15712 Final Report

cpahka qingyanl

April 2020

1 Introduction

With the end of Dennard scaling, a single-core chip can no longer promise the expected performance it once delivered. Motivated by the need to meet the ever-increasing computation demand, most computation platforms have transitioned from single-core chip to multicore chip. This transition signifies that a program's performance is also affected by how it utilizes the available hardware parallelism. One efficient way to use hardware parallelism is to distribute concurrent tasks to idle cores. The job of distributing these tasks is done by a scheduler. The size of the task itself is defined by the granularity. The focus of this work is OS support for fine-grain parallelism programs using a work-stealing scheduler. One of the benefits of fine-grain parallelism is preventing underutilized core.

1.1 Work Stealing - Background

In this project we look at work-stealing scheduler as a means to redistribute work to idle core. Work stealing is the idea where an idle core (worker) can steal work from another worker. The stolen work is usually the continuation of a suspending function. The worker that initiate the stealing is refer as the thief, while the worker that has its continuation stolen is refer as the victim. Work-stealing scheduler has shown success in balancing work load between processor.

1.2 Work Stealing - Problem Statement

One source of overhead we aim to reduce is synchronizing the deque between thief and victim. The deque is a doubly ended queue data structure owned by one worker and contains work that can be stolen. Synchronization is needed to ensure that the victim and thief do not execute the same work / task. Various methods, using shared memory, have been implemented to avoid this race condition. However, these methods require either a lock, synchronization primitive, or memory barrier, all of which may have negative impact on the sequential performance. If no stealing occurs, most of this deque management overhead is wasted.

We believe that we can reduce the overhead by using interrupts instead of shared memory. The main idea is to not let the thief have access to the victim's