CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Energy-Performance Balancing Task Scheduler for Asymmetric Platforms

Master's thesis in Computer Science and Engineering

Henrik Andersson

Carl Wiede

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

Energy-Performance Balancing Task Scheduler for Asymmetric Platforms

Henrik Andersson
Carl Wiede

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

Energy-Performance Balancing Task Scheduler for Asymmetric Platforms

Henrik Andersson
Carl Wiede

Supervisors:
Jing Chen, Department of Computer Science and Engineering
Bhavishya Goel, Department of Computer Science and Engineering

Examiner:
Miquel Pericàs, Department of Computer Science and Engineering

Master's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2023

Energy-Performance Balancing Task Scheduler for Asymmetric Platforms

Henrik Andersson
Carl Wiede
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Sustainability is a growing concern for society and computer science is no exception. Power consumption by computers may be reduced by lowering the frequency of the processor as well as meticulously limiting the hardware resource usage. An optimally energy efficient computation may, however, cause an impractically long execution time.

Previous work has successfully provided a framework that minimizes energy consumption using task-based computation. One way to develop the framework concerns efforts to strike a balance between performance and energy efficiency to find the optimal trade-off between increased execution time and reduced energy cost. An option to utilize such a trade-off could incentivize a greater adoption of aforementioned energy reduction techniques.

This thesis presents various efforts to modify an existing energy efficient task scheduling framework in order to balance energy efficiency and performance. The framework was further generalized and tested on multiple platforms for the sake of affirming its generic applicability. The Simics hardware simulator was assessed in hopes of enabling the possibility to test the framework on a myriad of virtual platforms.

The evaluation shows that the modified framework can successfully determine the task scheduling decisions that yield optimal trade-off between performance and energy efficiency. After some additional modifications, the framework could seamlessly run on other platforms than the one it was designed for. Although the attempts to use the framework within the select virtual environment were somewhat futile, promising directions for future research were discovered.

Keywords: Computer science, sustainable computing, task scheduling, task-parallel processing, asymmetric platforms, thesis, project.

Acknowledgements

Firstly, we would like to thank our supervisors, Jing Chen and Bhavishya Goel, for their continuous support during the course of the thesis. Their regular advice and feedback proved to be crucial in order to successfully complete the project. Secondly, we would like to express our gratitude to our examiner, Miquel Pericàs, for great intermittent assistance when we needed guidance regarding the direction of the thesis. Thirdly, we'd like to thank Jakob Engblom, Director of Simulation Technology Ecosystem at Intel, for offering aid concerning the Simics virtual hardware platform environment.

Henrik Andersson, Gothenburg, 2023-06-26

Carl Wiede, Gothenburg, 2023-06-26

Contents

List of Figures

1

Introduction

In an increasingly digital world, the demand for extensive computational effort is on the rise. This trend implies a growing energy consumption, where IT is estimated to represent over 10% of global energy consumption [1]. A need to find ways to reduce energy usage is becoming progressively more important not only to lower costs but also to minimize the environmental impact of computer systems on a global scale.

In the context of computer program evaluation, a fundamental metric is the execution time. This metric is commonly referred to as the *performance* of a program, with a lower execution time indicating better performance. In the field of sustainable computing, an essential metric used is the total *energy cost* of a program, obtained by multiplying the average power consumption during the execution by the execution time. Both performance and energy cost hold significant importance within the scope of this thesis.

Modern computers chips are able to throttle the execution speed of programs in order to consume less power using a technique called *dynamic voltage and frequency scaling* (DVFS) [2], which unveils an interesting trade-off opportunity between execution time and total energy cost. This trade-off is especially intriguing for heterogeneous (asymmetric) systems that comprise various types of processing components with different capabilities of energy efficiency and performance. Recent developments indicate that use of asymmetric platforms will increase due to their inherent adaptability [3][4]. Using as many resources as possible at the highest available frequency is typically the fastest way to execute a parallel program, but it is seldom the most energy efficient.

While possible to naively run programs at a setting that renders a minimal power consumption, the overall energy consumption doesn't necessarily decrease due to the implicit increase in execution time. Moreover, programs with time constraints cannot afford a drastically increased execution time. This thesis will explore the idea of executing programs at an ideal balance between performance and energy consumption on asymmetric platforms in order to reduce overall energy consumption without significantly lowering the performance.

