Developing and Evaluating Power Models of Heterogeneous Computer Systems

Master Thesis

Author(s):
Martsenko, Kristina

Publication date:
2021

Permanent link:
https://doi.org/10.3929/ethz-b-000488930

Rights / license:
In Copyright - Non-Commercial Use Permitted

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Master's Thesis Nr. 314

Systems Group, Department of Computer Science, ETH Zurich

Developing and Evaluating Power Models of Heterogeneous Computer Systems

by

Kristina Martšenko

Supervised by

Dr. Michael Giardino
Dr. David Cock
Prof. Timothy Roscoe

September 2020 – March 2021

D INFK

Abstract

As computational demands in the data center increase, so does the prevalence of heterogeneous systems that offload part of the application workload onto a secondary coprocessor. Recently FPGA based systems have become particularly popular due to their energy efficiency and reconfigurability. A central problem in such systems is the task of optimally splitting an application between the CPU and FPGA, which should maximize performance and minimize power consumption. This dictates the need for performance and power models of the system.

Enzian is a new and open CPU-FPGA research platform, which among other things can measure the power of each system component separately. In this thesis we study the power consumption of three Enzian components: CPU, memory and FPGA. We develop simple but accurate analytical energy models for the CPU and memory using hardware performance counters, and we study how properties of an FPGA design affect FPGA power consumption. This constitutes the first study on the power consumption of Enzian and takes a step towards the development of whole system power models for Enzian and heterogeneous systems in general.

Acknowledgements

I would like to thank Dr. Michael Giardino for supervising this thesis, for many discussions and helpful advice during the course of the thesis, and for providing the original research idea and the FPGA benchmark used in the thesis. I would also like to thank Dr. David Cock and Prof. Timothy Roscoe for helpful discussions and advice, for providing access to the Enzian platform, and for the opportunity to write my thesis in the ETH Systems Group. Finally, I would like to thank other members of the Systems Group for help with Enzian and its power management software.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cloud computing is increasingly used for processing large datasets, which requires increasing amounts of computational power. As improvements in the performance of conventional computers have slowed, new heterogeneous computing architectures have emerged. These include systems with GPUs (graphics processing units), FPGAs (field-programmable gate arrays) and ASIC (application-specific integrated circuit) coprocessors. FPGAs are particularly promising due to their high power efficiency and reconfigurability. In recent years a number of CPU-FPGA systems have been developed. These include systems with a PCIe connection between the CPU and FPGA (for example Microsoft Catapult, Amazon F1), as well as systems with shared memory and a coherent interconnect (for example Intel Xeon+FPGA, IBM CAPI) [13].

Recently a new research CPU-FPGA platform called Enzian has been developed [2]. Enzian is a coherent shared memory computer combining a Cavium ThunderX 48-core Arm CPU with a Xilinx Ultrascale+ FPGA. It is an open platform designed for computer systems software research [25].

An important problem for CPU-FPGA platforms is how to allocate work between the CPU and FPGA. Recent work has explored work scheduling from multiple perspectives, such as offloading database operations to the FPGA [2] or sharing the FPGA between multiple applications [38].

Scheduling decisions are driven by considerations such as performance and power consumption. In this thesis we focus on power consumption, as power efficiency has in recent years become an increasingly important issue in cloud data centers [42]. In order to make good scheduling decisions, a power model of the system is needed. In an effort to study power modeling of CPU-FPGA systems, we begin by developing a power model for Enzian. One possible way to do this is to first create power models of each hardware component in the system and then combine them. In this thesis we focus on developing power models for the Enzian CPU, memory and FPGA.

Our main contributions are as follows.

- The first study on the power consumption of Enzian. This includes a demonstration of the per-component power measurement capabilities of Enzian and their potential for future research.
- A study of the correlations between hardware performance counters on a 64-bit Arm CPU and the power consumption of the CPU and DDR4 memory. This includes simple yet highly accurate analytical energy models for the Enzian CPU and memory.
- A study of FPGA power consumption based on utilization and clock frequency. This includes an accurate analytical power model for very simple FPGA designs.

The rest of this thesis is structured as follows. Chapter 2 discusses the related work that has been done on power models and places our work in context. Chapter 3 covers the background concepts necessary for understanding our work. Chapter 4 provides an overview of the Enzian platform. Chapters 5, 6 and 7 present our methodology and results on the power modeling of the CPU, memory and FPGA, respectively. Chapter 8 describes possible future research directions. Chapter 9 concludes the thesis.

Chapter 2

Related Work

This chapter provides an overview of power models that have been developed in the literature. While our focus is on heterogeneous systems, we first look at models for each system component separately (CPU, memory, FPGA) and then conclude by looking at whole system models.

2.1 CPU Power Models

The central processing unit (CPU) has been found to consume more power than any other component in a data center server [4]. The ability to estimate CPU power consumption has many uses, such as energy aware scheduling, dynamic voltage and frequency scaling (DVFS), or taking advantage of low power states [22, 5].

Recent Intel processors report CPU power consumption through its Running Average Power Limit (RAPL) interface [53]. Recent AMD processors support a similar feature called Application Power Management (APM) [15]. However, to our knowledge similar interfaces are not available on other CPU architectures.

Many different CPU power models have been proposed in the literature. Some make use of hardware performance counters provided by the CPU to estimate power [5, 15]. Others use CPU utilization as a predictive metric [26]. Low level tools use simulation and circuit properties to estimate power [39]. In this thesis we focus on the use of performance counters for power modeling.

The use of CPU performance counters to estimate power consumption was first demonstrated by Bellosa [5]. He observed a linear relationship between the number of instructions executed per second and the power use of the CPU. In this thesis we explore similar relationships but on a modern processor with a different architecture (Arm). We also demonstrate the relationships for real-world workloads rather than microbenchmarks, and we are able to measure the CPU power in isolation, rather than whole system power.

Performance counter based power models have been created for Intel XScale PXA255 [16], Intel Pentium 4 [9] and Intel Core 2 Duo processors [6]. Performance counters have also been used to create analytical power models for each CPU component (such as the cache or floating point unit) [34, 6]. Recently, a toolkit was proposed that uses a learning technique to identify which performance counters correlate most with CPU power consumption [15]. In this thesis we do not aim to identify the best counters, but rather to understand and characterize the system using specific counters.

2.2 Memory Power Models

The main memory is one of the biggest consumers of power in a data center server [4]. Memory power models are useful for modeling whole system power [8], and could also enable power saving techniques such as dynamic voltage and frequency scaling (DVFS) in memory [20].

In addition to CPU power, recent Intel processors also report memory power consumption through their RAPL interface [21]. However as already mentioned, not all CPU architectures have similar interfaces.

Several methods exist for estimating memory power. The simplest is to use datasheets and spreadsheet models provided by memory vendors [51]. A more accurate method is to use simulations of the memory or entire memory hierarchy [57, 52, 59, 51]. In this thesis we focus on memory power models based on CPU performance counters.

Bellosa [5] was the first to demonstrate using CPU performance counters to estimate memory power consumption. He found a linear relationship between the number of second-level cache misses and memory power consumption. In this thesis we explore similar relationships but on a modern system with DDR4 memory (and an Arm CPU). We also demonstrate the relationships for real-world workloads rather than microbenchmarks, and we are able to measure the memory power in isolation, rather than whole system power, possibly leading to more accurate estimates.

Other work has explored predicting memory power using instruction fetch miss and data dependency events on an Intel XScale PXA255 processor [16]. Performance counters on an UltraSPARC CPU have also been used to estimate energy used by memory and caches [37]. More recent work has used cache misses, TLB misses and DMA accesses to model the power use of memory [8]. In this thesis we use cache misses on an Arm processor to model memory power consumption.

2.3 FPGA Power Models

While it is possible to complete an FPGA design and measure its power use with physical measurements, power estimation techniques allow power to be known early in the design process and enable power optimization techniques [11]. For example, FPGA power models can be useful for early design space exploration as part of HLS workflows [45].

Power is consumed by the routing fabric, logic blocks and clock network [11, 28, 23]. Power consumption can be broken down into switching power, short-circuit power and static power [11]. Power consumption is affected by FPGA parameters such as clock frequency, utilization, switching activity and interconnect capacitance [1, 3]. A linear relationship has been observed between clock frequency and dynamic power [35]. In this thesis we explore the effects of utilization and clock frequency on the power use of a modern FPGA.

FPGA EDA software contains functionality to estimate the power consumption of a design. Both Xilinx and Intel provide spreadsheet-based tools for early pre-RTL estimation [33, 17], as well as utilities within their EDA tools for more accurate RTL-based estimation [32, 19]. The spreadsheets estimate power based on design parameters input by the user, such as the number of components (clocks, LUTs, registers, BRAM, DSP, I/O) and their properties (clock frequency, toggle rate, other features). The EDA tools provide a power estimate based on a synthesized or placed-and-routed design, which can be further improved with simulation or detailed activity information input by the user. As the tools are proprietary, the exact algorithms used are not publicly known.

In addition to production uses, EDA software estimates are often used to assess the accuracy of new power models and FPGA tools [12, 10, 36]. Previous work has compared Xilinx software estimates with real measurements and found that the tool significantly overestimates power [35, 43]. In this thesis we compare Xilinx power estimates to real power measurements to further study this.

Academic work has explored power estimation methods as alternatives to commercial software. Common approaches use board measurements, statistical models or simulations [11]. Some approaches estimate power consumption using circuit-level simulations and low-level parameters [23]. Others use simulations and device parameters extracted from FPGA vendor tools [3, 12]. Simulation-based power models also exist for evaluating new FPGA architectures [49, 28]. Power models for HLS designs have also been proposed [12]. This includes power models for RTL operators (e.g. adders or multipliers) in HLS [36]. In this thesis we do not explore simulation nor models for HLS.

Previous work presented a methodology based on physical measurements to calibrate a power model for a specific platform [35]. This included using a set of microbenchmarks as well as simulations and design tool information. In this thesis we use a somewhat similar approach, in that we use a microbenchmark to calibrate a model. However, unlike all of the above models, we do not attempt to perform absolute power estimation, but rather explore how design properties affect power use.

Finally, Section 2.1 describes power models for CPUs based on performance counters. Unlike CPUs, FPGAs do not yet include performance counters, so no counter based models exist [46]. An alternative approach would be to predict FPGA power consumption based on execution properties (e.g. performance counters) on another platform, such as a CPU or GPU. To our knowledge no such models exist either [45].

2.4 Heterogeneous System Power Models

Heterogeneous systems include systems with GPUs, FPGAs, ASIC coprocessors, as well as asymmetric multiprocessors. As our focus is on coherent shared memory CPU-FPGA systems like Enzian, we do not look at power models for other kinds of systems here.

One way to create a power model for a CPU-FPGA system is to combine models of system components. This has been done for non-heterogeneous systems, to combine models of the CPU, memory, disk, network and other system components [8, 24]. In this thesis we create simple power models for three components of Enzian (CPU, memory and FPGA).

Several works have studied the performance of cache coherent CPU-FPGA systems [14, 13]. Unfortunately we are aware of few studies on the power consumption of such systems [27].

Chapter 3

Background

3.1 Power Consumption

The *power consumption* of a computer refers to the amount of energy it uses per second. Power is measured in watts (W) and energy in joules (J). The power consumption can be calculated as the product of voltage and current:

$$P = IV \tag{3.1}$$

Voltage is measured in volts (V) and current is measured in amperes (A).

Power consumption is made up of *static power* and *dynamic power*.

Dynamic power refers to the power used when there is circuit activity, to charge the capacitance of gates and wires when signals change state. The dynamic power consumption of each element can be calculated as

$$P_{dynamic} = \frac{1}{2} C {V_{DD}}^2 f \tag{3.2}$$

where V_{DD} is the supply voltage, C is the capacitance of the element, and f is how many times per second the signal changes state [29].

Static power refers to the power drawn when the circuit is idle and the signals do not change state. Static power consumption can be calculated as

$$P_{static} = I_{DD} V_{DD} \tag{3.3}$$

where V_{DD} is the supply voltage and I_{DD} is the leakage current [29].

The approximate energy consumed by an application can be calculated by measuring power at regular intervals, multiplying the measured power values with the elapsed time between measurements, and adding up the resulting energy values.

3.2 Power Measurement

The power consumption of a system can be measured with an external (or internal) power metering device. These include wattmeters, power analyzers, and combinations of galvanometers and voltmeters [42, 46]. They can be attached to the power supply unit (PSU) to measure the power used by the whole system.

In addition, some server models contain dedicated power data acquisition systems connected to their baseboard management controller (BMC) [42]. These can monitor the power of individual system components, such as CPU or memory.

3.3 Performance Counters

Hardware performance counters are a feature in modern CPUs. They provide the user with information about activities happening in hardware, and are intended for performance analysis of software. They were first introduced in the 1990s and today are available on most CPUs. In this thesis we refer to them as simply *performance counters*.

Performance counters count *events* occurring in the CPU and related components. For example, they may count the number of instructions executed or the number of times a cache was accessed. Most CPUs support tens or hundreds of types of events, which are documented in their manuals (for example, [18] or [41]). However, each CPU contains a fixed number of *counters* which must be configured to count one type of event at a time. This limits how many types of events can be counted at once. To count more events than there are counters, the counters need to be multiplexed, resulting in each event only being counted some proportion of the time.

To use performance counters, system software needs to configure them and periodically read (sample) the counts from CPU registers. A number of profilers exist that provide access to the counters. One of the most commonly used tools is the Perf profiler in Linux [48].

3.4 FPGAs

A *Field Programmable Gate Array* (FPGA) is a reconfigurable hardware platform. Unlike an ASIC (application-specific integrated circuit), it can be reprogrammed with different hardware designs. FPGAs are widely used in consumer electronics and prototyping, and have begun to be used as accelerators aside CPUs.

An FPGA consists of an array of *configurable logic blocks* (CLBs). Each logic block contains a number of *lookup tables* (LUTs) and 1-bit *registers.* An N-bit register is a set of N *flip-flops.* The LUTs can be configured to perform various logical functions, while the registers store values. The CLBs are connected through configurable routing networks of wires. FPGAs also contain clock networks for delivering clock signals, input/output elements, digital signal processor (DSP) blocks for arithmetic functions, block RAM (BRAM) memory, and other resources.

The design for an FPGA is created using a *hardware description language* (HDL) such as SystemVerilog or VHDL. HDLs implement the *register-transfer level* (RTL) abstraction. The design is then synthesized, placed and routed to use the resources available on the FPGA. This process produces a *bitstream* file with information on how to configure the LUTs, routing network, and other parts of the FPGA. The bitstream is loaded onto the FPGA to configure it. In recent years *high-level synthesis* (HLS) has become a higher level alternative to HDLs.

