Accurate Run-Time Prediction of Performance Degradation under Frequency Scaling

David C. Snowdon
NICTA[*]
University of New South Wales
Sydney, Australia

Godfrey Van Der Linden
NICTA
University of New South Wales
Sydney, Australia

Stefan M. Petters
NICTA
University of New South Wales
Sydney, Australia

Gernot Heiser
NICTA
University of New South Wales
Open Kernel Labs
Sydney, Australia

ABSTRACT

Dynamic voltage and frequency scaling is employed to minimise energy consumption in mobile devices. The energy required to execute a piece of software is highly depedent on its execution time, and devices are typically subject to timeliness or quality-of-service constraints. For both these reasons, the performance at a proposed frequency setpoint must be accurately estimated. The frequently-made assumption that performance scales linearly with core frequency has shown to be incorrect, and better performance models are required which take into account the effects, and frequency setting, of the memory architecture. This paper presents a methodology, based on off-line hardware characterisation and run-time workload characterisation, for the generation of an execution time model. Its evaluation shows that it provides a highly accurate (to within 2% on average) prediction of performance at arbitrary frequency settings and that the models can be used to implement operating-system level dynamic voltage and frequency scaling schemes for embedded systems.

1. INTRODUCTION

Energy consumption is an increasingly important factor in the design of computing systems. This is particularly true in embedded systems, where a lower energy consumption improves battery life and reduces size and cost, and has a significant impact on the commercial viability of a product.

Dynamic voltage and frequency scaling (DVFS) is a technique for reducing a circuit's energy consumption by modifying clock frequencies. Reducing frequency results in a lower power consumption and increased software execution time. It generally allows a circuit's supply voltage to be reduced, leading to quadratic savings.

While reducing the frequency to a particular circuit can improve its energy efficiency, other circuits may use more energy as a result of the longer execution time. Therefore the slowest frequency is not necessarily energy-optimal [13], and the energy-optimal frequency can only be chosen via knowledge of the expected execution time.

For example, in a totally memory-bound system, a reduction in CPU frequency will not result in an increase in execution time as the CPU is constantly stalled waiting on the memory bus (which is unaffected by CPU frequency). The reverse is also true: CPU bound applications' execution time will not be reduced by an increased bus or memory frequency, but the overall energy consumption will increase due to the higher bus or memory idle power. Furthermore, the dependence of the total execution time on the frequency is specific to the workload. Figure 1 shows the normalised execution cycles for two applications running at various CPU, bus and memory frequency combinations on an Xscale based processor (see Section 4 for further details). The difference between CPU-bound and memory-bound tasks is striking. Knowledge of the performance effects of frequency scaling is essential for choosing an energy-optimal setpoint.

The policy which selects when to switch frequency, and which frequency to switch to, is generally selected by the operating system.

Estimating and predicting the runtime of a piece of software is an important component in an effective power management system. The energy required for a task is heavily dependent on time ($E = Pt$). While the CPU's power consumption will be reduced by frequency scaling, core frequency is unlikely to have an effect on the power for the rest of the system. The energy required by components other than the CPU will be proportional to the overall execution time, leading to a complex relationship between core frequency and overall energy use. This is further complicated by the adjustment of memory, bus and IO interface clock frequencies.

In addition, accurate estimation of the execution time of a task is essential for meeting *real-time* (RT) deadlines and *quality-of-service*

[*]NICTA is funded by the Australian Government's Department of Communications, Information Technology, and the Arts and the Australian Research Council through Backing Australia's Ability and the ICT Research Centre of Excellence programs.

(QoS) requirements while employing DVFS.

A further factor complicating execution-time estimation is the increasingly dynamic nature of embedded systems. The applications themselves, the nature of their input data and stimuli, and the environment in which they are run are dynamic. It is therefore not practical to characterise the applications' behaviour a-priori, and any estimation must be performed at run-time.

