Resource Conscious Scheduling for Energy Efficiency on Multicore Processors

Andreas Merkel Jan Stoess Frank Bellosa
University of Karlsruhe

Extended Abstract

Today's operating system schedulers treat cores of a chip multicore processor (CMP) largely like distinct physical processors. Yet, there are some interdependencies between cores that need be taken into account for optimal performance and energy efficiency.

The cores of a multicore chip share resources such as caches and memory interfaces. This is likely to cause contention between the cores if activities with similar characteristics, for example several memory-bound programs, are running together. Contention slows down the execution of the programs, and, aside from the performance penalty, also induces inefficient use of energy, since cores waiting for a resource to become available dissipate power without making progress.

A second cross-effect, also related to energy efficiency, is the fact that many chips only allow setting a single frequency and voltage for the entire chip, meaning that all cores need to run at the same frequency and voltage, since allowing multiple frequencies and voltages would introduce additional hardware complexity.

The efficiency in terms of energy and runtime with which the processor can execute a program at a certain frequency and voltage setting, again strongly depends on the program's characteristics, in particular on the frequency of memory accesses. Memory-bound programs do not suffer much slowdown if the processor frequency is reduced, since the speed of their execution is limited by the speed of memory rather than by processor speed, which allows energy-efficient execution at low processor frequencies.

Since the scheduler is the component of the operating system responsible for deciding which tasks run on the cores simultaneously, scheduling is crucial for performance and energy efficiency. For a multicore chip that offers only chip-wide frequency scaling, the question arises whether it is advantageous to run tasks with similar characteristics together in order to run the chip at the corresponding optimal frequency. For instance, we could co-schedule memory-bound tasks, which run most efficiently at a low frequency, and co-schedule compute-bound tasks, which run most efficiently at a high frequency.

We investigate the cross-effects between applications running on a multicore system, considering resource contention and the technical constraint of chip-wide frequency and voltage settings. Our analysis of an Intel Core2 Quad processor finds that memory bandwidth is the most crucial resource on this platform, and that scheduling for avoiding contention is more important than being able to select a common best frequency. We find that in order to optimize the product of runtime and expended energy (energy delay product, EDP), the main goal must be to avoid contention by combining tasks with different characteristics. Only if nothing but memory-bound tasks are available, it is beneficial to apply frequency scaling.

In previous work, we have proposed the concept of *task activity vectors* to represent utilization of chip resources caused by tasks. Based on the information about resource utilization provided by activity vectors, we propose policies for avoiding resource contention by co-scheduling tasks with different characteristics. Our co-scheduling policy synchronizes the scheduling decisions of cores belonging to the same chip, and selects tasks utilizing different resources for simultaneous execution. In addition, a modified load balancing policy that is aware of task characteristics makes sure that tasks of different characteristics are available on each core.

In situations when contention cannot be avoided because of workloads containing too many memory-bound tasks, we take advantage of frequency and voltage scaling as a fallback solution. For this purpose, we apply a heuristic that lowers the frequency when only memory-bound tasks are running.

We have extended the Linux 2.6.22 kernel scheduler to support our new strategies. We further use a recent version of the KVM virtual machine monitor to extend the scheduling support from individual tasks to complete software environments running in virtual machine instances, and leverage virtual machine live migration to make placement decisions not only between different CPUs of the same node, but also across individual nodes. An evaluation of our policies using SPEC CPU 2006 benchmarks reveals that our policies manage to reduce EDP considerably in comparison to standard Linux scheduling.