The two leading FPGA manufacturers are Xilinx and Intel (formerly Altera). They provide EDA (electronic design automation) tools for synthesizing the FPGA bitstream, namely Vivado Design Suite from Xilinx and Intel Quartus Prime from Intel. Vivado calls the place-and-route phase the *implementation* phase.

3.5 Regression Analysis

Regression analysis is a mathematical method for finding relationships between a target quantity and a number of features that it might depend on. The most common type of regression is *linear regression*, which finds the closest linear relationship, meaning that the target is a linear combination of the features. There are several ways to determine 'closeness', the most common of which is *ordinary least squares*, which minimizes the sum of the squares of the differences between the target values and the values predicted by the linear model.

A regression model is built based on a dataset consisting of observed features and corresponding observed target values. If the model is built based on multiple features, it is called *multiple linear regression*, otherwise in case of one feature it is *simple linear regression.* Other types of regression besides linear include *polynomial regression*, which finds more complex relationships.

The accuracy of a regression model can be assessed using a number of metrics. One of the simplest is *mean squared error* (MSE) which calculates the average of the squares of the differences between the target and predicted values. A lower MSE is better, with 0 being best. The unit of MSE is the unit of the target quantity. Another metric is the *coefficient of determi-*

nation or R^2 score, which shows how much of the variance in the target quantity is predicted by the features. The best possible score is 1, with lower values being worse.

A number of software packages exist for performing regression analysis. In this thesis we use the scikit-learn [47] Python library to create all regression models.

Chapter 4

Enzian Platform

Enzian is a research computer built at ETH Zürich for systems software research [25, 2]. It is intended to be a open system and available to anyone. This chapter describes the Enzian platform and its power measurement infrastructure.

4.1 Architecture

Enzian is a heterogeneous NUMA (non-uniform memory access) system with a CPU and an FPGA and a cache-coherent interconnect between them. Figure 4.1 illustrates the Enzian architecture.

The CPU in Enzian is a Marvell Cavium ThunderX-1 CN8890-NT, which consists of 48 ARMv8.1 cores [25]. We will refer to it as simply ThunderX in this thesis. The CPU runs at a clock frequency of 2 GHz. During the course of this thesis Enzian did not support voltage or frequency scaling of the CPU. The CPU has local access to 128 GiB of DDR4 SDRAM memory. The Enzian used as part of this thesis was running the Ubuntu 18.04.4 Linux distribution.

The FPGA is a Xilinx Virtex UltraScale+ XCVU9P. It has local access to 512 GiB of DDR4 SDRAM memory.

The interconnect supports a shared physical address space between CPU and FPGA, similarly to the Intel Xeon+FPGA and IBM CAPI based systems [13]. Coherency is handled by the Enzian Coherence Interface (ECI). Enzian also supports up to 480Gb/s of network bandwidth.

4.2 Power Measurement

Enzian has been designed to enable research on power consumption.

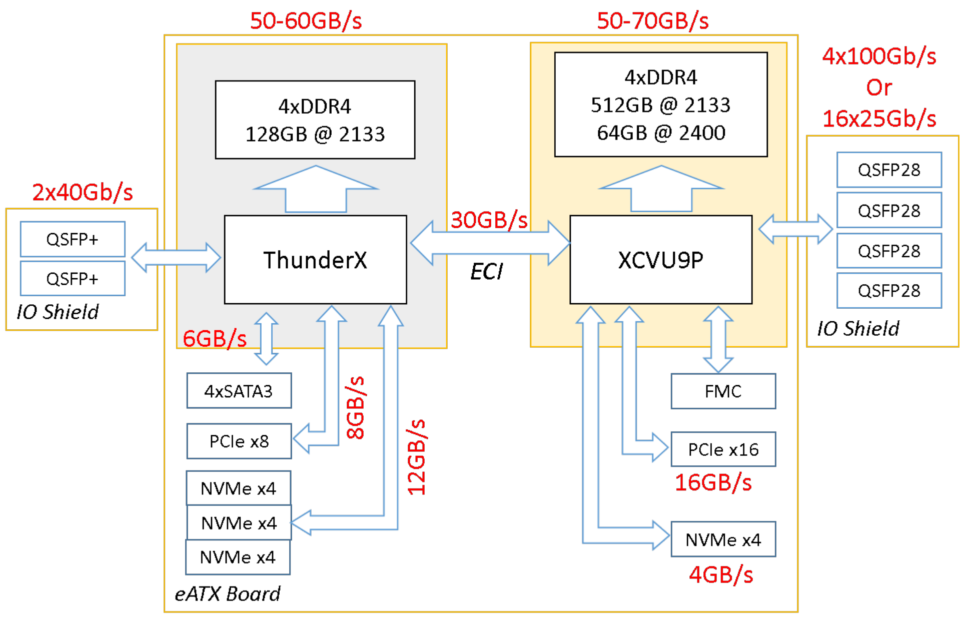


Figure 4.1: Enzian architecture. Reproduced from [25].

As in all systems, power is distributed to the system components over power rails (also *voltage rails*) delivering different voltages. Regulators on the rails guarantee the desired voltage. Enzian differs from most other systems in that it has been designed with voltage and current monitors (sensors) on most power rails in the system. This enables the power consumption of each system component to be measured in isolation of other components. The power consumption of the CPU, CPU DRAM, FPGA and FPGA DRAM can all be measured separately. We make use of this functionality in this thesis by measuring the power of the CPU, CPU DRAM and FPGA independently.

To be specific, the voltage on the CPU and FPGA rails is monitored by Lattice Semiconductor ispPAC-POWR1220AT8 monitors and on the DRAM rails by Texas Instruments INA226 monitors. The current on the CPU rails is monitored from Infineon IR3581 and Maxim MAX15301 regulators, on the FPGA rails from Maxim MAX20751 and Maxim MAX15301 regulators, and on the DRAM rails by Texas Instruments INA226 monitors.

Like most server-class systems, Enzian contains a baseboard management controller (BMC). On Enzian this is a Xilinx Zynq MPSoc CPU running the OpenBMC Linux distribution. Among other things, the BMC is responsible for reading measurements from the voltage and current monitors. This is accomplished by power management software running on the BMC. The software is written in Python and uses kernel drivers for measurements.

Chapter 5

CPU Power Model

As described in Chapter 1, one possible first step in defining a power model for a heterogeneous system is to define models for each of its components. The CPU is a central component of every computing system and consumes a significant amount of power. In this chapter we introduce a power model for the Enzian CPU.

As we saw in Section 2.1, a number of different methods exist for estimating the power use of an application running on a CPU. One way is to rely on metrics about the application that correlate with power use. For example, it has been found that some CPU performance counters are highly predictive of power consumption. In the following sections we examine how performance counters on the Enzian CPU correspond to its power consumption and derive a power model based on them.

Section 5.1 describes the application workloads we use for our experiments. Section 5.2 describes performance counters on the Enzian CPU and how we make use of them. Section 5.3 describes how we perform power measurements. Section 5.4 presents our findings on how performance metrics and power use change during the course of an application run. Section 5.5 introduces a power model based on performance counters that correlate with power.

5.1 Benchmarks

In order to collect and compare performance counters with power measurements, we need a set of benchmarks to run. In this section we describe our choice of benchmarks and how they were run.

5.1.1 Choice of Benchmarks

The chosen benchmarks should be diverse, with different profiles. They should run for a significant amount of time, to discount setup time. They should use a significant proportion of CPU resources. For instance, the ThunderX CPU in Enzian has 48 cores, so the benchmarks should be highly parallel to make use of them. Ideally the benchmarks would exercise the different performance counters that we are interested in. As such, synthetic benchmarks tailored to the performance counters would be ideal. However, real-world workloads would provide more realistic results.

We chose to use the PARSEC benchmark suite [7], rather than write our own synthetic benchmarks. PARSEC is a suite of multithreaded programs created at Princeton University in 2011 to better represent realistic parallel workloads. It is somewhat similar to the SPEC benchmark suites [54] but is freely available to use. The PARSEC suite consists of a diverse set of benchmarks representing real life workloads. Table 5.1 describes some of the benchmarks. All benchmarks are highly parallel and can be run with many threads. The runtime of the benchmarks is on the order of minutes, which is sufficient for our purposes. Table 5.1 shows approximate runtimes on the Enzian CPU.

5.1.2 Benchmark Configuration

For our study we used 9 benchmarks from PARSEC version 3.0. Table 5.1 lists these benchmarks. Out of the 13 benchmarks in PARSEC, we were not able to get the remaining 4 to either compile or run, so did not use them. (We also did not use the SPLASH-2 HPC benchmarks or network benchmarks present in PARSEC 3.0, although they could be interesting.) A few changes were needed to compile PARSEC for the 64-bit Arm architecture. Specifically, we applied a change provided by the Arm Research Starter Kit [55] to enable 64-bit Arm atomics and to statically link the benchmarks. In addition, a few minor fixes were needed to enable some benchmarks to be compiled with a newer compiler and tools. The benchmarks were compiled with GCC version 7.5. We ran all benchmarks with the largest input size available (called the *native* input set in PARSEC). All times shown in Table 5.1 were attained with this input set.

We initially tried to run benchmarks as single-threaded, however we found that then the power use of the 48-core CPU varied by only a few watts. This made it difficult to compare benchmarks, given that there was also a few watts of noise in the measurements.

Instead we ran all benchmarks with 64 threads. This applies to all results presented in the remainder of this chapter. When running benchmarks with 64 threads, the power use of the CPU varied significantly (up to 60

		Runtime	
Benchmark	**Description**	**1 thread**	**64 threads**
blackscholes	Option pricing with Black-Scholes Partial Differential Equation (PDE)	6m 8s	1m 17s
bodytrack	Body tracking of a person	10m 42s	0m 44s
canneal	Simulated cache-aware annealing to optimize routing cost of a chip design	5m 59s	1m 14s
dedup	Next-generation compression with data deduplication	0m 58s	0m 33s
facesim	Simulates the motions of a human face	32m 42s	1m 57s
fluidanimate	Fluid dynamics for animation purposes with Smoothed Particle Hydrodynamics (SPH) method	23m 57s	1m 24s
streamcluster	Online clustering of an input stream	26m 0s	2m 40s
swaptions	Pricing of a portfolio of swaptions	16m 39s	0m 24s
vips	Image processing	7m 17s	0m 10s

Table 5.1: Some PARSEC benchmarks and their approximate runtimes on Enzian. Benchmark descriptions are copied verbatim from the PARSEC website [56].

watts). While the Enzian CPU has 48 cores, we could not run all benchmarks with 48 threads as some of them only supported a power-of-2 number of threads.

We always ran benchmarks multiple times to ensure that results were consistent.

5.2 Performance Counters

In this section we describe how we use the performance counters on the Enzian CPU.

5.2.1 Event Types

As described in Section 3.3, performance counters enable the measurement of some aspect of a program's execution. This takes the form of counting CPU events of a certain type. Each CPU supports a specific set of event

Event#	Architectural name	`perf` name	Description
0x11	CPU_CYCLES	`cycles`	Number of CPU clock cycles elapsed
0x08	INST_RETIRED	`instructions`	Number of instructions executed
0x0C	PC_WRITE_RETIRED	`branches`	Number of branches taken
0x10	BR_MIS_PRED	`branch-misses`	Number of branches mispredicted or not predicted (or not taken)

Table 5.2: A subset of ThunderX performance counter event types. Details can be found in the Arm architecture manual [40].

types that can be recorded. We could experiment with all of these event types to find out which of them correlate with CPU power use. However, there are too many, as most CPUs support tens or hundreds of event types. Table 5.2 shows examples of some of the types of events available on the ThunderX CPU in Enzian.

We would like to investigate events that are likely to correlate with power consumption. A common CPU performance metric is Instructions Per Cycle (IPC), which is the average number of instructions executed per CPU clock cycle. It seems possible that the more instructions are executed in a given time period, the more energy is used in that time period. Indeed it has been shown on other systems that this is the case [5]. To see whether this is true for the Enzian CPU, we count INST_RETIRED and CPU_CYCLES events (Table 5.2) and calculate IPC from them. Of course there are many other types of events that could correlate with power use, but here we start by looking at IPC.

5.2.2 Counter Configuration

We collected performance event counts using the Linux Perf profiler [48] version 4.15.17 while running benchmarks on a Ubuntu 18.04.4 system. We used the `perf stat` command, either on its own for a full benchmark run (Section 5.5), or with the `--interval-print` parameter for time series measurements (Section 5.4).

Performance events were counted on all 48 cores. All relevant events were counted (for example all instructions executed in both user and kernel mode), because all of them affect power use. The ThunderX CPU has 6 performance counters [50], so up to 6 events can be monitored simultaneously. To avoid multiplexing the counters, we monitored up to 6 events in a single run.

For time series measurements (Section 5.4), we recorded event counts every 20 milliseconds. Initially we tried recording them at longer intervals, such as every 1000 ms or 100 ms. For three of the benchmarks (blackscholes, canneal, and swaptions) we found that 1000 ms was enough, as the event counts changed slower than every 1000 ms, so it was possible to accurately match power measurements to event counts. However, for the remaining six benchmarks, we found that a shorter interval of 20 ms was necessary in order to capture the frequently changing phases of the benchmark, in particular when performing time series measurements. The interval could have been reduced further to be lower than 20 ms (down to 2 ms), but we chose 20 ms to remain consistent with chosen power measurement interval, which could not be reduced (Section 5.3.1).

5.3 Power Measurements

To study how CPU performance counter events relate to CPU power use, we need to be able to measure the CPU power consumption. This section describes how we perform power measurements and how we synchronize the measurements with performance counters.

5.3.1 Voltage and Current Monitors

As described in Section 4.2, Enzian has separate power measurement devices on each component of the system. This means we can measure the CPU power directly, without deriving it from whole system power.

The CPU is powered by 3 power rails: VDD_CORE providing 0.964 V, 0V9_VDD_OCT providing 0.9 V, and 1V5_VDD_OCT providing 1.5 V. All of the rails have voltage and current monitors. From voltage and current measurements we can calculate power use at specific timepoints, and from these we can calculate total energy used across the run of the application (Section 3.1). As we found the voltage on the rails to be nearly constant (and this is the purpose of voltage regulators), we only read the current monitors.

Monitors were read every 20 milliseconds. Initially we tried reading them at longer intervals, such as every 1000 ms or 100 ms. Similarly to performance counter events (Section 5.2.2), we found that for three of the benchmarks 1000 ms was enough, as the power use changed slower than every 1000 ms. However, for the remaining six benchmarks, a shorter interval of 20 ms was necessary to capture the frequently changing phases (and power use) of the benchmark.