This paper presents a methodology and mechanism for the accurate run-time estimation of the performance of a given piece of software at an arbitrary frequency setpoint. Our specific contributions are: (i) a model providing accurate estimates of the runtime of a workload at an arbitrary frequency; (ii) a sound methodology for generating an execution-time model from *performance monitoring counter* (PMC) measurements; (iii) a sound methodology for selecting the optimum PMCs; (iv) an efficient algorithm for the calculation of the performance at any frequency setpoint; and (v) an evaluation of our approach using an extensive and representative benchmark suite.

We first summarise related work in Section 2. Section 3 describes our model for the execution time of an application, before detailing the parameter selection methodology, model generation and finally discussing runtime performance prediction. We describe our evalutation environment in Section 4 and present the results we have obtained in Section 5 before concluding with a summary and an outlook into our future work.

2. RELATED WORK

The performance benefits of DVFS has been an active area of research ever since the pioneering work of Weiser et al. [15]. The work related to operating-system level scheduling can be divided into two broad categories based on the OS's assumed a-priori knowledge.

Real-time systems, which are required to deliver results by a deadline, require knowledge of the system timing, frequently in the form of *worst case execution times* (WCET). Previous work explored the potential for CPU frequency scaling without missing real-time deadlines [1,12]. This was extended to use the CPU's memory stall rate in a feedback loop with the scheduler [9,11].

Off-line techniques [5,17] use a detailed static analysis of a workload by the compiler. Other approaches include an *a priori* characterisation of a workload by running it at two different frequencies, in order to derive a slowdown relation [14]. These off-line results are then used by a DVFS-aware scheduler to scale the processor frequency.

Systems where no a-priori characterisation is performed generally aim to get the best energy efficiency for a given performance impact [7,8]. Such early work was typically based on the incorrect assumption that performance was proportional to CPU frequency. The non-linear dependency has since been the subject of considerable investigation.

Most of this research uses PMCs as a guide to predicting the likely performance impact of a frequency change. Process cruise control [16] used instructions, memory accesses and cycles to index a precomputed table of frequency settings which led to a constant performance impact. Other research groups have investigated a more flexible technique, using on-line regression to calculate the ratio

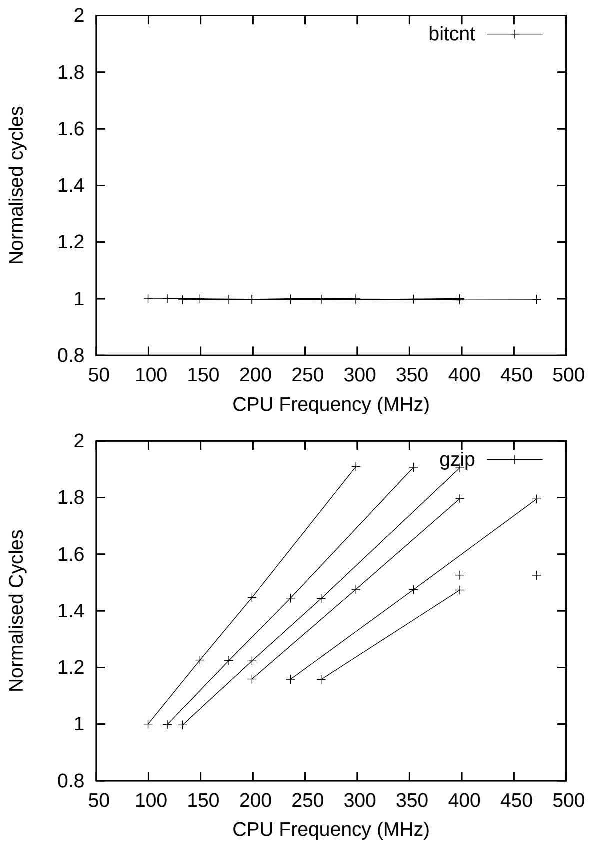

Figure 1: Execution cycles vs. CPU frequency, normalised to the slowest frequency cycles, for `bitcnt` and `gzip`, grouped by constant bus/memory frequency

of off-chip (CPU frequency independent) to on-chip cycles [2,4], however the computational overheads and response time are only cursorily discussed and there is reason to believe that they are substantial (i.e. the evaluation of a regression requires significant CPU time).

