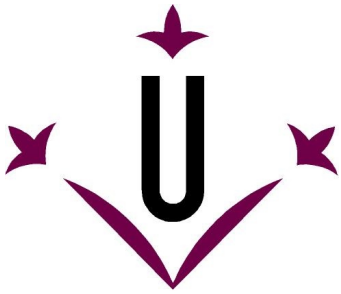

University of Lleida

Master's Degree in Informatics Engineering

Higher Polythecnic School

Parallel Benchmarking

High Performance Computing

Francesc Contreras

Albert Pérez

March 28, 2021

Table of contents

List of Tables

1 Introduction

This work contains the description and the analysis of the results obtained with the execution of two different benchmarks created by the NASA Advanced Supercomputing (NAS) Division [1], when ran against the UDL Moore Cluster[2], which is the educational cluster of the Polytechnic School[3].

The main aim of this practice was to learn to execute a parallel benchmark (application) written in the both most extended used parallel languages, the Message Passing Interface (MPI) and the Open Multi-Processing (OpenMP) Interface. And thus compare and analyze their performance results, as well as with respect to the Serial execution.

Next, the technologies used will be presented as well as their organizations if any, then the benchmarks chosen and finally the benchmarks' results obtained with the analysis and conclusions extracted.

[1]NAS Division: `https://www.nas.nasa.gov/about/about.html`
[2]University of Lleida Cluster Moore: `http://moore.udl.net/wordpress/`
[3]University of Lleida Polythechnic School: `http://www.udl.cat/ca/en/faculties/polytechnic_school`

2 NASA Advanced Supercomputing Division

The NASA Advanced Supercomputing (NAS) Division is located at NASA Ames Research Center[4], Moffett Field in the heart of Silicon Valley in Mountain View, California.

It has been the major supercomputing and modeling and simulation resource for NASA missions in aerodynamics, space exploration, studies in weather patterns and ocean currents, and space shuttle and aircraft design and development for over thirty years.

The NAS Division is enabling advances in high-end computing technologies and in modeling and simulation methods to tackle some of the toughest science and engineering challenges facing NASA today.

2.1 NAS Parallel Benchmarks

The NAS Parallel Benchmarks (NPB)[5] are a small set of parallel programs designed and developed with the aim of evaluate the performance of highly parallel supercomputers and even have also been widely used in distributed environments such as Cluster systems.

The benchmarks are derived from computational fluid dynamics (CFD) applications and consist of five kernels and three pseudo-applications in the original "pencil-and-paper" specification (NPB 1).

Moreover, it must be said the use of these benchmarks in the performance evaluation of the different co-scheduling models is certainly a validation guarantee.

About the benchmark suite, it has been extended to include new benchmarks for unstructured adaptive meshes, parallel I/O, multi-zone applications, and computational grids.

Problem sizes in NPB are predefined and indicated as different classes. And reference implementations of NPB are available in commonly-used programming models like MPI and OpenMP (NPB 2 and NPB 3).

3 Cluster Moore

The cluster used was provided by Universitat de Lleida (UdL), and is remotely accessible, for instance, with the connection using the **S**ecure **SH**ell at the server "moore.udl.cat".

The cluster Moore is an homogeneous Cluster with a total of 32 processing units and 32 GB of memory, distributed over 8 nodes with an Intel Core i5 processors with 4 cores at 3.1 Ghz and 4GB of main memory.

In order to avoid memory issues and never-ending jobs, all jobs are launched through **Sun Grid Engine (SGE) queue system** and are limited to 1GB of main memory and a maximum of 3 hours of execution time.

[4]NASA Ames Research Cen-ter: `https://www.nasa.gov/ames`
[5]NAS Parallel Benchmarks: `https://www.nas.nasa.gov/publications/npb.html`

4 Chosen Benchmarks

This section will describe the chosen benchmarks in order to present and better understand them, but also the NPB defined problems executed for each.

First of all, the ones chosen are included into the well-called NAS Kernel Benchmarks, but they weren't the first option because of we got some problems.

As we first wanted to test the Unstructured Adaptive mesh, dynamic and irregular memory access (UA) benchmark, but realized there are some problem classes not compatible and thus not being able to compare all tests properly decided to bet for another alternative that seems just as interesting.

So, the following were the chosen:

Short Descriptions

- **Embarrassingly Parralel (EP):** generate independent Gaussian random variates using the Marsaglia polar method.
- **Conjugate Gradient (CG):** estimate the smallest eigenvalue of a large sparse symmetric positive-definite matrix using the inverse iteration with the conjugate gradient method as a subroutine for solving systems of linear equations.

We considered selecting EP in the first place, because we can see what really happened, as there is little or no dependency and no need for communication between the parallel tasks, or results between them.

On the other hand, there is CG, because it solves an unstructured sparse linear system by the conjugate gradient method, characterized by an irregular memory access and communication.

So basically, we selected these two benchmarks as they provide us a big difference about the method used and also a different levels of communication ranges. That's the main reason why we bet to apply these ones, but also because they are definitely interesting.

Moreover it is worth mentioning that both of them can be run on multiple problem NPB classes, which differ from each other mainly in the sizes of the principal used data structures called array, so the complexity as well.

To conclude, each benchmark was executed using the class **S**, for small quick test purposes, and the classes **A** and **C**, which correspond to standard test problems being A lower complex than C which is 16X size bigger.

4.1 Embarrassingly Parallel

The NAS Embarrassingly Parallel (EP) benchmark estimates the upper achievable limits of floating-point performance without significant inter-processor communication. That means is an entirely CPU bound benchmark.

