Concurrency and Parallelism
Cilk$^+$ Parallel Patterns Implementation

André Rosa
48043
af.rosa@campus.fct.unl.pt

João Geraldo
49543
j.geraldo@campus.fct.unl.pt

Rúben Silva
47134
rfc.silva@campus.fct.unl.pt

***Abstract*—In recent years, the parallel computation paradigm has been emerging, as a consequence to: the huge growth on the amount of data that needs to be processed and analyzed, and to the switching of processors' architecture evolution process from becoming smaller and having faster clock speeds, to integrate parallel functionalities, such as multiple cores, hardware threads and vector operations. Therefore, it is imperative to build algorithms that explore these functionalities to increase the efficiency on processing such huge amounts of data. However, building algorithms on top of these functionalities can sometimes be a difficult and complex task, due to them being low level primitives and sometimes platform dependent. Thus, parallel functions and libraries can provide an easy and generic way to better utilize these resources, when available. Therefore, we developed a library, that implements some of the most well known parallel patterns, and that can be easily integrated in any already existing sequential program, to improve its efficiency. We developed each algorithm to try to achieve the maximum possible parallel slack and ...**

In order to be able to use it in a vast amount of use cases, these implementations are independent both from the data types they are manipulating and the parallel functionalities provided by each specific hardware platform.

What was our approach? What were the results? What did you learn?

Finally, we conducted a preliminary experimental evaluation on the performance of the different implemented alternatives, comparing them with their corresponding sequential version, that showed ... .

***Index Terms*—Parallel Algorithms, Cilk$^+$**

1. Introduction

Nowadays, the parallel paradigm is a matter of great importance, and the demand for more scalabel and efficient data processing algorithms has increased. [dizer um pouco mais]

In the one hand, the amount of data produced every day grows exponentially [meter uma citação], requiring more computational power to process it at the same speeds.

On the other hand, due to the limitations of processors designed to run serial code such as [Meter links para a power wall, ILP wall e memory wall penso nao ser preciso explicar o que são se tiver links] a revolution happened in the processor's architecture design. Architectures switched to have more parallel functionalities, such as multiple cores and hardware threads as well as vector operations, that allow the simultaneously execution of processes and parallel manipulation of multiple data.

For these reasons, it's crucial to take advantage of these features to build algorithms that allow the processing of such huge amounts of data, having good levels of scalability and latency. However, build such algorithms can be a very difficult and complex task, not only because they have to make use of the previous low level parallel primitives, but also because those primitives can be platform dependent and are highly heterogeneous, e.g., not all the processors provide vector operations and hardware threads and the amount of cores may vary greatly, even within the same manufacture, take Intel as an example - Intel's i7 lineup has chips with only 2 cores up to chips with a whopping 16 cores.

On that account, we designed and implemented a library that contains some of the most well known parallel patterns. [meter citação anteriror] This library provides an easy way to update already-built sequential programs to make use of the hardware parallel functionalities, allowing the application-level developer to integrate such patterns and give its application a boost of performance with little effort. We resorted to Intel's Cilk$^+$ to implement those patterns. [meter citação para os dreds] In order to be able to use it in a vast amount of use cases, these implementations are independent both from the data types they are manipulating and the parallel functionalities provided by each specific hardware platform. We developed each algorithm to try to achieve the maximum possible parallel slack and ... [mudar um bocado isto]

2. Architecture

Our library has the following patterns implemented: Map, Reduce, Scan, Pack, Split, Gather, Scatter, Pipeline and Farm. [citação para os patterns]

2.1. Map

O que dizer aqui?

2.2. Reduce

Explicar brevemente o que é We implemented 2 versions of reduce: (regular) reduce and tiled reduce. Explicar a difenreça Explicar que quisemos aumentar oa máximo parallel slack e por isso quando o array de ouptu é grande este é continuamente dividido na versaõ tiled até ter um tamanho minimo. Formando uma árvore n-ária em vez de binária. Explicar a técnica dos dois arrays para fazer a árvore de execução mas sem a armazenar toda em memória e poupar espaço sem ter apontadores para os pais e filhos, etc.

2.3. Scan

2.4. Pack

2.5. Split

Função extra(secalhar dizer isto lá em cima)

2.6. Gather

2.7. Scatter

2.8. Pipeline

3. Implementation

4. Experimental Evaluation

In this Section we present an experimental evaluation of the implemented parallel algorithms, comparing their performance against their sequential version.