A theoretical model of a classification system between memory-bound and CPU-bound applications [18] assumes the availability of a large number of concurrently-usable performance counters. Limitations of this work include a lack of a detailed justification of performance-counter selection, insufficient evaluation with only a small number of benchmarks, and lack of statistical rigour (the evaluation is performed with the same data that is used to obtain the parameters of the model).

The models and methodology in this paper represent a significant advance over the above-outlined work. Software is characterised at run-time and our technique can therefore be used for arbitrary workloads with dynamic input data (in contrast with off-line techniques [5,14,17]). Compared with the previous state of the art [2,4], our model and methodology can be applied on arbitrary platforms, we do not require a run-time regression (resulting in a low overhead), we present our method of selecting the performance counters best suited to the task of performance estimation, and an evaluation has been performed with a much more extensive range of benchmarks.

3. APPROACH

The objective of this work is to predict the runtime of a workload at one frequency, given measurements at another frequency. This execution time model can then be used for making true energy vs. performance frequency scaling decisions as discussed in Section 1.

3.1 Execution Time Model

While the CPU core is a major contributor to a system's power consumption, other subsystems, such as memory and I/O, are also significant and can in some cases even dominate the CPU. Moreover, such contributions generally are independent of the core frequency.

For example, the time the CPU stalls while waiting on main memory depends on the bus and memory clocks, not the core clock. On a processor with a single issue, in-order pipeline and a simple cache architecture (the typical scenario in embedded systems) the CPU is always stalled during memory activity. In this paper, we focus on this class of system. The effects of a superscalar architecture are expected to be small, but will be subject to future research.

For this class of system, we have an overall execution time T which is a function of the various clock rates f_x, used in the system:

$$T = \frac{C_{cpu}}{f_{cpu}} + \frac{C_{bus}}{f_{bus}} + \frac{C_{mem}}{f_{mem}} + \frac{C_{io}}{f_{io}} + \dots \quad (1)$$

The coefficients $C_{cpu}, C_{bus}, C_{mem}, C_{io} \dots$ depend on the instruction stream of the actual workload. The task of execution-time estimation comes down to obtaining good estimates for those parameters at run-time, without *a priori* analysis of the particular workload. I/O effects are beyond the scope of the paper, so we will, from here on, focus solely on the CPU and memory subsystem.

3.2 Application characterisation

While the coefficients C_x depend on the workload, they represent the total number of cycles used for particular actions (eg. CPU-only instructions or memory reads); each is the product of the number of such actions and the cycle cost of such an action. The former is a characteristic of the workload, the latter of the system architecture.

The fundamental idea behind our approach is to perform an off-line characterisation of the architectural parameters, and use run-time measurements, using performance monitoring counters (PMCs), for workload characterisation. Specifically, we postulate that each coefficient can be represented by some linear combination of PMC readings:

$$C_{bus} = \alpha_1 PMC_1 + \alpha_2 PMC_2 + \dots \quad (2)$$
$$C_{mem} = \beta_1 PMC_1 + \beta_2 PMC_2 + \dots \quad (3)$$

The accuracy of the model will depend on the architecture and the suitability of the PMCs. The architecture-specific coefficients α_x, β_x, …are properties of the hardware platform and can be determined once, by evaluating a suitable set of benchmarks representing the range of different workloads. Linear regression on Equation 2 and Equation 3 will establish the values of the coefficients and allow selecting the most suitable PMCs. This is important, as the hardware typically supports the concurrent use of only a small number of PMCs, and the correct choice is not obvious as will be shown.

The total number of CPU cycles, C_{tot}, can be directly obtained from the CPU's cycle counter. It is the product of total execution time, T, and core frequency, f_{cpu}, allowing us to rewrite Equation 1 as

$$C_{cpu} = C_{tot} - \frac{f_{cpu}}{f_{bus}} C_{bus} - \frac{f_{cpu}}{f_{mem}} C_{mem} \quad (4)$$

Hence, we only need to determine C_{bus} and C_{mem} from PMCs.

