SYSTEM LEVEL ADAPTIVE FRAMEWORK FOR POWER AND PERFORMANCE SCALING ON INTEL® PXA27X PROCESSOR

Priya N. Vaidya, Moinul H. Khan, Bryan Morgan, Premanand Sakarda
{Priya.n.vaidya,Moinul.h.khan}@intel.com
Intel Corporation

ABSTRACT

Next generation Phone and PDAs face stringent power and performance requirements. In order to take advantage of dynamic voltage and frequency management software driven adaptive power management methods are emerging as the key to performance and power scaling. This paper demonstrates an adaptive power management framework for Intel XScale™ Microarchitecture based platforms, which dynamically characterizes executing workloads based on system level events and adapts frequency and voltage in order to save power. In this paper we discuss the overall framework and analysis behind the optimal policy to adapt processor frequency and voltage. The paper also illustrated benefits of using this framework for MP3 playback, memory data transfer, phone idling etc. real life case studies.

1. INTRODUCTION

In the recent past, the frequency of operation for wireless application processors has been trending upward to accommodate higher performance demand. Higher frequency typically results in higher power. On top of that, these processors have been integrating various functionality on-chip leading to higher die-size and increased leakage. On the other hand, battery longevity constraint is increasing. Dynamic voltage and frequency management (Wireless Intel® SpeedStep Technology) is being deployed to meet the power and performance constraints. This paper introduces one such software framework based on Intel® XScale™ Microarchitecture technology. The paper is organized as follows: Section 2 describes the hardware power management features. Section 3 gives high level overview of the software framework. Section 4 discusses in detail the analysis behind the policy manager. Sections 5 and 6 describe the experimental results and data analysis.

2. H/W POWER MANAGEMENT SOLUTION

To meet the power performance scaling challenges, system on-chip solutions for handheld devices such as, PXA27x[7], an Intel® XScale™ Microarchitecture based wireless application processor, deploys two key technologies:

a) Dynamic voltage and frequency control: In this method, operating frequency of the core, interconnect, memory and other sub-system can be adjusted at run-time. Reducing frequency during inactive period saves power. Depending on the frequency of operation, the external voltage supply can be dynamically adjusted. Lowering voltage saves power even further.

b) Power modes: The Intel® PXA27x processor family supports six power modes. The power modes are primarily differentiated by available functionality, total power consumption, and the amount of time to enter and exit a mode. As a function of workload and resource utilization, the device can transition between the power modes to minimize power consumption. The state transition diagram between the modes is shown in Fig. 1.

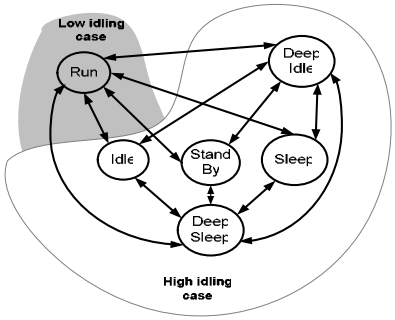

Figure 1: State transition diagram between power modes

Each of the six modes provides different levels of power consumption and resource availability in addition to variation in the transition time to the "Run" mode. The details of the power modes are not furnished for the sake of brevity (details in [7]). HW solution assumes that, software (OS/Applications) are in charge of taking advantage of the above power features. During inactive period the system uses low power modes, such as sleep, standby, where as during active operation, the system resides in "Run" mode where the frequency and voltage are adjusted dynamically.

3. SOFTWARE POWER MANAGEMENT SOLUTION

In terms of taking advantage of the HW features, typically, operating system, application and users perspective are

limited to high level of abstraction and are not fine tuned to the HW platform. Hence, most of the software power management solutions developed thus far are tied to the Operating systems and utilize the information that the operating system provides [1][4][5][6]. They are inherently predictive methodologies based on task or scheduler information, which works well for compute intensive applications, since, they cannot distinguish between CPU or memory load, i.e. usage contribution of various system components. This paper proposes a generic software solution framework that uses a system-level approach and works closely with embedded operating system's power management interface. A system level perspective allows the power-manager to understand the resource demand by the core and other system components (e.g. DMA, LCD, Video, Graphics Controller and Camera etc.) and fine tune the core, peripheral, memory and interconnect frequency individually. OS/application independent solution allows further policy adaptation based on the system usage. The proposed framework is composed of two basic components: (a) Profiler (b) Policy Manager. The profiler is responsible for probing the system, collecting the statistics and making them available to the policy manager. The policy manager uses that input and decides the system operating point (core freq., bus/mem frequencies and processor state). Fig. 2 below shows the overall framework.

