epEBench: True Energy Benchmark

Simon Holmbacka* †
Robert Müller*
*Faculty of Mathematics and Computer Science, FernUniversität in Hagen
Universitätsstrasse 11, 58084, Hagen, Germany
simon.holmbacka@fernuni-hagen.de
† Faculty of Science and Engineering, Åbo Akademi University
Vattenborgsvägen 5 20500 Turku, Finland
firstname.lastname@abo.fi

ABSTRACT

This paper presents epEBench – a multi-core benchmark specifically designed for evaluating energy efficiency. It supports workload modeling and workload level modification to generate realistic case studies not earlier supported by benchmark tools. We show how the workload of two real-world application has been extracted and used as a model input to epEBench.

1. INTRODUCTION

From tiny embedded systems to large scale server farms and diverse IoT systems, energy efficiency is becoming the main struggle for system usability, expansion, reliability and scalability. In order to verify energy efficiency in computer systems, the tools and methods must be tested against a strict set of parameters, which usually consists a benchmark. Benchmarks normally run a number of standardized tests in order to stress a system and hereby to provide a method for proving a certain quality of the system. Existing benchmarks usually aim to analyze the achievable performance at maximum capacity rather than focusing on energy. This is not very realistic in everyday applications such as video decoders or other applications with QoS requirements, since the streaming media seldom requires the full capacity of the processor. For systems handling such a fluctuating and unpredictable workload, few benchmarks have obtained the ability to replicate a realistic scenario.

The problem with such workloads is its very unpredictable nature. Since e.g. a video can consist of many formats, resolutions, framerates, encodings etc. it is not possible to generalize the workload. Instead, the workload should adapt to the input format used, and the stress on the processor should follow this format. One solution is to use real-world applications as benchmarks, but repeatability of the experiments is then a problem, as well as automatically configuring the benchmark for other possible scenarios such as scaling the number of threads, thread affinity etc.

Benchmarks for computing systems have existed for many decades, but most benchmarks focus on other characteristics than energy efficiency. Popular benchmark suites such as SPEC, PARSEC, EEMBC and stress-ng contain a large set of computationally intensive microbenchmarks, in other words they spawn high intensity threads with load levels close to 100%. The purpose of such benchmarks is usually to compare the computational power of two systems or the difference in energy efficiency between two systems. In contrast, to compare the energy efficiency of internal mechanisms, such as the scheduler, computationally intensive benchmarks are not able to fully represent interesting case studies because of a constant high workload. For example mapping four threads, each with 100% load, on a quad-core system is trivial for a scheduler. On the other hand, mapping four threads, each with 20% workload, on a quad-core system will result in very different energy consumption depending on the policy chosen.

For the aforementioned reasons, a benchmark which can more closely model a fluctuating and realistic workload is needed. This paper presents epEBench, a multi-core benchmark capable of replicating realistic workloads based on real-world applications. The paper focuses on creating load models by introducing a methodology for gathering and analyzing data and characteristics from everyday applications in order to acquire the notion of an accurate model. We measure the accuracy of the modeled workload compared to the real-world application and we study the energy consumption of several benchmarks executing on a real embedded hardware platform.

2. RELATED WORK

A large number of benchmarks have been presented in previous work. The majority of them, however, aim to analyze the achievable performance by running a number of standardized tests at maximum CPU work load. Furthermore, a minority of the benchmarks are intended for evaluating energy efficiency, which we focus on in this section.

Standard Performance Evaluation Corporation (SPEC)[5] is the largest and a widely accepted benchmark consortium. The SPEC suite contains a branch for measuring energy efficiency, called "SPEC Power". This is theoretically a viable option for evaluating energy efficiency, since the suite contains several microbenchmarks with various workloads and workload levels. However, the benchmark is intended for high-end Intels, and when executing the calibration scripts are run on a low-end ARM device, the performance of the host itself is not sufficient to pass the calibration phase of the benchmark. Secondly, the SPEC benchmarks are only available with a commercial license.

EEMBC [2] is an industry alliance which develops benchmarks to evaluate performance and energy consumption of computer systems, for different device architectures. Even though this benchmark is intended for evaluating energy efficiency, workload level manipulation is not possible which

one of our requirements.