1.1 Problem statement

This thesis will focus on task-parallel processing which essentially signifies that programs are broken down in smaller parts referred to as *tasks*. Programs in this type of processing are typically expressed as directed acyclic graphs where each node represents a task, which is further elaborated in Section 2.4. When possible, tasks are usually run in parallel to increase computational throughput. One of the main problems regarding task-parallel processing concerns task scheduling, i.e., deciding in which order and where on the hardware the tasks are executed. Scheduling plays a crucial role in maximizing the utilization of system resources and achieving high performance. Challenges in implementing a task scheduler include handling inter-task dependencies, managing load balancing across the system, and ensuring energy efficiency.

To achieve optimal task allocation, it is beneficial that the task scheduler is aware of two hardware concepts referred to as *static asymmetry* and *dynamic asymmetry*. Static asymmetry relates to the fixed hardware components, where multiple types of cores are integrated in order to provide divergent energy-performance features. Dynamic asymmetry refers to the capabilities of utilizing DVFS to adjust voltage and frequency when accommodating for differences in processing needs while running programs. In addition to understanding the underlying hardware platform on which it operates, a task scheduler must also possess knowledge relating to details of the tasks to be scheduled. This concept can be denoted as *task heterogeneity*, encompassing attributes such as computational demand, moldability (ability of inter-core execution), and granularity of each task. The three concepts are explained further in Section 2.1, 2.2, and 2.4, respectively.

A recently developed approach to the issue of energy efficiency in task scheduling is the STEER framework, as proposed by Jing Chen et al. [5], which is a state-of-the-art task scheduler focusing on energy efficiency on asymmetric platforms. The framework considers the static and dynamic asymmetry of the platform as well as the task heterogeneity of the input so that the total energy cost of the workload is kept at a minimum. STEER is built on top of XiTAO, which is a lightweight development platform for testing scheduling and resource management algorithms [6]. Although STEER has demonstrated a notable reduction in energy consumption compared to similar frameworks [5], it may not be suitable for all high-performance programs. This is due to the potential drawback of significantly increased execution time, rendering it impractical for programs with high-performance requirements.

1.2 Goals

This thesis will investigate potential trade-offs between performance and energy efficiency using the STEER framework. The original STEER framework will be modified in order to strike a balance between the two factors, reducing energy consumption while still acknowledging performance demands. The modified framework will also be tested on various hardware configurations to assess its generic applica-

bility.

Decomposing the goals, the first objective is to alter the scheduler with the aim of minimizing the energy-delay product (EDP, explained in Section 2.5), instead of the total energy cost, in order to attain reasonable balance between performance and energy cost. Specifically, the fusion of the two predictive models within STEER enables the creation of a new model that reveals the optimally balanced asymmetry configuration(s), which STEER then utilizes when executing the workload.

The second objective is to investigate the generic applicability of STEER. While the study conducted by Jing Chen et al. utilized a single asymmetric setup consisting of two contrasting sets of core types, this thesis will characterize STEER on multiple physical and virtual platforms with varying degrees of static and dynamic asymmetry. The results from this characterization will ultimately be examined in order to identify potential inefficiencies in the STEER framework.

1.3 Scope

This thesis focuses on systems that incorporate static and dynamic asymmetry. Platforms lacking these asymmetries are excluded from consideration since they do not experience the benefits offered by the task scheduler. Such platforms have limited or significantly fewer options regarding resource usage and processor frequency, thereby diminishing the effectiveness of the STEER framework.

Due to the fact that STEER is built upon the XiTAO runtime, it necessitates the design of the workload to align with this framework. To function properly, each task type must be implemented utilizing XiTAO libraries. Consequently, any task type that is not explicitly developed using the XiTAO library will require modification.

1.4 Significance

Should the thesis objectives be completed, the outcomes will effectively demonstrate the substantial potential of STEER in reducing the energy consumption of asymmetric systems. The likelihood of widespread adoption increases if the system performance remains largely unaffected by the implementation of the framework. This broader adoption can have far-reaching implications, potentially resulting in substantial reductions in energy consumption across various systems. Utilizing EDP as the primary metric will prove to be a successful approach to striking a balance between performance and energy efficiency.