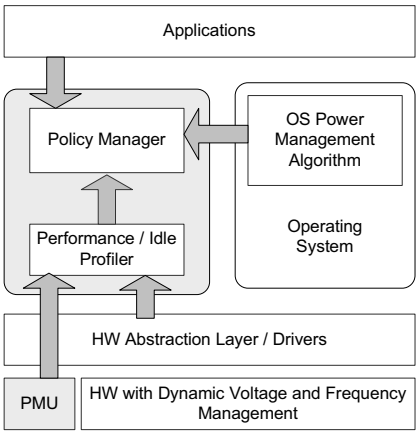

Figure 2: Architecture of IPM framework

3.1 Profiler

The profiler collects 2 different types of statistics. One based on the OS idle thread that calculates %CPU utilization, and the other using the Processor PMU (Performance monitoring unit) that calculates the %memory utilization. At the end of each sampling window the statistic is delivered to the policy manager. Probing window size determines the speed of adaptation. Depending on the rate of change in the profile, the probing window can be adapted dynamically. It is important that this code is executed as close to the end of a window as possible to ensure timely statistics are generated. For example, installing this code in an ISR meets this requirement (each OS has its own way of achieving this goal). For calculating the %CPU utilization based on idle thread the algorithm is as below:

```
(enter idle task)
        store the idle start time
        enter processor idle power mode
        store the idle stop time
        calculate and aggregate time spent in
        processor idle
(exit idle task)
```

PMU monitors various processor specific events (Detailed list can be found in [7]). For calculating the memory utilization statistic (% Memory utilization) at end of each window, we monitor the data cache misses and use that as an indication of system memory activity.

% Memory Utilization = %Data Cache-misses rate - This metric is calculated by counting the PMU event for (Data Cache accesses) and (Data Cache Misses)

$$D_{miss\_rate} = D_{miss} / D_{access} \; ; \; I_{miss\_rate} = I_{miss} / I_{access}$$

Similarly, communication fabric utilization, processor utilization and demand of other system components can be tracked. These system level events (which we capture via profiler) are stochastic processes, which can change in short term or long term (slow varying or rapid varying). The PMU counters mentioned earlier run concurrently to the execution of application and do not require any counting mechanism in software. This allows a clock-by-clock accuracy of the events. The profiler can keep track of events at any level granularity (as programmed by the policy).

4. POLICY DESCRIPTION

Most critical component of this framework is the policy manager. It uses the profiler information as its input. The output of the policy manger is the system operating point. We define the operating point as: *OP = {PM(State/Mode), f(Freq), V(Voltage)}*. Processor frequency and voltage are varied when the processor state is "Run", when in any other lower-power state which is not "Run", the frequency and voltage is usually fixed and not varied. Based on the system activity and profiles we can make 2 high-level distinctions between the overall policies:

(a) Policy for low idling CPU – System Run Mode

Most software applications/workloads can be generalized into two main categories:

- *CPU (compute) bound applications*
- *Memory bound applications*

In order to achieve dynamic power/performance scalability, the user demand (performance) should be achieved at the lowest cost (power). Dynamic voltage and frequency scaling of the processor allows us to attain an optimal operating point. There have been several research papers in this area [2][3]. Memory latency and throughput

are dependent on the frequency of the memory bus or the system bus which is usually decoupled from the core frequency. Thus, memory and system bus frequency can be controlled for the memory-bound applications independent of the core frequency. For this Freq is a three tuple number *f = {fc, fb, fm}*, representing core, bus and memory frequency respectively. Typically any single application can go through a set of distinct phases, (e.g. internet explorer downloading, video decoding etc.). However, given complex application mixes, the phases may not be as distinct. We model the demand of system usage is an auto-regressive process. Where the demand for resource utilization at nth probing window, is a weighted sum of the same over last Nth window. $\hat{W}_R[n]$ as the prediction of demand or usage of a resource *R* at the nth probing window.