3.3 Performance prediction

Being able to estimate a workload's C_x values from PMC readings at a particular setpoint (characterised by a particular combination of clock frequencies, f_x) it is possible to predict the performance of the same workload at a different setpoint with frequencies f'_x:

$$\begin{aligned} C'_{tot} = {} & C_{tot} \\ & - \frac{f_{cpu}}{f_{bus}} C_{bus} - \frac{f_{cpu}}{f_{mem}} C_{mem} \\ & + \frac{f'_{cpu}}{f'_{bus}} C_{bus} + \frac{f'_{cpu}}{f'_{mem}} C_{mem} \end{aligned} \quad (5)$$

We define the performance, s, as the execution time at a particular setpoint normalised to execution time at maximum frequency setpoint, f_x^{max}:

$$s = \frac{t^{max}}{t'} = \frac{f'_{cpu}}{f_{cpu}^{max}} * \frac{C_{tot}^{max}}{C'_{tot}} \quad (6)$$

For the present setpoint, $f'_{cpu} = f_{cpu}$ and $C'_{tot} = C_{tot}$, the latter being the performance counter reading. Hence, the performance at the current frequency settings is a linear equation of performance counter and frequency cross terms. This allows a single regression to be applied across all workloads to calculate α_x and β_x, given a pre-calculated s avoiding the intermediate step of C_x.

$$s_{cur} = \frac{f_{cpu}}{f_{cpu}^{max}} * \frac{C_{tot}^{max}}{C_{tot}} \quad (7)$$

Similarly, we can calculate performance relative to the current setpoint using C_{tot} in place of C_{tot}^{max}.

4. EVALUATION

Our model and methodology were evaluated on a typical embedded systems platform (Section 4.1) using a number of benchmarks (Section 4.2). The model was used to implement an on-line estimation system (Section 4.3).

4.1 Platform

The hardware platform used in all experiments was PLEB 2, a single board computer based on an Intel PXA255 processor [6]. The PXA255 is based on an Xscale core, with split L1 caches and TLBs, write and fill buffers. The data cache supports a write-back policy. PLEB2 integrates 64MB SDRAM and 8MB Flash memory. The core voltage remained constant at 1.5V.

Only specific combinations of f_{cpu}, f_{bus} and f_{mem} can be generated by the processor. For our experiments, we use 22 setpoints which are detailed in Table 1. Note that most of these setpoints are outside the manufacturer's recommended limits, but were found to work reliably and used in order to obtain more data.

	f_{cpu} (MHz)	f_{bus} (MHz)	f_{mem} (MHz)
1	99.531	49.766	99.531
2	117.964	58.981	117.964
3	132.71	66.354	132.71
4	149.299	49.766	99.531
5	176.946	58.981	117.964
6	199.064	49.766	99.531
7	199.064	66.354	132.71
8	199.064	99.531	99.531
9	235.929	58.981	117.964
10	235.929	117.964	117.964
11	265.420	66.354	132.710
12	265.420	132.710	132.710
13	298.598	49.766	99.531
14	298.598	99.531	99.531
15	353.894	58.981	117.964
16	353.894	117.964	117.964
17	398.131	66.354	132.71
18	398.131	99.531	99.531
19	398.131	132.71	132.71
20	398.131	199.064	99.531
21	471.858	117.964	117.964
22	471.858	235.929	117.964

Table 1: PXA255 frequency setpoints

The PXA255 provides three performance counters – a cycle counter and two configurable performance counters. The configurable counters can each count any one of 14 events [6] (outlined in Table 2).

PMC	Description
0x0	ICache miss
0x1	ICache stall cycles
0x2	Data dependency stalls
0x3	ITLB miss
0x4	DTLB miss
0x5	Branch instruction executed
0x6	Branch mispredicted
0x7	Instruction executed
0x8	DCache buffer stall cycles
0x9	DCache buffer stall
0xa	DCache access
0xb	DCache miss
0xc	DCache write-back
0xd	Software changed the PC

Table 2: PXA255 performance counter events

We conducted all experiments on Linux 2.4.19, having written kernel modules and modifications to support per-process performance counter reading. The PMCs were read and accumulated after each scheduler invocation.