For some of the benchmarks, it may have been beneficial to reduce the measurement interval even further below 20 ms. Unfortunately this was not possible, as performing reads of the monitors required a certain amount of time. In particular, reading all three CPU current monitors required at least 15 ms. It's worth noting that this is likely to be a limitation in the hardware design of Enzian. A different design might enable the monitors to be read faster, for example by having separate hardware buses for the monitors, thus enabling more frequent power measurements.

5.3.2 Synchronization with performance counters

As described in Section 4.2, power measurements on Enzian are performed from the BMC. In the meantime, CPU performance counters must be read on the CPU. We want to see how the CPU events at some timepoint affect the CPU power use at that timepoint. This means that we need a mechanism to synchronize the data collected from the BMC and CPU. The easiest way to do this is by using timestamps.

As our measurements are performed every 20 ms, we need the time on the CPU and BMC to be accurate to at least that granularity. In our experimental setup, we use the widely used Network Time Protocol (NTP) to keep time on the CPU and BMC synchronized. We set up a local NTP server to increase accuracy. NTP has been found to be accurate to less than a millisecond in local networks [44], so is sufficient for our use case. Figure 5.1 illustrates our setup.

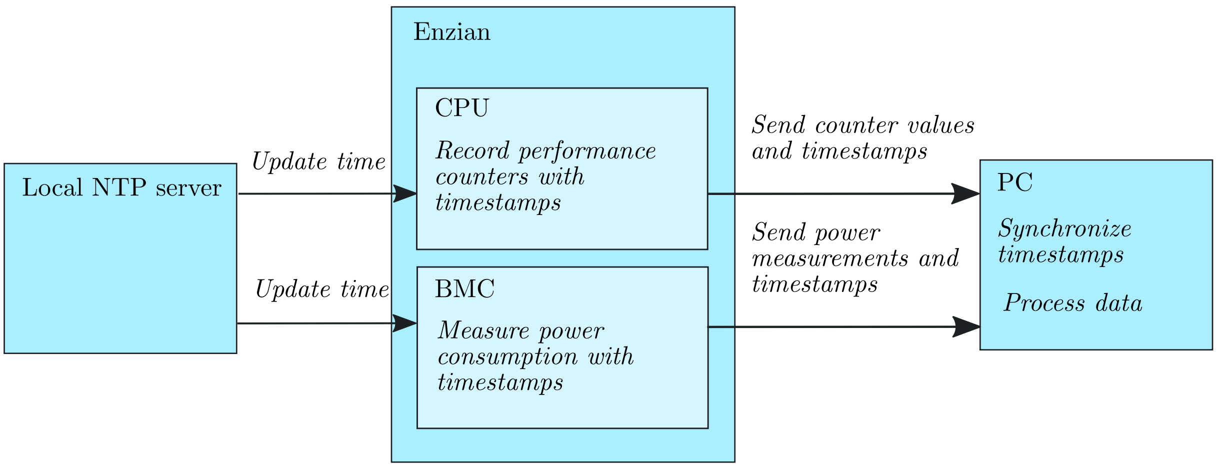

Enzian
CPU
Record performance counters with timestamps
BMC
Measure power consumption with timestamps
Local NTP server
Update time
Update time
Send counter values and timestamps
Send power measurements and timestamps
PC
Synchronize timestamps
Process data

Figure 5.1: Synchronization of CPU and BMC measurements

5.4 Time Series Analysis

This section presents our experimental results on how the IPC and power consumption of the chosen benchmarks change over time. Section 5.5 will present a model based on all benchmarks, but here we look at each benchmark's profile individually. This can provide an intuitive understanding of what is happening during the execution of the benchmarks.

Figures 5.2 to 5.5 show our results for 4 of the 9 benchmarks. The plots for the remaining 5 benchmarks are mostly similar, and can be found in Appendix A.

Each plot shows time series measurements with datapoints 20 ms apart. The x-axis shows the runtime of each benchmark, while the y-axis shows both the IPC and CPU power consumption. (Note that the runtimes as well as the y-axis ranges differ between benchmarks.)

IPC is calculated by dividing total instructions executed on all 48 cores of the CPU by total clock cycles executed by all 48 cores. Therefore it is the average per-core IPC.

For each benchmark, two plots are shown. One is the entire run of the benchmark, while the other ('zoomed') is a smaller time window extracted from the same run, showing the very short execution phases of some of the benchmarks. (Note that the length of the displayed time window differs between benchmarks.)

From the plots we can see that CPU power consumption varies between about 70 and 130 watts. Meanwhile, the average IPC of a core varies between 0 and 0.7.

For 8 benchmarks out of 9, there is a strong correlation between IPC and CPU power use, similar to the *blackscholes* and *facesim* benchmarks shown here. This suggests that IPC would likely make a good predictor of power use. In Section 5.5 we will derive a model of power use based on IPC.

An exception is the *streamcluster* benchmark, where IPC and power seem to correlate less. It is not clear why this is, but it is possible that IPC is not measured frequently enough to capture the very short phases of the benchmark. As power is a snapshot from a single timepoint, but IPC represents all events in a time window, it is possible that shorter IPC windows would display spikes similar to power. As explained in Section 5.2.2, we could not reduce the sampling period below 20 ms.

It is also visible that in the case of some benchmarks, such as *blackscholes* or *swaptions*, power use gradually increases over a few seconds or even tens of seconds after IPC sharply increases and remains high. This could potentially be explained by a gradual increase in temperature, which in turn would cause an increase in current flow and therefore power.

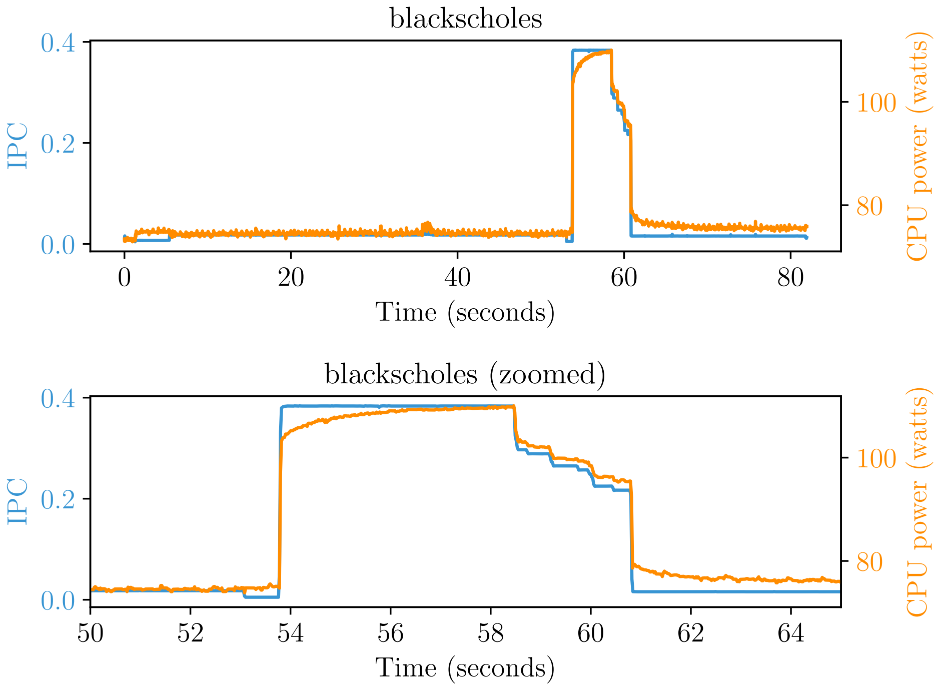


Figure 5.2: IPC and CPU power use of *blackscholes* over time

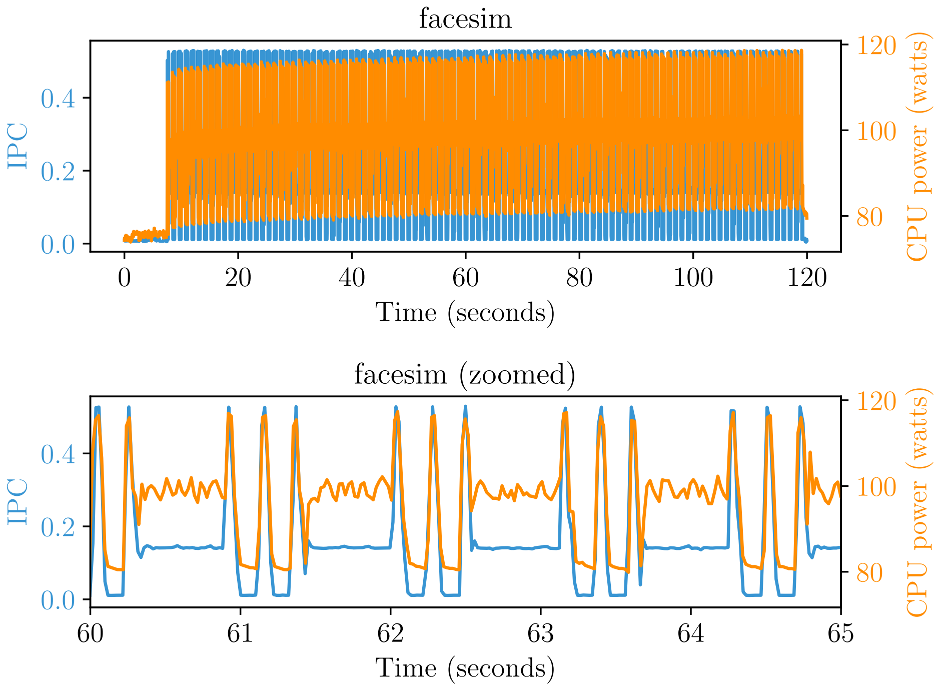


Figure 5.3: IPC and CPU power use of *facesim* over time

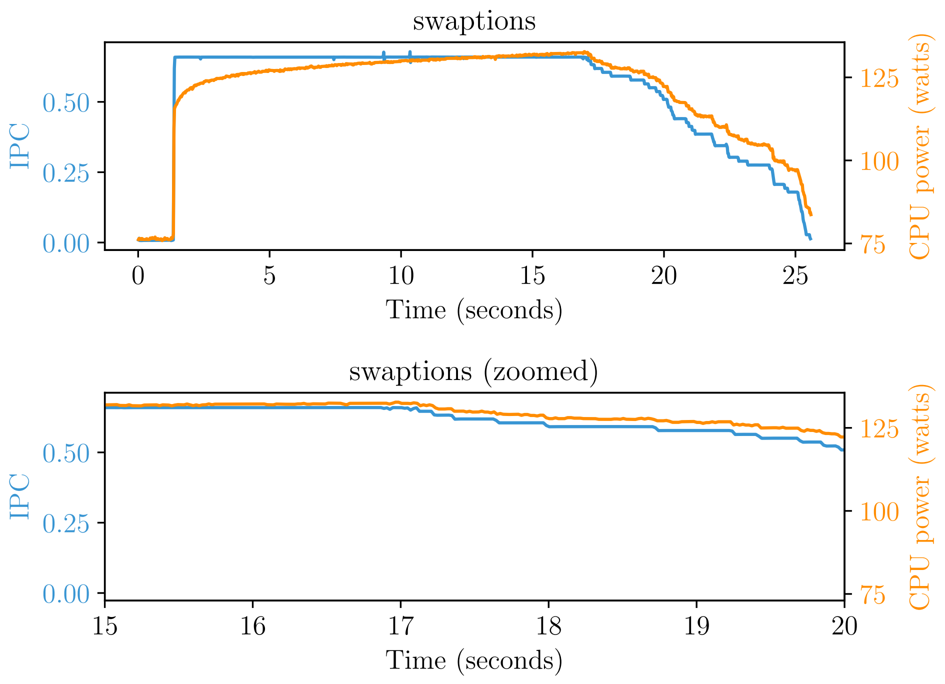


Figure 5.4: IPC and CPU power use of *swaptions* over time

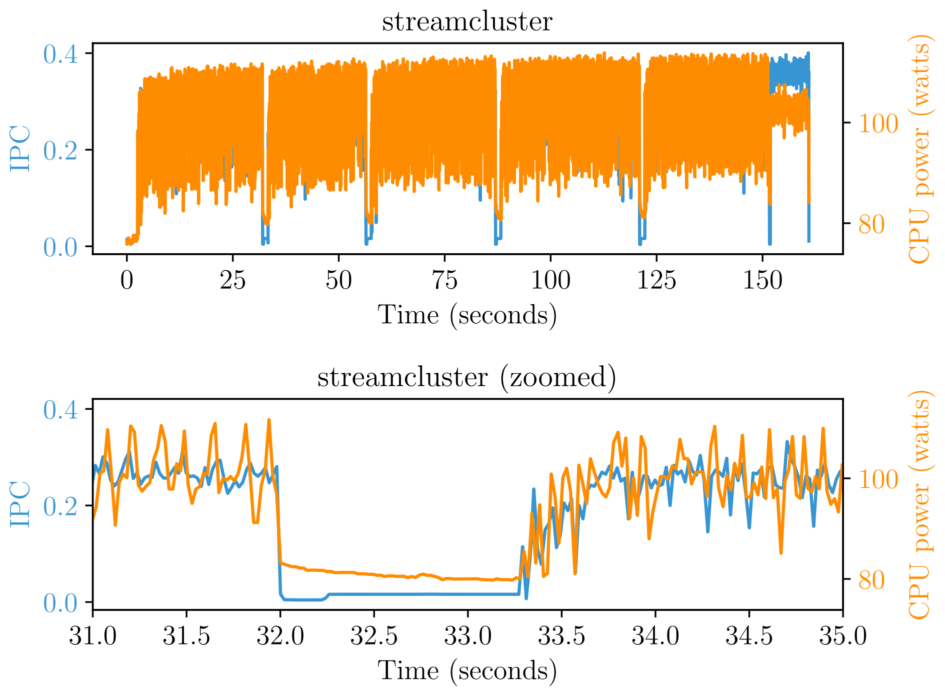


Figure 5.5: IPC and CPU power use of *streamcluster* over time

5.5 Power and Energy Models

We would like to create a model that could be used to predict the power consumption and energy used by an application based on recorded performance event counts. From time series measurements we have seen that IPC appears to correlate well with CPU power use. We can therefore create a CPU power model based on IPC and test how well it predicts power use of an application.

We create a simple model using all of our benchmarks. The model is based on aggregate measurements from a complete run of a benchmark, rather than individual timepoints like in the previous section. For example, we use the total number of instructions executed by a benchmark, or the total IPC across all timepoints. Similarly, we use total energy used by a benchmark, or average power use across timepoints. (Individual timepoints could be used as further datapoints, but we do not do that here.)

