EMBECOSM

The Software Drained my Battery

Jeremy Bennett, Embecosm
Kerstin Eder, University of Bristol, Department of Computer Science

Has downloading and running the latest applications also drained your smartphone's battery? In this article we look at the great potential of making software engineering more energy aware. We start at the hardware.

In recent years, hardware designers have become very good at low power design. Multiple voltage domains, clock gating, dynamic frequency scaling and a host of other techniques have helped reduce power consumption. It is a never ending battle as dimensions shrink to just 10s of atoms, and leakage becomes an ever more pressing problem.

For a long time, energy efficiency has been seen as a hardware problem. Yet software can undo all the design efficiency at a stroke. Famously a Linux implementation wasted 70-90% of its power, simply because a blinking cursor woke up the entire system several times a second [1]. One of the authors (Bennett) was involved in a commercial project, where the design team found they had to increase clock frequency (and hence power consumption) three fold because a standard audio codec caused excessive processor stalls through cache conflicts. That project was canceled shortly afterwards.

There are three main factors contributing to power loss:

- static leakage—mitigated by reducing voltage;
- dynamic leakage—mitigated by reducing frequency and switching; and
- number of components—mitigated through smaller, simpler silicon and less memory.

Note in particular that reducing voltage is a quadratic gain, and that reducing frequency is a double gain because it also allows voltage to be reduced. With chip voltages ranging from 0.6V to 1.5V, there is the potential of 10x gain to be had.

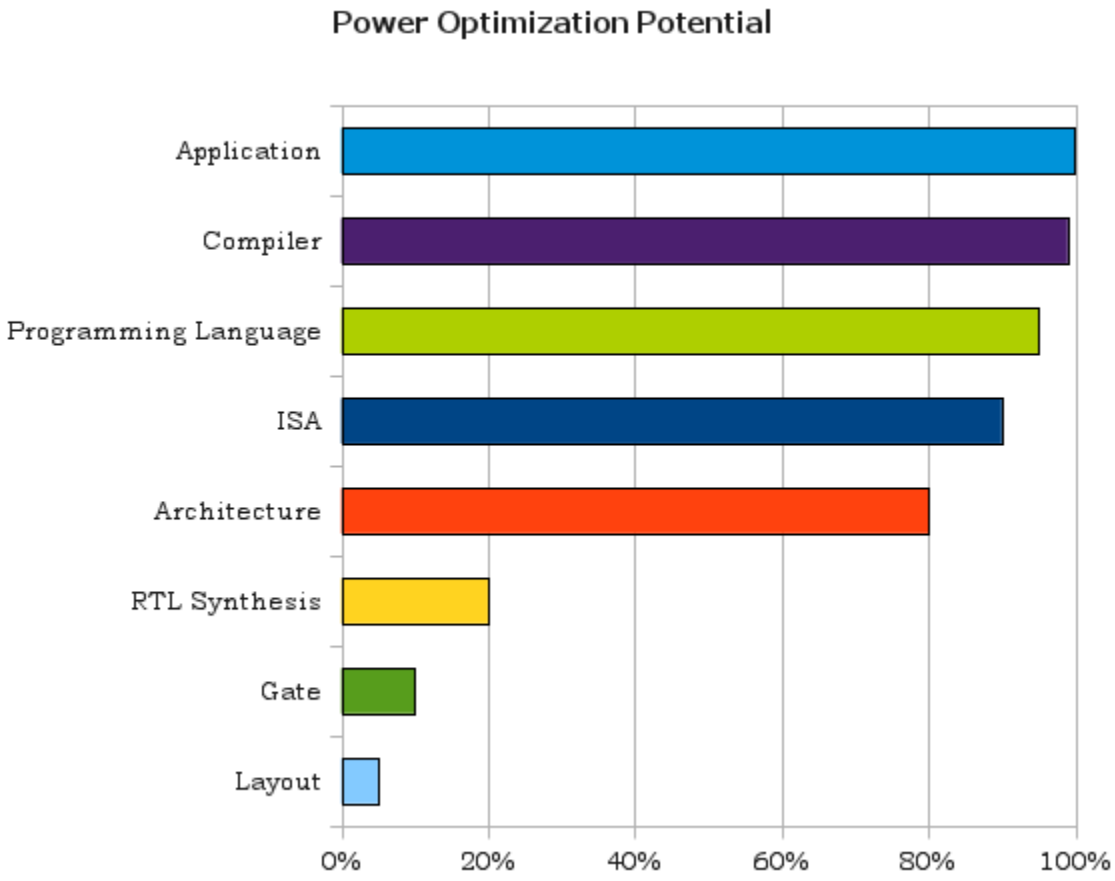
How to tackle energy efficiency at a system level has been known for well over a decade. In their 1997 paper [2], Roy and Johnson summed up how to align software design decisions with energy efficiency as a design goal. Their key steps are (in the given order):

- choose the best algorithm to fit the hardware;
- manage memory size and memory access through algorithm tuning;
- optimize for performance, making best use of parallelism;
- use hardware support for power management; and
- generate code that minimizes switching in the CPU and data path.

Seeing that this understanding already existed back in 1997, it is perhaps surprising to see that no major advances have been made since then. Much of the current literature is focused on advancing one or several of the above steps with marginal returns. In the context of multi-threaded software running on multi-core hardware many of these approaches no longer work. A step change is needed to push technology towards the 10x improvements that result in a noticeable difference in practice. This requires a fundamental re-evaluation of system design methods in the context of energy efficiency.

