Dynamic Power Management for Embedded Systems

Bishop Brock and Karthick Rajamani
IBM Research 11501 Burnet Road Austin, Texas 78758
Email: bcbrock@us.ibm.com, karthick@us.ibm.com

***Abstract*— This paper discusses several of the SOC design issues pertaining to dynamic voltage and frequency scalable systems, and how these issues were resolved in the IBM PowerPC 405LP processor. We also introduce DPM, a novel architecture for policy-guided dynamic power management. We illustrate the utility of DPM by its ability to implement several classes of power management strategies and demonstrate practical results for a 405LP embedded system.**

I. Introduction

Advances in low-power components and system design have brought general purpose computation into watches, wireless telephones, PDAs and tablet computers. Power management of these systems has traditionally focused on sleep modes and device power management [1]. Embedded processors for these applications are highly integrated system-on-a-chip (SOC) devices that also support aggressive power management through techniques such as programmable clock gating and dynamic voltage and frequency scaling (DVFS). This paper describes one of these processors, and the development of a software architecture for policy-guided dynamic power management.

II. 405LP Design and Power Management Features

The IBM PowerPC 405LP is a dynamic voltage and frequency scalable embedded processor targeted at high-performance battery-operated devices. The 405LP is an SOC ASIC design in a 0.18 μm bulk CMOS process, integrating a PowerPC 405 CPU core modified for operation over a 1.0 V to 1.8 V range with off-the-shelf IP cores. The chip includes a flexible clock generation subsystem, new hardware accelerators for speech recognition and security, as well as a novel standby power management controller [2]. In a system we normally operate the CPU/SDRAM at 266/133 MHz above 1.65 V and at 66/33 MHz above 0.9 V, typically providing a 13:1 SOC core power range over the 4:1 performance range. From a system design and active power management perspective the most interesting facets of the 405LP SOC design concern the way the clocks are generated and controlled. These features of the processor are described in the remainder of this Section.

A. PLL

The 405LP PLL exists in a separate voltage island, internally regulated to the minimum voltage specified for the CPU core. As long as the core voltage slew rate is below 10 mV/μs the PLL is immune to relocking requirements during voltage scaling. Hence, the 405LP continues to execute code and respond to interrupts even during voltage scaling. With low-cost power supply designs we find voltage scaling times typically to be on the order of several hundred μs, so it is advantageous to be able to continue execution while waiting for voltages to stabilize. The CPU clock is generated by a clock divider outside of the PLL feedback path, and the circuit design allows divide ratios to be changed without glitches on the output clock. This allows CPU frequency scaling over a wide range with low latencies.

B. System Clocking

The 405LP CPU clock is further divided down to generate clocks for the internal and external busses: the high-speed Processor Local Bus (PLB, also SDRAM frequency), and lower speed On-board Peripheral Bus (OPB) and External Peripheral Bus (EBC). These clock dividers are glitch-free, and the divide ratios can be changed at any time as long as the resulting overall frequency settings are lower than the maximum frequency allowed by the current operating voltage. Clocking flexibility is important for optimal power management. In a CPU-intensive application like media decoding we find that lowering bus frequencies from their maximum performance points can result in system-wide energy savings without impacting real-time performance. However, power management policies are typically defined to only scale the CPU and PLB (memory) frequencies, leaving the lower-speed busses at fixed frequencies. This avoids the need to reprogram bus controller parameters and IP cores that have some sensitivity to peripheral bus frequencies.

In the 405LP design changing the CPU and bus clocking scheme requires updating anywhere from one to three control registers and may also require reprogramming the SDRAM controller and other bus controllers. The total operating system overhead for typical frequency and voltage changes varies from 13 μs to 95 μs in our Linux implementation, depending on the initial and target frequencies and whether voltage management is required. Low-latency frequency scaling is a crucial part of the dynamic power management equation, and it is especially important that frequency changes can be made without impacting system operations. Some scalable SOC designs require all memory and DMA traffic to be suspended for extended periods during certain frequency changes [3]. Aggressive scaling strategies under these constraints could lead to undesirable artifacts in the embedded display and audio