We use ordinary least squares multiple linear regression to construct the models. The MSE and R^2 scores reported here are based on a test set randomly selected by scikit-learn, where the training set consisted of 6 datapoints and the test set of 3 datapoints. As there are only 9 datapoints (one for each benchmark), the R^2 scores varied somewhat.

Note that we are interested in the relative change in power depending on IPC. We do not attempt to estimate the absolute CPU power use, but rather assume that baseline power is measured and see how IPC affects it.

We create separate models for the average power use and the total energy used by an application.

5.5.1 Average Power Model

It can be useful to predict the average power consumption of an application based on its IPC. Here IPC refers to total executed instructions divided by total elapsed cycles during the application run.

Figure 5.6 shows the relationship between IPC and average CPU power use obtained from our benchmarks. The line shows the fitted linear model, which is

$$P_{CPU} = IPC \times 89.57 + 77.65 \ (W)$$

This model suggests that the baseline power used by the processor is 77.65 watts, while an additional instruction executed in every cycle would use an additional 89.57 watts of power.

For this model we obtain an MSE of 7.55 and an R^2 score of 0.98, which indicates high accuracy (although this is limited by the very small sample

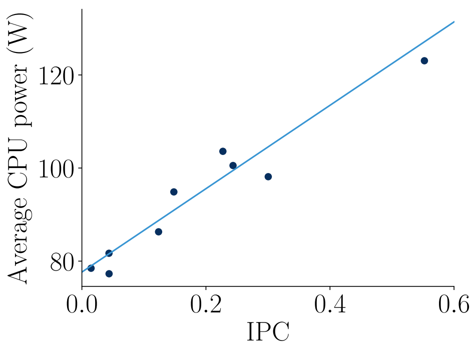


Figure 5.6: Model of average CPU power based on IPC

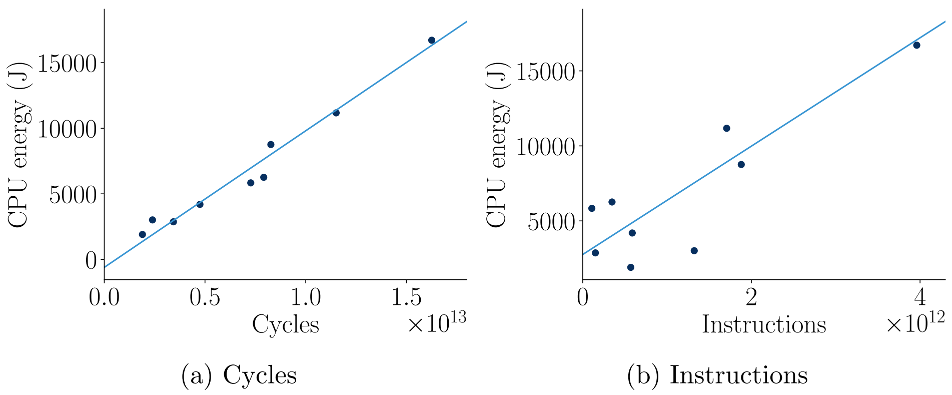


Figure 5.7: CPU energy model based on the number of (a) cycles or (b) instructions

size).

5.5.2 Energy Model

In addition to the average power consumption, it would be useful to know the total energy consumed by the whole application run.

For an energy model, using IPC does not make sense as it does not take into account the runtime of the application. The longer a program runs, the more energy it consumes. The runtime can be factored in by counting the number of cycles elapsed.

Figure 5.7a shows the relationship between total energy consumption and total cycles elapsed. The model obtains an MSE of 854203.04 and an R^2 score of 0.36, indicating that cycles alone are not very predictive of energy consumed.

Another option is to look at the total number of instructions executed. A model based only on the number of instructions is of course even worse, as it does not take the runtime into account. The model in Figure 5.7b has an MSE of 9384570.08 and an R^2 score of -6.00.

We can however factor in both the number of cycles and instructions, to create the following energy model:

$$E_{CPU} = N_{cycles} \times 7.87 \times 10^{-10} + N_{instructions} \times 1.06 \times 10^{-9} - 41.61 \ (J)$$

where N_{cycles} is the total number of cycles elapsed and $N_{instructions}$ is the total number of instructions executed by the application. According to this model, each cycle consumes 0.787 nanojoules of energy, and each instruction 1.06 nanojoules. The intercept (-41.61) is near 0, as expected, since executing for 0 cycles should consume no energy.

This model performs much better, obtaining an MSE of 23852.18 and an R^2 score of 0.98. (Again with the caveat that the sample size is small.) This means that the model could be used to accurately estimate the energy consumed by any application running on the ThunderX. Similar models could be constructed for other CPUs. Knowing the CPU energy use could be useful for deciding when to run an application on the CPU or when to offload it onto an FPGA.

Chapter 6

Memory Power Model

In the previous chapter we defined a power model for the Enzian CPU. In addition to the CPU, the other important component of any computer system is its main memory. The power consumed by the memory subsystem can be large, especially if the system contains large amounts of physical memory (as Enzian does) and needs to process large datasets. Therefore it is important to be able to estimate the power consumed by the memory subsystem. In this chapter we introduce a power model for the Enzian DRAM.

We adopt a similar approach to Chapter 5 and use CPU performance counters. We examine how performance counter events on the CPU correspond to the power consumption of the memory and use them to derive a power model for the memory.

As in the previous chapter, Section 6.1 describes the benchmarks, Section 6.2 the performance counters, and Section 6.3 the power measurements. Section 6.4 examines runtime changes in power and Section 6.5 presents a power model.

6.1 Benchmarks

We need a set of workloads to collect performance counters and memory power measurements from.

6.1.1 Choice of Benchmarks

Our requirements for memory benchmarks are similar to those of CPU benchmarks (Section 5.1.1). The benchmarks should be diverse and sufficiently long-running. Some benchmarks should spend a large proportion of their time accessing memory while others should be compute-bound. A highly parallel benchmark would enable many concurrent memory requests, which could increase memory power consumption.

We found that the PARSEC benchmark suite (described in Section 5.1.1) was also suitable for modeling memory power. It is highly parallel and the benchmarks exhibit differing memory access patterns. We used the same set of benchmarks as for CPU modeling (listed in Table 5.1).

6.1.2 Benchmark Configuration

For the results presented in this chapter, we compiled and ran the benchmarks in the same way as described in Section 5.1.2.

6.2 Performance Counters

In this section we describe how we choose CPU performance counters for memory power estimation.

6.2.1 Event Types

We would like to find which CPU performance counter events (if any) are correlated with memory power consumption. As mentioned in Section 3.3, most CPUs support counting many types of events. While it would be possible to study them all, it is unlikely that most of them significantly predict memory power consumption.

Memory power is likely to increase when memory accesses are made, so the most promising candidates are counters which count memory access operations or events. Table 6.1 lists some events of this kind that are available on the ThunderX. Of these, memory read and write operations may result in cache hits and not proceed to generate memory accesses. Cache miss events, on the other hand, are likely to generate memory accesses in most cases. As the ThunderX contains two levels of cache [58], with a unified Level 2 cache, then misses in the Level 2 cache in particular are likely to lead to memory accesses. We therefore start by counting the L2D_CACHE_REFILL events to see how well they predict memory power use.

6.2.2 Counter Configuration

We collected performance event counts using the same tools and methodology as described in Section 5.2.2.

6.3 Power Measurements

This section describes how we perform memory power measurements.

Event#	Architectural name	`perf` name	Description
0x06	LD_RETIRED	`ld_retired`	Number of memory read instructions executed
0x07	ST_RETIRED	`st_retired`	Number of memory write instructions executed
0x13	MEM_ACCESS	`mem_access`	Memory read or write operations for data
0x04	L1D_CACHE	`cache-references`	Level 1 data cache accesses
0x03	L1D_CACHE_REFILL	`cache-misses`	Level 1 data cache misses
0x16	L2D_CACHE	`l2d_cache`	Level 2 unified cache accesses
0x17	L2D_CACHE_REFILL	`l2d_cache_refill`	Level 2 unified cache misses
0x18	L2D_CACHE_WB	`l2d_cache_wb`	Write-backs to memory from the Level 2 unified cache

Table 6.1: A subset of ThunderX event types related to memory accesses. Details can be found in the Arm architecture manual [40].

6.3.1 Voltage and Current Monitors

We measure memory power similarly to CPU power (Section 5.3.1). The memory is powered by 2 power rails: VDD_DDRCPU13 providing 1.2 V, and VDD_DDRCPU24 also providing 1.2 V. Both rails are equipped with voltage and current monitors, so the power consumed by the memory can be measured separately from the power consumed by the rest of the system. As the voltage was again nearly constant, we only measured current.

We measured the current every 20 milliseconds. A longer interval would not have been ideal, as we found that, for the benchmarks we used, the memory power use changed too frequently (similarly to CPU power use, Section 5.3.1). It was also not possible to make the interval significantly shorter, as reading the 2 memory current monitors required at least 12 ms. We chose 20 ms to remain consistent with the interval chosen for CPU power measurements.

6.3.2 Synchronization with performance counters

The memory power measurements on the BMC are synchronized with the performance counter readings on the CPU using timestamps. Our experimental setup is the same as described in Section 5.3.2 for CPU power measurements.

6.4 Time Series Analysis

In this section we describe our results on how the cache miss rate and memory power consumption change over time during a benchmark run.

Figures 6.1 to 6.4 show our time series measurements for 4 of the 9 benchmarks. These 4 are also representative of the plots for the other 5 benchmarks, which can be found in Appendix B.

The plots consist of datapoints that are 20 ms apart. As in Section 5.4, both the whole run of a benchmark as well as a 'zoomed' shorter time window are shown. (Note that axis ranges differ between benchmarks.)

The y-axis shows both the cache misses as well as the memory power consumption. Cache misses refer to LLC (last-level cache) misses, which are likely to access memory. Recall from Section 6.2.1 that the last-level cache on the ThunderX CPU is the Level 2 cache. The number of cache misses on the plots refers to the number of misses in a 20 ms time window.

It can be seen from the plots that memory power consumption varies between about 7 and 12 watts.

For 8 benchmarks out of 9, there is a strong correlation between the number of last-level cache misses and memory power use, similar to what we see for the *blackscholes* and *facesim* benchmarks here. For example, for *facesim* we see that a spike in LLC misses is followed by a roughly 40% increase in power. This means that LLC misses could be a strong predictor of memory power use. Section 6.5 will explore creating a model based on it.

One exception is again the *streamcluster* benchmark, where the power use does not seem to be related to the number of LLC misses during certain phases. As also explained in Section 5.4 with regards to *streamcluster*, this could be due to the relatively long performance counter sampling period (20 ms), which could be hiding spikes in the LLC miss counts.

In Section 5.4 we found that the CPU power sometimes gradually increases after a rise in IPC, which could be due to an increase in temperature. This could be seen in the *blackscholes* benchmark, for example. Here we see that DRAM power does not increase gradually after an increase in LLC misses, but rises immediately and then remains constant. This suggests some differences in the way CPUs and DRAM operate.

Another difference is that unlike CPU power use, which increases as soon as IPC increases, DRAM power use increases about 25 milliseconds after memory accesses miss in the CPU last-level cache. This can be seen in the *fluidanimate* benchmark, for example, where the spikes in power lag behind spikes in cache misses. This may be explained by memory access latency, as it takes some time for a memory access request to reach the memory, or

