Programming Languages and Compilers (CS516) - Project #1

Hari Amoor

April 1, 2020

Contents

1 Introduction

In the theory of programming languages, concurrency is a property exhibited by certain algorithms that allow individual parts, or *contexts*, to execute in an arbitrary order and still procure a correct result. Concurrency can be conceptualized both as a formalized theory of computation, i.e. with the actor model, as well as a criterion for optimization by an optimizing compiler. This paper explores the latter, in particular by exploring the following questions:

Given a synchronous program, how can an optimizing compiler output an equivalent program with the maximum degree of concurrency possible? Moreover, what are the tradeoffs between different *runtimes* for concurrency?

How does a program that exploits hardware-level concurrency, i.e. by utilizing a graphical processing unit (GPU), differ from a program that utilizes concurrency at the operating system level?

We analyze the supplied Quickshift program, which is intended to *smoothen* an input image w.r.t. hyperparameters σ and τ. The program is capable of either running synchronously on a single CPU thread or asynchronously, using either a concurrency backend provided by the C++ OpenMP library or one backed by a 256-core GPU. Particularly, we explore the tradeoff necessitated by each concurrency runtime in terms of execution time and power consumption.

2 Background and Initial Insights

Naturally, the synchronous program runtime backed by a single CPU thread is the simplest to predict and conceptualize. With this, only a single CPU core can be utilized at a single point in time; thus, it is clear that execution time will grow linearly w.r.t. the parameters σ and τ. Furthermore, we hypothesize that only a constant factor of power can be consumed relative to the hyperparameters, given that a single core likely cannot consume more than a fixed amount of power.

The OpenMP concurrency runtime, on the other hand, provides C++ `#pragma` directives that enable multithreading, i.e. with the POSIX thread API. This runtime enables computation on multiple CPU cores. Thus, we hypothesize that the program's use of the OpenMP runtime will result in a greater consumption of power than that with a single CPU thread. Naturally, it can be expected for use of multiple CPU cores to result in a greater factor of power consumption; it is also reasonable to expect a decreased factor of growth in execution time.

The most unique runtime, and also the least predictable, is left as that backed by the 256-core GPU. Naturally, there can only be a fixed amount of power that a single GPU, regardless of the number of cores, can consume; however, it is unclear what that threshold is or at what point an increase in hyperparameter values will stop resulting in increased power consumption.

We hypothesize that heightened degrees of parallelism reuslt in a decrease in execution time at the tradeoff of an increase in power consumption, though at most by a constant factor.

3 Experiment Design

We design an experiment to analyze execution time and power of the Quickshift program in each the [single-threaded] CPU mode, the [multi-threaded] OpenMP mode, and the GPU mode. The program was run for each provided PNM image for each $(\sigma, \tau) \in \{(3i, 5i) \mid i \in 1 \ldots 5\}$, and each data point was taken as the average of five separate trials.

Execution time was output through by the Quickshift program through `stdout`, and power was measured with the provided `PowerStrip.sh` script. The power measurements were taken as the maximum measured power consumption throughout the duration of the program's runtime.

The goal of the experiment is to observe the real-time behavior of the Quickshift program relative to the hypothesis provided in (2). In particular, we seek to validate the hypothesis that concurrency reduces the execution time of a program, with the tradeoff of increased power consumption; the degree to which this effect is amplified is specific to the concurrency runtime used by the application in question.

4 Analysis of Execution Time

As specified in (3), the execution time was measured for various sets of hyperparameters by the Quickshift program itself. We present Figure 1, which specifies time taken for execution across each mode for each set of hyperparameters.

We observe that, while execution time grows exponentially w.r.t. the hyperparameters in CPU mode, it grows quadratically at best in OMP mode and linearly at best in GPU mode.

Theoretically, the hyperparameters could be increased to a point where a single growth rate would persist across all modes, since each degree of parallelism conjecturally only reduces execution time by a constant factor; however, at that point, the result would not be meaningful relative to the purpose of the Quickshift program. Thus, it is true w.l.o.g. that each degree of parallelism is an order of magnitude *faster* w.r.t. execution time than the last.

5 Analysis of Power Consumption

Furthermore, we analyze power consumption across modes. We present Figure 2, which specifies the aforementioned.

