On The Implications of Language Constructs for Concurrent Execution in the Energy Efficiency of Multicore Applications

Gustavo Pinto
Informatics Center, Federal University of Pernambuco
Recife, PE, Brazil
ghlp@cin.ufpe.br

Fernando Castor
Informatics Center, Federal University of Pernambuco
Recife, PE, Brazil
castor@cin.ufpe.br

Abstract

Our study analyzed the performance and energy consumption of multicore applications, using a number of techniques to manage concurrent execution. We concluded that language constructs for concurrent execution can impact energy consumption. Nonetheless, the tradeoff between performance and energy consumption in multicore applications is not as obvious as it seems.

Categories and Subject Descriptors D.1.3 [*Programming Techniques*]: Concurrent Programming, Parallel programming

Keywords Energy-Efficiency, Language Constructs, Concurrent Execution

1. Introduction

Measuring the energy consumption of an application and understanding where the energy usage lies provides new opportunities for energy savings. In order to understand the complexities of this approach, we specifically look at multithreaded applications. The performance of the existing constructs for concurrent execution is reasonably well-understood [3, 5]. Furthermore, since parallel programming enables programmers to run their applications faster, a common belief is that this application will also consume less energy [2]. We call this as the "Race to idle" philosophy. In summary: faster programs will theoretically consume less energy because they will have the machine idle fast.

This paper presents an empirical study consisting of the evaluation of the performance and energy consumption of four applications that use three concurrent constructs plus a sequential implementation with the goal of demonstrating that is hard to trace tradeoffs between energy-efficiency and performance. By an evaluation of these applications in multiple environments, we show how basic parameters, such as clock frequency, threading options, and different VMs can impact energy consumption. Our study highlights the non-obvious and context-dependent nature of tradeoffs between performance and energy consumption.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

SPLASH '13, October 26–31, 2013, Indianapolis, Indiana, USA.
Copyright is held by the owner/author(s).
ACM 978-1-4503-1995-9/13/10.
http://dx.doi.org/10.1145/2508075.2514880

2. Study Setting

To achieve the goal of this study, we ran a set of benchmark applications from different domains in a number of different configurations while varying a group of attributes. We can divide these attributes in two groups: internal (programming language construct, number of threads in use and resource usage - CPU and/or IO) and external (the clock frequency and the JVM implementation). We then provide a set of variant implementations for each benchmark using four concurrent constructs, plus a sequential implementation[1].

In this experiment, we used commodity hardware: an Intel(R) Xeon(R), 2.13 GHz, 4 cores/8 threads and with 16Gb of memory, running Linux 64-bit, kernel 3.0.0.-31-server. These experiments were run using three JVM: i) OpenJDK version 1.7.0_09, ii) HotSpot JDK version 1.7, and iii) JRockit version 1.6. When the experiments were performed using JRockit, we provided an external jar file containing the `ForkJoin` implementation.

Our experiments consisted of running these four applications in each of the three JVM implementations, scaling the CPU frequency, and limiting the number of threads in use. We ran each experiment twelve times for each workload, while measuring the system using the *powertop* utility[2]. We discarded the executions with the lowest and the highest to reduce bias caused by outliers. Therefore, we have two main metrics that evaluate the experiments: the energy consumed (in Joules) and the execution time (in seconds).

3. Study Results

Table 1 shows the overall view of our experimental results. The column named "Energy Consumption (J)" organizes the results of the 10 executions. The value contained in each cell represents the median of the 10 executions, where each sample represents the total of energy consumed in this given execution. We use **boldface** to highlight the best result for the given benchmark application. The column "Time (s)" works similarly.

As Table 1 shows, we can notice that the results of the concurrent constructs can have significant differences. For instance, the `Thread` results are always more inefficient in terms of both consumption and performance, when compared to `Executors`. Thus, since the usage of `Thread` and the `Executors` is very similar, with a little effort, a programmer could use `Executors`. Table 1 also shows that the same technique can have different impact on the energy consumed. If we take into consideration the `ForkJoin` variant, we have, for the N-Queens and Knucleotide benchmark applications, a highly efficient trend for both performance and energy. On the other hand, considering the LargestImage application, we notice that the same `ForkJoin` variant consumed more energy than

[1] The implementation details are available at http://bit.ly/parallel-construct

[2] https://01.org/powertop/

	Energy Consumption (J)			
	N-Queens	**LargestImage**	**Mandelbrot**	**Knucleotide**
Sequential	**732.8**	679.8	1978.1	5395.5
Thread	1023.7	766	1290.8	4808.2
Executors	984.2	**633.5**	**1287.5**	3281.2
Fork/Join	753.1	749.3	1292.4	**2993.4**
	Time (s)			
	N-Queens	**LargestImage**	**Mandelbrot**	**Knucleotide**
Sequential	85	78	71	106
Thread	44	42	41	68
Executors	38	**31**	41	62
Fork/Join	**27**	54	**35**	**55**

Table 1. The comparision between the language constructs in terms of energy consumption and time. The obtained values for Energy consumption and time are the medians of 10 executions.

every other variant except for the one using threads. Nonetheless, LargestImage is a benchmark application that made heavy use of IO operations, and it is well know that `ForkJoin` is not adequate for this kind of computation. This fact is a possible reason for the poor energy consumption.