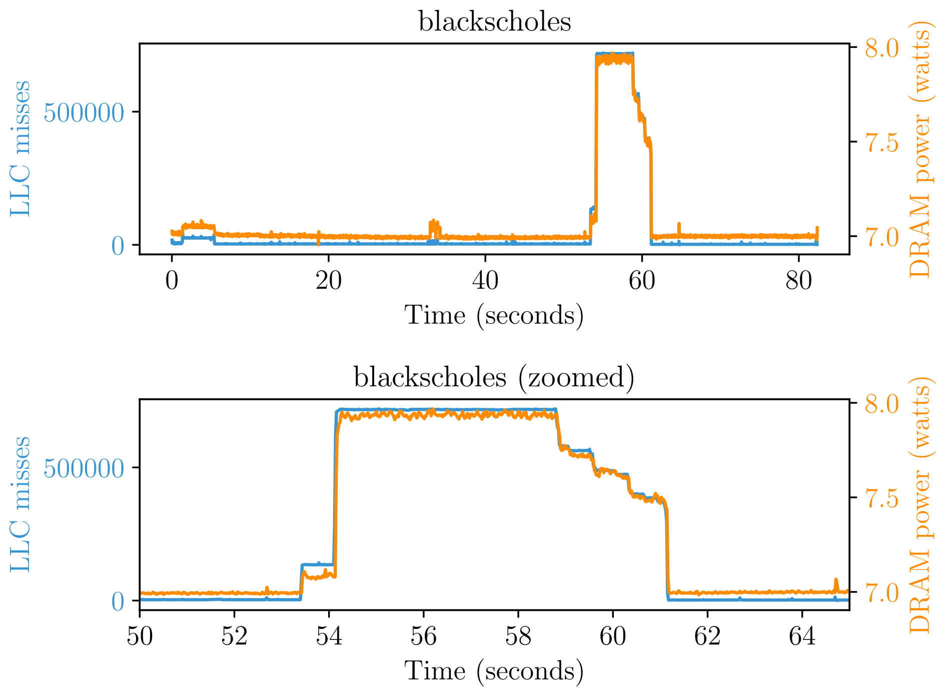


Figure 6.1: Cache misses and memory power use of *blackscholes* over time

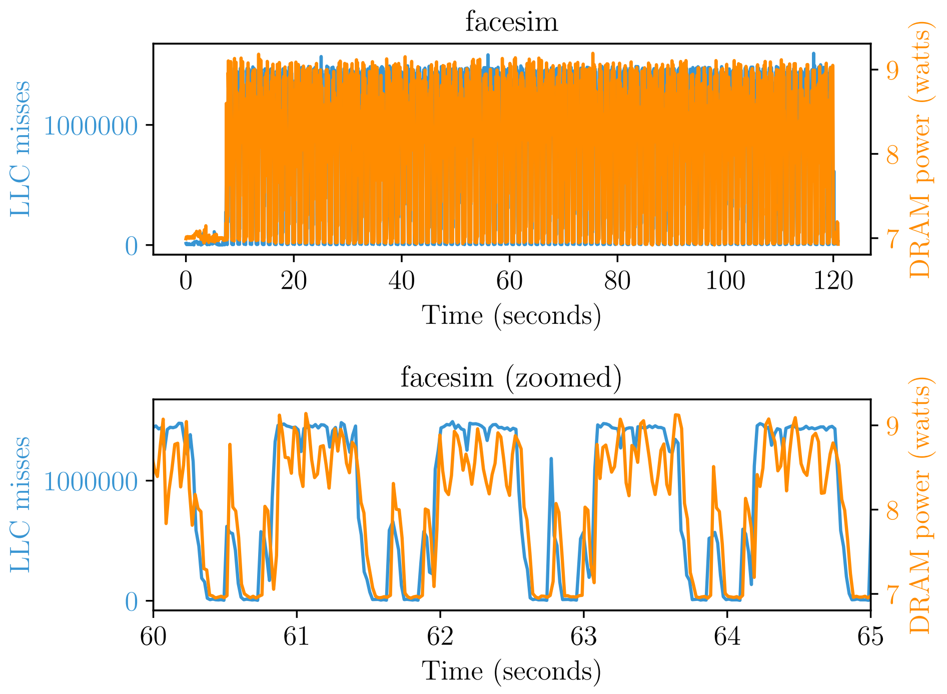


Figure 6.2: Cache misses and memory power use of *facesim* over time

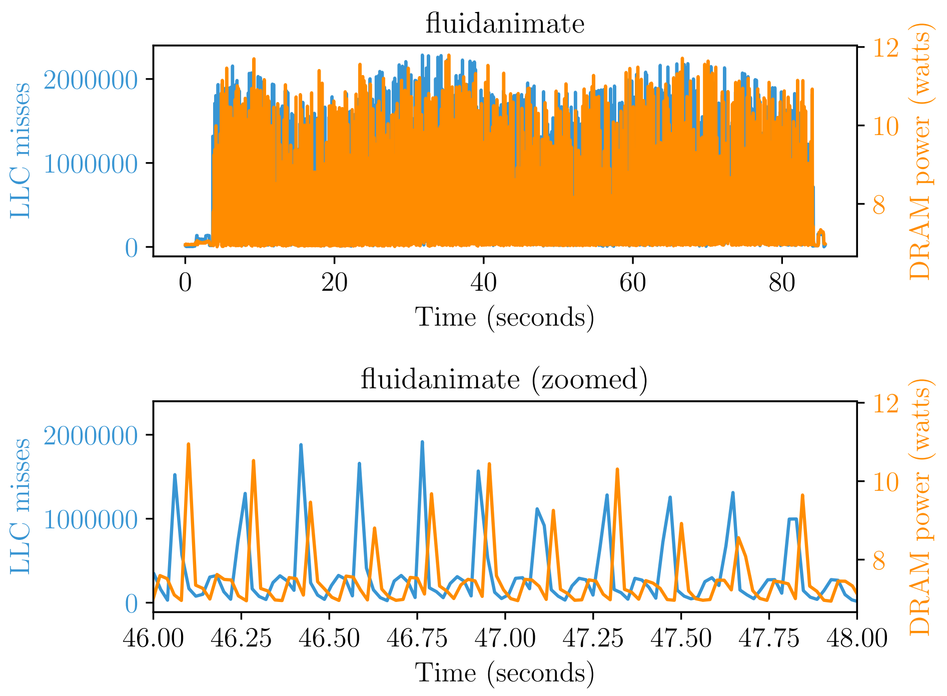


Figure 6.3: Cache misses and memory power use of *fluidanimate* over time

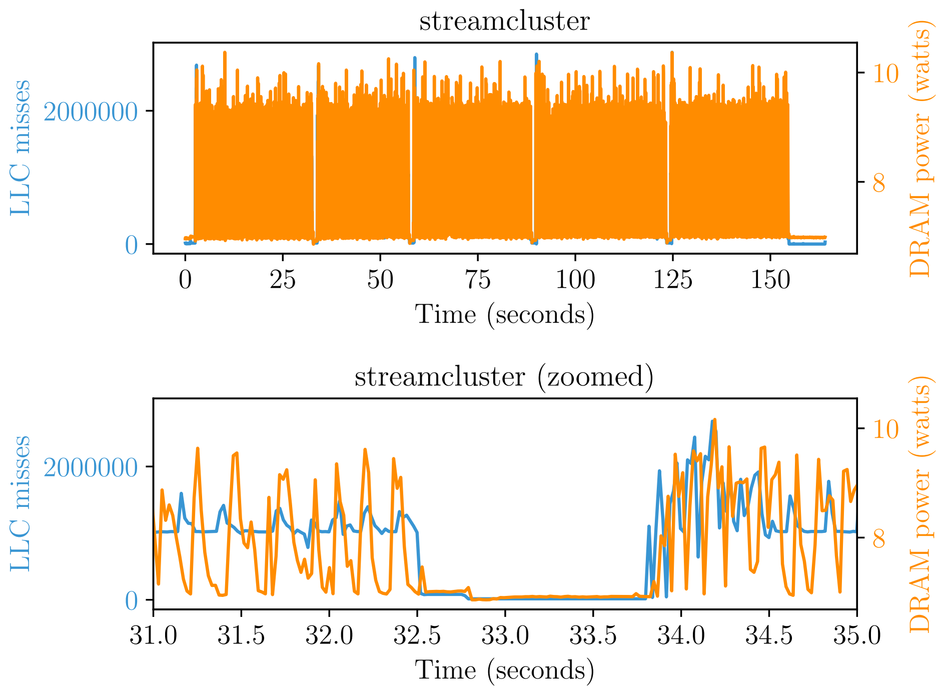


Figure 6.4: Cache misses and memory power use of *streamcluster* over time

possibly other aspects of DRAM which cause power consumption to increase later. Systems with very large amounts of physical memory might exhibit even larger delays between accesses and power use.

6.5 Power and Energy Models

As we have seen from time series measurements, the number of last-level cache misses seems to correlate well with memory power use. We would therefore like to create a model based on recorded LLC misses to predict the memory power consumption and energy use of an application.

We create the model in a similar way as the CPU power model in Section 5.5. We use all benchmarks, and base the model on aggregate measurements from a whole benchmark run, rather than on individual timepoints. We use ordinary least squares multiple linear regression, with a 6–3 random split of benchmarks for the train and test set to evaluate the model. The very small dataset limits the reliability of the model as well as the reported MSE and R^2 scores. (Using more benchmarks or individual timepoints as further datapoints would remove this limitation.)

We create separate models to predict average memory power use and energy. The models are tuned for the Enzian CPU.

6.5.1 Average Power Model

We first look at predicting the average memory power consumption of an application based on the number of times it misses in the LLC. As power refers to energy used per second, we need a similar measure for cache misses. The obvious choice is LLC misses per second. To calculate average LLC misses per second, we divide total LLC misses with total elapsed seconds during the application run (derived from total cycles and CPU frequency).

Figure 6.5 shows the relationship between average LLC misses per second and average memory power use obtained from our benchmarks. The line represents the fitted linear model, which is

$$P_{DRAM} = N_{LLCMpS} \times 9.89 \times 10^{-7} + 7.06 \ (W)$$

where N_{LLCMpS} is the average number of LLC misses per second.

This model implies that the static power used by the memory is 7.06 watts, while an additional LLC miss every second would use an additional 989 nanowatts of power.

This model obtains an MSE of 0.01 and an R^2 score of 0.92, indicating high accuracy (but again limited by the small dataset size).

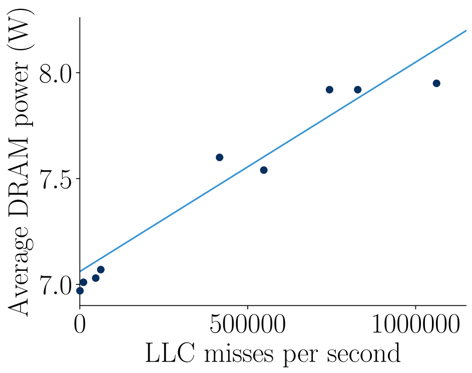


Figure 6.5: Model of average memory power based on average LLC misses per second

6.5.2 Energy Model

In addition to average power use of memory, it would be useful to know how much energy is consumed by the memory during a whole application run.

The memory consumes static energy throughout the run of a program, so application runtime needs to be accounted for in the memory energy model. This can be achieved by counting the number of cycles elapsed.

Figure 6.6a shows the relationship between total energy consumed by memory and total cycles elapsed, based on our benchmarks. This model obtains an MSE of 149.43 and an R^2 score of 0.99, indicating very good accuracy. Based on this it seems that static power dominates memory energy use. However, this could be due to our choice of benchmarks. It is possible that the PARSEC benchmarks do not stress memory enough, leading to little energy consumed by memory accesses. One way to test this hypothesis would be to use synthetic benchmarks with many memory accesses.

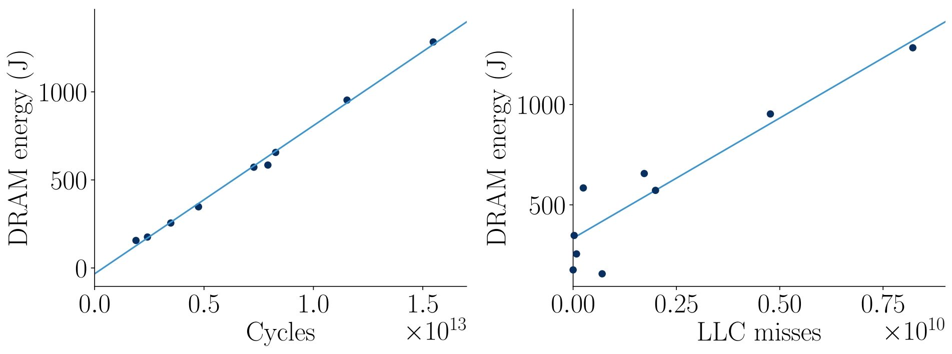
For completeness, Figure 6.6b shows the relationship between the number of LLC misses and energy use. As it does not take runtime into account, it is not a very good model (MSE = 8351.80, R^2 score = 0.68).

We can also create a model based on both cycles an LLC misses, which gives us

$$E_{DRAM} = N_{cycles} \times 7.40 \times 10^{-11} + N_{LLCM} \times 1.76 \times 10^{-8} + 2.59 \ (J)$$

where N_{cycles} is the total number of elapsed cycles and N_{LLCM} is the total number of LLC misses during an application run.

This model performs better than the model based on cycles alone, obtaining an MSE of 35.50 and an R^2 score of 1.00. This indicates that LLC misses



(a) Cycles

(b) LLC misses

Figure 6.6: Memory energy model based on the number of (a) cycles or (b) LLC misses

do have a significant effect on memory energy consumption. According to the model, each cycle consumes 0.074 nanojoules of energy, while each LLC miss consumes 17.6 nanojoules. The intercept (2.59) is near 0, as expected, as running for 0 cycles should not consume any energy.

While the dataset size is small, these findings do nevertheless suggest that counting cache misses is a reliable way to predict memory power and energy consumption of an application. The obtained model could be used to predict the energy used by the memory on Enzian by any application. When combined with other system components, such as CPU and disk power models, the total energy cost of an application can be calculated. On a system like Enzian, a similar energy model could be made for the FPGA and FPGA memory. Knowledge of the energy cost can be used for scheduling decisions, such as which system or which parts of a system (such as CPU or FPGA) to run an application on, or to identify high energy parts of an application that could be optimized or offloaded to accelerators.

Chapter 7

FPGA Power Model

In the previous chapters we created power models for the Enzian CPU and memory. In addition to these, the Enzian system also contains an FPGA (Chapter 4). It is known that the power consumed by an FPGA depends on the properties of the FPGA design that it implements (Chapter 2). In this chapter we explore how basic properties of an FPGA design affect its power use.

Specifically, we are interested in two properties:

- **Clock frequency**, which determines how many times per second an FPGA changes state. Using a higher clock frequency results in higher performance.
- **Utilization**, which measures what proportion of FPGA resources (e.g. CLBs or registers) are used by a given FPGA design.

We first describe the benchmark we implemented for our study (Section 7.1). We also describe how we performed the power measurements (Section 7.2). We then present our findings on how utilization and clock frequency affect power use, as well as a simple power model based on them (Section 7.3). As part of this we also examine how closely FPGA vendor software is able to predict the power use of a design (Section 7.3.4).

7.1 Benchmark

We implemented a single benchmark for our experiments. The goal was to vary clock frequency and utilization (independently) while keeping other factors constant.

We were interested in the utilization of the two most basic resources of an FPGA: LUTs and registers. To achieve high utilization, the FPGA was filled with pairs of D flip-flops (using registers) and inverters (using LUTs). The flip-flops were made to invert their value on each clock edge, consuming power. The FPGA was then divided into 16 equally sized regions, each of which could be enabled or disabled independently (using the Virtual Input/Output core in Vivado [31]). In Section 7.3 we will see how the number of enabled regions affects power use.

Table 7.1 shows the utilization of the whole design as well as a single region (*Baseline* is explained later). As the design was not precisely calibrated to use every single resource of the FPGA, the total utilization is less than 100%. In addition, fewer flip-flops than LUTs were used. This is because a CLB on the chosen FPGA contains 8 LUTs and 16 flip-flops. Due to the way the design was made, only 8 of the 16 flip-flops in any CLB were used. Nevertheless, the benchmark covers a significant part of the FPGA and enables the comparison of relative power changes.

	LUT count	LUT utilization	FF count	FF utilization
Baseline	97,000	8%	128,000	5%
Benchmark	835,000	71%	737,000	32%
Total	**932,000**	**79%**	**865,000**	**37%**

(a) Whole benchmark

	LUT count	LUT utilization	FF count	FF utilization
Region	52,000	4.5%	46,000	2%

(b) Single region

Table 7.1: Number of FPGA LUTs and flip-flops used by the benchmark. Utilization shows the proportion used out of all FPGA resources of that type. Precise counts varied based on clock frequency (by less than 1%), so approximate counts are given. Numbers were obtained from Vivado based on a fully implemented design.

Each of the 16 regions was driven by a clock signal. Initially a single clock was used, however this caused the power supply unit (PSU) of the board to reset, possibly due to too much power being drawn at a single instant. Instead, 4 out-of-phase clock signals (with the same frequency) were used, each driving 4 regions. The clock frequency could be chosen when synthesizing the design. In section 7.3 we will see how the clock frequency affects power use.

The benchmark was implemented on top of a baseline design for the Enzian FPGA. This consisted of a Xilinx MicroBlaze core and 4 memory controllers. None of these were used as part of the benchmark. Table 7.1a shows what proportion of the FPGA resources were taken up by the baseline implementation.