Traditionally, researchers and engineers work within one or perhaps two layers of the system stack with very limited overlap, e.g. software engineers, computer architects or hardware designers. However, energy-aware computing is a challenge that requires investigating the entire system stack from application software and algorithms, via programming languages, compilers, instruction sets and micro architectures, through to the design and manufacture of the hardware. In 2010 Mentor Graphics and LSI Logic identified

that, while optimization at synthesis level could potentially save 5% power, optimization at architectural level could save 80%. But even then they did not consider the potential for savings even further up in the compiler, programming languages and software. An updated version of their graph might look like this:



After Mentor Graphics and LSI Logic 2010

This is because energy is consumed by the hardware performing computations, but the control over the computation ultimately lies within the software and algorithms, i.e. the applications running on the hardware. Industry is waking up at last to the fact that, while hardware can be designed to save a modest amount of energy, the potential for savings is far greater at the higher levels of abstraction in the system stack [3].

The greatest savings are expected from energy-consumption-aware software. Addressing the challenge of energy-aware computing requires collaboration between engineers and researchers from all the above named areas and a good understanding of the applications that will drive software and hardware development in the future.

Any engineering team must work together across hardware *and* software disciplines to address energy issues throughout the entire system stack.

- Synthesis/RTL. This comes late in the process, and power estimation tools while accurate are fabulously slow (weeks to run for a big chip).
- Functional block level. Tools like *Wattch* [4] offer architectural power estimation that runs 1000x faster then synthesis tools, yet is accurate to within 10% of layout level estimation tools.

- Instruction Set Architecture. Minimizing the Hamming distance between pairs of instructions, and partitioning register files are clear wins here.
- Compiler. Profile directed optimization is essential to be able to take advantage of both models of hardware power and (once silicon is available) measurements of power. The EU funded MILEPOST project [5], in which the UK was a big player offers an ideal framework for this.
- Programming languages. Annotations to specify power consumption budgets during programming need to become integrated parts of new programming languages, which will be dedicated to giving programmers full control over software power consumption. Additionally, programmers may decide to trade accuracy for power by utilizing advanced approximate data types [6] that take advantage of new optimizations in hardware [7].
- Algorithms and applications. The tool chain must allow early design space exploration. In the first instance a programmer must get a report on the power consumption of the program they write. Longer term, the tools should be able to optimize for power, just as today they optimize for time and/or space.

In summary, there is huge potential for power savings when looking at the entire system stack, especially at the higher levels. **To unlock this potential, energy efficiency must be promoted to a first class system design goal.** We can identify some key steps to achieve this:

- We need to bridge the gap between hardware and software design. This requires a change of culture in engineering teams and tool providers.
- New tools are required. Initially to communicate power consumption to developers, especially software engineers. Later, to automatically optimize so that the specified power budgets are being met. Tool providers will play a key role in bridging the hardware-software gap.
- Education and training of engineers will need extending. Just as today a hardware designer knows how to optimize for area, performance or power, future software engineers will need to work in a multi-dimensional design space that includes power in addition to traditional software design metrics such as memory usage and performance.
- A critical part to drive this forward is raising end user awareness. If end users made non-functional requirements such as energy budgets part of the specification for an application, then engineers would need to design to meet these expectations from the start.

Making system design more energy efficient is a considerable challenge for both the engineering and the research community. This challenge calls for collaboration across the board. We have a long way to go. So, next time the software drains your battery, you know we've still not quite made it. When we can select from new apps based on their energy rating, similar to the way we buy light bulbs or white goods today, and when smartphones don't need recharging for weeks, then we'll be almost there.

References

1 E. Sperling and P. Chatterjee. 16 June, 2011. The Tao of Software. *Chip Design Magazine Low Power Engineering* online community blog post. chipdesignmag.com/lpd/blog/2011/06/16/the-tao-of-software.

2 K. Roy and M.C. Johnson. 1997. Software design for low power. In W. Nebel and J. Mermet (Eds.) *Low power design in deep submicron electronics*. Kluwer Nato Advanced Science Institutes Series, Vol. 337, Norwell, MA, USA, pp 433-460.

3 Chris Edwards. 15-21 June, 2011. Lack of software support marks the low power scorecard at DAC. *Electronics Weekly*.

4 D. Brooks, V. Tiwari, M. Martonosi, “Wattch: A framework for architecture-level power analysis and optimizations,” Proc. 27th International Symposium on Computer Architecture (ISCA), pp. 83-94, 2000.

5 Fursin et al. MILEPOST GCC: Machine learning based research compiler. *GCC Summit*, Ottawa, Canada, 2008.

6 A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze and D. Grossman, “EnerJ: Approximate Data Types for Safe and General Low-Power Computation” In Proc. of PLDI, June 2011.

7 J.Y.F. Tong, D. Nagle and R.A. Rutenbar, Reducing Power by Optimizing the Necessary Precision/Range of Floating-Point Arithmetic. *IEEE Transactions on VLSI Systems*, 8(3), pp 273-286, June 2000.

About the Authors

Dr Jeremy Bennett is Chief Executive of Embecosm Limited. Embecosm (www.embecosm.com) provides open source services, tools and models to facilitate embedded software development with complex systems-on-chip. Contact him at jeremy.bennett@embecosm.com.

Dr Kerstin Eder is a Senior Lecturer in Computer Science at the University of Bristol, specializing in Design Verification. During 2011 she has been on a Royal Academy of Engineering funded Fellowship at XMOS in Bristol, investigating “State of the Art Power-Aware System Design”. She gratefully acknowledges the support of both the RAE and XMOS in carrying out the work described here. Contact her at kerstin.eder@bristol.ac.uk.

This article was originally published in the NMI Yearbook 2011-12.