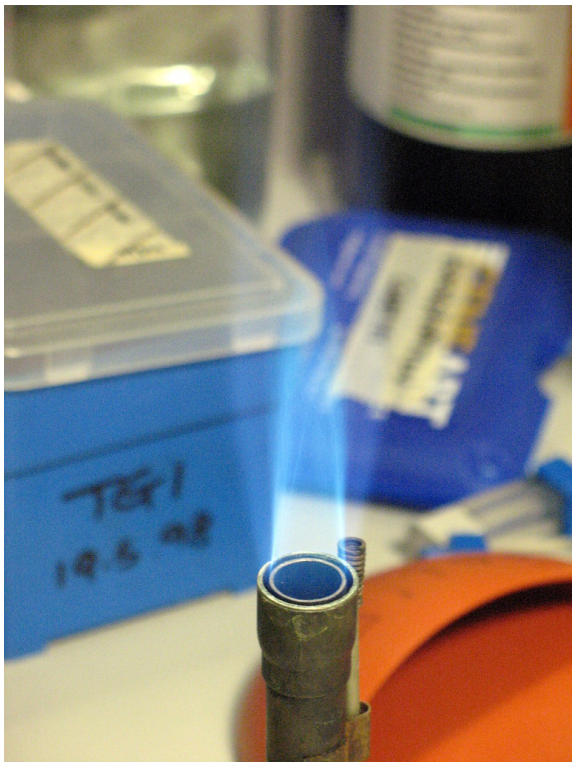

Figure 1.1: Bunsen flame by El Bingle licensed under CC BY-NC-SA 2.0

In general terms the simulation consists of:

- The centre of the 2D domain contains the bunsen flame.
- We compute environmental parameters such as temperature and pressure produced during the combustion for all the grid points.
- Following transport equations defined, compute the motion of chemical species, and if the environmental conditions are ideal, compute the chemical reactions.

Developers of the application gave us two versions of the bunsen 2D flame:Developers of the application gave us two versions of the bunsen 2D flame:

- **Detailed:** It has a significant number of transport equations, which means that the case is industry-relevant and the chemical part of the simulation, will be the most relevant part of the code.
- **Reduced:** Has a short number of transport equations, meaning the computational relevance of the chemical part of the code is low, and the case is less industry-relevant but interesting to compare to the Detailed version.

1.3 Analysis

As said in Section 3, we will follow the Performance Optimization and Productivity[5] methodology. The POP Efficiency metrics are the base of the methodology. First, we usually run the application many times in a *strong scaling* manner. Strong scaling means leaving the problem's size constant and increasing the resources, for example, running the same Alya input set with 1, 2, 4 and Nodes. Then we gather necessary data for computing the efficiency metrics from the previous runs. Once we have the metrics, we possess insight into the factors limiting the code's performance when scaling in the number of resources. From this insight, we can locate the zones in the code responsible for it and try to propose a better implementation.

1.3.1 Efficiency metrics

Pop efficiency metrics try to describe the sources of inefficiency. Figure 1 shows the representation of the metrics. The main feature of the metrics is that they are multiplicative. Multiplicative means that the multiplication of the children of a metric computes its value.