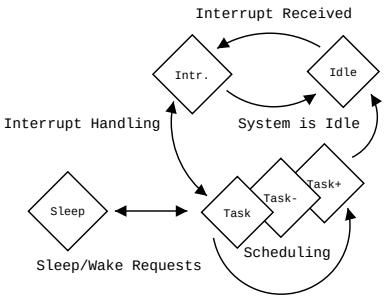

Fig. 1. Operating states and state transitions

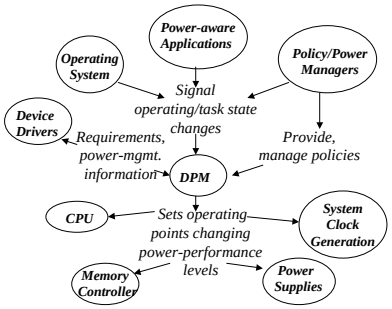

Fig. 2. DPM Architecture

subsystems if insufficient DMA buffering were available. In the 405LP design, memory and DMA transactions are never suspended for long periods during frequency shifts. When changes in the PLB frequency require reprogramming the SDRAM controller, all transactions to SDRAM are temporarily and transparently suspended, and resume once the reprogramming is complete. This latency ranges from 3 μs to 14 μs at 133 MHz and 33 MHz PLB frequencies respectively, and does not appear to cause any insurmountable problems in the applications we have studied.

III. DPM Architecture

A number of strategies for saving energy using DVFS techniques have been developed [4]–[7]. We believe that since the most effective way to manage energy consumption is highly dependent on the particulars of the embedded system, a generic power management architecture needs to be flexible enough to support multiple platforms with differing requirements. Part of this flexibility is the requirement to support "pluggable" power management strategies that allow system designers to easily tailor power management for each application. Although excellent results have been obtained with kernel-level approaches for DVFS, we believe that the requirements for simplicity and flexibility are best served by leaving the workings of the DVFS system completely transparent to most tasks, and even to the core of the OS itself. These considerations led to the development of an architecture for policy-guided dynamic power management called DPM.

It is important to note at the outset that DPM is *not* a DVFS algorithm, nor a power-aware operating system such as described in [8], nor an all-encompassing power management control mechanism such as ACPI. Instead, DPM is an independent module of the operating system concerned with active power management. DPM policy managers and applications interact with this module using a simple API, either from the application level or the OS kernel level. Although not as broad as ACPI, the DPM architecture does extend to devices and device drivers in a way that is appropriate for highly integrated SOC processors (but not discussed here). A key difference with ACPI is the extensible nature of the number of power-manageable states possible with DPM.

While DPM is proposed as a generic feature for a general-purpose operating system, our practical focus has been the implementation of DPM for Linux. DPM implementations are expected to be included in embedded Linux distributions for the 405LP and other processors. The following is a brief overview of the DPM architecture. For more complete information see [9].

A. DPM Policies

The DPM *policy* abstraction models the executing system as a state machine. It involves two key concepts - operating points and operating states.

With the fine grained control over system clocking available in processors like the 405LP, a single CPU frequency and/or voltage setting is not suitable to denote a system state for power management. Instead, DPM policies are based on abstract *operating points* which encapsulate the minimal set of interdependent, physical and discrete parameters that define a specific system performance level along with an associated energy consumption. For example, DPM operating points for the 405LP are derived from a dozen separate parameters that include the CPU operating voltage and frequency, bus frequencies etc. Under DPM, the system designer is responsible for defining the useful operating points for the system since they form the basis of a system-specific power management strategy.

Figure 1 shows how the system moves from state to state in response to events: tasks are scheduled, interrupts are handled, the system becomes idle, and sleep mode is entered and exited. Under DPM, each such state is an *operating state*. In an aggressive power management policy, each operating state may be associated with an operating point specific to the requirements of that state. A DPM *policy* data structure is simply a mapping from operating states to operating points. The locations of the state transitions are well defined in the OS, and it is straightforward to insert the appropriate calls to DPM at these points.

A DPM strategy begins with a set of one or more *policies* defined by the system designer. DPM policies are data structures registered with and interpreted by the DPM implementation in the kernel. Policy activation is controlled by an application-specific, executable *policy manager*, also provided by the system designer. The policy manager is optional, as some systems may be effectively power managed by a single policy installed at system initialization. Figure 2 shows the different components and their interactions in a DPM-enabled system.

TABLE I

REPRESENTATIVE DPM POLICIES; CPU FREQUENCIES IN MHZ, VOLTAGE SCALING IMPLIED

Policy	Task+	Task	Task-	Idle
Default	266	266	266	266
IS	266	266	266	33
LS266	266	266	266	33
LS200	266	200	200	33
LS166	266	166	166	33
LS133	266	133	133	33
LS100	266	100	100	33

B. DPM Task States

Several researchers have arrived at the conclusion that an optimal power management policy will require the active participation of power-aware tasks in an operating system capable of task-/event-specific power and performance management [10], [11]. However, we believe that in many systems only a small number of application programs will ever be modified to be power-aware. Therefore, the system must be able to effectively handle mixed workloads of power-aware and conventional tasks. We also do not believe that a single task running on a general purpose system should be able to unilaterally set the system operating point, unless that task also has the authority to act as a complete power-policy manager for the system. The DPM *task state* mechanism for task-specific power management was arrived at as a compromise between competing concerns for simplicity, flexibility, performance and optimum power efficiency.