The benchmark was implemented in VHDL and synthesized with the Vivado software. The benchmark was implemented by Dr. Michael Giardino as part of this project.

7.2 Power Measurements

The FPGA is powered by a number of voltage rails. On Enzian, several of the rails have voltage and current monitors, as described in Section 4.2. We measured the power on these rails. This included the FPGA internal power (V_{CCINT} in Xilinx documentation [30]), as well as power for BRAM, I/O, transceivers and other components.

While using the benchmark, we observed the voltage to be almost constant, so only current measurements were taken. Most of the current was drawn by the internal power rail. The current on the other rails remained almost constant and totaled less than 10 A.

7.3 Effects of Utilization and Clock Frequency

We would like to see how changes in utilization or clock frequency affect the power consumption of an FPGA. Note that we are interested in the relative power change. We are not aiming to estimate absolute power consumption, as that would require knowledge of further properties of the FPGA (such as the capacitance of various components). Here we measure the base power use and study how utilization and clock frequency affect it.

7.3.1 Utilization

Figure 7.1 shows the results of our measurements. As we can see, power use increases linearly with utilization. That is, using more FPGA resources (LUTs and flip-flops) results in higher dynamic power use, and the increase is proportional to the number of resources used. For example, with a clock frequency of 150 Mhz, the power use of our design could be estimated as

$$P = 0.26U + 24.82\ (W)$$

where U is the utilization percentage of the FPGA (as defined in our benchmark as a certain proportion of LUTs and flip-flops active on every clock edge). Therefore, a 1% increase in utilization increases power use by 0.26 W.

7.3.2 Clock Frequency

We ran our benchmark at four clock frequencies. As we can see from Figure 7.1, a higher clock frequency results in higher power use. Dynamic power increases roughly linearly with clock frequency, which matches what

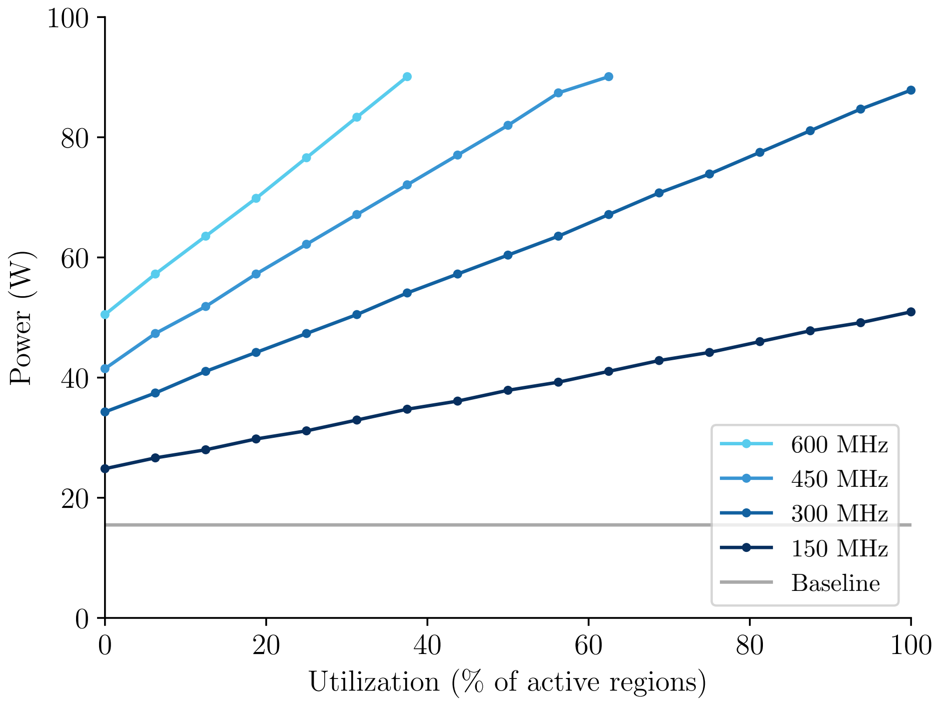


Figure 7.1: Effects of utilization and clock frequency on total FPGA power use. The size of a region is described in Table 7.1b. The baseline implementation is described in Section 7.1.

we would expect based on Equation 3.2. For example, at 25% utilization, the power use of our design could be estimated as

$$P = 0.1f + 16.52\ (W)$$

where f is the clock frequency in MHz. This means that changing the state of flip-flops a million more times per second uses an additional 0.1 W of power.

Note that static power can be seen as part of the baseline implementation power. Also note that even with no regions enabled, the power use is higher at a higher frequency. This is likely because the clocks are still running, even if they are not driving the logic in a region.

The maximum current limit on the internal power rail of the FPGA limited how much current could be drawn, which is why not all regions could be activated at 450 MHz and 600 MHz frequencies. As we can observe, it is often not possible to use all resources of an FPGA due to power constraints. In the case of our design for example (where we change state on every clock edge), with a 450 MHz clock frequency we were only able to use about

62.5% of the resources (approximately 45% of LUTs and 20% of flip-flops), and only 37.5% with a 600 MHz clock frequency (27% LUTs and 12% flip-flops).

7.3.3 Combined Power Model

As we have seen, FPGA power consumption depends on both resource utilization and clock frequency. We can therefore use both properties to estimate the power use, if we know how each of them affect power on a given FPGA and design, as well as the static power. In the case of our design, we could use the following model:

$$P = 0.002Uf - 0.012U + 0.055f + 16.931\ (W)$$

where f is the clock frequency in MHz and U is the utilization percentage. As in the previous sections, utilization is defined as the proportion of active regions, where 1% utilization corresponds to 0.71% LUTs and 0.32% flip-flops (changing state on every clock edge) due to the design of our benchmark (Table 7.1).

The above model obtained an MSE of 0.13 and an R^2 score of 1 (using 80% of the measurements as a training set and 20% as a test set), suggesting very high accuracy.

Having calibrated the model to the hardware and design, it can then be used to estimate the power consumption for any choice of utilization or clock frequency.

7.3.4 Comparison to Vivado Estimates

In this section we look at how closely an FPGA synthesis tool estimates the power consumption of a design. Specifically, we take a look at the Vivado Design Suite version 2020.1 from Xilinx.

Vivado can provide a power estimate based on either a synthesized or a fully implemented design. In order to make the estimate more accurate, the user can set various parameters to describe the operation of the design. These include properties of the FPGA (e.g. precise voltages, heat sink size) and the environment (e.g. ambient temperature, airflow), activity of the components (toggle rate and static probability) or a simulation activity file.

We were not able to set all of these parameters due to lack of information, which limits the accuracy of the estimates. We modified the toggle rate of the LUTs and registers, based on knowledge of our benchmark. Toggle rate refers to how frequently the component changes state, for example a toggle rate of 100% means that on average it changes once per clock edge, while

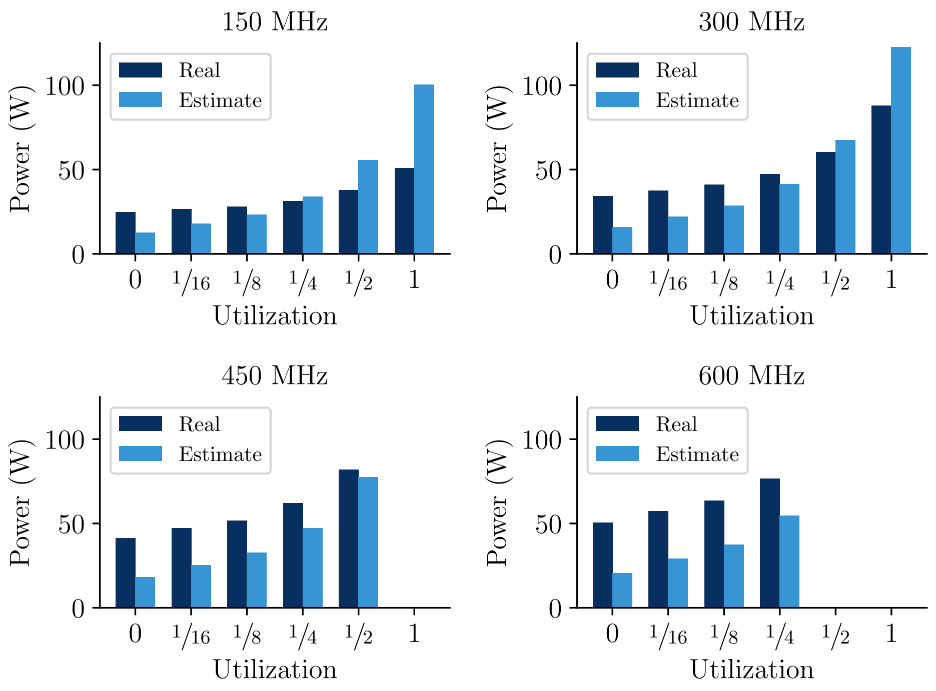


Figure 7.2: Comparison of Vivado power estimates with real power use

25% means that on average it changes once per 4 clock edges [32]. We set the toggle rate based on how many regions were enabled. For example, if 8 regions (out of 16) were enabled, we set the rate to a little under 50% (taking into account that the baseline implementation was also not active).

Figure 7.2 shows how Vivado power estimates compare to real power measurements. Estimates were taken based on a fully implemented design.

We see that Vivado underestimates the power use at low utilization but overestimates it at high utilization. In addition, the error is greater at larger clock frequencies. In the worst cases it underestimates by 59% (29.8 W), and overestimates by 97% (49.5 W). It is not clear why this is, but it suggests that some parameter in the design is not being taken into account or we have not specified it properly.

Previous work has found that Vivado consistently overestimates power use (Chapter 2), but here we find that it can also underestimate. In addition we see that the error can be quite large. This suggests that real power measurements should always be preferred, and relying on tool estimates may lead to wrong decisions (e.g. at early algorithm design stages).

Chapter 8

Future Work

This chapter discusses possible future research directions in power modeling.

8.1 CPU and Memory Power Model Improvements

There are several possible ways in which the developed CPU and memory power models could be improved.

As mentioned in Chapters 5 and 6, the small dataset limits the reliability of the models. This could be improved by evaluating the models on further benchmarks, such as the SPLASH-2 HPC benchmarks or memory-intensive benchmarks like STREAM. The datapoints of the time series measurements could also be used.

Additional types of performance events could be evaluated for correlations with power use. These could include the memory access events listed in Table 6.1 or events used in related work (Chapter 2) such as stall cycles.

The effect of other parameters could also be investigated, such as CPU temperature, and the model improved with them.

8.2 FPGA Power Model Improvements

As we only studied two basic properties of an FPGA design, there are many others to study, such as switching activity. In addition, further benchmarks could be written to stress specific components of the FPGA, such as BRAM or I/O, especially since Enzian has separate power monitors on each FPGA power rail, and the rails power different components of the FPGA, so the power use of each component could be measured separately.

An ambitious goal would be to explore ways to estimate the power consumption of an FPGA design based on CPU performance counter recordings of the same application.

8.3 Models of Other System Components

As a step towards a full system model, power models of other system components could be developed. This includes common components such as the disk or network interface. More interestingly, the power consumption of Enzian-specific components could be investigated. The DRAM memory of the FPGA could be benchmarked, and relationships found between the bandwidth used and the power consumed. The CPU-FPGA coherent interconnect could be exercised by passing large amounts of data through it, possibly through FPGA filters and DRAM, and the power consumption measured.

8.4 Composite Power Model

The power models developed in this thesis could be combined into a composite power model. This model could then be evaluated with workloads that exercise the whole Enzian system. At present no such workloads are available, but in the future they could, for example, include database or streaming applications that offload parts of their computation to the FPGA.

Chapter 9

Conclusion

In this thesis we have studied the power consumption of the CPU, memory and FPGA of Enzian and developed power models for them.

We performed and examined time series measurements of power consumption and hardware performance counters for a range of benchmarks, which provided insights into the correlations between performance events and power. We found IPC to correlate well with CPU power consumption and LLC misses to correlate well with memory power consumption. We developed analytical power and energy models for the ThunderX CPU based on IPC and for the DDR4 memory based on LLC misses. While simple, the models proved to be highly accurate for estimating the power and energy consumption of an application.

We studied how the utilization of LUTs and flip-flops and the clock frequency of an FPGA design affect its power use. We used this to develop an accurate analytical power model for very simple FPGA designs. We also examined the accuracy of Xilinx Vivado power estimates.

We have performed the first study on the power consumption of Enzian. As part of this, we identified limitations in the sampling frequency of the power measurement devices, which can be improved in future revisions of Enzian.

This study demonstrates the utility of the per-component power measurement devices available on Enzian. This motivates future research on power consumption on Enzian, which will be able to make use of the devices for detailed analyses on how the power consumption of each component is affected by different workloads or different divisions of work between the CPU and FPGA.

Having models of system components is a step towards creating a whole system power model for Enzian. From there, similar models can be created for other heterogeneous systems. Having an accurate power model will enable better work allocation policies and power savings in CPU-FPGA systems in data centers and elsewhere.

References

[1] Nabil Abdelli et al. “High-level power estimation of fpga”. In: *2007 IEEE International Symposium on Industrial Electronics*. IEEE. 2007, pp. 925–930.

[2] Gustavo Alonso et al. “Tackling Hardware/Software co-design from a database perspective.” In: *CIDR*. 2020.

[3] Jason Helge Anderson and Farid N Najm. “Power estimation techniques for FPGAs”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 12.10 (2004), pp. 1015–1027.

[4] Luiz André Barroso and Urs Hölzle. “The datacenter as a computer: An introduction to the design of warehouse-scale machines”. In: *Synthesis lectures on computer architecture* 4.1 (2009), pp. 1–108.

[5] Frank Bellosa. “The benefits of event: driven energy accounting in power-sensitive systems”. In: *Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system*. 2000, pp. 37–42.

[6] Ramon Bertran et al. “Decomposable and responsive power models for multicore processors using performance counters”. In: *Proceedings of the 24th ACM International Conference on Supercomputing*. 2010, pp. 147–158.

[7] Christian Bienia. “Benchmarking Modern Multiprocessors”. PhD thesis. Princeton University, Jan. 2011.

[8] William Lloyd Bircher and Lizy K John. “Complete system power estimation using processor performance events”. In: *IEEE Transactions on Computers* 61.4 (2011), pp. 563–577.

[9] William Lloyd Bircher et al. “Runtime identification of microprocessor energy saving opportunities”. In: *Proceedings of the 2005 international symposium on Low power electronics and design*. 2005, pp. 275–280.

[10] Andrew Canis et al. “LegUp: high-level synthesis for FPGA-based processor/accelerator systems”. In: *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*. 2011, pp. 33–36.

