# Energy-efficient parallel software for mobile hand-held devices

Antti P. Miettinen
*Nokia Research Center*

Vesa Hirvisalo
*Helsinki University of Technology*

## Abstract

Energy efficiency is an essential design criterion for mobile hand-held devices. We underline the issue that parallel programs for mobile devices must be efficiently executable on the development platforms in addition to their efficient execution on the target hardware. Further, we argue that with the proliferation of increasingly complex heterogeneous MPSoC architectures, seamless performance and energy estimation needs to become an integral part of the software development flow.

We review the present reality of mobile software development and the run-time behavior of the software on the devices. Considering the future development environments, we highlight how they can benefit from modern compiler technology and discuss the feasibility of providing rapid feedback about performance and energy consumption characteristics of the software while it is being developed.

## 1 Introduction

In this paper, we consider the development of parallel software for mobile hand-held devices. On one hand, parallel programming has the potential to offer *both* performance and energy-efficiency gains for the devices. On the other hand, parallel programming combined with novel architectures (e.g., many-core with NUMA), raises the risk of creating software with high energy consumption and low performance. The outcome is often in the hands of the software developer.

Cellular phones have for a long time employed multiprocessing and energy-efficient designs. Increasing parallelism is also the trend for the mobile application subsystem [1]. Despite their small size, the descendants of cellular phones include complex subsystems embedded inside the device. Especially when the forthcoming hardware architectures are considered, the complexity of the software/hardware interactions is making the performance and energy consumption behavior increasingly non intuitive. Therefore the developer must rely on the tools available even in the era of novel parallel programming methods that are now emerging.

Cross-platform simulators are often used in the software development flow for mobile hand-held devices. Speed is essential for such simulators, but also difficult considering the forthcoming architectures. Recently, simulators for massive parallelism have been studied and demonstrated [2]. Also, using partial execution for performance prediction has been studied [3]. Further, performance prediction and regression models have been considered to implement fast simulation [4]. Our work is based on such developments, our experience with mobile systems, and our previous research.

The structure of this paper is the following. In the context of parallel programming (Section 2), we review the hardware and software found in mobile hand-held devices and the related software development practices (Sections 3 and 4). Among the challenges in the development (Section 5), we underline the importance of efficient execution of programs on the development platforms and the need for integrating seamless performance and energy estimation into the normal software development. After outlining the difficulties involved, we discuss the use of parameterized abstract models and highlight how the development environments can benefit from modern compiler technology (Section 6) and then present some concluding remarks.

## 2 Background

Energy efficiency is a central theme in the design of mobile hand-held devices. The increasing use of always online applications, multimedia, high speed wireless networking, large displays, etc. are making the challenge continually more demanding. Also, software trends towards increasing use of e.g., web applications and dynamic programming languages are making the optimiza-

tion of energy efficiency ever more important. Additionally, even if the available energy would not be a limiting factor, the power budget of roughly three Watts for a hand-held device remains a valid rule because of thermal concerns [5].

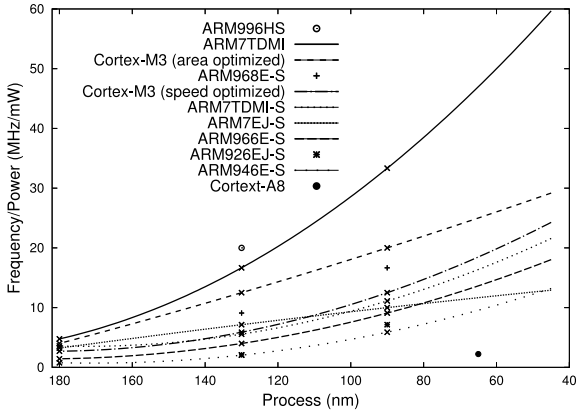


*Figure 1: Clock speed per power for a collection of ARM processor cores.*

Even though parallel hardware is often viewed as a challenge, it is also an opportunity for mobile devices because of the better energy efficiency of parallel processing when compared to sequential designs of comparable performance. Figure 1 shows an overview of data collected from ARM public web pages about various ARM processor cores with polynomial extrapolation curves for cores with three data points. As can be seen, the clock speed achievable with given power budget is at least an order of magnitude higher for small low performance cores than for bigger high performance cores. Even though larger cores have the potential to perform more work within one cycle, this advantage is often diluted by the fact that the performance of modern software tends to be limited by memory effects as will be shown later in this paper.