Under DPM, the various task states are recognized as separate operating states of the system. The task state index of each task is stored in the task structure, and is not interpreted by the core OS. Instead, DPM policies map operating points to the various task states, and whenever a task is scheduled the context switch invokes the DPM layer to actuate the operating point associated with the task's DPM task state. The task state mechanism allows privileged, power-aware tasks to indirectly set their own operating point or the operating points of other tasks by changing task state assignments. Only the policy manager determines the task state to operating point assignment, however, by choosing to activate an appropriate DPM policy.

IV. DPM STRATEGIES

We argue that DPM is a useful abstraction because of its ability to easily implement a wide variety of effective power management strategies. In the following we describe several strategies that we have implemented and tested on experimental platforms based on the 405LP. Some of the strategy descriptions refer to Table I, which abstracts the key mappings of several possible DPM policies for the platforms.

A. Single-policy Strategies

The simplest DPM strategies require only a single policy and no run time policy manager. An example is the *idle scaling* (IS) strategy. In an IS strategy the system runs at full speed when processing the workload (in *task* state), and at reduced frequencies (and perhaps voltages) when idle. The goal in an SOC environment is to reduce CPU and bus frequencies to the lowest possible point that still meets continuous DMA requirements (e.g., LCD refresh) and provides acceptable latency when resuming from idle. For the 405LP we generally use 33 MHz as the idle operating point. Another simple one-policy strategy could assign a different operating point to each task state, and power-aware tasks or the policy manager would manage system power and performance by adjusting task state assignments. This would be one way of implementing a *job-classifying* power manager [12].

B. Multi-policy Strategies

Multi-policy strategies under DPM would include the well-known load scaling (LS) strategies, also referred to as interval methods [4], [6], [12], [13]. These strategies attempt to balance the system operating point with current or predicted processing demands, in an attempt to run the system at the most efficient operating point with minimum idle time. LS strategies are simple and effective for general workloads. In our experimental setup examining load scaling, each voltage/frequency level is represented by a DPM policy that maps that voltage frequency/level to the default *task* state. In Table I, LS266, LS200, LS166, LS133, and LS100 are examples of such policies. Our LS policy manager sets the policy using simple heuristics based on system load over a previous interval. Note that all of our LS policies also incorporate idle scaling, which helps mitigate any inefficiencies caused by the selection of a non-optimal operating point during a load-scaling interval. Other types of multi-policy strategies that would be straightforward to implement with DPM include strategies that manage the system based on the state of the system energy source.

C. Task-scaling Strategies

We are particularly interested in strategies that combine load scaling, which has proved to be "good enough" for general-purpose applications without real-time constraints, and task-specific requirements. We refer to these hybrid systems as *application scaling* (AS) strategies. Video decoding is commonly used as an example of an important workload that is difficult to power manage without application participation [10], [11], [14], so we developed an AS strategy for an experimental, multi-threaded MPEG4 video/audio decoder for Linux. This power-aware video decoding approach does not require extensive analysis and prediction of processing requirements or the implementation of new scheduling mechanisms, but still achieves good results. We present this AS example to illustrate an interesting use of DPM's policy mechanisms without which the application would need significant additional complexity to make it power-aware.

We made two simple changes to the video decoding threads to make it easier to power manage this application with our LS policy manager. We modified the video decoder to begin processing the next frame immediately upon completion of the current frame, rather than idling. For this *work-ahead* phase the decoder sets its operating state to *task–*, which hints to the LS policy manager that the *work-ahead* activity is not

TABLE II
AVERAGE POWER CONSUMPTION FOR MP3 (IN MW)

Strategy	System	Base Card (BC)	BC Normalized to Default
Default	1417.7	548.0	100%
IS	1096.4	286.5	52%
LS	1026.7	232.7	42%

essential. In turn, this lowers the system load as computed by the LS policy manager, letting it reduce the operating point by switching to a policy with a lower *task* operating point. If the video thread begins to fall behind, the thread marks itself to run in the *task+* state, which is always mapped to the highest performance operating point. The thread remains in the *task+* state until it has caught up again. Our approach does not modify the audio thread, however, its impact is captured by the system load as monitored by the LS policy manager and the recognition of *work-ahead* and *catch-up* phases in the video threads.