When executing benchmarks, the only other runnable thread was kupdated, which flushes the file system buffers. The network device was disabled during benchmark execution. The timer tick was not disabled, which will be a small source of error in the measurements since the number of timer ticks which occur during a benchmark run is dependent on its real-world execution time.

4.2 Benchmarks

Sound experimental methodology requires that the workloads used for evaluation must be different from those used for calibration. A further requirement on the benchmarks is that the total amount of work is the same in each run, independent of the frequency settings.

For calibration we used a total of 37 benchmarks. Most are from the MiBench suite [3], a set of real-world applications which are representative of the tasks found in different types of embedded systems. Several benchmarks (`djpeg`, `susan_corners`, `susan_edges`, `tiff2bw`) were removed as their total execution time was too short to be useful (less than 0.25s at maximum frequencies). Two short-running ones (`adpcm` and `stringsearch`) were modified to iterate a number of times in order to extend the overall run time. Two others (`sphinx`, `pgp`) were removed as their amount of work differs between runs under identical circumstances.

We added further benchmarks to the calibration set. Four (`gzip`, `mpg123`, `vision` and celp32c) have been previously described [13]. We also added three synthetic benchmarks to cover extreme behaviour: `cpubound` executes an unrolled loop of NOP instructions entirely in cache. `membound` and `readbound` execute an unrolled loop of out-of-cache writes and reads respectively.

For validation we used SPEC CINT95 benchmarks. `vortex` was excluded because of memory constraints, `go` and `m88ksim` due to runtime errors. The "test" dataset was used and the input data size for `compress` was reduced to 420000 bytes to reduce overall execution time. This leaves 5 benchmarks used for validation.

All benchmarks were compiled or assembled using gcc 3.4.4 with softfloat. The linux kernel and ramdisk was compiled using gcc 3.3.2.

4.3 Implementation

We used the approach presented in Section 3 to estimate, while executing at a particular frequency setpoint, the performance that would be achieved at the maximum-frequency setpoint. This can then be compared to direct measurements of an execution at f_x^{max}.

For evaluation purposes, we also used the techniques to aim for a particular pre-determined performance, which can then also be compared to the actual performance, obtained by measuring the overall execution time and comparing to the execution time at f_x^{max}. While this technique is unlikely to choose an energy-optimal setpoint, it does demonstrate the ability of the system to predict the performance of a benchmark at run-time, which we have discussed as being crucial to energy-optimality.

The approach is based on the well-established model of temporal locality which underlies many operating-system policies. In our case this means that we assume that the behaviour of a particular task does in most cases not change significantly between subsequent time slices. At the end of each time slice, the OS collects the PMC readings and *estimates* the slowdown at the present setpoint using our model. When the task is next scheduled, the slowdown estimate is compared to the target slowdown, and the frequency setting adjusted if necessary.

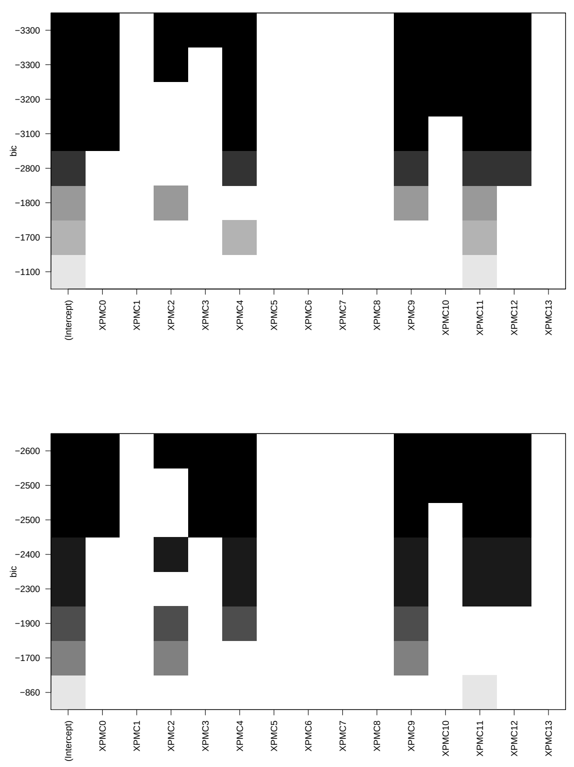

Figure 2: Parameter selection for C_{bus} and C_{mem} models

A key advantage of our approach is the lightweight nature of the estimation calculations. The XScale is typical for embedded processors in that it lacks an FPU and a hardware divide instruction. We therefore use only fixed-point arithmetic and avoid divisions.