Performance scaling is an important aspect to parallelism for mobile workloads, which exhibit varying levels of required performance. DVFS [6] has been used to optimize the device operation based on this variation. Figure 2 shows the energy per compressed data for the deflate compression algorithm at various operating points for an ARM11 based SoC along with the corresponding data point for a statically clocked ARM9 based SoC. Even with voltage scaling, the ARM11 based chip fails to achieve the energy efficiency of the ARM9 based one. This suggests that controlling the activity of cores could offer a more efficient performance scaling mechanism than DVFS.

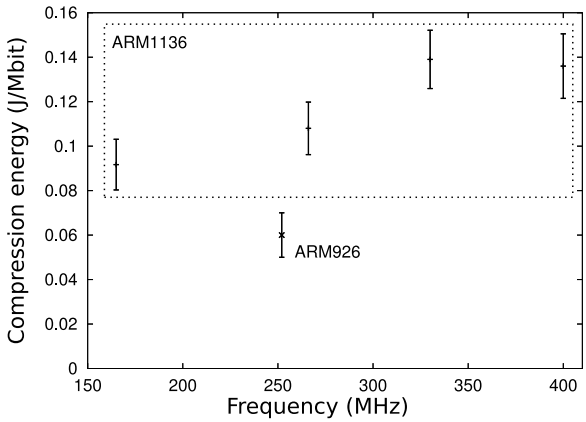


*Figure 2: Deflate compression energy efficiency.*

## 3 Mobile hardware

Parallel architectures are common for mobile hand-held devices. A general purpose processor coupled with a digital signal processor has been the standard configuration for even the most basic mobile phones. Energy efficiency optimization requirements have steered the configurations towards heterogeneous designs where each core has been dedicated for a particular purpose. However, trends in hardware development are towards employing parallel processing also within the subsystems of a device.

In contemporary mobile hand-held devices, the two major subsystems requiring high processing performance are the application subsystem and the wireless modem. Despite similarities in their deployment structure, the design process behind the subsystems has significant differences. The wireless modem is a reasonably well defined specialized environment where for example model-based system-level design can be relatively painlessly applied. Contrary to that, the application subsystem is often an open environment which should be more aligned towards software development paradigms allowing more freedom in many respects.

In addition to programmable processor cores, mobile devices employ hardware acceleration for the functionalities requiring high computational processing power. Within the wireless subsystem, these include signal filters and channel coding. For the application subsystem, it is also desirable to utilize hardware acceleration, e.g., for multimedia codecs and 3D rendering.

## 4 Mobile software

A distinctive characteristic for the mobile software development environments is their *cross-development* setup. As e.g., the input and output capabilities of physically small devices are not optimal for software development,

native development is a rare practice. The software development kits employ simulators for testing and debugging while the software is developed on a separate host system.

Even though the cross-development setup seems to be a persistent feature for mobile software development, the emphasis on *design for a particular purpose* is currently under rapid change towards more open ended software development process [7]. The software, the development practices and the tools commonly used in desktop environment are finding their ways into mobile hand-held devices. Therefore, the mobile domain shares many trends with mainstream software development.