Moreover, it is interesting to discuss about the `Sequential` variant of the N-Queens benchmark application, which presented the best energy consumption result, in spite of presenting the worst performance. This benchmark's result can vary according to the input data (in case, the NxN size of the matrix). In this experiments, we realised that, for a small matrix, the overhead caused by the thread creation led to an increase in energy consumption. But, for example, if we doubled the matrix size, the `ForkJoin` variant becomes the most energy efficient as well.

Furthermore, we analyzed how the benchmark applications scale with respect to the number of threads. Both the Mandelbrot and Knucleotide benchmark applications scale well. This means that: the more cores available, the faster the applications run, and more energy is saved. Nonetheless, for the other ones, it is not true. For instance, in the LargestImage benchmark, the more cores we have, more inefficient the `ForkJoin` variant is, in terms of both performance and energy. In summary, we collected a total of 128 samples (4 benchmarks x 4 variants x 4 nr. of threads x 2 clock frequencies), and for 36 of those, the variation which achieve the best performance were not the same that consumes less energy.

We then repeated the experiments varying the CPU frequency from 1.2 GHz to 2.13 GHz. We notice that even after reducing the clock frequencies, the results seems to be fairly similar to the latter one, in terms of the better technique remains the better. However, for the LargestImage benchmark, we found out that the lowest frequency consumes the same amount of energy as a middle clock frequency. It is interesting and acceptable, since it is an application that does not use a huge amount of CPU. Thus, for IO-bound applications, a low cpu frequency could reduce energy usage, without sacrificing performance.

Finally, we have also analyzed whether different JVM had different impacts on performance and energy consumption. We observed that, in general, results are very similar, specially for OpenJDK and HotSpot. It does not surprise us, since the HotSpot is the primary reference implementation of JVM, and OpenJDK is heavily inspired by them. On the other side, the JRockit JVM presents the worst case scenario, for all variants. For example, taking into consideration the `Thread` variant, the results increased in more than 10%. For the other benchmark applications, the JRockit also exhibited the worst results among the JVMs. Although the different JVMs did affect execution time and energy consumption, similarly to different clock frequencies, they did not significantly change the behavior of the variants, e.g. the fastest and slowest variants in one JVM were the fastest and lowest variants for all of them.

4. Related Work

To the best of our knowledge, only two studies have dealt with the topic of understanding the impact of concurrent constructs on the energy efficiency of applications [1, 4]. Gautham et al. [1] explore the following synchronization techniques to find an ideal solution for synchronization intensive workloads: i) spin lock ii) mutexes iii) software transactional memory. They show that Software Transactional Memory (STM) systems can perform better than locks for workloads where a significant portion of the execution time is spent in the critical sections. Trefethen [4] studies the behaviour of the NAS Benchmark suite for its energy and runtime performance. The benchmark suite considered includes I/O and compute-intensive applications. The authors concluded that there is a clear interaction between execution time and energy but this is not a simple relationship and can be affected by the computer environment and algorithmic approach used in the application. Nonetheless, none of these papers compare the energy-efficiency between techniques to manage concurrent execution.

5. Conclusion

This paper presented an empirical study targeting four benchmark applications using a number of concurrent programming constructs with the goal of finding interesting tradeoffs between energy-efficiency and performance. Our approach indicates that it is possible to switch from one technique to another in order to consume less energy. Nonetheless, we also conclude that it is very hard to identify which technique is the better for a given scenario. Moreover, our experiments show that factors such as the nature of the problem to be solved, the technique used to manage concurrent execution, the CPU clock frequency, and the JVM implementation can create variations. These results lead us to conclude that it is very difficult to generalize a relationship between performance and energy consumption for concurrent systems. For these systems, winning the race to idle does not imply more energy efficiency. In the future, we intent to address this problem by conducting a broader-scoped study. The results of this new study will provide input for us to derive a catalog of energy code smell for concurrent software. Then we plan to proceed with the design of refactoring catalog that will enable application programmer to safely restructure their applications to use less energy.

6. Acknowledgments

Gustavo is supported by CAPES and Fernando is supported by CNPq (306619/2011- 3), FACEPE (APQ-1367-1.03/12), and by INES (CNPq 573964/2008-4 and FACEPE APQ-1037- 1.03/08).

References

[1] A. Gautham, K. Korgaonkar, P. Slpsk, S. Balachandran, and K. Veezhinathan. The implications of shared data synchronization techniques on multi-core energy efficiency. In *Proceedings of the*, HotPower'12, Berkeley, CA, USA, 2012.

[2] J. Jelschen, M. Gottschalk, M. Josefiok, C. Pitu, and A. Winter. Towards applying reengineering services to energy-efficient applications. In *CSMR*, pages 353–358, 2012.

[3] L. A. Smith, J. M. Bull, and J. Obdrzálek. A parallel java grande benchmark suite. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing*, Supercomputing '01, pages 8–8, New York, NY, USA, 2001. ACM.

[4] A. Trefethen and J. Thiyagalingam. Energy-aware software: Challenges, opportunities and strategies. *Journal of Computational Science*, 1(0):–, 2013. ISSN 1877-7503. .

[5] W. Zhu, J. del Cuvillo, and G. R. Gao. Performance characteristics of openmp language constructs on a many-core-on-a-chip architecture. In *IWOMP*, pages 230–241, 2006.