Asymmetric Multicore Processor Scheduling

Gavin Austin, Nicholas Rust, Ren Wall

November 22

1 Introduction

1.1 Why We Chose Asymmetric Multicore Processors

It has been hypothesised that ARM processors will begin to be used outside of mobile phones, and when this becomes popular, it would be nice to have a scheduler that is both efficient *and* fast. Current implementations are extremely focused on power management, leaving significant room for improvements in speed (throughput).

1.2 Current Asymmetric Multicore Process Scheduling Implementation

The current ARM implementation is known as Energy Aware Scheduling or EAS. As the name implies, it takes into account the energy usage of each process and allocates the process to a core for the duration of the process. It should also be noted that EAS for Linux is being developed jointly by Arm and Linaro, it still has a way to go.

1.3 Previous Scheduler

Linux previously had three parts that all worked next to eachother in order to try to improve energy efficiency without compromising throughput.
These are the three parts:

1. Linux scheduler (Completely Fair Scheduler - CFS)
2. Linux cpuidle
3. Linux cpufreq

1.3.1 CFS

The linux Completely Fair Scheduler is very good at optimizing throughput by throwing processes onto open cores. However, by default it does not have any knowledge of energy usage or efficiency. When implementing EAS it is given the ability to access and read the power usage values of every process.

1.3.2 cpu-idle

The cpuidle subsystem of the CFS attempts to use heuristics to save energy by putting the processor into an idle state without affecting performance. The cpuidle system has absolutely no control over the processes that are put onto specific cores and is only capable of reacting to what the CFS throws at it.

1.3.3 cpu-freq

The cpufreq (cpu frequency) subsystem also uses heuristics to throttle up or down the frequency of a core in order to

2 Types of Asymmetric Scheduling

3 Optimization Targets

4 WASH Implementation

5 Alternatives