[11] Deming Chen, Jason Cong, and Peichan Pan. *FPGA design automation: A survey*. Vol. 2. Now Publishers Inc, 2006.

[12] Deming Chen et al. “High-level power estimation and low-power design space exploration for FPGAs”. In: *2007 Asia and South Pacific Design Automation Conference*. IEEE. 2007, pp. 529–534.

[13] Young-Kyu Choi et al. “In-depth analysis on microarchitectures of modern heterogeneous CPU-FPGA platforms”. In: *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 12.1 (2019), pp. 1–20.

[14] Young-kyu Choi et al. “A quantitative analysis on microarchitectures of modern CPU-FPGA platforms”. In: *Proceedings of the 53rd Annual Design Automation Conference*. 2016, pp. 1–6.

[15] Maxime Colmant et al. “The next 700 CPU power models”. In: *Journal of Systems and Software* 144 (2018), pp. 382–396.

[16] Gilberto Contreras and Margaret Martonosi. “Power prediction for Intel XScale/spl reg/processors using performance monitoring unit events”. In: *ISLPED’05. Proceedings of the 2005 International Symposium on Low Power Electronics and Design, 2005.* IEEE. 2005, pp. 221–226.

[17] Intel Corporation. *Early Power Estimator for Intel Stratix 10 FPGAs User Guide (UG-20069)*. July 25, 2019.

[18] Intel Corporation. *Intel 64 and IA32 Architectures Performance Monitoring Events (335279-001)*. December, 2017.

[19] Intel Corporation. *Intel Quartus Prime Standard Edition User Guide: Power Analysis and Optimization (UG-20184)*. December 7, 2020.

[20] Howard David et al. “Memory power management via dynamic voltage/frequency scaling”. In: *Proceedings of the 8th ACM international conference on Autonomic computing*. 2011, pp. 31–40.

[21] Howard David et al. “RAPL: Memory power estimation and capping”. In: *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*. IEEE. 2010, pp. 189–194.

[22] Miyuru Dayarathna, Yonggang Wen, and Rui Fan. “Data center energy consumption modeling: A survey”. In: *IEEE Communications Surveys & Tutorials* 18.1 (2015), pp. 732–794.

[23] Vijay Degalahal and Tim Tuan. “Methodology for high level estimation of FPGA power consumption”. In: *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*. 2005, pp. 657–660.

[24] Dimitris Economou et al. “Full-system power analysis and modeling for server environments”. In: International Symposium on Computer Architecture (IEEE). 2006.

[25] *Enzian*. URL: http://enzian.systems/ (visited on 03/21/2021).

[26] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. “Power provisioning for a warehouse-sized computer”. In: *ACM SIGARCH computer architecture news* 35.2 (2007), pp. 13–23.

[27] Heiner Giefers, Raphael Polig, and Christoph Hagleitner. “Accelerating arithmetic kernels with coherent attached FPGA coprocessors”. In:

2015 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE. 2015, pp. 1072–1077.

[28] Jeffrey B Goeders and Steven JE Wilton. "VersaPower: Power estimation for diverse FPGA architectures". In: *2012 International Conference on Field-Programmable Technology*. IEEE. 2012, pp. 229–234.

[29] David Harris and Sarah Harris. *Digital design and computer architecture*. Morgan Kaufmann, 2010.

[30] Xilinx Inc. *Virtex UltraScale+ FPGA Data Sheet: DC and AC Switching Characteristics*. December 8, 2020.

[31] Xilinx Inc. *Virtual Input/Output v3.0 LogiCORE IP Product Guide*. April 4, 2018.

[32] Xilinx Inc. *Vivado Design Suite User Guide: Power Analysis and Optimization (UG907 v2020.2)*. November 24, 2020.

[33] Xilinx Inc. *Xilinx Power Estimator User Guide (UG440 v2020.2)*. December 4, 2020.

[34] Canturk Isci and Margaret Martonosi. "Runtime power monitoring in high-end processors: Methodology and empirical data". In: *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36*. IEEE. 2003, pp. 93–104.

[35] Ruzica Jevtic and Carlos Carreras. "Power measurement methodology for FPGA devices". In: *IEEE Transactions on Instrumentation and Measurement* 60.1 (2010), pp. 237–247.

[36] Tianyi Jiang, Xiaoyong Tang, and Prith Banerjee. "Macro-models for high-level area and power estimation on fpgas". In: *International Journal of Simulation and Process Modelling* 2.1-2 (2006), pp. 12–19.

[37] Ismail Kadayif et al. "vEC: virtual energy counters". In: *Proceedings of the 2001 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*. 2001, pp. 28–31.

[38] Dario Korolija, Timothy Roscoe, and Gustavo Alonso. "Do {OS} abstractions make sense on FPGAs?" In: *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*. 2020, pp. 991–1010.

[39] Sheng Li et al. "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures". In: *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. 2009, pp. 469–480.

[40] Arm Limited. *Arm Architecture Reference Manual. Armv8, for Armv8-A architecture profile. DDI 0487F.c.* 2020.

[41] Arm Limited. *Arm Cortex-A76 Core Revision r3p0 Technical Reference Manual*. 2018.

[42] Weiwei Lin et al. "A Taxonomy and Survey of Power Models and Power Modeling for Cloud Servers". In: *ACM Computing Surveys (CSUR)* 53.5 (2020), pp. 1–41.

[43] Dimitrios Meidanis, Konstantinos Georgopoulos, and Ioannis Papaefstathiou. “FPGA power consumption measurements and estimations under different implementation parameters”. In: *2011 International Conference on Field-Programmable Technology*. IEEE. 2011, pp. 1–6.

[44] *NTP FAQ. How does it work? 5.1.3.1. How accurate will my Clock be?* URL: http://www.ntp.org/ntpfaq/NTP-s-algo.htm#Q-ACCURATE-CLOCK (visited on 03/14/2021).

[45] Kenneth O’Neal and Philip Brisk. “Predictive modeling for cpu, gpu, and fpga performance and power consumption: A survey”. In: *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE. 2018, pp. 763–768.

[46] Kenneth O’brien et al. “A survey of power and energy predictive models in HPC systems and applications”. In: *ACM Computing Surveys (CSUR)* 50.3 (2017), pp. 1–38.

[47] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[48] *perf: Linux profiling with performance counters.* URL: https://perf.wiki.kernel.org (visited on 10/16/2020).

[49] Kara KW Poon, Steven JE Wilton, and Andy Yan. “A detailed power model for field-programmable gate arrays”. In: *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 10.2 (2005), pp. 279–302.

[50] Miloš Puzović et al. “Quantifying energy use in dense shared memory HPC node”. In: *2016 4th International Workshop on Energy Efficient Supercomputing (E2SC)*. IEEE. 2016, pp. 16–23.

[51] Freeman Rawson and I Austin. “Mempower: A simple memory power analysis tool set”. In: *IBM Austin Research Laboratory* 3 (2004).

[52] Paul Rosenfeld, Elliott Cooper-Balis, and Bruce Jacob. “DRAMSim2: A cycle accurate memory system simulator”. In: *IEEE computer architecture letters* 10.1 (2011), pp. 16–19.

[53] Efraim Rotem et al. “Power-management architecture of the intel microarchitecture code-named sandy bridge”. In: *Ieee micro* 32.2 (2012), pp. 20–27.

[54] *SPEC Benchmarks.* URL: https://www.spec.org/benchmarks.html (visited on 03/09/2021).

[55] *The Arm Research Starter Kit: System Modeling using gem5.* URL: https://github.com/arm-university/arm-gem5-rsk (visited on 03/08/2021).

[56] *The PARSEC Benchmark Suite: Overview.* URL: https://parsec.cs.princeton.edu/overview.htm (visited on 03/09/2021).

[57] Shyamkumar Thoziyoor et al. “A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies”. In: *ACM SIGARCH Computer Architecture News* 36.3 (2008), pp. 51–62.

[58] *ThunderX CN8890 - Cavium - WikiChip*. URL: `https://en.wikichip.org/wiki/cavium/thunderx/cn8890` (visited on 03/17/2021).

[59] David Wang et al. “Dramsim: a memory system simulator”. In: *ACM SIGARCH Computer Architecture News* 33.4 (2005), pp. 100–107.

Appendix A

CPU Time Series

This appendix lists the remaining 5 plots not shown in Section 5.4. These plots show the time series measurements of IPC and CPU power use for 5 PARSEC benchmarks.