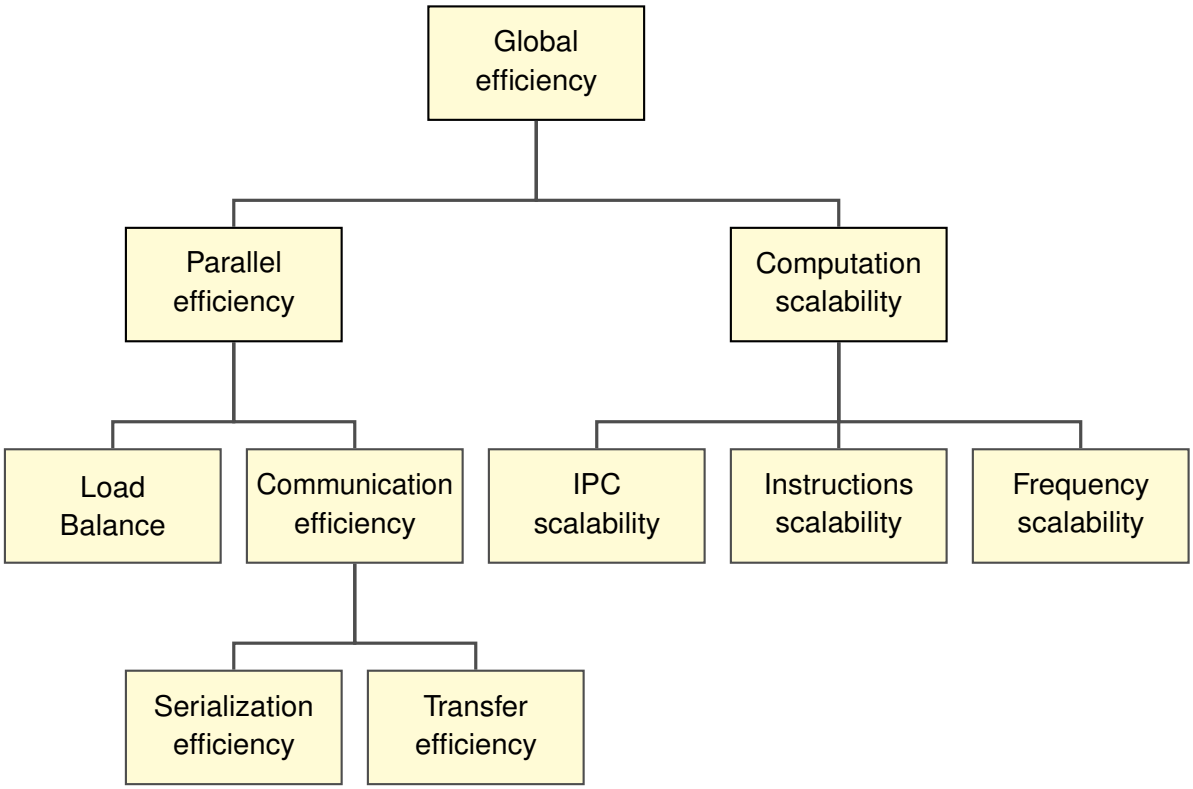

Figure 1.2: POP Efficiency metrics

The global efficiency describes how good is the parallelization in general. Then, the global efficiency is represented by:

- **Parallel efficiency:** Shows the efficiency lost due to splitting the work between processes and the effects of adding communications.
- **Computation scalability:** Represents the efficiency lost due to computation side effects of the parallelization of the job. For example, dividing the data among processes

adds more instructions, running with more processes in the same machine can lower the IPC because of the resource sharing.

Parallel efficiency

Before entering in-depth on the metrics, we need to explain a few assumptions. The efficiency metrics estimate that each process can be in two states:

- Useful state means that the process is performing computation. We usually represent a process in useful with the colour blue.
- Not useful state means that the process does not perform computation, such as sending an MPI message or waiting in a synchronization. We usually represent a process that is not useful with the colour red.

Figure 1.3 shows the graphical representation in a timeline of processes and their states. The x-axis represents the time, y-axis processes, and the colour of a process means the state that the process is in that time. The Figure shows four processes following the pattern: All computing, processes 3, 2 and 1 finish early and wait for process 0, then process 0 finishes and re-establish the computation.

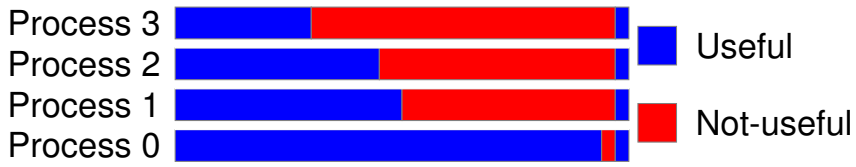

Figure 1.3: Example of useful and not useful states. Own compilation

We define E as the total execution time, $P = \{p_1, p_2, \ldots, p_n\}$ as the set of processes and for each process the set of time intervals where the process is in useful $U_p = \{u_1^p, u_2^p, \ldots, u_{|U|}^p\}$ and the same set but when the process is in not-useful $\overline{U}_p$. Equation 1.1 shows the definition of T_{U_p} which is the total time the process is in useful. Graphically expressed the sum of the blue chunks. The time the process is in not-useful $T_{\overline{U}_p}$ is defined similarly.