These trends have clear implications for the computing architecture. Table 1 shows an example highlighting the significance of the dynamic content in current browser workloads. As has been demonstrated by e.g., the recently introduced Google V8 JavaScript engine, high gains can be achieved with dynamic compilation.

| Module | cycles / % | comment |
|---|---|---|
| `libflashplayer` | 43 | Adobe Flash |
| `libxul` | 27 | XML UI |
| `libmozj` | 7 | Javascript |
| `libc` | 6 | C library |
| `libgcc` | 5 | compiler support |
| `libpthread` | 2 | threading |
| `libm` | 2 | math |
| `libnspr` | 2 | porting layer |

*Table 1: CPU cycle distribution within the browser process while opening CNN.com in Nokia N810.*

As mobile devices are by nature embedded systems interacting in many ways with their environment, the processing that they perform should have more inherent parallelism than for example batch mode computation. However, the software development tools and programming models used for developing software for mobile devices do not offer any better support for expressing the parallelism than what is common in e.g., desktop world. This means that the parallelism of mobile workloads does not differ radically from general purpose application software. Figure 3 shows the histogram for the operating system ready queue length for various workloads running in an OMAP2420 based device. As can be seen, some workloads show potential for decent utilization on a few cores. However, for higher levels of parallelism, operating system scheduling alone is clearly not enough.

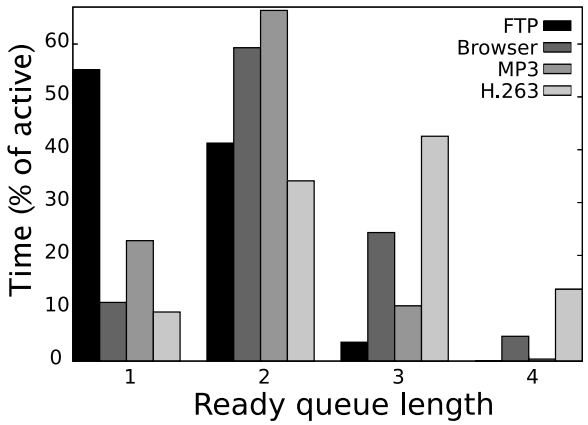


*Figure 3: Linux ready queue length for different workloads in OMAP2420 based hardware.*

## 5 Challenges

### 5.1 The programmability walls

From the view point of a software developer, programming for parallel hardware is challenging – especially, when heterogeneous hardware and tightly coupled asymmetric execution environments are considered. On one hand, there are obvious possibilities of improving performance and energy-efficiency, e.g., by proper allocation of computational tasks and the related data on the units. On the other hand, the novel architectures bring a high risk of significantly degrading the performance and energy-efficiency (e.g., by bad allocation). Automated methods for increasing the parallelism of workloads have been proposed, e.g., thread-level speculation [8]. However, the reliance on statistical properties of workloads makes the exact run-time behavior increasingly hard to predict.

For mobile devices, a crucial aspect requiring special attention is the *observability wall* related to performance and energy consumption effects. The simulation tools that are an integral part of the mobile software development process usually employ only functional simulation in order to maintain the high simulation speeds required by interactive debugging. Therefore, performance and energy aspects remain virtually unobservable during the normal edit-compile-debug cycle.

Parallelism introduces additional challenges for performance analysis. At high level, the performance behavior of a uniprocessor system is reasonably straight forward and subtle performance anomalies are rare. However, the performance behavior of multiprocessing systems is significantly more complex. A simple example of the subtleness of performance effects is false sharing. With false sharing parallelized code can show full utilization while performance of parallel execution

can be lower than the performance of the same code executed sequentially. Diagnosing such situations can be quite challenging with traditional debugging tools.

Energy consumption effects can also be quite subtle and hard to debug even with current mobile platforms. For rough analysis, on-target measurement can be invaluable [9]. For detailed analysis, more elaborate hardware instrumentation is usually employed, but these measurement setups are generally not suitable for large scale adoption.

## 5.2 Heterogeneous systems

While increase in programmability is continually sought for, mobile devices are likely to require hardware acceleration also in future. From performance estimation point of view, the hardware accelerators are an especially challenging topic. The software development kits might in the best case implement functional emulation of the accelerators. This enables operating system device driver development without the actual hardware. However, often the accelerators are simply short-circuited by using development host resources to provide similar functionality. This means that it is impossible to observe the performance effects of the acceleration within the simulator.