$\hat{W}_R[n] = \sum_{i=1}^{N} c_R[i] W_R[n-i]$; where $W_R[i]$ is the observed demand and $C_R[i]$ is the coefficient of prediction. At the end of each probing window based on the demand forecast of the resource respective frequency is adjusted. The frequency forecast is considered directly proportional to the demand forecast. Thus, at each probing window the adjustment to the frequency is done as follows

$$\Delta f_c[n] = \alpha \sum_{i=1}^{N} c_c[i] W_c[n-i] - f_c[n-1]$$

$$\Delta f_b[n] = \beta \sum_{j=}^{M} \sum_{i=1}^{N} c_b[i,j] W_b[n-i,j] - f_b[n-1]$$

$$\Delta f_m[n] = \gamma \sum_{j=1}^{M} \sum_{i=1}^{N} c_m[i,j] W_m[n-i,j] - f_m[n-1]$$

Here α, β, γ are the proportional constants. For *fb* and *fm* adaptation, demand for the other resources are also considered and hence the second summation over M number of resources. However, based on the Si constraints, these frequency changes may be done at discrete steps (i.e. as multiples of external oscillator frequency etc.). The coefficients for prediction $C_R[i]$, method of selection of α, β, γ and order of prediction *N* is open to experimentation in the framework and also, varies with the probing window size. If the prediction is accurate, the window size is increased, otherwise, it is reduced. Thus, probing window length Tw is computed as follows:

$$T_W[n+1] = \left(\hat{W}_C[n] - W_C[n]\right)\lambda + T_W[n]$$

For larger window size, the order of prediction can be reduced. Along with the window size $C_R[i]$ adaptation scheme can also be programmed in the framework. For the experiments presented in the paper, some simplified set up was used to the above algorithm for better study. The impact of window size on performance is under investigation.

(b) Policy for high idling CPU – System Idle Mode

When the profilers report high idling activity, the device can be turned into low-power state for maximum power savings. The policy manager then queries the device input. If the devices time-out, the policy decides to enter processor low-power state. The events that are required for this policy decision to happen are the profiler input and the device time-out. The devices inputs are random events and do not necessarily follow the MA model. The device-timeout events can arrive randomly, following a Poisson distribution. The wake-up events or events that bring the system out of this low-power state are events like key-press, RTC wake-up etc. these also again are random and can follow Poisson distribution. Fig. 3 gives a very high-level flow graph example of the policy.

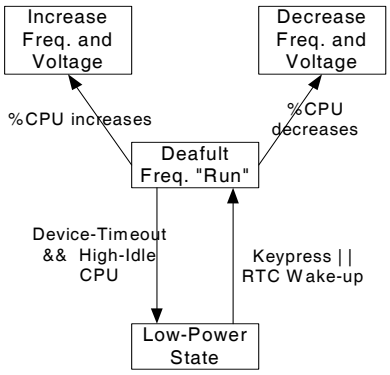

Figure 3: Policy Flow graph

5. EXPERIMENTAL SET UP

After careful characterization of the various usage models, we decided to focus on multimedia workloads including MP3 and data movement operation to demonstrate the policy framework. PXA27x processor based on Intel® Development platform running Linux based Operating System was used as a test-bed. QVGA LCD display was enabled and refreshed at 60 fps. The workloads were downloaded as executables on the system and were running out of SDRAM. MP3 workload was playing audio at 44.1 KHz and 128 kbps. Data Transfer workload performed large amount of data/file copying between two locations in the external memory SDRAM. The video decoder was playing a 320x240 sized clip at about 256 kbps. Multimedia workloads were based on the Intel IPP Performance Suite [8]. The profilers and the policy were running on the system during the operation of these workloads. The sampling power data was captured using National Instruments Data Acquisition. In all the cases the sampling was done for about 6 seconds while running the workloads. Power savings is measured based on the following definition:

$$P_{Saving} = \frac{\left(P_{RUN} - P_{Policy}\right)}{P_{RUN}} \cdot 100\%$$

The policy frame work was programmed at a fixed probing window and a 1st order of autoregressive prediction was used.

6. RESULTS

All the results follow a similar color scheme. The black denote the power data without the profilers and the policy and the white denotes the power data with the policy and the profilers running. Fig. 4 shows the power acquisition profile for workload running MP3. The data is power captured on the processor core only not system-wide. The prediction model for the experiment assumes *N=1* and $\alpha = \beta = \gamma = 1$.

