AUTOMATIC PERFORMANCE-SETTING FOR LINUX

Vivek Gusain, Varsha Chauhan
Department of Information technology
Dronacharya College of Engineering, Gurgaon, India

Abstract- **Combining high performance with low power consumption is becoming one of the primary objectives of processor designs. Instead of relying just on sleep mode for conserving power, an increasing number of processors take advantage of the fact that reducing the clock frequency and corresponding operating voltage of the CPU can yield quadratic decrease in energy use. However, performance reduction can only be beneficial if it is done transparently, without causing the software to miss its deadlines. In this paper, we describe the implementation and performance-setting algorithms used in Vertigo, our power management extensions for Linux. Vertigo makes its decisions automatically, without any application-specific involvement. We describe how a hierarchy of performance-setting algorithms, each specialized for different workload characteristics, can be used for controlling the processor's performance. The algorithms operate independently from one another and can be dynamically configured. As a basis for comparison with conventional algorithms, we contrast measurements made on a Transmeta Crusoe-based computer using its built-in LongRun power manager with Vertigo running on the same system. We show that unlike conventional interval-based algorithms like LongRun, Vertigo is successful at focusing in on a small range of performance levels that are sufficient to meet an application's deadlines. When playing MPEG movies, this behavior translates into a 11%-35% reduction of mean performance level over LongRun, without any negative impact on the framerate. The performance reduction can in turn yield significant power savings.**

I. INTRODUCTION

Power considerations are increasingly driving processor designs from embedded computers to servers.Perhaps the most apparent need for low-power processors is for mobile communication and PDA devices. Devices are battery operated, have small form factors and are increasingly taking up computational tasks that in the past have been performed by desktop computers. The next generation 3G mobile phones promise always-on connections, high-bandwidth mobile data access, voice recognition, video-on demand services, video conferencing and the convergence of today's multiple standalone devices—MP3 player, game machine, camera, GPS, even the wallet— into a single device. This requires processors that are capable of high performance and modest power consumption. Moreover, to be power efficient, the processors for the next generation communicator need to take advantage of the highly variable performance requirements of the applications they are likely to run. For example an MPEG video player requires about an order of magnitude higher performance than an MP3 audio player but optimizing the processor to always run at the level that accommodates the video player would be wasteful. Dynamic Voltage Scaling (DVS) exploits the fact that the peak frequency of a processor implemented inCMOS is proportional to the supply voltage, while the amount of dynamic energy required for a given workloadis proportional to the square of the processor's supply voltage. Running the processor slower means that the voltage level can also be lowered, yielding a quadratic reduction in energy consumption, at the cost of increased run time. The key to making use of this trade-off are performance-setting algorithms that aim to reduce the processor's performance level (clock frequency) only when it is not critical to meeting the software's deadlines. The key observation is that often the processor is running too fast. For example, it is pointless from a quality-of-service perspective to decode the 30 frames of a video in half a second, when the software is only required to display those frames during a one second interval. Completing a task before its deadline is an inefficient use of energy .While dynamic power currently accounts for the greatest fraction of a processor's power consumption, static power consumption, which results from the leakage current in CMOS devices, is rapidly increasing. If left unchecked, in a 0.07 micron process, leakage power could become comparable to the amount of dynamic power . Similarly to dynamic power, leakage can also be substantially reduced if the processor does not always have to operate at its peak performance level. One technique for accomplishing this is adaptive reverse body biasing (ABB), which combined with dynamic voltage scaling can yield substantial reduction in both leakage and dynamic power consumption .The pertinent point for this paper with respect to DVS and ABB is that lowering the speed of the processor results in better than linear energy savings. Vertigo provides the main lever for controlling both of these tech-niques by providing an estimate for the necessary performance level of the processor.Most mobile processors on the market today already support some form of voltage scaling; Intel calls its version of this technology SpeedStep. However,due to the lack of built-in performance-setting policies in current operating systems, the computers based on these chips use a simple approach that is driven not by the workload but by the usage model: when the notebook computer is plugged in a power outlet the processor runs at a higher speed, when running on batteries, it is switched to a more power efficient but slower mode. Transmeta's Crusoe processor sidesteps this problem by building the power management policy—called LongRun—into the processor's firmware to avoid the need to modify the operating system. Vertigo is implemented as a set of kernel modules and patches that hook into the Linux kernel to monitor program execution and to control the speed and voltage levels of the processor One of the main design objectives of this system has been to be minimally intrusive into the host operating system. Vertigo coexists with the existing scheduler, system calls, and power manager (which controls the sleep and awake modes of the processor), however it needs certain hooks within these subsystems. A unique feature of Vertigo is that instead of a single performance-setting algorithm, it allows the composition of multiple algorithms, all specializing in different kinds of run-time situations.