# 6 Discussion

The observability wall in mobile software development is related to the ubiquitous use of simulators. In the advent of many-cores, the need for increasing the visibility of performance and energy consumption effects within the natural software development-flow is obvious.

Fast and accurate simulation is a challenging topic. For example, traditional cache simulation [10] is slow, because it is based on explicit simulation of the memory traffic and maintaining accurate information about memory system status. However, immediacy of feedback is often much more important than accuracy, because of product process issues, development flows, etc. Also, the overall performance of software can often be explained by high-level metrics. This makes it appealing to use simple approximative methods that are based on empirically adjusted abstract models. Such abstract models can be connected to the compiler technology that is used in typical functional simulators.

Figure 4 shows the cumulative data access distributions for three different traces from a mobile browser. Several observations can be made. The most immediate one is the relatively low MIPS metric for all the cases (the CPU frequency was locked to 330 MHz during the traces). Another observation can be read from the maximum value reached by the curves, which tells that the data to instruction ratios for all the traces are quite high.

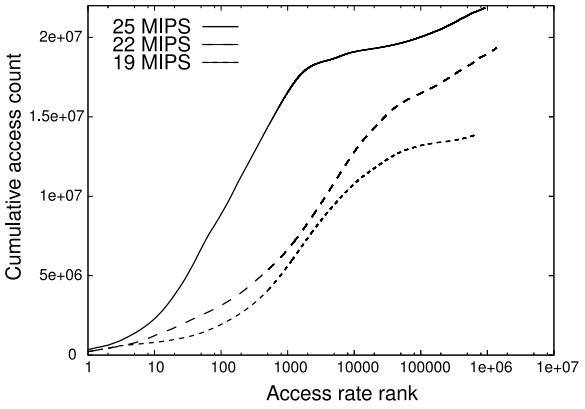


*Figure 4: Cumulative counts of data accesses to unique addresses for three one-second traces from the Mozilla based browser in Nokia N800.*

Finally we can observe that the wider the accesses are spread, the lower the achieved MIPS metric, i.e., the larger the memory footprint, the lower the performance. In our experience, this kind of behavior is typical.

To demonstrate that even simple models can provide quite accurate estimates for performance, Figure 5 shows estimated and measured MIPS metrics for the deflate compression algorithm. The cycle and instruction counts were collected from the RealView Instruction Set Simulator with a memory model calibrated with micro benchmarks against real hardware. The estimated MIPS metric was calculated with a simple linear model from data cache line fetch rate. The coefficients for the model were obtained by least squares fit to the data collected from the ASCII compression case. As can be seen, the estimate for the ASCII case is almost perfect. When the input data is changed from ASCII to JPEG, a visible systematic error in the estimate can be seen, but the mean square error for the estimate is still below 1%.

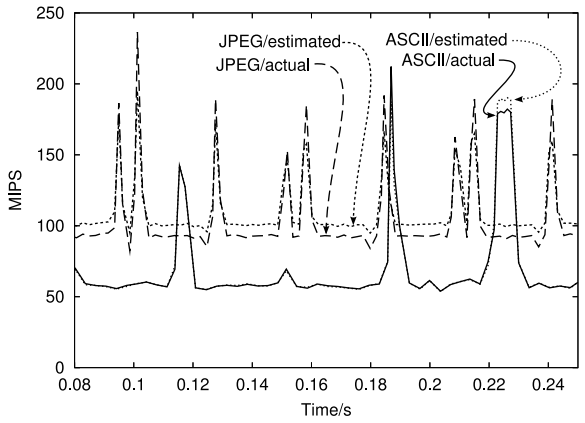


*Figure 5: Actual and estimated MIPS metric for the deflate compression algorithm.*

Accurate energy consumption estimation is also a challenging task, and like accurate timed simulation, continues to be a relevant research topic [11]. However, e.g., for contemporary CPUs, activity time seems to be a good first-order estimate for the consumed energy. Even though individual instructions can have clear differences in their energy cost, the differences tend to average out during execution.