1.5 Outline

Following the introduction, this thesis is structured into four chapters. Chapter 2 explains concepts essential to the thesis, some of which have already been mentioned. Chapter 3 contains detailed explanations of how results will be produced through

various modifications to the STEER framework, as well as using a hardware simulation tool. Chapter 4 displays aforementioned results and brief mentions of how they are significant to the thesis. Finally, Chapter 5 discusses and concludes the thesis based on the results ascertained on the previous chapter.

2

Background

There are a few core concepts related to computer architecture and sustainable computing that this thesis relies on. When combined in the context of using STEER, the concepts concern the structure of the target system, how to utilize the target system, and the format of the input.

2.1 Static asymmetry

One of the two fundamental characteristics found in an asymmetric platform is static asymmetry, which essentially denotes the utilization of multiple processor core types within a single chip. Each type of processor core is typically grouped into core *clusters*, each featuring its own cluster-wide cache, enabling largely independent runtime environments. Commonly, one type of core on the chip exhibits higher speed but consumes more power, while another type of core consumes less power at the expense of performance. Notable examples of architectures employing a cluster-based core layout which will be important to the thesis include the Nvidia Jetson TX2 [7] and Intel Alder Lake [8]. The TX2 features four energy efficient "A57" cores and two performant "Denver" cores, while Alder Lake features eight energy efficient "E-cores" and eight performant "P-cores".

When discussing static asymmetry, it will typically be mentioned in the form of what cluster is used and at which *resource width*, which basically conveys how many cores on that cluster are used to execute a task. This is also called *task moldability* which was briefly mentioned in section 1.1 as one of the attributes of *task heterogeneity*. This tuple of cluster and width is commonly referred to as the *execution place*. Typically, the execution time of a task will be shorter when given an increased amount of cores to run on. However, the energy consumption of a task is a more complex question, as a greater resource width will generate a higher power consumption. Some types of applications consume less energy running on the performant cores while some consume less energy running on the slower cores. This interesting attribute of static asymmetry is one of the reasons why asymmetric platforms are particularly focused in this thesis.

2.2 Dynamic asymmetry

The other fundamental characteristic found in an asymmetric platform is dynamic asymmetry, describing at which extent the system can adjust its voltage and frequency to various workloads during runtime using DVFS. Adjusting the voltage and frequency levels are one way to regulate the power consumption of a system. This becomes apparent once the dynamic power equation is considered:

$$P = \frac{1}{2} \times C \times V^2 \times frequency$$

Where P is power, C is load capacitance and V is the voltage level [9]. What essentially happens is that the CPU is throttled, reducing the frequency at which it executes instructions. Or by reducing the supplied voltage which will have a non-linear impact on the resulting power consumption, however this can reduce stability if done incorrectly. DVFS is usually governed by the operating system on a per-cluster basis in a way that causes continuous fluctuations in CPU frequency, increasing when under stress and decreasing when idle. The concept of adjusting the processor frequency using DVFS is commonly referred to as *frequency* or *voltage scaling*.

A system's capability of frequency scaling interacts with energy efficiency in an interesting demeanor. When scaling down a processor's frequency, it consumes less power. At a lower frequency, however, an application typically runs for a longer period of time. This entails that the total energy cost heavily depends on the characteristics of the application. Memory intensive applications display a less pronounced correlation to CPU frequency changes than a compute intensive application, for instance.

2.3 Memory-boundedness

The measure of memory-boundedness(MB) can be described as the ratio of an application that doesn't scale with processor frequency. This means that the memory-bound portion of an application is limited by the amount of memory it uses, regardless of the computational steps required.

The memory-boundedness of an application gives an indication of how its execution time changes upon frequency scaling. An application near 0% memory-boundedness could be expected to almost halve its execution time when doubling the processor frequency. Similarly, an application near 100% memory-boundedness could be expected to practically not differ in execution time when changing processor frequency. Simply put, 0% memory-boundedness implies a 100% compute-boundedness, and vice versa.