The benchmark inner problem is based on generate pairs of Gaussian random deviates according to a specific scheme[6] and tabulate the number of pairs in successive square annuli.

The conception form of an EP problem is that its input is easy to parallelize and if you are not able to find a way "it embarrasses you". So it's basically a problem where the input can be divided into small subsets (usually into 'n' subsets of the same size, 'n' as the number of computational nodes) and processed individually without communication between the processing units, and with each individual result, calculate the final result.

4.2 Conjugate Gradient

The NAS Conjugate Gradient (CG) benchmark is often used to evaluate machine performance and compare characteristics of different programming models.

It solves an unstructured sparse linear system by the conjugate gradient method.

In more detailed words, this benchmark uses the inverse power method to find an estimate of the largest eigenvalue of a symmetric positive definite sparse matrix with a random pattern of non-zeros.

In addition, the CG benchmark is quite memory intensive; it tests irregular long distance communication and employs unstructured matrix vector multiplications, which are major matrix multiplication bottlenecks for paralleled scientific computation.

[6] EP Scheme: Section 2.3 of `https://www.nas.nasa.gov/assets/pdf/techreports/1994/rnr-94-007.pdf`

5 Benchmarking results

According to Intel Speedup is a metric to determine how much faster parallel execution is versus serial execution, while efficiency indicates how well software utilizes the computational resources of the system.

This section exhibits the results found with the execution of each benchmark and each programming model, as well as the calculated Speedups and Efficiencies. Each of those metrics is displayed in one of the following tables.

Benchmark	Class	Serial	OMP		MPI		
			2 Threads	4 Threads	2 Proces.	4 Proces.	8 Proces.
Embarrassingly Parallel	S	0.96	0.50	0.26	0.49	0.26	0.15
	A	15.47	7.88	4.07	7.88	4.09	2.06
	C	247.24	125.83	64.90	126.08	64.98	32.56
Conjugate Gradient	S	0.04	0.02	0.02	0.02	0.01	0.1
	A	1.14	0.60	0.50	0.59	0.59	0.57
	C	211.52	109.39	73.50	117.17	77.82	55.13

Table 1: Benchmarks' Time in seconds

Benchmark	Class	OMP		MPI		
		2 Threads	4 Threads	2 Proces.	4 Proces.	8 Proces.
Embarrassingly Parallel	S	1.92	3.69	1.96	3.69	6.4
	A	1.96	3.87	1.96	3.78	7.51
	C	1.97	3.81	1.96	3.80	7.59
Conjugate Gradient	S	2	2	2	4	0.4
	A	1.9	2.28	1.932	1.93	2
	C	1.93	2.87	1.805	2.718	3.836

Table 2: Benchmarks' Speedup by number of threads/processes

Benchmark	Class	OMP		MPI		
		2 Threads	4 Threads	2 Proces.	4 Proces.	8 Proces.
Embarrassingly Parallel	S	0.96	0.92315	0.9796	0.92307	0.8
	A	0.9816	0.966825	0.9815	0.945575	0.9387
	C	0.9824	0.9524	0.9804	0.9512	0.9491
Conjugate Gradient	S	1	0.5	1	1	0.05
	A	0.95	0.57	0.96	0.48	0.25
	C	0.965	0.72	0.9025	0.68	0.48

Table 3: Benchmarks' Efficiency

6 Analysis of the benchmarking results in relation to the characteristics of the benchmarks

In order to better comprehend the acquired results, We created the below table that contains the averages Speedup, Efficiency, and Efficiency in percentage, by the level of parallelization.

Benchmark	Measure	OMP		MPI		
		2 Threads	4 Threads	2 Proces.	4 Proces.	8 Proces.
Embarrassingly Parallel	Speedup	1.95	3.79	1.96	3.76	7.17
	Efficency	0.97	0.94	0.98	0.94	0.89
	Efficiency(%)	3	6	2	6	11
Conjugate Gradient	Speedup	1.94	2.32	1.912	2.88	2.08
	Efficiency	0.97	0.6	0.95	0.72	0.26
	Efficiency(%)	3	40	5	28	74

Table 4: Benchmarking results comparison

The results obtained in the EP benchmarks have generally been excellent. Of these we have to highlight the following points:

- The higher the number of threads/processes, the faster the problem is solved.
- It maintains a linear progression and a high level of parallelization, thus making execution faster.
- The indices of parallelization is great in all the cases studied.
- The best efficiency is found when it is executed with 2 threads/proces. independently of the size of the problem and if it is OMP or MPI.

Apart from that, the EP benchmark also demonstrated a very high level of efficiency, this means that, on average, throughout the execution, each of the cores is idle about only 5,6% of the time.

Regarding the results obtained with Benchmark CG, we can observe that it works well when the size of the problem is little and the parallelization hardly exist. As we can observe in the different tables, it has a terrible parallelization index and efficiency when we want to work with more than 2 threads/proces. So, this makes an inefficiency method when we want to work with problems that depend on parallelization.

7 Conclusions

This task allowed us to perform some benchmark testing in different parallel algorithms, enabling us to assimilate more efficiently the concepts learned in class, and also to be initiate into the cluster environment used.

The execution and comparison between the Embarrassingly Parallel and Conjugate Gradient benchmarks in the three types of computation, Serial, MPI and OMP proved to us that is important to know well the input data characteristics as well as the type of parallelization in order to successfully and efficiently resolve the different challenges that we face in our profession.