$$T_{U_p} = \sum_{U_p} \blacksquare = \sum_{i=1}^{|U_p|} u_i^p \tag{1.1}$$

The parallel efficiency (PE) represents the time lost due to the parallelization. Then we can express it as the factor of the total time in useful by the total CPU time consumed. As we said, as the metrics are multiplicative it can also be computed with the product of the Load Balance (LB) and the Communication Efficiency (CE). Equation 1.2 shows the formalization.

$$PE = \frac{\sum_{i=1}^{|P|} T_{U_i}}{E * |P|} = LB * CE \tag{1.2}$$

The Load Balance measures the efficiency lost due to improper work distribution. Expressed mathematically the factor between the average time in useful by the maximum

time in useful. Equation 1.3 shows the formalization.

$$LB = \frac{\sum_{i=1}^{|P|} T_{U_i}}{|P| * max_{i=1}^{|P|} T_{U_i}} \tag{1.3}$$

The Communication Efficiency measures efficiency lost due to spending time in not-useful state, namely, communications. Equation 1.4 defines the CE. Expressed in words, is the maximum time in useful, divided by the total runtime. Also the Communication Efficiency is the multiplication of it's children, the Serialization Efficiency (SE) and Transfer Efficiency (TE).

$$CE = \frac{max_{i=1}^{|P|} T_{U_i}}{E} = TE * SE \tag{1.4}$$

As we said, the metrics are multiplicative. Therefore we can check that $PE = LB * CE$. Equation 1.5 shows the check, we can see that the expression of efficiency lost by distributing improperly the work by the time lost in communications, expresses the Parallel Efficiency.

$$LB * CE = \frac{\sum_{i=1}^{|P|} T_{U_i}}{|P| * max_{i=1}^{|P|} T_{U_i}} * \frac{max_{i=1}^{|P|} T_{U_i}}{E} = \frac{\sum_{i=1}^{|P|} T_{U_i}}{E * |P|} = PE \tag{1.5}$$

As said, the Communication Efficiency can be expressed in terms of the Transfer Efficiency and the Serialization Efficiency.

The Serialization Efficiency measures time lost in communications where there is not data to send or receive. Therefore, for calculating the metric we need to simulate an ideal network so no time because of transferring data is taken into account. Then the Serialization efficiency is the maximum computation time on an ideal network divided by the total runtime on an ideal network. Equation 1.6 shows the expression where the suffix ideal means the time is on an ideal network.

$$SE = \frac{max_{i=1}^{|P|}(T_{U ideal_i})}{E_{ideal}} \tag{1.6}$$

The Transfer Efficiency measure time lost in not-useful due to data transfer. This can be computed with the factor of the runtime on ideal network by the runtime. Equation 1.7 shows the mathematical expression.

$$TE = \frac{E_{ideal}}{E} \tag{1.7}$$

As done with the parallel efficiency, we can do the same check with the communication efficiency. Equation 1.8 shows the check. We can see that the communication efficiency is represented by time lost due to the network not being ideal, which may seem it is not a thing we can not improve, but packing communications can solve it. And also time lost in not-useful without sending/receiving data.

$$SE * TE = \frac{max_{i=1}^{|P|}(T_{U ideal_i})}{E_{ideal}} * \frac{E_{ideal}}{E} = \frac{max_{i=1}^{|P|} T_{U_i}}{E} = CE \tag{1.8}$$

Computation scalability

In the previous Section 1.3.1 we have been discussing about efficiency. In this section we present the Computation Scalability, which this metric and its children refer to scalability, not efficiency.

This difference is important because the numbers depend on the baseline case. The Computation Scalability in strong scaling is defined as the ratio of the sum of all the useful computation time by the number of processes divided by the baseline ratio.