MultiBench [3] utilizes workloads from out of hundred combinations and can be parameterized to vary the amount of concurrency being implemented by the algorithm. This allows the user to test the system and find out bottlenecks. Despite the various features for multi-core architectures and the possibility to scale data input, it still focuses on optimizing program performance as it lacks the capability to scale workloads regarding energy consumption. Similarly to EEMBC, MultiBench lacks the support for workload modification, which is one of the key issues in an energy benchmark. All of the currently available benchmark suites, to our knowledge, lack the support of at least one of the following features:
(1) *Ability of user defined workload modification.* This feature is needed to replicate the actual action of the CPU.
(2) *Ability of workload level manipulation.* This feature is needed to replicate the fluctuating workload levels in real-world applications.
(3) *Automatized execution.* This feature is needed to enable repeatability in experiments.

3. WORKLOAD MODELS

To replicate realistic energy consumption, the power dissipated by the CPU must be accurate to the real-world case. This means that the CPU must operate in a similar way when executing the real-world application and when executing the workload model.

Finding a realistic workload model depends significantly on the application which the platform under analysis is intended for. If, for example, a computing device is most of the time idle waiting for user interaction and only performs I/O operations, the stress on the CPU will be much lower than when using an application which is used for heavy computations such as ALU or SIMD operations. Also the instruction mix will differ from application to application, and as a consequence the ratio in occupying different processing units (e.g. ALU, FPU, SIMD). Rather than fixing the workload parameters like Prime95[1] and CoreMark[2], epEbench is providing a modular workload interface to which workload configuration files can be attached.

[1]www.mersenne.org
[2]www.coremark.org

Extracting instructions. In order to replicate a real-world application, the instructions in the chosen application must be extracted in order to be used for building the workload model. One alternative to extract instruction mixes in applications is to use the Dynamic Rio [1]. The tool allows program analysis such as profiling, performance evaluation, and bug detection. In order to collect run-time information, it uses a technique to insert code into an executable, **in our case we inserted a counter after each instruction**. The code is instrumented dynamically *just in time*, so no recompilation or re-linking is required.

In this work, the instruction mix of the VLC media player [7] extracted. This application was chosen based on the streaming behavior of media playback, and because it is a very popular and mature platform. The VLC[7] media player was compiled with SIMD extensions on both an x86 and on an ARM platform, after which Dynamic Rio was used to analyze the decoding of a 720p video "*The Godfather trailer*". Figure 1 shows the opcode break-down of the VLC playback on the x86 and the ARM platform. As seen in the figure, the workload differs depending on the platform used (and on the compiler used).

Workload generation. Based on the previous analysis, we model workloads with similar characteristics as the extracted opcode mixes. For this purpose *load functions* are created executing the prime characteristic instruction types such as `add` or `mov`. epEBench current supports 24 different load functions. These functions are called to run a certain amount of operation loops in accordance to the ratio identified in the binary analysis above. Each load function should therefore ideally execute only one instruction type in large numbers, in order to simplify the modeling process. However in practice, a fair amount of other instructions are executed beside the target operation type as well, needed to provide the functionality of the whole framework itself, such as load estimation and controlling, operation loop structure and thread management (context switch – store and recover registers). These effects must be isolated to correct the influence on the demanded model characteristic. Listing 1 shows the C implementation of a fixed point `add` operation. To reduce the overhead of the required while-loop, the `add` instructions in the while-loop are performed 20x for each instruction. epEBench then automatizes the execution of each load function according to the input workload model given.