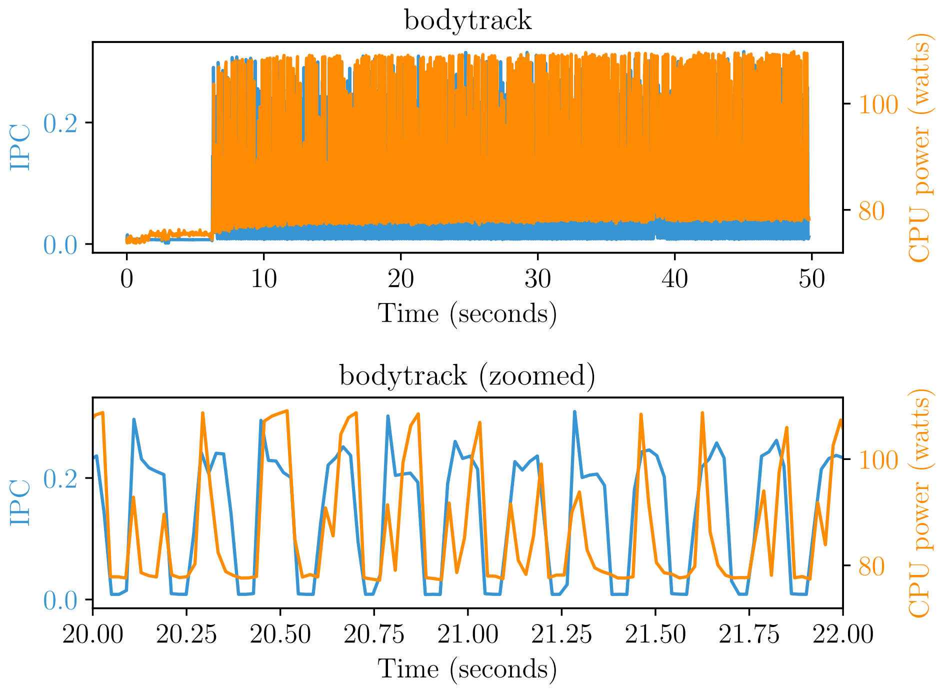


Figure A.1: IPC and CPU power use of *bodytrack* over time

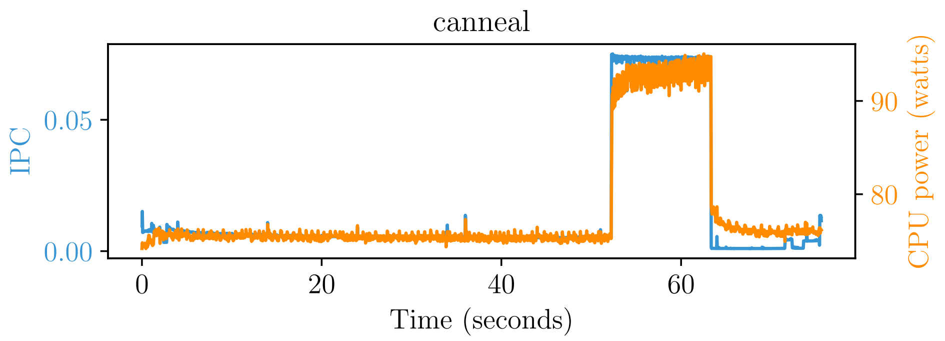


Figure A.2: IPC and CPU power use of *canneal* over time

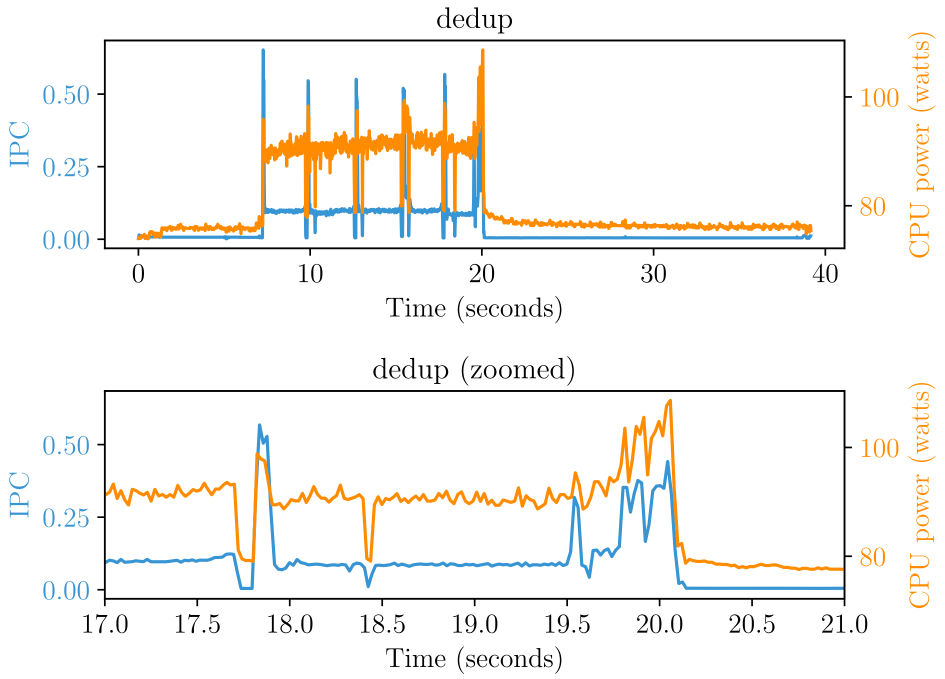


Figure A.3: IPC and CPU power use of *dedup* over time

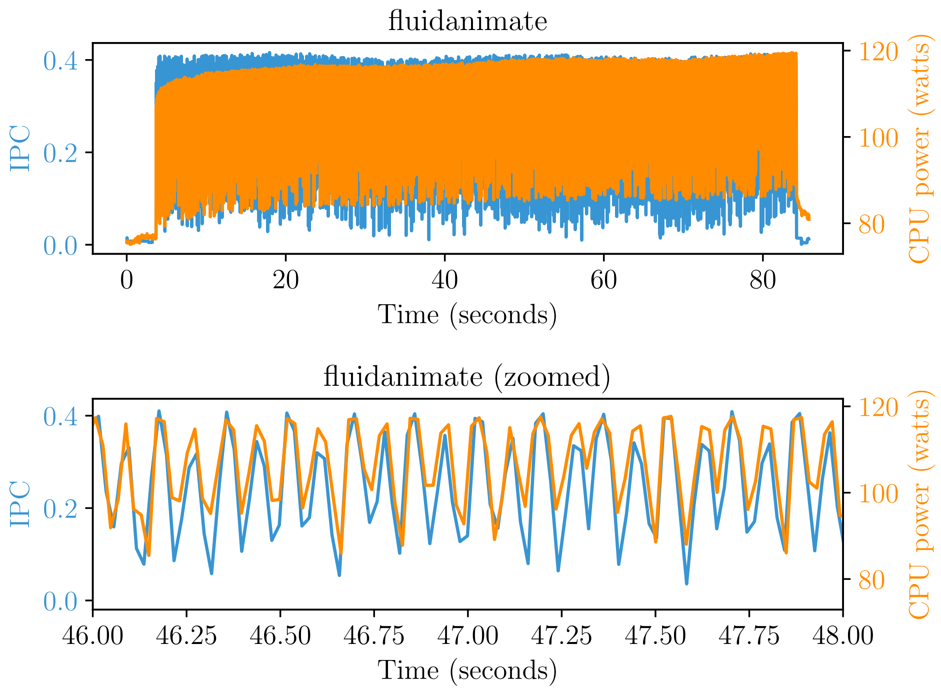


Figure A.4: IPC and CPU power use of *fluidanimate* over time

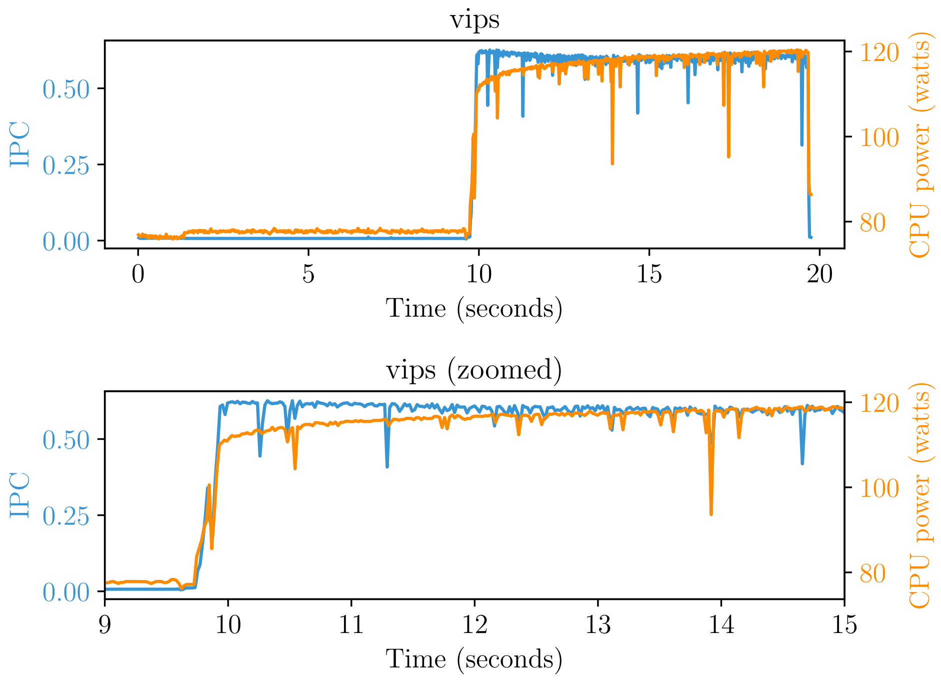
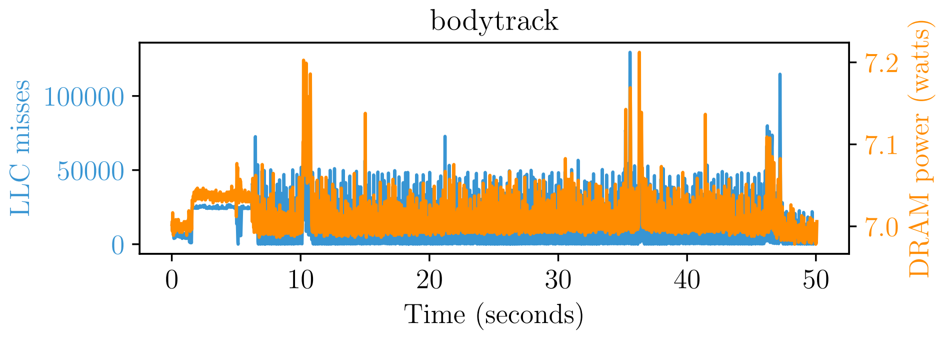


Figure A.5: IPC and CPU power use of *vips* over time

Appendix B

Memory Time Series

This appendix lists the remaining 5 plots not shown in Section 6.4. These plots show the time series measurements of last-level cache misses and DRAM power use for 5 PARSEC benchmarks.



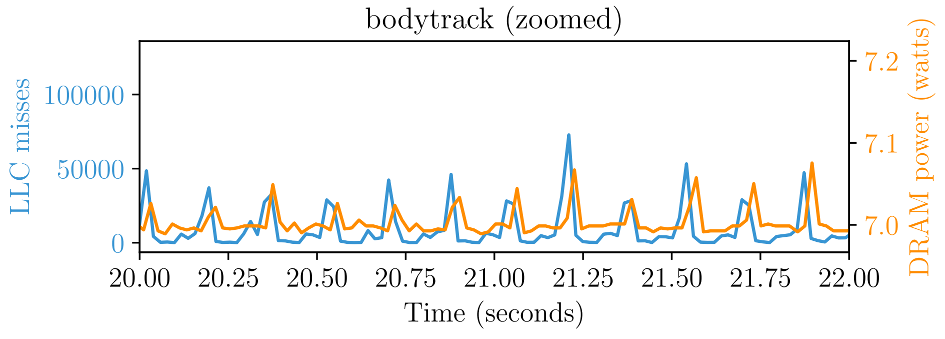


Figure B.1: Cache misses and memory power use of *bodytrack* over time

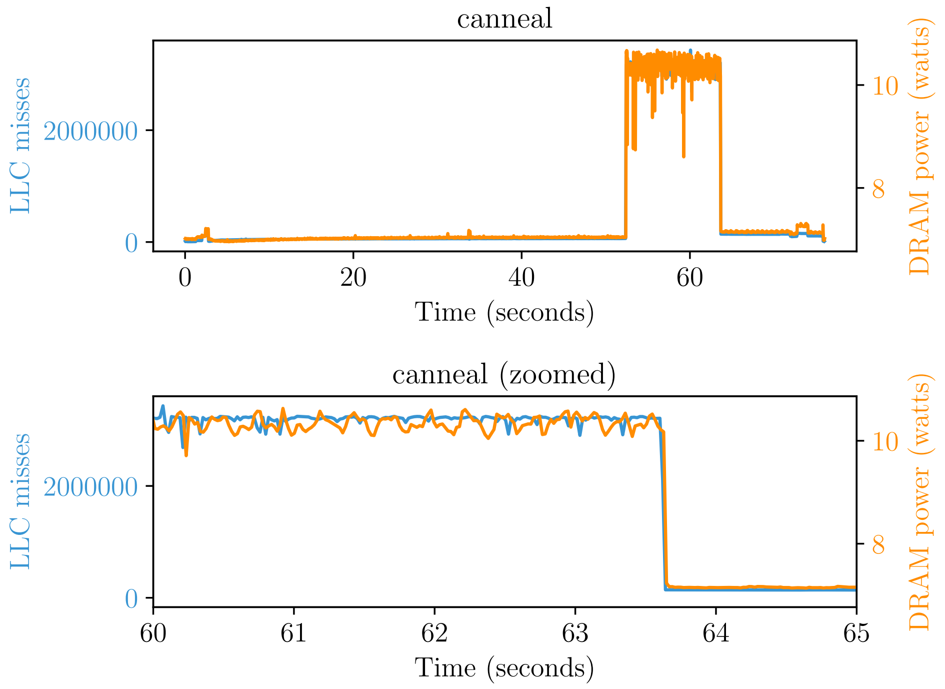


Figure B.2: Cache misses and memory power use of *canneal* over time

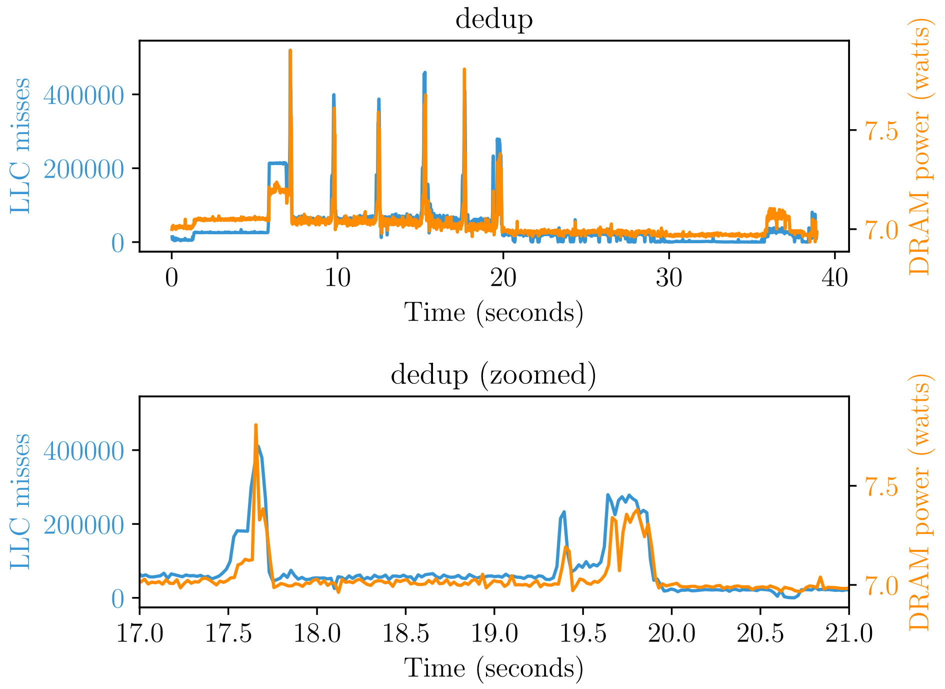
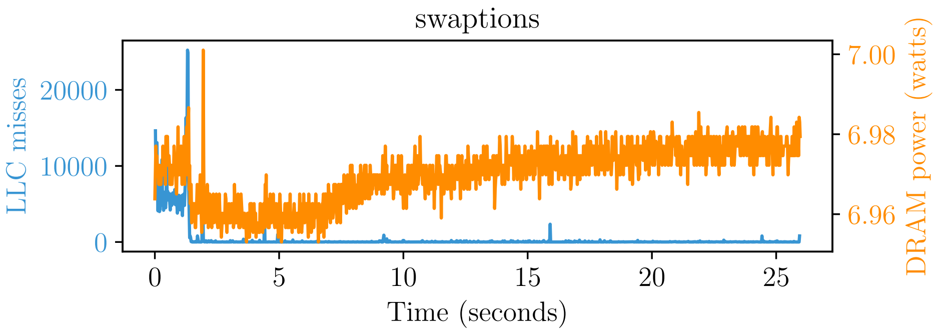


Figure B.3: Cache misses and memory power use of *dedup* over time



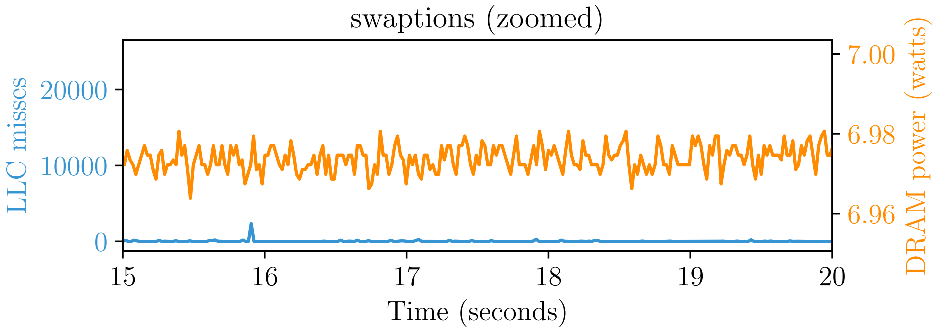
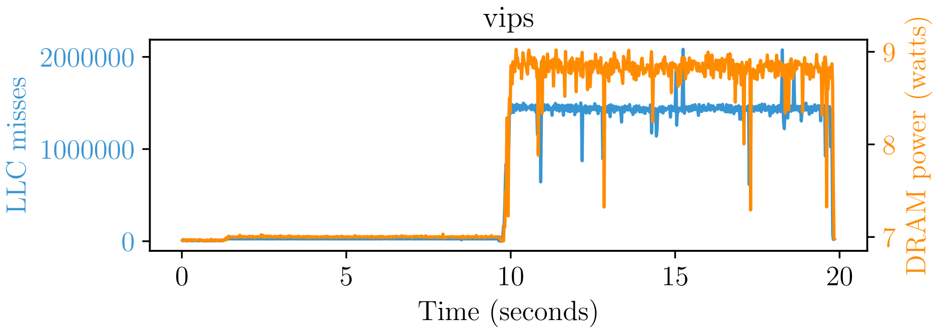


Figure B.4: Cache misses and memory power use of *swaptions* over time



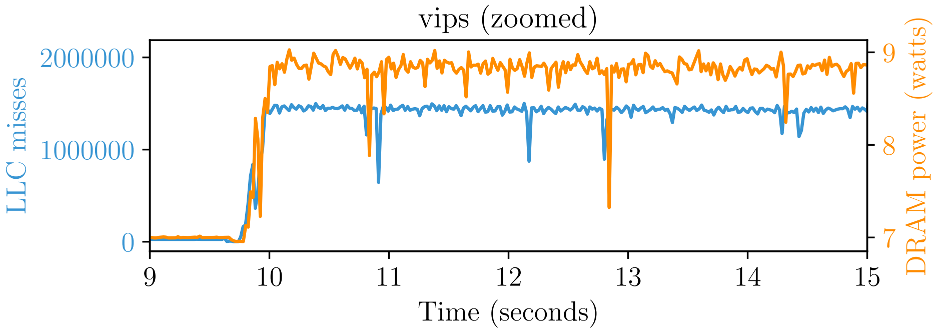


Figure B.5: Cache misses and memory power use of *vips* over time

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Developing and Evaluating Power Models of Heterogeneous Computer Systems

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Martšenko	Kristina

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Dübendorf, 22.03.2021	[illegible]

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.