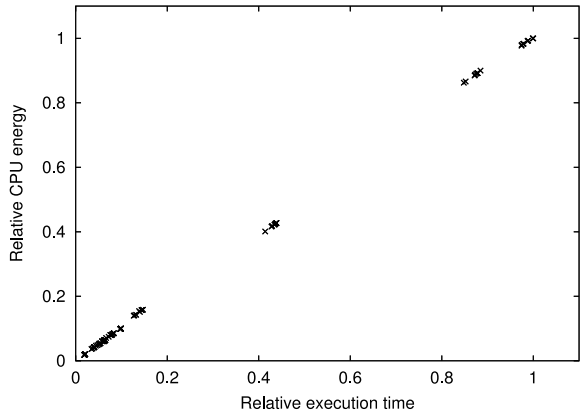


*Figure 6: Energy consumption as function of execution time for various workloads on OMAP5912.*

To demonstrate the feasibility of simple models for energy consumption estimation, Figure 6 presents various workloads compiled with various optimization levels as function of their relative execution time. As can be seen, the CPU energy consumption depends quite linearly on the execution time. This allows rough estimation of the CPU energy consumption to be based on the activity times obtained by performance estimation models and the explicit modelling of the power behavior of the hardware can be avoided.

For current hardware platforms this kind of simple models could be integrated into the functional simulators used for mobile software development. These production simulators are typically based on dynamic binary translation, which translates fragments of target code into development host code and executes the fragments on-the-fly by using a small code buffer [12]. Further, more advanced methods are emerging for simulation that are better suitable for parallel software.

Performance of simulating parallel hardware targets can be significantly improved with unsynchronized execution and direct use of simulation host shared memory [2]. The unsynchronized execution means that the simulation is non-deterministic, but the consistency of host shared memory ensures functionally correct execution. We propose augmenting such an approach with simulation of non-functional behavior (see Figure 7). However, it is not obvious that approximate models can expose performance phenomena specific to parallel execution, e.g., false sharing. Clearly further research is needed to find the balance between high simulation speed and meaningful estimates for e.g., timing and memory traffic when simulating parallel execution.

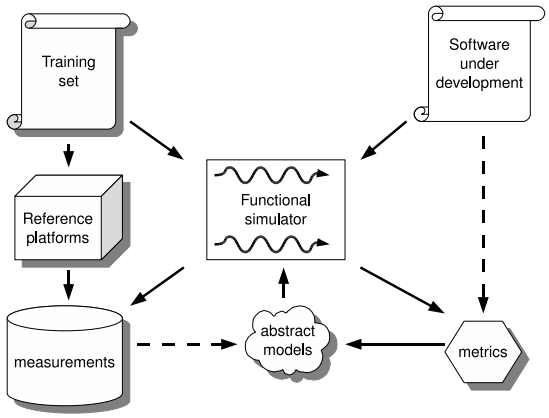


*Figure 7: Applying empirically adjusted abstract models in mobile software development with a setup similar to [4].*

Memory behavior is essential with respect to energy-efficiency. Abstract models can be used to speed-up simulation of memory behavior [13]. In such an approach, abstract models are used to partially resolve the memory traffic during static compilation. The remaining memory traffic is resolved during run-time along the dynamic binary translation for the functional emulation. According to our previous experience with such compiled simulation methods, the majority of cache hits and misses will be resolved during static compilation.

The use of abstract models during the dynamic phase means that the simulation will not be accurate, but we expect it to be sufficient for embedded software development based on other studies (e.g., [4]). We anticipate that the accuracy of the performance estimates provided by the abstract models could be improved by optimizing the composition of the training set programs used for calibrating the models. Also the choice of metrics that are exported from the functional simulator could be optimized, e.g., based on static analysis of the software under development.

For parallel software and hardware, it is necessary to model the effects of synchronization and sharing. Memory, being the most common sharing mechanism, represent the main challenge for performance and energy estimation. We advocate further studies on modelling the performance and energy consumption effects of the memory subsystem in parallel platforms for the purpose of combining static analysis with simulators utilizing dynamic compilation and light weight run-time data collection suitable for rapid feedback during software development.