To derivate the formula for memory-boundedness, the total execution time at a certain frequency f_0 for an application can be divided into the memory-bound portion and the non-memory bound portion:

$$Time_{f_0} = Time_{f_0} \times (MB + (1 - MB))$$

The compute-bound part of the application will scale with processor frequency, while the memory-bound will not. This attribute can be used by introducing another frequency f and solving for the differences in execution time:

$$Time_f = Time_{f_0} \times (MB + (1 - MB) \times \frac{f_0}{f})$$

Solving for memory-boundedness, the previous formula is equivalent to:

$$MB = \frac{\frac{Time_f}{Time_{f_0}} - \frac{f_0}{f}}{1 - \frac{f_0}{f}}$$

With the final formula available, the process of determining the memory-boundedness is quite trivial. To characterize an application, the application is executed at a base frequency f_0, measuring its execution time $Time_{f0}$. The process is then repeated with a lower frequency f, measuring its execution time $Time_f$. With these data points, you can then calculate the memory-boundedness of the application using the formula above.

2.4 Task heterogeneity

In order to proficiently schedule tasks, three attributes of each task set must be considered that are jointly referred to as the task *heterogeneity*. The first aspect concerns *arithmetic intensity*, i.e., the number of arithmetic operations executed per byte of data. This is a metric that gives a pointer to whether an application is memory-bound or compute-bound, which is useful as different levels of memory-boundedness may entail different amounts of power consumption.

The second aspect of task heterogeneity is referred to as the task's *moldability*. The moldability of a task depends on its capability of being parallelized. A task that isn't moldable will only ever be able to run on a single core. A higher moldability will allow a task to run across multiple cores, possibly reducing execution time as more computing resources may be utilized. Figure 2.1 visualizes how a task graph may be structured with different task types, also known as *kernels*, and the dynamics of inter- and intra-task parallelism.

The third and final aspect in regard to heterogeneity is the granularity of a task set. Each individual task has a *workload size* which essentially means the size of the input data. As with the other aspects, this value can drastically impact the performance depending on the hardware structure of the target system. The task scheduler must consider the total workload size of each currently running task so as the execution time won't increase considerably due to cache misses. This is especially important for applications that intensively utilize caching.

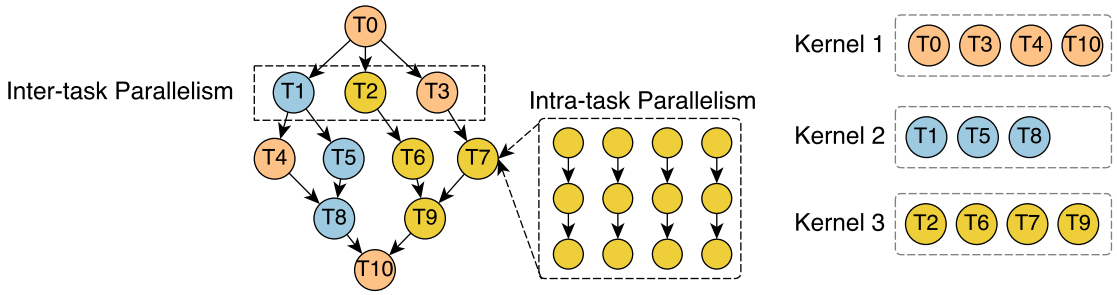


Figure 2.1: Task graph that visualizes inter- and intra-task parallelism, interdependencies, as well as kernel work queues.

Image source: Jing Chen et al. [10]

