Monitoring interactive applications

The goal of the computer when running interactive applications is to respond to user events under human perception bounds. For these applications the key performance metric is not overall throughput but response time. However, due to the difficulty of isolating execution episodes that have direct bearing on response time, even benchmarks for interactive applications (e.g. Sysmark and Winstone) only measure end-to-end throughput. The results of these benchmarks can be misleading because they ignore two key properties of interactive applications: not all parts of the program’s execution are equally important and faster response-time is not always better. The human cognitive system sets an upper bound for necessary performance. There is a point beyond which improvements are imperceptible to the user, and, thus are unnecessary. To gain insight into the behavior of interactive applications, I developed a technique that provides automatic feedback to the operating system about the interactive performance of executing programs. This methodology quantifies response time by monitoring the communication between executing threads (both system and application) and by observing the applications’ interactions with the kernel through system calls. This mechanism is implemented in the Linux kernel and requires no modification of user programs.

Impact of multiprocessing on interactive performance

Multiprocessing is already prevalent in servers where multiple clients present an obvious source of thread-level parallelism. However, the case for multiprocessing is less clear for desktop applications. Nevertheless, architects are designing processors that count on the availability of thread-level parallelism. As a reality check, we applied our monitoring technique to measure the impact of multiprocessing on interactive performance. The results show that running our benchmarks on a dual-processor machine improves response times of perceptible events by as much as 36% and 22% on average—out of a maximum possible 50%. The benefits of multiprocessing are even more apparent when background tasks are considered. Running a simple MP3 playback program in the background degrades response times by an average of 14% on a uniprocessor vs. only 4% on a dual-processor machine. While four processors would be underutilized, dual-processor machines provide an effective way of improving response times.

Performance-setting for dynamic voltage scaling

My investigations into the effects of multiprocessing show that, while there is still a need to improve processor performance to reduce response-times of a few long-running episodes, the majority of interactive episodes—even on a uniprocessor—fall under the human perception threshold. When response-times are fast enough for further improvements to be imperceptible, the processor can be slowed down to save energy. Some contemporary processors (e.g. Transmeta Crusoe and Intel XScale) support dynamic voltage and frequency scaling which allows one to make fine granularity trade-offs.
between power-use and performance, provided there is a mechanism for controlling that trade-off. I developed an algorithm, based on the combination of the episode detection technique and a performance-level predictor, for automatically controlling dynamic voltage scaling to optimize energy use. Unlike previous automated approaches, this method works equally well with irregular and multiprogrammed workloads. Moreover, it has the ability to ensure that the quality of interactive performance is within user-specified parameters. Experiments show that as a result of this algorithm, processor energy savings of as much as 75% can be achieved with only a minimal impact on the user experience.

**Directions for future research**

Due to the lack of availability of systems that support dynamic voltage scaling, the power management results presented in my dissertation are based only on simulations. I am interested in further refining the performance-setting technique and evaluating it on a system that supports dynamic voltage scaling.

While the power management methodology does not require the modification of user programs, an optional API augmenting the existing prediction and communication-tracking mechanism would allow critical applications to take full advantage of power management. One of the biggest shortcomings of the current performance-setting strategy is its inability to distinguish episode instances from one another (i.e. distinguishing when an interactive episode is the result of the same UI event as before). The proposed API would allow the programmer to delineate and name critical episodes and to optionally specify their deadlines.

The core idea of the episode detection technique—on-line monitoring and dynamic adaptation—could be applied to other algorithms. For example the scheduler could be made aware of which tasks interact with the user and could use this information to further improve interactive performance. Perhaps this idea could be generalized to derive deadlines for repetitive real-time tasks automatically.

My dissertation has shown that multiprocessing can be used effectively to improve the response-times of long interactive episodes and that dynamic voltage scaling can yield significant power savings when response-times are fast enough. Can these two ideas be combined to yield a power-efficient multiprocessor?