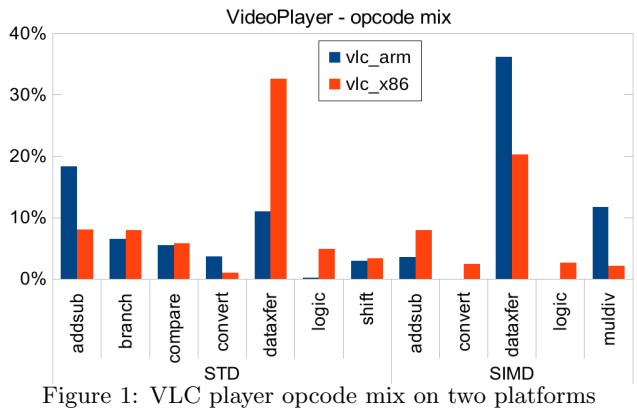

Figure 1: VLC player opcode mix on two platforms

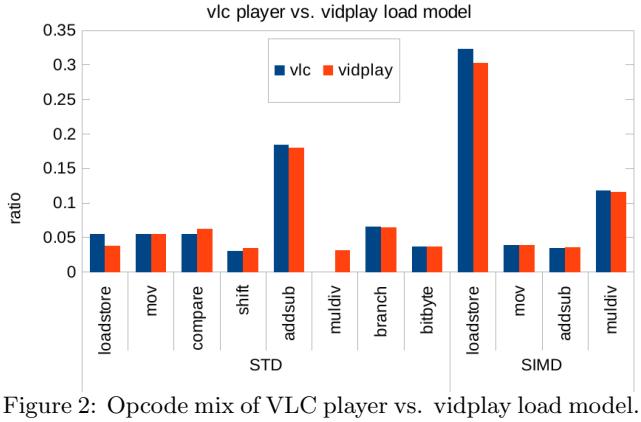

Figure 2: Opcode mix of VLC player vs. vidplay load model.

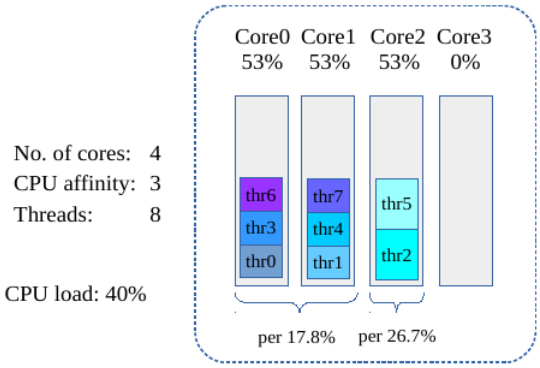

Figure 3: Unison thread and load distribution onto the cores

Dynamic RIO was then used to analyze the opcode mixes of both the real VLC player execution and the model execution on the ARM platform. Figure 2 shows a model "vidplay" comparable with the real execution with an average (absolute) deviation to the VLC binary of less 0.7%. The small number (3%) of muldiv instructions of the vidplay load is effected by the memory access pattern implementation in the relevant load functions.

```
void *run_iadd(int counts){
  volatile int x = 0;
  instcnt += counts * 24;
  while (counts--) {
    x+=1;
    x+=2;
    (...)
    x+=20;
  }
}
```

Listing 1: iadd function (C code)

4. EPEBENCH

epEBench[3] is an opensource energy benchmark for POSIX compatible systems and is freely available at `https://github.com/RobertMueller/epEBench` The benchmark integrates synthetic load models and enables an analytic approach in analyzing the energy efficiency of embedded multi-core systems. The key aspects of the benchmark is mainly the design to allow a wide range of parametrization to give the user the ability to stress certain aspects of the CPU while still being repeatable and realistic enough through simulating real application behavior. The most notable parameters are the **Opcide mix**, which is the actual instructions to replicate in the benchmark. And the **Workload level manipulation** which mimics the load level of the real application threads.

Figure 3 illustrates a benchmark scenario in which eight benchmark threads are mapped on 3 CPUs; six of them having a load level of 17.8% and two threads having a load level of 26.7%. This setup is used to replicate the behavior of for example a media decoder, with a load level depending on the framebuffer size, threads used, video file used etc.

Load controller. The major functionality of the benchmark tool is mainly provided by the model routine running in parallel unsynchronized threads. It features the model implementation, cascaded CPU load controller comprising of the sleep time estimation, and period control loop. The control loop architecture consists of the CPU load controller used to throttle the workload level of a benchmark thread at a fixed level. As illustrated in Figure 4 the mechanism consists of the CPU load controller (top) used to minimize remaining load offset. It receives the CPU *load_w* as feedback signal and provides a correction value to all load generator threads. The period control loop (bottom) regulates the operation time to keep a constant measurement interval for the periodic load estimation. The load generator calculates the sleep interval *tsleep*, in order to gain the demanded CPU load. It receives the correction value *cpucorr_k* from the load controller, and used as input to the inner PI control loop in the lower part in Figure 4. Due to the fluctuation of the operation time, caused by interference of other model tasks and OS processes, the load estimation is performed periodically with an interval of 50ms.

