LART Lessons Learned: cpufreq

J.A.K. (Erik) Mouw, Koen Langendoen, Johan Pouwelse
UbiCom program
Delft University of Technology
PO BOX 5031, 2600 GA Delft, The Netherlands
{erik,koen,pouwelse}@ubicom.tudelft.nl

Abstract

In order to run as long as possible on a single battery, battery-powered computers need to be efficient. A large part of that efficiency can be gained by using low-power hardware, but software can also help to reduce power consumption. One way to do that is to let the OS control the CPU frequency and core voltage. This paper will explain the backgrounds of power consumption in CPUs and how clock and voltage scaling can help to decrease the power consumption. It will show the current Linux implementation (cpufreq) and compare it with other implementations.

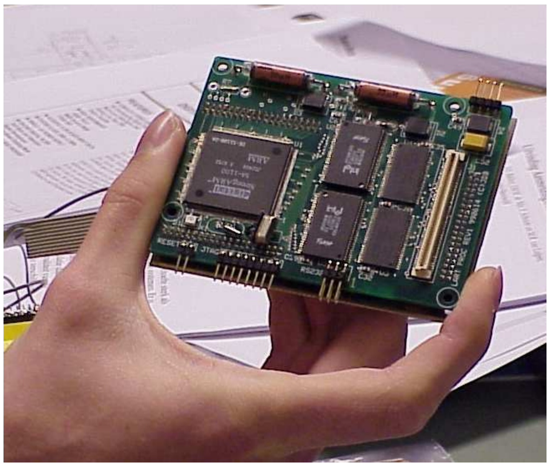

Figure 1: LART processor board

1 Introduction

The Mobile Multimedia Communications project (MMC, 1996 – 2000)[MMC] and the Ubiquitous Communications program (UbiCom, 1998 – 2002)[UbiCom] at the Delft University of Technology are two related projects that research high data rate cellular networks. The MMC project focused on multimedia communication protocols and applications (text, audio, video) for mobile use, while the UbiCom program extended this to augmented reality and wearable computer systems. Both projects needed a mobile computer platform to test their theories. This platform had to be small, low power, powerful, affordable, and extendible. To solve the tension between these requirements, the emphasis was put on best computing power per watt. Unfortunately there was not such a computer platform available on the market around 1997, so MMC project members decided to design such a system themselves: the Linux Advanced Radio Terminal (LART)[LART].

2 LART

The LART is build around the Intel StrongARM SA-1100 CPU, an embedded processor with an excellent power/MIPS ratio and a large set of built-in peripherals[SA-1100]. The CPU normally runs at 221 MHz, at which speed it delivers a performance comparable to an Intel Pentium 200. The SA-11x0 CPU family is well supported by the Linux operating system, and the mature userland utilities (gcc, etc.) and openness of Linux allows for easy integration of special purpose device drivers.

Figure 1 shows the LART processor board (7.5×10 cm), holding the CPU, 32 MB of EDO DRAM, 4 MB of Flash boot ROM, a connector for two (simple) serial ports, a JTAG debug interface connector, a high-speed extension connector and a low-speed extension connector (at the back of the board). The complete LART processor board weighs only 50 g.

An extension board known as the Kitchen Sink Board (KSB) can be connected and provides a PS/2 interface (2×) for keyboard and mouse, USB client interface, IrDA infra-red link, IDE disk interface, stereo 16 bit 48 kHz audio output, mono 12 bit 26 kHz audio I/O (speakers and microphone), telephony interface, touch panel interface, and an LCD interface. Both the LART and the KSB design files are available under an open license allowing everybody to build boards for themselves or even improve the designs.

At full CPU utilization the processor board consumes about 1 W, which allows it to run for several hours from a single 4.5 V battery. However, the LART design was flexible enough that frequency and voltage scaling could be added at a later stage. This allows the CPU to run at lower frequencies and voltages thereby saving energy. The amount of energy saved depends on the type of application: applications with different CPU load patterns yield different amounts of energy savings. This paper will focus on the Linux implementation of frequency and voltage scaling, Pouwelse et. al. discuss the power saving techniques for different kinds of workloads[Pouwelse].

3 Frequency and voltage scaling

To understand the advantages of frequency and voltage scaling, we will first discuss the theory behind it. Digital CMOS (Complementary Metal-Oxide Semiconductor) circuits as used in the majority of modern microprocessors have both static and dynamic power consumption[Pouwelse][Burd][Ishira]. The static power consumption is caused by bias and leakage currents, and can usually be ignored for designs that consume more than 1 mW of power.

