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.

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 $(\sigma, \tau) \in \{(3i, 5i) \mid i \in 1 \dots 5\}$, 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.

4 Analysis of Execution Time

5 Analysis of Power Consumption

6 Conclusion