Org Mode Template for an IEEE Conference Article

CONTENTS

***Abstract*—Reproducible research has become increasingly important, just like parallel architectures so we propose a novel experimental study of a parallel implementation of the quicksort algorithm that builds on reproducible research technology.**

INTRODUCTION

With the advent of parallel architecture, it is tempting to propose parallel implementations of classical operations. In this wonderful article, we report the performance gain of the well known quicksort algorithm, whose parallelization seems natural.

CONTEXT

Quicksort (sometimes called partition-exchange sort) is an efficient sorting algorithm, serving as a systematic method for placing the elements of an array in order. Developed by Tony Hoare in 1959 with his work published in 1961 [1], it is still a commonly used algorithm for sorting. Quicksort is a divide and conquer algorithm. Quicksort first divides a large array into two smaller sub-arrays: the low elements and the high elements. Quicksort can then recursively sort the sub-arrays. We propose to sort the sub-arrays in parallel on two threads and to bound the recursion level.

RELATED WORK

Well, this is such a brilliant and novel idea that none has worked on this cutting-edge topic. Parallel architecture barely existed in 1959, when Hoare proposed the sequential version of this algorithm [1]. Some colleagues mentioned some obscure parallelization of the partitioning phase but it is too complex to implement and certainly too costly to provide any performance gain.

METHODOLOGY

We used a Dell Latitude 6430u with 16Gb of RAM running a Debian with Linux 3.14.15. The CPU is an Intel(R) Core(TM) i7-3687U CPU @ 2.10GHz comprising two physical cores and hyperthreading. The `performance` frequency governor was used. We used FCC 5.3.1 with the following compilation flags: `-g -Wall -O2 -pthread -lrt`. Since we care a lot about reproducibility of our research, all the sources and detailed information regarding our experiments are provided on Github [1].

We used the wonderful Org-mode format [2] to both document our experiments and create a replicable article that can be obtained on Github as well. We could have used the Ipython approach [3] but we dishonestly ruled it out because we wanted to be in full control of our C experimental setup and did not know how to properly use it to write a replicable article.

[1]Or figshare, or zenodo, or whatever platform you prefer!

Finally, it is important to explain that each measurement was repeated five times and the set of experiments was absolutely not randomized, which may compromise the validity of our observations and conclusions.

EXPERIMENTAL RESULTS

As we can see in Figure 1, to benefit from the parallel version of our quicksort implementation, significantly large arrays (over a million entry) are required. Of course, it is difficult to conclude from so few measurements and a better experiment design would be useful.

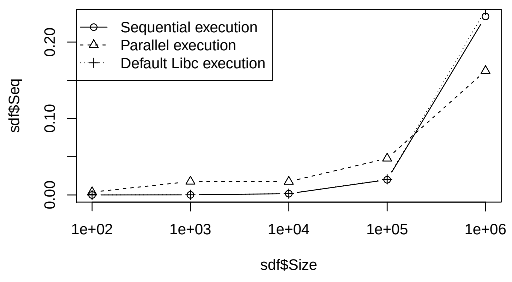

Figure 1. Comparing performances of several implementation of the quicksort algorithm

CONCLUSION

Exploiting parallel machine can be quite difficult and tedious. From our experience, such architecture are useful only when processing sufficiently large data sets. As a future work, we intend to consolidate our study with more experiments.

ACKNOWLEDGMENTS

This work is partially supported by the FOO and BAR projects. Experiments presented in this paper were carried out using our own experimental testbed with support of our university.

REFERENCES

[1] C. A. R. Hoare, "Algorithm 64: Quicksort," *Commun. ACM*, vol. 4, no. 7, pp. 321–, Jul. 1961. [Online]. Available: http://doi.acm.org/10.1145/366622.366644

[2] E. Schulte and D. Davison, "Active document with org-mode," *Computing in Science & Engineering*, vol. 13, no. 3, pp. 66–73, May/June 2011.

[3] F. Pérez and B. E. Granger, "IPython: a system for interactive scientific computing," *Computing in Science and Engineering*, vol. 9, no. 3, pp. 21–29, May 2007. [Online]. Available: http://ipython.org