The results for MP3 show that due to the static window size there is slight delay between the start of MP3 application and the change in the system operating point. Fig. 5 shows the power sampling data for the data transfer workload. The profiler detects that the workload is memory bound and policy manager makes a decision to reduce to power by lowering the core frequency but increases the memory bus frequency. That's how the total power dissipation is reduced but the required performance is delivered. There is approximately 8% reduction in the memory copy throughput that is achieved. The table below gives the actual power savings percentages and corresponding performance degradation.

Table 1: Power Savings and performance impact

Workload	Savings with Policy	Impact on Performance
OEMIdle	60% Savings	0%
MP3	47% Savings	0%
Data Transfer	70% Savings	~8%
MPEG4 Video Playback	50% Savings	0% fps degradation

7. CONCLUSION

This paper details the dynamic power and performance scaling framework that can be adopted. It is adaptable to any operating system. The policy can be modified and adjusted for a specific usage model. A single policy cannot cover all the various usage models that a PDA/phone can have. Further work is ongoing on analyzing the impact of higher order prediction on the overall policy impact. Characterization of adaptive probing window is also currently studied.

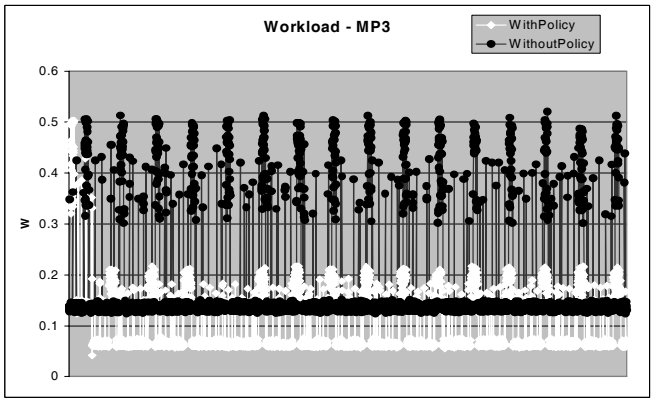

Figure 4: Power Data for MP3

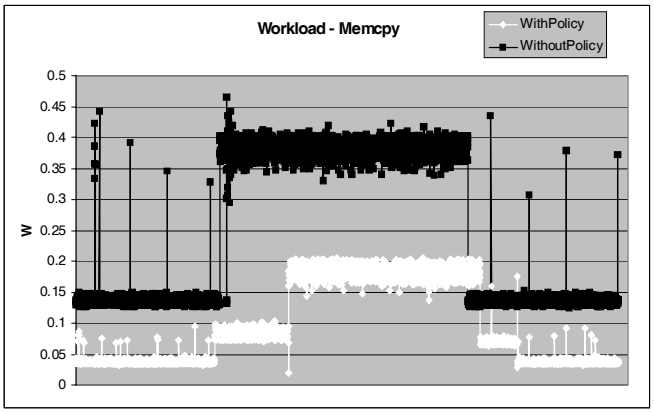

Figure 5: Power Data for Memory Copy

8. REFERENCES

[1]Krisztian Flautner and Trevor Mudge,"Vertigo: Automatic Performance Setting for Linux" Proceedings of the 5th Symposium on Operating Systems Design and Implementation 2002.

[2] Matthew W. Alsleben and Jeanine Cook, "Toward Dynamic Recognition of Workload Phases" 4th Annual Austin CAS Conference, February 21, 2003

[3]J. Cook, R.L. Oliver, E. E. Johnson, "Examining Performance Differences In Workload Phases", Proceedings of the 4th IEEE International Workshop on Workload Characterization, pages 82-90, December 2001

[4]Y. Shin, K. Choi, and T. Sakurai. "Power Optimization of Real-Time Embedded Systems on Variable Speed Processors." Proceedings of the International Conference on Computer-Aided Design, pages 365--368, November 2000

[5]A. Manzak, C. Chakrabarti, "Variable Voltage Task Scheduling for Minimizing Energy or Minimizing Power", IEEE International Conference on Acoustics, Speech, and Signal Processing, (ICASSP'00), 2000, pp. 3239-3242.

[6]Oman S. Unsal, Israel Koren, "System- level Power Aware Design techniques in Real-time Systems", Proceedings of the IEEE, Vol. 91, July 2003.

[7]Intel® PXA27x Application Processor Architecture Reference Manual. http://www.intel.com

[8] Intel® Integrated Performance Primitives on Intel® Personal Internet Client Architecture Processors Reference Manual. http://www.intel.com