To generate a specific CPU load, the load-generator modulates the run and sleep intervals of the threads. The sleep period is calculated by measuring the operation time within one period to achieve the demanded average load. It is simply regarded as PWM-modulated on-off operation states t_{on} and t_{off}. The average CPU utilization U is then calculated with the following formula: $U_{CPU} = \frac{1}{T}\int_0^T u(t)dt = \frac{1}{T}\int_0^{t_{on}} U_{op}(t)dt$ where t_{on} is the operation time, U_{op} the current CPU-load and T the total period $t_{op} + t_{sleep}$. To estimate the idle period per thread, the operation time t_{op} is measured and the sleep period t_{sleep} is then calculated with: $t_{sleep} = t_{op} \cdot \left(\frac{1}{U_{thread}+k_{CPU}} - 1\right)$, with the correction value k_{CPU} from the load controller and U_{thread} the demanded thread load.

Usage. The main routine provides a text based user-interface to configure the testbench, with the command-line arguments shown in Figure 5.

A standard test run can be invoked by entering for example.
`./epebench -m vidplay -n 8 -a 3 -u 30 -t 20`
This allocates eight threads distributed on three cores with

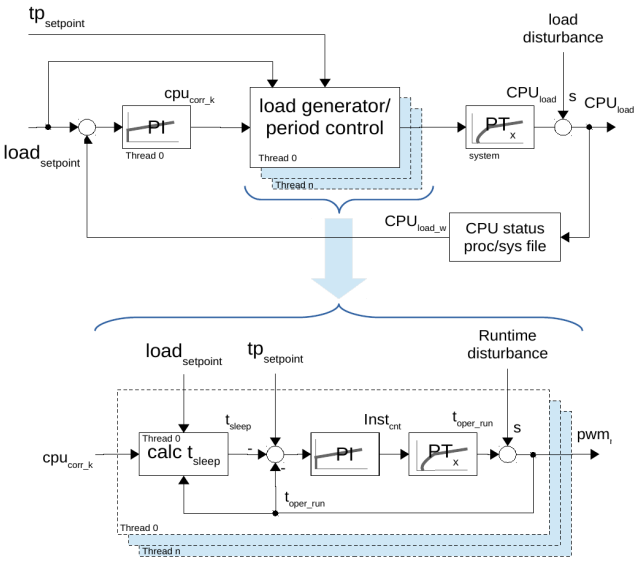

Figure 4: Load level controller in epEBench

[3]More technical details is found in [4].

a total CPU load of 30% for 20 seconds. Figure 5 shows a summary of the used parameters.

```
usage: epebench [-h] [-v] [-tTIME] [-iINSTR] [-mMODEL] [-nTHREADS]
[-aCPU] [-uUSAGE] [-d] [-p] [-r]

Options:
   -t T1 {T2...Tn}        Time in seconds
                          Opt: Enter a list used as test pattern [-p]
   -m MD1 {MD2...Mdn}     Model(s) to execute (see ebmodels.ini)
   -n N {N1...Nn}         Number of threads to allocate
                          Opt: Enter a list used as test pattern [-p]
   -u U {U1...Un}         CPU-load setting (0 - 100)[%]
                          Opt1: Enter a list used as test pattern[-p]
                          Opt2: List of CPU-loads per thread (normal)
   -a CPU {CPU1...CPUn}   CPU affinity; amount or list of used cores
   -i N                   Maximum No. of instructions in Mio
   -d                     CPU-usage process disable. Secondary load
                          controller will not be working
   -p                     Test pattern mode
   -r                     Enable thread pinning mode
   -h                     show this help message and exit
   -v                     show epEBench version
```

Figure 5: epEBench commands

Model accuracy. To determine the ability to replicate a real workload in terms of energy consumption, we compared the real application to the model executed in epEBench. As benchmark target, an Odroid Exynos 5410 board was used, which is an octa core ARMv7-A SoC based on a Cortex-A15 and Cortex-A7 MPCore. It is equipped with a power monitoring unit containing four INA231 current/voltage sensors from TI to measure the power consumption of the A15 cores, A7 cores, GPUs and DRAMs individually. The standard ondemand[6] CPU frequency governor in Linux was used during all experiments.