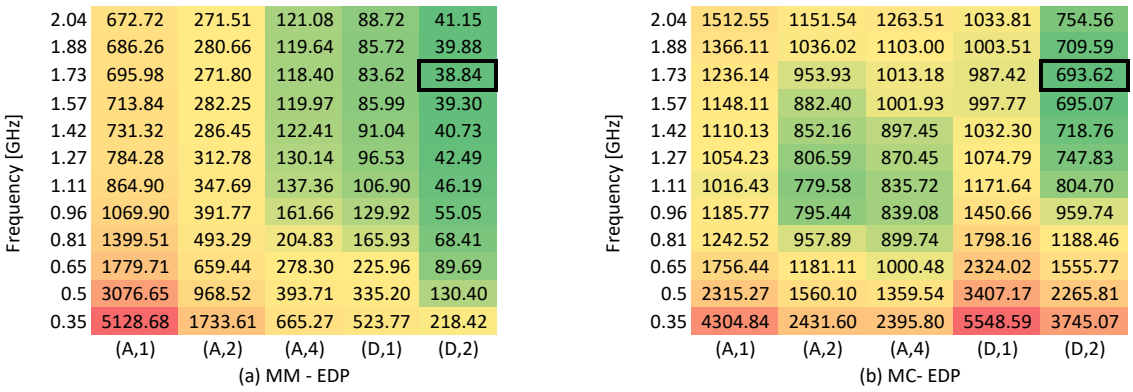


Figure 2.2: Benchmark EDP values of matrix multiplication and memory copy on Nvidia Jetson TX2. The columns are named on the form (cluster, width), where 'A' designates the A57 cluster and 'D' designates the Denver cluster. The cells marked with thick borders indicate the optimal configurations.

2.5 Energy-delay product

Energy-delay product (EDP) is a metric that will be used to evaluate the trade-off between performance and energy consumption in runtime simulations. EDP is defined as the product of the energy consumed by the framework (E, measured in joules) multiplied by the execution time (D, measured in seconds) [11]. A lower EDP value indicates a more efficient frequency setting and execution place that can perform the same operation in less time and/or with less energy than a setting with a larger EDP value.

The need to use a metric like EDP arises from the fact that some applications need to be executed at very low clock frequencies to be as energy efficient as possible. Such low frequencies may render a significantly increased execution time, making the framework infeasible to use should there be any strict time constraints.

2.5.1 Motivation

To demonstrate the usefulness of EDP, benchmark results are presented in Figure 2.2. The two tables are compiled from the motivational data presented by Jing Chen et al. [5]. The columns indicate different resource widths of the A57 cluster and Denver cluster of the NVIDIA Jetson TX2 platform. The rows indicate different frequency settings of the clusters. The energy consumption and execution time results from Jing Chen et al. can be found in Appendix A.1.

The figure unveils that running the benchmarks on (D,2) at 1.73GHz yields the lowest EDP values, and thus, the best trade-off between energy efficiency and performance for both matrix multiplication(MM), a default compute-bound benchmark, and memory copy(MC), a more memory-bound benchmark. Compared to the optimal execution time setting (D,2 at 2.04GHz), the optimal EDP setting renders a 20% reduction in energy consumption while increasing execution time by 18% for matrix multiplication. Similarly, the optimal EDP setting for memory copy displays a 21% reduction in energy consumption at a 16% increase in execution time.

For reference, the best setting for execution time can be compared to the values of the (D,2) at a 1.11GHz frequency. For matrix multiplication, the energy consumption is reduced by 38%, but the execution time increases by 81%. As for memory copy, the energy consumption is reduced by 39%, and the execution time increases by 75%. Thus, one can observe that even though the energy efficiency is improved, the relatively larger increase in execution time renders a worse trade-off. This indicates the usefulness of the EDP metric in that the differences in EDP values display the differing trade-off quality between settings.

2.6 NAS benchmark suite

Various benchmarks are available to test the framework. The benchmarks used throughout the project are the NASA Advanced Supercomputing (NAS) Parallel Benchmarks [12], which is a small set of programs derived from computational fluid dynamics designed to help evaluate the performance of parallel computers. For instance, the suite contains an embarrassingly parallel benchmark, which is near 0% memory-bound, as well as an algebraic multigrid benchmark, which is memory intensive and thus quite memory-bound.

2.7 STEER

As mentioned in Section 1.1, STEER is a task scheduling framework that emphasizes energy efficiency. The framework is central to the project as it is greatly involved in both of the main goals explained in Section 1.2. With all necessary theoretic concepts of the thesis explained, STEER may be described in greater detail.

2.7.1 Predictive power modeling

One of the optimizations that decrease the operational overhead of STEER is the usage of predictive power models. A measure for power is needed in order to calculate the energy cost which is used to determine the optimal runtime settings. An alternative to using predictive models is to generate an average power measure during runtime. This process could, however, add an infeasible amount of overhead at scale and be unreasonably inaccurate. These issues are of no concern when using a predictive power model.