II. Performance-Setting Algorithms

Unlike previous approaches, Vertigo includes multiple performance-setting algorithms that are coordinated to find the best estimate for the necessary performance level. The various algorithms are organized into a decision hierarchy, where algorithms closer to the top have the right to override the choices made at lower levels. Currently we have three levels on the stack:

• *At the top:* an algorithm for automatically quantifying the performance requirements of interactive applications and which ensures that the user experience does not suffer. This algorithm is based on our previous one described in.
• *In the middle:* an application specific layer, where DVS-aware applications can submit information about their performance requirements.
• *At the bottom:* an algorithm that attempts to estimate the future utilization of the processor based on past information. This *perspectives-based* algorithm differs from previous interval-based algorithms in that it derives a utilization estimate for each task separately and adjusts the size of the utilization-history window on a per-task basis. Moreover, since the algorithm in the top layer ensures the high quality of interactive performance, the baseline algorithm does not have to be conservative about the size of the utilization- history window, the consideration of which has led to inefficient algorithms for even simple workloads (e.g. MPEG playback) in the past .
In this paper our focus is on the *interactive* algorithm at the top of the stack and the *perspectives-based* algorithm at the bottom. The application-specific layer is currently only used for debugging: we have instrumented certain applications such as the X server and our mpeg player to submit application specific information to Vertigo (through a system call) and then this information can be used to correlate Vertigo's activities with that of the applications.

III. Implementation Issues

3.1 Policy stack

The policy stack is a mechanism for supporting multiple independent performance-setting policies in a unified manner. The primary reason for having multiple policies is to allow the specialization of performance-setting algorithms to specific situations instead of having to make a single algorithm perform well under all conditions. The policy stack keeps track of commands and performance-level requests from each policy and uses this information to combine them into a single global performance-level decision when needed.

3.2 Work tracking

Our algorithms use the processor's utilization history over a given interval to estimate the necessary speed of the processor in the future. The idea is to maximize the busy time of the processor by slowing it down to the appropriate performance level. To aid this, Vertigo provides an abstraction for tracking the work done during a given time interval which takes performance changes and idle time into account regardless of the specific hardware counter implementations. To get a work measurement over an interval, a policy needs to allocate a vertigo_work struct and call the vertigo_work_start function at the beginning, and the vertigo_work_stop function at the end of the interval. During the measurement, the contents of the structs are updated automatically to reflect the amount of idle time and the utilized time weighted by the corresponding performance levels of the processor.

3.3 Monitoring, timers and tracing

One design goal of Vertigo has been to make it as autonomous from other units in the kernel as possible.Another design goal emerged as we selected the platform for our experiments. The Transmeta Crusoe processor includes its own performance-setting algorithm and we wished to compare the two approaches. The first requirement has already yielded a relatively unobtrusive design, the second focused us on turning the existing functionality into a passive observation platform.

IV. Evaluation

Our measurements were performed on a Sony Vaio PCG-C1VN notebook computer using the Transmeta Crusoe 5600 processor running at 300Mhz to 600Mhz with 100Mhz steps. The operating system used is Mandrake 7.2 with a modified version of the Linux 2.4.4- ac18 kernel. The workloads used in the evaluation are the following: Plaympeg SDL MPEG player library , Acrobat Reader for rendering PDF files, Emacs for text editing, Netscape Mail and News 4.7 for news reading, Konqueror 1.9.8 for web browsing, and Xwelltris 1.0.0 as a 3D tetris-like game. The interactive shell commands benchmark is a record of a user doing miscellaneous shell operations during a span of about 30 minutes. To avoid variability due to the Crusoe's dynamic translation engine, most benchmarks were run at least twice to warm up the dynamic translation cache, and data was used from the last run.

V. CONCLUSIONS AND FUTURE WORK

We have shown how two performance-setting policies implemented at different levels in the software hierarchy behave on a variety of multimedia and interactive workloads. We found that Transmeta's LongRun power manager, which is implemented in the processor's firmware, makes more conservative choices than our Vertigo algorithms, which are implemented in the Linux kernel. On a set of multimedia benchmarks, the different design decisions result in a 11%-35% average performance level reduction by Vertigo over LongRun. Being higher on the software stack allows Vertigo to make decisions based on a richer set of run-time information, which translates into increased accuracy. While the firmware approach was shown to be less accurate than an algorithm in the kernel, it does not diminish its usefulness. LongRun has the crucial advantage of being operating system agnostic. Perhaps one way to bridge the gap between low and high level implementations is to provide a baseline algorithm in firmware and expose an interface to the operating system to optionally refine performance-setting decisions. The policy stack in Vertigo can be viewed as the beginnings of a mechanism to support such design, where the bottom-most policy on the stack could actually be implemented in the processor's firmware.

We believe that aside from dynamic voltage scaling, performance-setting algorithms will be useful for controlling other power reduction techniques, such as adaptive body biasing. These circuit techniques cut down on the processor's leakage power consumption, which is an increasing fraction of total power as the feature sizes of transistors are reduced. While the power consumption of the processor is a significant concern, it only accounts for a fraction of the system'stotal power consumption. Future work will extend our technique to managing the power of all the devices in an integrated system.

References

[1] M. Aron and P. Druschel. Soft Timers: Efficient Microsecond Software Timer Support for Network Processing. *Proceedings of the 17th Symposium on Operating Systems Principles (SOSP-17)*, December 1999.

[2] T. Mudge. Power: A First Class Architectural DesignConstraint. IEEE Computer, vol. 34, no. 4, April 2001.

[3] T. Okuma, T. Ishihara, and H. Yasuura. Real-Time Task Scheduling for a Variable Voltage Processor. *Proceedings of the International Symposium on System Synthesis*, November 1999.

[4] T. Pering, T. Burd, and R. Brodersen. The Simulationand Evaluation of Dynamic Voltage Scaling Algorithms.*Proceedings of International Symposium on Low Power Electronics and Design 1998*, pp. 76-81, June 1998.

[5] T. Pering, T. Burd, and R. Brodersen. Voltage Scheduling in the lpARM Microprocessor System.