The dynamic power consumption is caused by the logic transactions of the gates in the digital circuit: every charge and subsequent discharge of the gate capacitance drains power. The dynamic power consumption can be modeled by

$$P_{dynamic} = \sum_{i=1}^{N} C_i \cdot f_i \cdot V_{DD}^2 \qquad (1)$$

where N is the total number of gates in the circuit, C_i the load capacitance of gate g_i, f_i the switching frequency of gate g_i, and V_{DD} the supply voltage. Equation 1 clearly shows that lowering V_{DD} yields the largest reduction in power. However, reducing V_{DD} will increase the circuit delay, which can be described by

$$\tau \propto \frac{V_{DD}}{(V_G - V_T)^2} \qquad (2)$$

where τ is the propagation delay of the CMOS transistor, V_T the threshold voltage, and V_G the input gate voltage. The propagation delay restricts the maximum clock frequency for any clock driven digital CMOS circuit. Equations 1 and 2 show there is a trade-off between switching frequency and supply voltage: digital CMOS circuits (and hence microprocessors) can only operate at a lower supply voltage when the clock frequency is lowered to compensate for the increased propagation delay.

Equation 1 can be simplified by assuming that all gates g_i create a collective switching capacitance C operating at a common switching frequency f:

$$P = \alpha \cdot C \cdot f \cdot V_{DD}^2 \qquad (3)$$

This equation shows that lowering the clock frequency linearly decreases power, but that voltage reduction results in a squared power reduction. Figure 2 illustrates this conclusion for a LART running a CPU intensive workload at various clock frequencies.

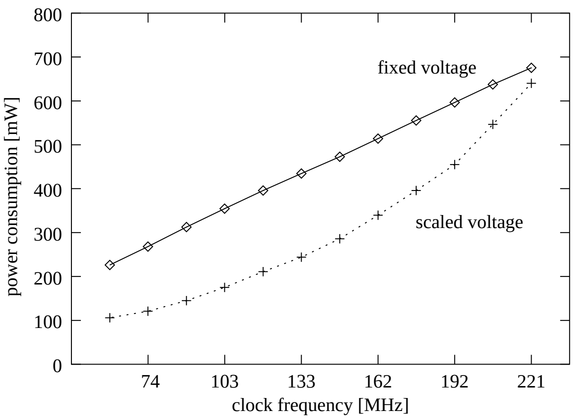

Figure 2: Total power consumption for a LART running a CPU intensive workload

An important observation is that frequency scaling alone only saves *power*, but not *energy*. Running a task at a decreased clock frequency makes that it takes longer to complete that particular task. The task completion time is proportional to $1/f$, and hence the total energy consumed remains the same. Combining frequency scaling with voltage scaling will save power *and* energy because V_{DD} can be scaled with respect to f.

4 Implementation

To exploit the potentials of frequency and voltage scaling, we implemented it on our LART computer platform. The LART frequency and voltage scaling consists of a hardware and software part. The SA-1100 natively supports frequency scaling: the clock frequency can be set in 15 MHz steps from 58 to 221 MHz. It does not, however, support voltage scaling. Therefore the LART design includes additional circuitry to control the core voltage supply.

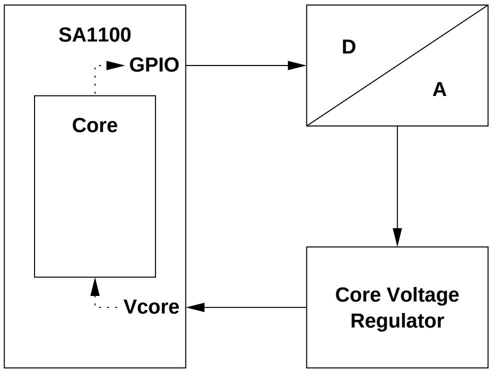

Figure 3: LART voltage scaling hardware

Figure 3 shows how the CPU controls the core voltage: eight General Purpose I/O (GPIO) pins are used to set the output voltage of an 8 bit digital to analog converter (DAC), which on its turn controls the core voltage regulator. The core voltage is thus completely software controlled, and there are a couple of hardware safety measures to prevent the CPU from exposing itself to excess voltages.

The SA-1100 is an embedded CPU and among its built-in interfaces is a memory controller, which should be programmed to generate the necessary waveforms for the memory connected to the system (e.g. SRAM and DRAM). This memory controller is directly driven by the core frequency oscillator, so it has to be reprogrammed at each clock speed change. The SA-1100 is special in that it needs software to reprogram the core voltage and memory settings: most other CPUs have external memory controllers independent of the CPU frequency and hardware controlled core voltage regulators. Figure 4 shows the order of events that have to happen when increasing the clock speed. Decreasing the clock speed reverses the order: decrease clock speed, decrease core voltage, tighten memory settings. When switching to a higher clock speed, the generated memory waveforms are too wide for the current frequency speed and hence decrease the available memory bandwidth. However, this situation only exists for such a small amount of time that it does not decrease the system performance.