To estimate a power model, benchmarks are run in order to generate accurate power usage averages for each frequency, execution place, and 10% interval of memory-boundedness from 0% to 100%. What resources are available as well as the intensity of computations directly affect the performance and power consumption, which justifies the classification. The need to also consider memory-boundedness arises from the fact that compute-bound and memory-bound applications generally consume power at a different rate, heavily depending on the runtime platform.

Because of variances in architecture and hardware layout, each unique system needs its own power model before being capable of running STEER. The generation of a power model may take from a few hours up to a full day depending on the desired accuracy of the model. Although this may appear tedious, it is important to note that STEER is designed to minimize overall energy consumption in the long term. Therefore, this one-time setup should not pose any significant issues.

2.7.2 Configuration

Before usage, the user needs to define some parameters that alter the functionality and behaviour of the framework. Concerning static asymmetry, the number of processor clusters and available widths need to be defined in order to let the framework know what resources are available. In regards to dynamic asymmetry, the user needs to specify what frequencies are available and how the framework may utilize the target system to change the processor frequency. This configuration process is similar to what is needed before generating the predictive power model. It is important to note that asymmetric differences between the two configurations may be error-prone.

2.7.3 Runtime

In order to run STEER, the user specifies what type of tasks to run, how many of each task, and the workload of each individual task (grouped by task type). Each type of task marks a set of tasks. Two things happen before executing the task sets. Firstly, a small sample of each task set is executed and measured at the highest and the midmost of the available processor frequencies. The estimated execution times at each remaining frequency are then extrapolated, generating the full *performance model*, i.e., the estimated execution time of a task at each available frequency and execution place. Secondly, the memory-boundedness of each task type is measured at each available execution place. The result of this measurement indicates what

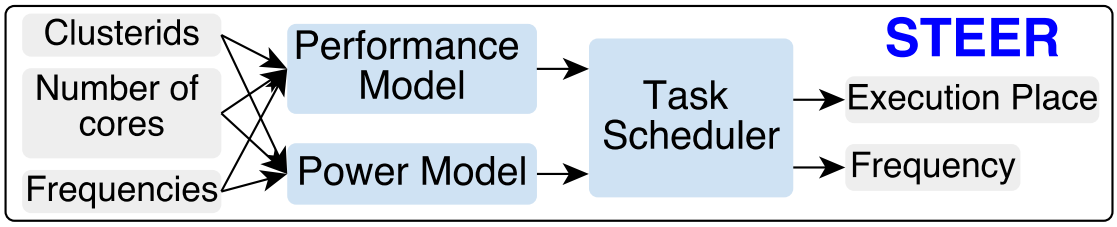


Figure 2.3: The runtime process of STEER, from where static and dynamic asymmetry is defined to where the optimal execution place and frequency is determined.

Image source: Jing Chen et al. [5]

average power value (from the power model) should be used for that specific type of task.

Once all the initial calculation is complete, both the predictive models of STEER are in place. The corresponding values at each individual frequency and execution place are then multiplied in order to generate the final model that reveals the estimated energy cost of executing tasks at each setting. Once the most energy-efficient setting has been determined, the tasks are prepared for execution. A somewhat simplified visual overview of the STEER runtime is presented in Figure 2.3.

2.8 Simics

Simics, originally developed as a research project by the Swedish Institute of Computer Science in 1991, is a framework for full-system simulation of computers and other electronic systems. It allows users to create a virtual model of a complete computer system, including processors, memory, peripherals, and other devices. This model can then be used to run software and firmware as if it were running on real hardware, allowing users to test and debug their code in a safe and controlled environment. Simics was commercially released in 1998 through the founding of Virtutech [13], a company that was later acquired by Intel in 2010. In 2021, Simics was made available as a public release [14] that will be used throughout the thesis.

