Sustainable Computing: Informatics and Systems xxx (2017) xxx–xxx

Contents lists available at ScienceDirect

Sustainable Computing: Informatics and Systems

journal homepage: www.elsevier.com/locate/suscom

Understanding the impact of task granularity in the energy consumption of parallel programs

Alcides Fonseca[a,*], Bruno Cabral[b]

[a] LASIGE, Faculdade de Ciências da Universidade de Lisboa, Lisboa, Portugal
[b] CISUC, Department of Informatics Engineering, University of Coimbra, Coimbra, Portugal

A R T I C L E I N F O

Article history:
Received 29 July 2016
Received in revised form 20 July 2017
Accepted 31 October 2017
Available online xxx

MSC:
00-01
99-00

Keywords:
Power consumption
Parallel
Granularity

A B S T R A C T

Recently, there is a concern about reducing the energy consumption of data centers and clusters for economical and environmental reasons. Furthermore, energy consumption on mobile devices is also important to improve battery life. In this work we address the performance-energy trade-off on shared-memory multicore devices in parallel programs. In particular, we assess the impact of task granularity in performance and energy consumption. Our aim is to give programmers the knowledge they need to understand how to maximize performance of parallel programs while minimizing energy spending.

Parallel programs typically divide work in subproblems that are solved in parallel. Each subproblem can then be recursively subdivided until it is no longer worthwhile to spawn smaller tasks. Ideally, the number of parallel tasks should match the number of hardware threads in order to maximize performance and reduce scheduling overheads. Cut-off algorithms are used to stop spawning new parallel tasks and, thus, switching to sequential execution. We evaluate cut-off approaches such as MaxTasks, MaxLevel, Surplus, Adaptive Task Control and LoadBased to understand how they influence performance and energy consumption. Additionally, we also introduce and evaluate three novel approaches: MaxTasksInQueue, StackSize and MaxTasksWithStackSize.

Our experiments and analysis show how branching, workload, depth and balance influence the execution time and energy spending over a set of synthetic and real world programs. We concluded that MaxLevel was the fastest overall, while MaxTasksInQueue was the most energy efficient algorithm. Also, despite MaxTasks being slower than the prior two, it can be used by a wider range of programs.

1. Introduction

In nowadays multicore platforms, to improve the performance of computationally intensive programs, they have to be designed to execute in parallel. Currently, all types of computers, from smartphones to supercomputers, have multiple CPU cores available. In both ends of the spectrum, lowering energy consumption has become an important goal. On smartphones, tablets and other mobile devices, good performance is important to improve user experience, but battery longevity is also crucial. On the other end, energy consumption has also been an important driver for supercomputer design, both for economical and environmental reasons.

On the software side, there are several attributes that influence both speed and energy. In this work, we focus on shared-memory multicore processors. Typically, the longer a program is running, more energy is being spent on that computation. However, the energy spent does not only depend on the time a program takes to execute, but also on the way CPU and memory are used.

One of the aspects of writing and optimizing parallel programs is granularity control. Most of times, we have more parallelism in the program than hardware threads. In that case, work must be grouped together to create an ideal match between tasks and hardware threads. A task is a representation of individual work that can be executed asynchronously on any given core. If the number of tasks is smaller than the number of available hardware threads, some cores can become idle but keep consuming energy. If the number of tasks is larger than the number of hardware threads, time and energy will be spent in non-profit scheduling operations. We consider tasks instead of Operative System (OS) threads because using system calls would introduce an undesirable overhead. Our model uses a one-to-one matching between software OS-level and hardware threads, similar to how green threads work but without preemption.

* Corresponding author.
E-mail addresses: amfonseca@fc.ul.pt (A. Fonseca), bcabral@dei.uc.pt (B. Cabral).

https://doi.org/10.1016/j.suscom.2017.10.014
2210-5379/© 2017 Elsevier Inc. All rights reserved.

```
int fib(int n) {
    if (n < 16) {
        return fib(n-1) + fib(n-2);
    } else {
        Future f1 = new Future(() => fib(n-1));
        Future f2 = new Future(() => fib(n-2));
        return f1.get() + f2.get();
    }
}
```

Listing 1. Example of LTC on the Fibonacci Example.

```
class SyntheticRecursiveTask {
    public TBody(long n, long side) {...}
    public static int[] allocate(int size) { return new int[size]; }
    public static int work(int its) {
      for (int i=0; i<its; i++)
        r *= hash("SHA−256", "Really long string...");
      return r;
    }
    public static void sequential(long depth, long side) {
      allocate(ALLOCATION_BEFORE);
      work((int) (BEFORE * (BEFORE_STATIC_FACTOR + side *
           BEFORE_SIDE_FACTOR + depth * BEFORE_DEPTH_FACTOR)));
      int br = (int) (BRANCHING * (BRANCHING_STATIC_FACTOR + side *
           BRANCHING_SIDE_FACTOR + depth * BRANCHING_DEPTH_FACTOR));
      if (depth < MAX_DEPTH && br > 0) {
        for (int i=0; i < br; i++)
          sequential(depth+1, i);
      } else {
        work((int) (LEAFS * (LEAFS_STATIC_FACTOR + side *
             LEAFS_SIDE_FACTOR)));
      }
    }
  }
```

Listing 2. Recursive synthetic program.

The aim of this paper is to understand the impact of cut-off algorithms in parallel programs. As such, our contributions are:

- A new methodology for evaluating the impact of cut-off algorithms in parallel programs.
- A synthetic program that emulates different parallel program behaviors.
- An evaluation of cut-off techniques using synthetic and real-world benchmarks, in terms of time and energy consumption.
- New insights that help programmers identify the best cut-off algorithms based workload, depth, branching and balance.

In this paper we introduce both existent and new cut-off algorithms used for granularity control and related techniques (Section 2). We also present the existing work on granularity control, and on performance/energy evaluation of parallel programs (Section 3). In order to evaluate the performance/energy impact, we define a methodology (Section 4), of which we present the results on both synthetic and real-world benchmarks (Section 5), from which conclusions are drawn (Section 6).

2. Cut-off mechanisms

Dividing computational work across tasks is not easy. It depends on the program structure and the amount of work, which may in turn depend on the data provided during execution. As such, dynamic systems for scheduling any number of tasks over a certain number of threads are required. Work-stealing schedulers [1,2]