The factors that affect directly the computation scalability are:

- **Instruction scalability:** Measures if the number of instructions scales with the number of processors, if we double the number of processors, the ideal number of instructions executed should be the double. More instructions could mean that we are repeating work, or adding overhead from the parallelization.
- **IPC scalability:** Measures how the IPC evolves from the baseline, when we increase the number of processors on a run we can expect that the IPC is lower since the resources are shared with more processors.
- **Frequency scalability:** Measures how the frequency evolves from the baseline, frequency of the processors should remain stable but pre-emptions and other side effects from running the application can affect the frequency and lower the performance.

1.3.2 Computing the efficiency metrics.

Now that we understand the efficiency metrics that explain the performance of the code, we need to know how to apply it in practice. All the data we need for computing the efficiency metrics is:

- Elapsed time.
- Maximum time in useful.
- Total time in useful.
- Elapsed time running in an ideal network.
- Average IPC in useful.
- Number of instructions executed in useful.
- Average frequency in useful.

Fortunately the Barcelona Supercomputing Center has a set of tools that allow us to gather the data, automatize the extraction and calculating the Efficiency Metrics. The process consists of:

1. Run the application with different number of processes with Extrae enabled.
2. Select a focus of analysis, this consists to the relevant part of the code that we want to study. Usually the program initialization and finalization is not taken into account, and if the application uses an iterative patter, we usually choose to focus on a set of iterations that represents the average iteration to study.

3. Once we have the focus of analysis and the traces, we have to chop the traces so they just have the data of the focus of analysis.

4. Finally we pass the chopped traces to the BasicAnalysis BSC tool which automatically extracts the data (including simulating the run with an ideal network using Dimemas) and computes the metrics.

The next sections bellow include a basic background on how to use the BSC tools for extracting the data for the Efficiency Metrics.

Extrae

Extrae[6] is a set of tools concerned to generate traces of executions for post-mortem analysis. There exist multiple ways for Extrae to intercept the program execution and get information. As we will be working with MPI we will use the `LD_PRELOAD` method with intercepts the MPI calls and dumps the information.

For using this method we need:

- A valid `extrae.xml` file including the different tracing options we want.
- Choose the correct dynamic library to preload according to the programming model we are using. In our use-case we will use the pure MPI dynamic library.

Listing 1.3 shows the integration of extrae with SLURM running an application `"BINARY"`. We add the call to `mpi2prv` tool because we disable automatic merging in the `extrae.xml` configuration file because we detected sometimes the tracing is stalled with the option enabled.

```
1  #!/bin/bash
2  #SBATCH --job-name=alya-trace
3  #SBATCH --output=alya.out
4  #SBATCH --ntasks=48
5  #SBATCH --cpus-per-task=1
6  #SBATCH --time=1:20:00
7  #SBATCH --exclusive
8
9  export EXTRAE_HOME=#path to extrae installation
10 export EXTRAE_CONFIG_FILE=#path to extrae.xml
11
12 EXEC="env LD_PRELOAD=${EXTRAE_HOME}/lib/libmpitracef.so BINARY"
13 mpirun -np ${SLURM_NTASKS} ${EXEC}
14 $EXTRAE_HOME/bin/mpi2prv -f TRACE.mpits -o Alya-trace.prv -no-keep-
       mpits
```

Listing 1.3: Running a binary with SLURM and Extrae.

Paraver

Paraver[7] is a visual inspection tool for analysis of parallel applications. The data the application needs is a trace in format `.prv` which are Extrae traces. This tool let us visually inspect the execution of applications. Therefore we can visually detect the pattern of the application, usually and in our use-case iterative and chop the trace having in the focus of analysis.

Dimemas

Dimemas[8] is a command-line tool that given an input trace of an execution allows to remake the trace changing some conditions, for example, simulating an ideal network or changing some of the computational resources conditions. We will not enter in depth on how to use Dimemas as it is automatically called by BasicAnalysis (Section 1.3.2).

BasicAnalysis

BasicAnalysis[9] is a suite of tools for, as the name says, performing a basic analysis. We will focus just on the `modelfactors.py` tool.