To initiate the experiment, the VLC application was started reading the same data files as for the binary analysis, and the power consumption and core-utilization was measured. The number of created threads was determined by use of `htop`. The workload of the multi-threaded VLC application is distributed onto eleven threads where one thread is liable for 86% core load. The other threads load the remaining cores with 15%. Then epEBench was initiated to create the same number of threads executing the vidplay workload, and loading the cores likewise. Energy consumption was considered for an execution time interval of 35 sec. The same procedure was also run for generating second a workload model, the `gzip` application, which is significantly more compute intensive than vidplay. This application was also executed for the same amount of time on a single thread and the difference between real application and model was calculated.

From the Figure 6 we can see that both models slightly underestimate the energy consumption of the real application. The energy consumption vidplay model is 10.6% lower than the VLC application, and the energy consumption of the gzip model is 1.6% lower than the GZIP application. As stated, the gzip model uses only a single thread and the workload level is constantly on 100% – this makes it more easy to replicate than the more fluctuating vidplay model. With a workload level constantly on 100%, the load generator is never loosing accuracy since no load level adjustments are needed. The complex vidplay model, on the other hand, reaches a slightly lower level of accuracy compared to the VLC application because of varying workload levels, which the workload generators in epEbench must constantly keep stable (even when affected by frequency scaling).

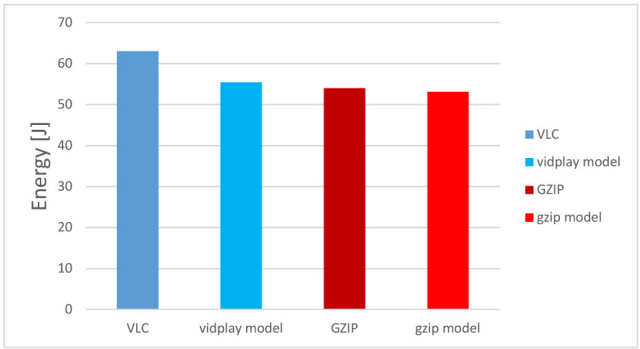

Figure 6: Energy comparison between real application and workload models

5. CONCLUSION

Previous benchmarks have failed to accurately replicate the behavior of streaming applications such as multi-media and signal processing applications. The two main reason for this is the inability of replicating the workload and the inability of replicating the workload level. This paper presented epEBench; a true energy benchmark for all range of multi-core devices. The benchmark is capable of executing workload defined by explicit input models, and it is able to manipulate the workload level of the benchmark threads using a described controller algorithm. This makes it highly repeatable while still keeping the characteristics of real-world workloads. Due to the open model interface and predefined workloads to stress only defined aspects of the workload on the CPU, developers are free to create own workload models based on the applications and parameters of interest. A study on creating workload models by the extractions of CPU opcodes was presented, and the opcode balance between the real application and the workload model was shown accurate. We finally evaluated the energy consumption of two real case-study applications together with the models created based on these applications. This evaluation confirms the representability of executing a workload model for streaming applications.

6. REFERENCES

[1] D. Bruening, T. Garnett, and S. Amarasinghe. An infrastructure for adaptive dynamic optimization. In Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization, CGO '03, pages 265–275, Washington, DC, USA, 2003. IEEE Computer Society.

[2] Industry-Standard Benchmarks for Embedded Systems . EEMBC benchmark. `http://eembc.org/`.

[3] Industry-Standard Benchmarks for Embedded Systems . MultiBench - Performance Suite for Multicore Architectures. `http://eembc.org/multibench/index.php`.

[4] R. Müller. Evaluation of energy consumption using diverse workload on multi-core systems. Master's thesis, FernUniversität in Hagen, 2016.

[5] Standard Performance Evaluation Corporation. SPECPower benchmark. `http://spec.org/`.

[6] V. P. A. Starikovskiy. The ondemand governor. In Proceedings of theLinux Symposium, 2006.

[7] VideoLAN Organization. Video LAN Client. `http://www.videolan.org/vlc/index.html`.