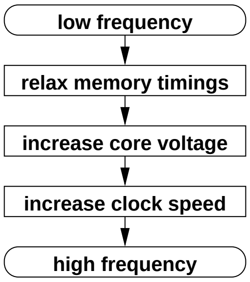

Figure 4: Order of execution for switching to a higher clock speed

The initial Linux driver for the LART clock and voltage scaling hardware exactly followed the procedure depicted in Figure 4. The switching was controlled from a file in the `/proc` file system: in this way the mechanism was implemented in the kernel, while the policy of *when* to change clock speed could be implemented in userland. The initial implementation worked well for a simple system with only the LART processing board, but it did not have enough flexibility to support a LART system with more hardware (like hard disk, PCMCIA interface, etc.), or a system build around a different kind of CPU.

5 Cpufreq

Quite some kernel drivers depend on the `udelay()` function for timed access to hardware. For the ARM family, this function is implemented as a busy wait that uses the `loops_per_jiffy` variable to check if the requested number of micro seconds already passed. The value of the `loops_per_jiffy` variable is derived during the famous *Calibrating delay loop* event when the kernel boots. Because `loops_per_jiffy` depends on the CPU frequency, it needs to be adjusted after a speed change. Fortunately the variable does not need to be recalibrated: it is directly proportional to the CPU frequency so it can be calculated.

When frequency and voltage scaling support for several 80x86 CPUs was added, it became clear that those CPUs use a timer independent from the CPU core frequency to calculate the amount of time to be spend in `udelay()`. Also, these CPUs did not need to reprogram their memory controller. Therefore, Russell King designed a flexible framework for clock and voltage scaling: cpufreq [Cpufreq].

Cpufreq separates the act of changing the CPU speed from the other measures that have to be taken upon a speed change. At kernel initialization, the CPU dependent driver needs to register its `validatespeed()` and `setspeed()` functions with cpufreq. All other hardware drivers that depend on the CPU frequency also need to register themselves with cpufreq so they can be notified for speed changes. A 80x86 cpufreq driver only need to register its `validatespeed()` and `setspeed()` functions, while the SA-1100 driver also has to register the functions that change the memory timings. The value of `loops_per_jiffy` is automatically changed by cpufreq; it is not necessary for 80x86 CPUs, but it is nice that `/proc/cpuinfo` gives an indication of the current CPU speed, even though it is a bogus one.

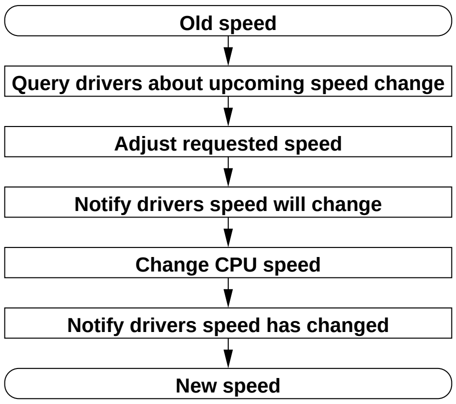

Figure 5: Cpufreq order of execution

Figure 5 shows the cpufreq order of execution at a CPU speed-change request. First of all, all registered drivers are queried about the speed range they can tolerate. A driver that for some reason (like the SA-1100 LCD controller that needs a certain amount of bandwidth) currently can't accept a speed range can limit the requested range to the range it is able to handle. If the new CPU frequency is out of the range the drivers can currently tolerate, it is adjusted to fall within the range. The drivers are then notified about the upcoming CPU speed change, so they can decide to adjust certain parameters. For example: when going to a faster speed, the SA-1100 memory driver will relax the memory timings. Next, the CPU speed will be changed to the requested value using the CPU `setspeed()` function. After that, all drivers will be notified that the CPU speed has changed, so they can adjust their parameters. For example: when going to a slower speed, the SA-1100 memory driver will tighten the memory timings. This completes the speed change and the system can continue to do whatever it was doing before the speed change. Again, the kernel only implements the switching mechanism; the policy can be controlled through a sysctl interface by a userland process.

6 Discussion

The flexible cpufreq framework supports StrongARM SA-1100, StrongARM SA-1110, ARM Integrator, VIA Longhaul, AMD Elan, AMD PowerNow K6, and Intel SpeedStep, while work is underway to add support for AMD PowerNow K7. The current cpufreq implementation is stable and scheduled to be included in Linux-2.5. Following the Unix design rules, cpufreq only implements the *mechanism* to change the CPU speed; the *policy* of when to change speed is left to userland.