## 7 Conclusions

While parallelism has tremendous potential especially for mobile devices, already the current platforms are suffering from severe observability challenges related to performance and energy consumption. The industry standard tools and practices are fundamentally lacking in this area especially considering the trends towards increasingly complex parallel architectures employing asymmetric multiprocessing coupled with hardware accelerators.

Although a large body of research exists on simulation, further research is needed on the interaction between cross-platform simulation and many-core platform implementation, especially considering memory-related developments such as transactional memory [14] and hardware interconnection structures [15].

Advanced simulation methods, modern compiler technology and empirically tuned models show promise for providing tools to seamlessly integrate performance and energy consumption estimation into the software development flow used to target mobile hand-held devices. In the context of contemporary software development practices, e.g., test-driven development, rapid feedback will be crucial for creating efficient software for the highly parallel platforms of the future. Considering software development, viability of the design decisions by software developers are more important than absolute precision of the tools used by them.

## 8 Acknowledgments

For support and review work we thank Kimmo Kuusilinna, Eero Aho and Jussi Ruutu at Nokia Research Center. For our tools of the trade we are indebted to numerous people in Nokia product development units, including Igor Stoppa, Sakari Poussa, Ari Aho, Rolf Kühnis, Stephen Wade and Andrew Baldwin. We are also grateful for Juhani Peltonen for rigorous measurement work supporting our research.

## References

[1] Symbian Symmetric Multiprocessing description, `http://www.symbian.com/symbianos/os_smp.html`

[2] B. Lanz, *Parallel SimOS: Performance and Scalability for Large System Simulation*, Ph.D. Dissertation. Stanford University, Stanford, CA (2007).

[3] L. T. Yang, X. Ma and F. Mueller, *Cross-Platform Performance Prediction of Parallel Applications Using Partial Execution*, Proc. ACM/IEEE Conf. Supercomputing (2005).

[4] B. Franke, *Fast cycle-approximate instruction set simulation*, Proc. int. Workshop on Software & Compilers For Embedded Systems (2008).

[5] Y. Neuvo, *Cellular phones as embedded systems*, Digest of Technical Papers, IEEE Int. Solid-State Circuits Conf. (2004) pp. 32–37.

[6] K. Flautner, S. Reinhardt and T. Mudge, *Automatic Performance Setting for Dynamic Voltage Scaling*, Wirel. Netw., vol. 8, no. 5 (2002) pp. 507–520.

[7] Maemo Software Platform, `http://maemo.org/intro/platform/`

[8] T. Ohsawa, M. Takagi, S. Kawahara, S. Matsushita, *Pinot: Speculative Multi-threading Processor Architecture Exploiting Parallelism over a Wide Range of Granularities*, Proc. IEEE/ACM Int. Symposium on Microarchitecture (2005) pp. 81–92

[9] Nokia Energy Profiler, `http://www.forum.nokia.com/main/resources/user_experience/power_management/nokia_energy_profiler/`

[10] J. Edler and M. D. Hill, Dinero IV Trace-Driven Uniprocessor Cache Simulator, `http://pages.cs.wisc.edu/~markhill/DineroIV/`

[11] D. Brooks, V. Tiwari, M. Martonosi, *Wattch: A Framework for Architectureal-Level Power Analysis and Optimizations*, Proc. Int. Symposium on Computer Architecture (2000) pp. 83–94.

[12] F. Bellard, *QEMU, a fast and portable dynamic translator*, Proc. USENIX Annual Technical Conf. (2005).

[13] V. Hirvisalo, *Using Static Program Analysis to Compile Fast Cache Simulators*, Doctoral Dissertation, Helsinki University of Technology, Espoo (2004).

[14] J. Larus, C. Kozyrakis, *Transactional Memory*, Commun. ACM, vol. 51, no. 7 (2008), pp. 80–88.

[15] V. Narayanan, *Network-on-Chip for 3D Architectures*, Mini keynote, Int. Forum on Appliation-Specific Multi-Processor SoC (2008).