5. RESULTS

5.1 Time vs. Frequency validation

Each benchmark in the calibration suite was run at each of the 22 setpoints. Equation 1 was fit to the data for each of the benchmarks using least-squares linear regression. The fit was extremely good, the value of $1 - R$, which indicates effects in the data that are not explained by the model, ranging between 5×10^{-4} and 3×10^{-8}. This is a strong indication that the model can accurately account for the architectural features of this class of processor. Furthermore, the intercepts were negligible, indicating that, for these benchmarks, in this system, the execution time depends solely on these frequencies.

5.2 Performance counter selection

We investigated the best choices of performance counters when only a small number of them can be used concurrently (as on most hardware, including ours). The models were formulated by equating C_{bus} and C_{mem} with all possible linear combinations of the available performance counters, as well as several potentially relevant cross terms.

Each model of each size was compared using a criterion function. i.e. every possible combination of performance counters was used to predict each of C_{bus} and C_{mem}. The best n-parameter model was selected using the BIC criterion function (a measure of the model's predictive capability), although R^2 would rank the models in the same way. (e.g. the n-parameter model with the highest BIC or R^2 is selected as the best). This procedure was performed using the `regsubsets` command in `R` [10]. In this way we determine the best performance counters for performance prediction of these calibration workloads. The results of the parameter selection for C_{bus} and C_{mem} are shown in Figure 2. The figures show which parameters are selected for each of the n-parameter models (each row represents a model with one more parameter than the row below). The differences in the graphs are likely due to noise in the data, insufficient variance in the benchmarks. Our future work has included a unified approach to the parameter selection.

We observe that, for this benchmark suite, the best single parameter model uses data cache misses (PMC11), the best dual parameter model also uses data TLB misses (PMC4). The best three parameter model uses PMC11, data cache buffer stalls (PMC9) and data dependency stall cycles. The best four parameter model uses PMC11, PMC4, PMC9 and data cache write-backs.

The results also show that the model does not improve significantly beyond four PMCs, and two PMCs perform almost as well as three (indicating strong correlations between cache and TLB misses). The PXA255 only supports two simultaneous measurements (in addition to the cycle counter), so our on-line prediction system in this platform is based on data TLB misses and data cache misses.

Importantly, unlike previous work [2, 4] since our model does not implicitly require the number of instructions, the parameter selection is free to choose any two events.

5.3 Slowdown prediction

While the actual parameters selected by this procedure depend on the benchmarks used for calibration, the BIC values indicate that this should not have a dramatic effect on the overall results. This can be verified by validation runs using independent benchmarks (our SPECINT subset).

An offline evaluation (i.e., using end-to-end data obtained running the validation suite over several executions) yields an average error of 1.7% and maximum error of 7% for the two parameter model. For comparison, the same data yields an average error of 10% and a maximum of 38% if the estimate is based only on the CPU frequency ("naive model"). Figure 3 shows the errors in the naive model, and Figure 3 for the 2-parameter model. The improvement over the naive model is obvious.

5.4 Frequency scaling error

We then ran each of the validation benchmarks aiming for 17 predetermined performance values ranging from 20 and 100%, each time recording actual and estimated run time at f_x^{max}. As the frequency settings are not continuous, the system cannot normally chose a setting that is estimated to produce exactly the target performance. Instead, the frequency selection policy simply chooses the setpoint which gives the closest approximation to the desired performance; the actual performance and desired performance will therefore differ, even if our estimates were totally correct. To account for this fact, we present the values again as the error in the estimated performance (estimated minus actual) against the estimated performance, Figure 5. These are on-line estimation errors: the performance is calculated for each time slice.