V. DPM RESULTS

DPM has been implemented under PowerPC Linux 2.4, and strategies and policy managers have been created for multiple 405LP platforms. One such platform is a high-function PDA reference design (32 MB SDRAM, 320x240 TFT display at 16bpp, USB and other peripherals). Table II shows the results of total system power and "base card" power for MP3 playback on this platform (with the display off). The "base card" power measurement includes the 405LP, memory, I/O power required to drive peripherals on a "personality" card, and power conversion losses. The default strategy runs the application at full voltage and frequency, and the IS strategy scales voltage and frequency at idle. Since MP3 is a relatively light workload for the 405LP, the LS policy manager normally runs the application at the lowest voltage, and further scales the frequency at idle, resulting in significantly lower power than the other strategies.

Table III shows the results for MPEG4 decode with different DPM strategies. We gauge the effectiveness of our AS policy for the MPEG4 decoder by comparing its performance to a synthetic best-case strategy we refer to as frame-scaling (FS). The FS strategy is assumed to have perfect knowledge of the processing requirements of each video frame (including audio decoding overhead), and the ability to select the most power-efficient operating point for each frame based on this knowledge. We synthesized the FS result by running the decoder on a test clip (30 fps at 352x288) at every operating point and collecting power and time measurements for every frame. The power for the optimal scaling strategy, FS, is then obtained by adding the lowest power for each frame while meeting the time constraint (33ms for 30fps). Our relatively simple AS strategy was able to achieve system-level power consumption very close to the "ideal" FS strategy.

VI. CONCLUSION

We have presented hardware and software architectures that provide practical, effective dynamic power management

TABLE III
AVERAGE POWER CONSUMPTION FOR MPEG4 (IN MW)

Strategy	System	Base Card (BC)	BC Normalized to Default
Default	2762.7	941.6	100%
IS	2630.4	834.9	89%
LS	2538.2	747.4	79%
AS	2463.6	703.8	75%
FS	2464.2	700.8	74%

for SOC processors and embedded systems. We expect to see continuing improvement in the dynamic power/performance range of embedded processors by advances in semiconductor technology, application-specific acceleration, and finer-grained clock and power gating. We will to continue to develop DPM and DPM implementations to take advantage of these new technologies.

ACKNOWLEDGMENT

The DPM architecture was developed and implemented in conjunction with Hollis Blanchard and Robert Paulsen of the IBM Linux Technology Center, and Matthew Locke and Mark Orvek of MontaVista Software.

REFERENCES

[1] L. Benini, A. Bogliolo, and G. D. Micheli, "A Survey of Design Techniques for System-Level Dynamic Power Management," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 10(2), pp. 299–316, June 2000.

[2] K. Nowka *et al.*, "A 32-bit PowerPC Syetem-on-a-Chip with Support for Dynamic Voltage Scaling and Dynamic Frequency Scaling," *IEEE Journal of Solid-State Circuits*, vol. 37(11), pp. 1441–1447, Nov. 2002.

[3] Intel Corporation, "Intel PXA255 Processor Developer's Manual," Mar. 2003.

[4] K. Govil, E. Chan, and H. Wasserman, "Comparing Algorithms for Dynamic Speed-Setting of a Low-Power CPU," in *Proc. ACM Int'l Conf. on Mobile Computing and Networking*, 1995, pp. 13–25.

[5] T. Pering, T. Burd, and R. Brodersen, "The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms," in *International Symposium on Low Power Electronics and Design (ISLPED)*, August 1998, pp. 76–81.

[6] Dirk Grunwald and Philip Levis and Keith I. Farkas amd Charles B. Morrey III and Michael Neufeld, "Policies for Dynamic Clock Scheduling," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, October 2000.

[7] K. Flautner and T. Mudge, "Vertigo: Automatic Performance Setting for Linux," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Dec. 2002.

[8] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat, "ECOSystem: Managing Energy as a First Class Operating System Resource," in *Tenth International Conference on Architectural Support for Programming Languages and Operating Systems*, October 2002.

[9] IBM and MontaVista Software, "Dynamic Power Management for Embedded Systems," http://www.research.ibm.com/arl/projects/dpm.html, November 2002.

[10] J. Pouwelse, K. Langendoen, and H. Sips, "Application-directed voltage scaling," *IEEE Transactions on Very Large Scale Integration Systems, (preprint)*, 2003.

[11] C. Poellabauer and K. Schwan, "Power-Aware Video Decoding using Real-Time Event Handlers," in *5th International Workshop on Wireless Mobile Multimedia (WoWMoM)*, September 2002.

[12] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for Reduced CPU Energy," in *First Symposium on Operating Systems Design and Implementation*, November 1994.

[13] M. Fleischmann, "LongRun Power Management," Transmeta Corporation Whitepaper, January 2001.

[14] D. Son, C. Yu, and H.-N. Kim, "Dynamic Voltage Scaling on MPEG Decoding," in *International Conference on Parallel and Distributed Systems (ICPADS)*, June 2001, pp. 633–640.