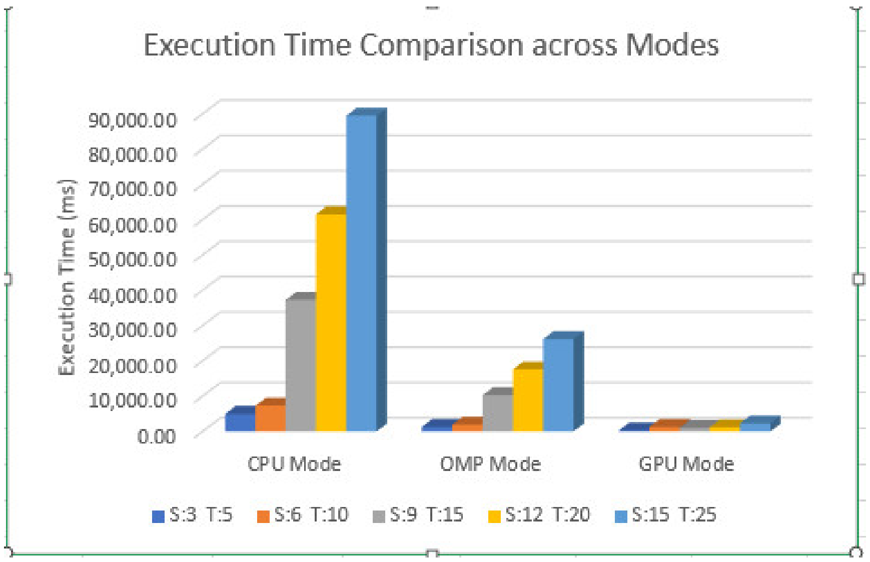

Figure 1: Execution time by mode of parallelism for each set of hyperparameters

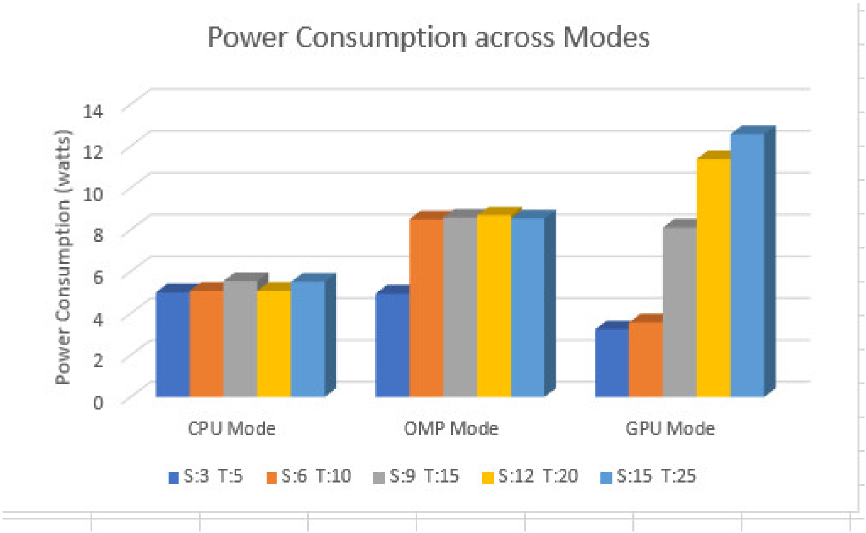

Figure 2: Power consumption (watts) by mode of parallelism for each set of hyperparameters

Here, we observe a trend that is similar to the inverse of that observed in (4). Power consumption in CPU mode is roughly constant at about 5-6 watts, and also constant, albeit at the slightly higher threshold of 8-9 watts, in OMP mode. However, in GPU mode, power consumption is roughly logarithmic, though with a high initial growth rate.

6 Analysis of Energy Consumption

Finally, we analyze energy consumption across modes of parallelism. Energy is measured in general as the (Riemann) integral of the program's power function, so we will naturally expect a correlation between energy consumption and each of execution time and power consumption.

We present Figure 3, which details the energy consumption of the Quickshift program in each mode of parallelism.

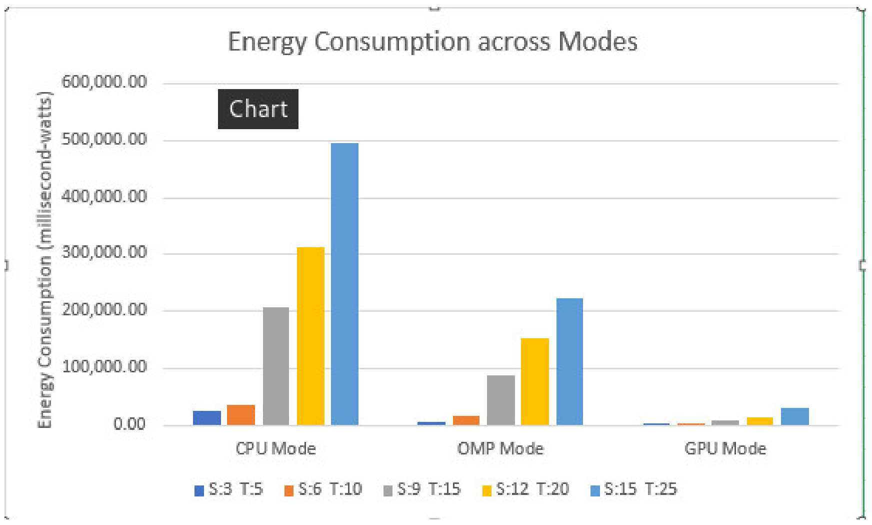

Figure 3: Energy consumption (millisecond-watts) by mode of parallelism for each set of hyperparameters

We observe that energy consumption grows non-trivially for each mode of parallelism. However, each mode of parallelism is shown to have less magnitude than the last by at least a constant factor.

7 Conclusion

The result observed in Figure 3 validates our hypothesis in (2) that heightened degrees of parallelism result in a decrease in execution time and an increase in power consumption by at most a constant factor.

However, the absolute decrease in energy consumption by heightened modes of parallelism is insufficient to say that utilization of heightened degrees of parallelism result in strictly more *performant* programs. This is specifically true in the case of power-constrained devices, e.g. smartphones and embedded systems.

Nonetheless, this experiment makes clear the tradeoff in performance made by the introduction of heightened modes of parallelism; execution time and energy consumption are observed to decrease by some constant factor, at the cost of increase in power consumption that is also by a constant factor.