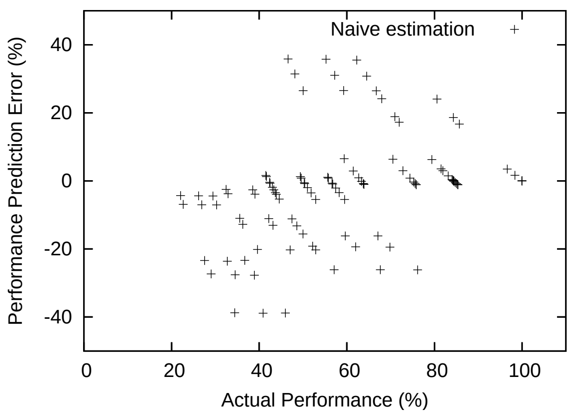

Figure 3: Naive model estimated performance error vs. actual performance

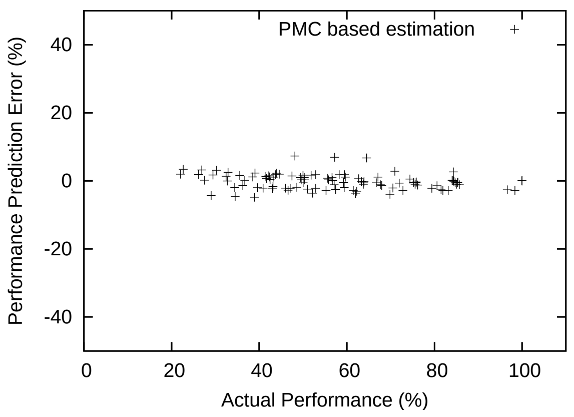

Figure 4: 2 parameter PMC-based model estimated performance error vs. actual performance

The maximum error observed was 7%, the average was 1.9%. These compare favorably with the most accurate published performance estimations (4–6% [2]), and are consistently more accurate than those presented in other work, despite having been tested with a much larger variety of workloads. In addition, previous work generally used the same benchmarks for calibration and validation. Thus the errors observed in most previous work are not indicative of their models' predictive capability.

5.5 Frequency scaling overheads

The cost of the frequency selection calculations were measured to be 5000—7750 cycles. This averages $24\mu s$, which compares favourably to Choi's $100\mu s$ [2]. That work requires an on-line regression calculation, yet does not consider multiple memory frequencies.

6. CONCLUSIONS

This paper has first motivated, and then presented a general and sound model of execution time under frequency scaling. It is based on an off-line characterisation of the hardware platform, combined with on-line evaluation of application characteristics using performance counters. The model has been implemented and validated

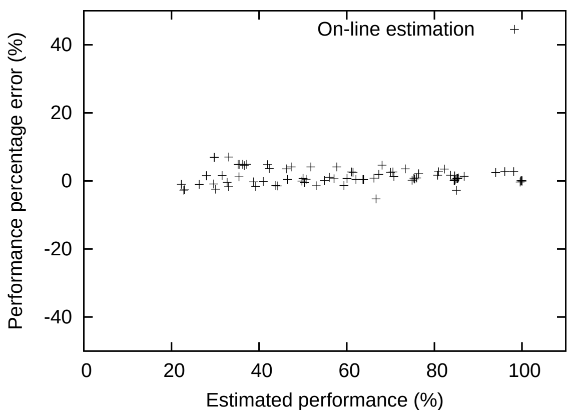

Figure 5: On-line estimated performance error vs. estimated performance

on a processor typical for use in high-end mobile systems.

The model, once developed on a set of representative benchmarks, has demonstrated an excellent ability to predict the performance of new applications. The on-line evaluation implies that the model can quickly adjust to changes in application behaviour. The approach is general in the sense that it should be readily portable to different processor platforms providing basic performance monitoring hardware. The model performed well with only two performance counters, without the need for time-multiplexing.

In addition, this work has taken a rigorous approach to the model evaluation, with two large, disjoint published sets of benchmarks used for calibration and validation. The system was tested with an order of magnitude more workloads than comparable work.