The `modelfactors.py` tool automatically extracts and computes all the data needed for computing the POP metrics. We just need our set of chopped traces for a set of processes P. Listing 1.4 shows how to run the tool inside a job script.

```
1 #!/bin/bash
2 #SBATCH --job-name=modelfactors
3 #SBATCH --output=modelfactors.out
4 #SBATCH --ntasks=1
5 #SBATCH --cpus-per-task=48
6 #SBATCH --time=1:20:00
7 #SBATCH --exclusive
8
9 modelfactors.py trace-48.chop1.prv trace-96.chop1.prv ...
```

Listing 1.4: Running modelfactors with SLURM.

Running it will output a table with the metrics computed and will output the following files:

- **modelfactors.csv**: The POP metrics in a csv format.
- **modelfactors.gp**: A gnuplot[1] script file that projects the efficiencies to a high number of processes. Figure 1.4 shows an example plot given by modelfactors tool.

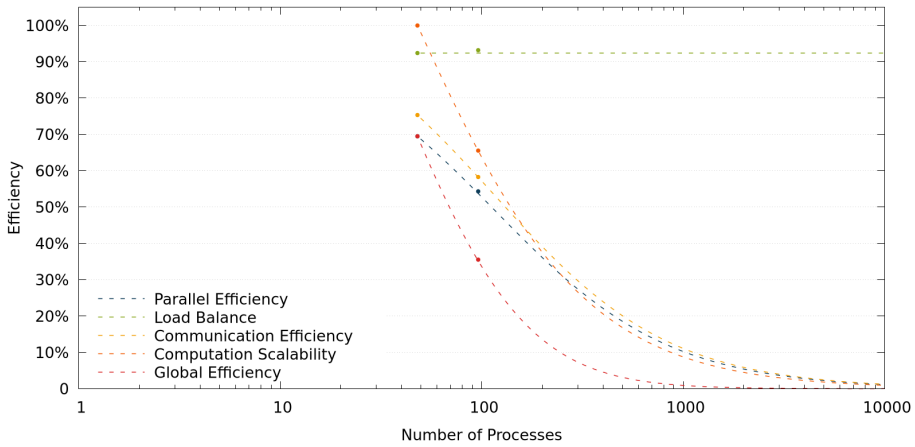

Figure 1.4: Modelfactors projected efficiencies.

[1] gnuplot site: http://www.gnuplot.info/

Bibliography

[1] "Marenostrum4 user's guide," Barcelona Supercomputing Center, accesed on 26/12/2020. [Online]. Available: http://www.bsc.es/support/MareNostrum4-ug.pdf

[2] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 2003, pp. 44–60.

[3] "Slurm documentation," Slurm, accesed on 23/12/2020. [Online]. Available: https://slurm.schedmd.com/documentation.html

[4] D. G. Goodwin, R. L. Speth, H. K. Moffat, and B. W. Weber, "Cantera: An object-oriented software toolkit for chemical kinetics, thermodynamics, and transport processes," https://www.cantera.org, 2018, version 2.4.0.

[5] M. Wagner, S. Mohr, J. Giménez, and J. Labarta, "A structured approach to performance analysis," in *International Workshop on Parallel Tools for High Performance Computing*. Springer, 2017, pp. 1–15.

[6] H. Servat, G. Llort, K. Huck, J. Giménez, and J. Labarta, "Framework for a productive performance optimization," *Parallel Computing*, vol. 39, no. 8, pp. 336–353, 2013.

[7] V. Pillet, J. Labarta, T. Cortes, and S. Girona, "Paraver: A tool to visualize and analyze parallel code," in *Proceedings of WoTUG-18: transputer and occam developments*, vol. 44, no. 1. Citeseer, 1995, pp. 17–31.

[8] S. Girona, J. Labarta, and R. M. Badia, "Validation of dimemas communication model for mpi collective operations," in *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*. Springer, 2000, pp. 39–46.

[9] M. Casas, R. Badia, and J. Labarta, "Automatic analysis of speedup of mpi applications," in *Proceedings of the 22nd annual international conference on Supercomputing*, 2008, pp. 349–358.