4.1. Experimental Setting

To evaluate the performance of our algorithms, we built a tester application that, for each algorithm, runs each of its versions (parallel, sequential and alternative, when there is one) one after the other a parameterizable amount of times, in order to compute the average of their results and thus filter the noise introduced by the ... . For each algorithm we measured its latency for multiple amount of jobs and its latency for a fixed amount of jobs, with variable computation work. In order to count the elapsed time between the start and finish of the algorithm, we resort to a system call [citar] that returns the amount of the process CPU time, and thus not counting the time when the process is interrupted by the OS's process scheduler, giving more accurate results for the latency. The units utilized are microseconds (us). We run the experiments on node9, a computer with 16 cores and .... .

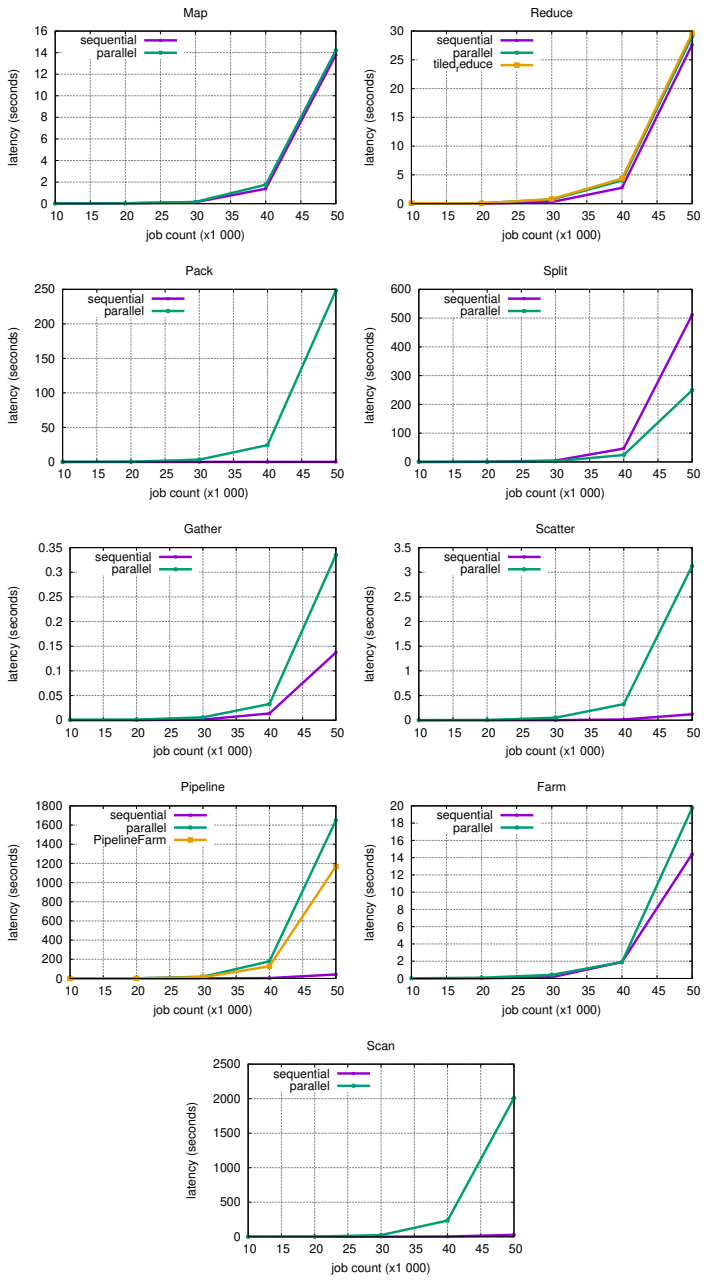

Figure 1. Latency of the algorithms.

4.2. Experimental Results

The Figure 1 has the results obtained, for each algorithm, on the experiments.

Análise comparatória da performance dizendo porque correu bem e porque correu mal para cada um. Análise geral, comparando a performance dos vaíos uns cons os outros. Ex o map é o que aprenseta melhores resultados comparativamente à versão sequencial, bla bla Podiamos medir a difenreça entre a paralela e a sequencial dividdo e assim tinhamos um valor que ea comparável com os outro salgoritmos

5. Conclusion

The conclusion goes here.

Acknowledgments

The authors would like to thank...
[1]

Comments

References

[1] J. Leitão, P. Ákos Costa, M. Cecília Gomes, and N. Preguiça, "Towards Enabling Novel Edge-Enabled Applications," *ArXiv e-prints*, May 2018.