A simple userland scheduler that changes the CPU speed by observing the CPU idle time works nice for most workloads, but it breaks down at bursty real-time tasks like real-time video decoding. The CPU speed scheduler will select a low clock frequency when the video decoder decodes low-complexity frames, but it will be too late to select a high clock frequency when the video decoder encounters a high-complexity frame. As a result, the frame will be decoded too late which will be visible to the user. The CPU speed scheduler can also decide to select a high clock frequency so all frames will be completed in time, but in this case the CPU will waste energy. Pouwelse et. al. show that a power aware video decoder is able to combine close-to-optimal energy savings with real-time decoding performance[Pouwelse2][Pouwelse3].

7 Related work

There are two software frameworks for CPU power management. Advanced Power Management (APM)[APM] is an older standard still widely in use that allows the CPU to enter a low power state when executing the idle loop. APM only implements an on/off power savings approach: intermediate power saving levels are not available, even when the CPU is able to switch to multiple performance levels.

The Advanced Configuration and Power Interface (ACPI)[APCI] is the successor of APM. ACPI has a fine-grained CPU power management interface that can be controlled by the OS. Unfortunately, the standard also allows the ACPI BIOS to control the CPU speed without notifying the OS thereby removing the ability for userland scheduling tools to control the CPU speed policy. Another disadvantage of ACPI is that it depends on the BIOS implementation. In many cases, frequency and voltage scaling is not implemented in the BIOS, thereby missing an opportunity to save energy. Fortunately, work is being carried out to fit cpufreq within the Linux ACPI subsystem.

A hardware approach to CPU power management is implemented in the Transmeta Crusoe TM5400 CPU[Crusoe] which implements frequency and voltage scaling in its microcode ("LongRun"). This means that the policy is implemented in the CPU and operates without knowledge about the applications. The scheduler works the same as the simple scheduler described in the previous section, and thus has the same limitations.

8 Conclusions

A well designed experimental computer platform can lead to interesting results: the flexible LART platform allowed to exploit the theoretical power and energy savings of frequency and voltage scaling. The resulting software framework was used, together with other implementations, to get at the generic cpufreq frequency and voltage scaling driver which allows the OS to control the CPU power consumption. Other approaches to control the CPU power either lack the fine grained control cpufreq offers, or try implement the power saving policy at the wrong place.

Cpufreq only implements the mechanism of frequency and voltage scaling. The policy of *when* to change CPU speed is still an active area of research. It is clear that the simple speed scheduler as described in Section 5 does not yield optimal power savings and fails for bursty real-time tasks, but the ideal scheduler still has to be written[Pouwelse3]. As Linus Torvalds remarked: "The really interesting things happen in userland".

9 Acknowledgements

This work was carried out within the MMC project and the UbiCom program and funded by the Dutch Foundation of Applied Sciences (STW) and the TU Delft DIOC research program. We would like to thank Jan-Derk Bakker for designing an excellent low-power platform and Russell King for the cpufreq framework and the many discussions we had.

References

[APCI] Compaq Computer Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., Toshiba Corporation, *Advanced Configuration and Power Interface, Revision 2.0*, July 2000.

[APM] Intel Corporation, Microsoft Corporation, *Advanced Power Management (APM) BIOS Interface Specification, Revision 1.2*, February 1996.

[Burd] T. Burd, R. Brodersen, *Processor design for portable systems*, Journal of VLSI Signal Processing, Aug/Sept 1996.

[Cpufreq] D. Jones, R.M. King, J.A.K. Mouw, J.A. Pouwelse, A. van der Ven, *Cpufreq homepage*, `http://www.lart.tudelft.nl/projects/scaling/`

[Crusoe] Transmeta Corporation, *The technology behind the Crusoe processor*, `http://www.transmeta.com/crusoe/download/pdf/-crusoetechwp.pdf`

[Ishira] T. Ishihara, H. Yasuura, *Voltage scheduling problem for dynamically variable voltage processors*, ISLPED, Aug. 1998.

[LART] J.-D. Bakker, M.A.H.G. Joosen, J.A.K. Mouw, *Linux Advanced Radio Terminal*, `http://www.lart.tudelft.nl/`

[MMC] *Mobile Multimedia Communications Project*, `http://www.mmc.tudelft.nl/`

[Pouwelse] J.A. Pouwelse, K. Langendoen, H. Sips, *Voltage scaling on a low-power microprocessor*, Mobile Computing Conference (MOBICOM), Jul. 2001.

[Pouwelse2] J.A. Pouwelse, K. Langendoen, R.L. Lagendijk, H. Sips, *Power-aware video decoding*, Picture Coding Symposium (PCS), 2001.

[Pouwelse3] J.A. Pouwelse, K. Langendoen, H. Sips, *Energy priority scheduling for variable voltage processors*, International Symposium on Low-Power Electronics and Design (ISLPED), Aug 2001.

[SA-1100] *Intel StrongARM SA-1100 microprocessor developer's manual*, available at `http://www.lart.tudelft.nl/doc.php3`

[UbiCom] *Ubiquitous Communications Program*, `http://www.ubicom.tudelft.nl/`