The model has clear applications as part of an energy-saving framework, which could enable an accurate trade between performance and energy. Our subsequent work will show the necessity of a performance model when building an accurate model for the prediction of energy consumption under frequency scaling.

In the future we plan to validate the model's generality by deploying it on other platforms. Another obvious next step is an evaluation in a multi-processing context. Performance loss related to IO devices, and the effect of interupts and DMA should be examined.

7. REFERENCES

[1] H. Aydin, V. Devadas, and D. Zhu. System-level energy management for periodic real-time tasks. *Proceedings of the 27th IEEE Real-Time Systems Symposium*, 0:313–322, 2006.

[2] K. Choi, R. Soma, and M. Pedram. Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access to on-chip computation times. *IEEE Transactions on CAD ICAS*, 24(1):18–28, Jan. 2005.

[3] M. R. Guthaus, J. S. Reingenberg, D. Ernst, T. M. Austing, T. Mudge, and R. B. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Proceedings of the 4th IEEE Annual Workshop on Workload Characterization*, Dec. 2001.

[4] C.-H. Hsu and W. chun Feng. Effective dynamic voltage scaling through CPU-boundedness detection. In *Proceedings of the 2004 Workshop on Power-Aware Computer Systems*, pages 135–149, 2004.

[5] C.-H. Hsu and U. Kremer. The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction.

SIGPLAN Not., 38(5):38–48, 2003.

[6] Intel Corporation. Intel PXA250 and PXA210 applications processors developers manual. http://www.intel.com/design/pca/products/pxa255/techdocs.htm, 2005.

[7] T. L. Martin and D. P. Siewiorek. Nonideal battery and main memory effects on cpu speed-setting for low power. *IEEE Trans. Very Large Scale Integr. Syst.*, 9(1):29–34, 2001.

[8] A. Miyoshi, C. Lefurgy, E. V. Hensbergen, R. Rajamony, and R. Rajkumar. Critical power slope: understanding the runtime effects of frequency scaling. In *Proceedings of the 16th International Conference on Supercomputing*, pages 35–44, New York, NY, USA, 2002. ACM Press.

[9] C. Poellabauer, L. Singleton, and K. Schwan. Feedback-based dynamic voltage and frequency scaling for memory-bound real-time applications. In *Proceedings of the 11th IEEE Real-Time and Embedded Technology and Applications Symposium*, volume 00, pages 234–243, 2005.

[10] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2006. ISBN 3-900051-07-0.

[11] D. Rajan, R. Zuck, and C. Poellabauer. Workload-aware dual-speed dynamic voltage scaling. In *Proceedings of the 12th International Conference on Embedded and Real-Time Computing and Applications*, pages 251–256, 2006.

[12] K. Seth, A. Anantaraman, F. Mueller, and E. Rotenberg. FAST: Frequency-aware static timing analysis. *ACM Transactions on Embedded Computing Systems*, 5(1):200–224, 2006.

[13] D. C. Snowdon, S. Ruocco, and G. Heiser. Power Management and Dynamic Voltage Scaling: Myths and Facts. In *Proceedings of the 2005 Workshop on Power Aware Real-time Computing*, New Jersey, USA, Sept. 2005.

[14] V. Venkkatachalam, C. Probst, and M. Franz. A new way of estimating compute boundedness and its application to dynamic voltage scaling. *International Journal on Embedded Systems*, 1(1):64–74, 2006.

[15] M. Weiser, B. Welch, A. J. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *Proceedings of the 1st USENIX Symposium on Operating Systems Design and Implementation*, pages 13–23, 1994.

[16] A. Weissel and F. Bellosa. Process cruise control—event-driven clock scaling for dynamic power management. In *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, Grenoble, France, Oct. 8–11 2002.

[17] F. Xie, M. Martonosi, and S. Malik. Compile-time dynamic voltage scaling settings: Opportunities and limits. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 49–62, New York, NY, USA, 2003. ACM Press.

[18] F. Xie, M. Martonosi, and S. Malik. Efficient behavior-driven runtime dynamic voltage scaling policies. In *Proceedings of the 3rd International Conference on Hardware/Software Codesign and System Synthesis*, pages 105–110, 2005.