2.8.1 Intel ISIM

Recent versions of the public release of Simics include the Intel Integrated Simulation Infrastructure with Modeling (ISIM), which allows a user to run an additional profiling layer on top of the virtual Simics platform in order to measure performance, power, and thermal profiles [15]. When configuring ISIM for use there are a number of parameters that needs to be defined in order for the simulator to accurately generate, and therefore measure, the aforementioned performance, power and thermal data. A sample of these parameters include thermal limits, thermal resistance, ambient temperature, but most importantly workpoints. Workpoints are defined as a tuple of (*voltage*, *frequency*) and are used by ISIM to determine the power consumed by each core at different frequencies. These measurements are made on a core by

core basis, giving an analyst a more detailed view of the simulated system than would normally be possible on most physical hardware.

3

Methods

There are two primary methods that will be applied in order to complete the objectives of the thesis. Firstly, STEER will be thoroughly analyzed and modified in order to support using EDP as its main metric when deciding upon an optimal setting of processor frequency and execution place. The framework will also be generalized to the point that it supports platforms other than the Nvidia Jetson TX2. Secondly, significant work will be put in to exploring the Simics full-system simulation framework with hopes of enabling the usage of a myriad of virtual platforms where STEER may be tested.

3.1 Necessary changes to STEER

Several modifications to the STEER framework are needed before results may be generated regarding the general and energy-performance balancing attributes of the framework. In order to accommodate the predictive power modeling of STEER, a method to generate power models on arbitrary platforms needs to be developed.

3.1.1 Generalization

The version of STEER as presented by Jing Chen et al. was designed to operate specifically on the Nvidia Jetson TX2, and as such, some academic shortcuts were taken to hasten development and produce results faster. Some portions of the source code were statically defined so as the framework would not function on a platform that doesn't feature two clusters with four and two cores, respectively. The instances of these static definitions will be replaced by variables that are defined in the configuration process. Rewriting the framework like this will ease later work while also granting a better understanding of the underlying algorithms within STEER.

In order to confirm the generic applicability of STEER, the modified framework will be run on a physical Intel Alder Lake platform. Should the results prove reasonable after some deliberation, this may be an indicator that STEER in fact is generically applicable. Virtual Simics platforms with various degrees of static and dynamic asymmetry will be used in attempts to provide additional results.

3.1.2 Power modeling

In order to flexibly run STEER on new platforms, a Python script will be used that utilizes the NAS benchmark suite in order to generate power models which can be used directly by STEER. Initially, the script estimates the memory-boundedness of each benchmark application in order to categorize which benchmark is most representative of each memory-boundedness interval (0%-10%, 10%-20%, ..., 90%-100%). With one benchmark categorized to each memory-boundedness level and resource width, each benchmark is run at its determined execution place at each available frequency in order to measure the average power consumption.

In order to measure power consumption, *running average power limit* (RAPL) [16] is used, which is a processor feature widely supported on Intel platforms. RAPL is readily available on the Nvidia Jetson TX2 as well as the Intel Alder Lake platforms. Even though Simics is an Intel product, however, the pre-built Simics platforms do not feature RAPL, which is addressed by using ISIM to generate power models for Simics platforms.

3.1.3 Energy-delay product

The effort to affirm that the framework can seamlessly determine the optimal energy-performance balancing setting is relatively trivial. STEER stores its estimated execution times and power costs in two dimensional lookup tables the size of the number of available frequency levels multiplied by the number of available execution places. Compiling these two tables, the lowest estimated EDP setting can be determined and used when executing the task set.

In order to confirm the functionality of the modifications, STEER will be run multiple times on the Nvidia Jetson TX2 using the very same settings as Jing Chen et al. used to generate motivational data. An average result will be produced and compared to the results presented in Figure 2.2. If the new data displays general patterns where the same execution places prove to yield similar improvements in EDP compared to the fastest setting, this would indicate that the changes were successful.

3.2 STEER changes in practice

As mentioned, both of the objectives of the thesis required changes to the STEER framework. This section contains practical details on the changes that were made in order to enable the completion of the thesis.

3.2.1 Cluster selection

The original version of STEER contained several assumptions regarding static asymmetry. Namely, the hardware layout was assumed to be equal to the Nvidia Jetson TX2, i.e., two cluster with two and four cores, respectively. For example, with 6 threads at runtime, the cluster in which the thread belongs to could be decided by

simply testing if the ID of the thread was lower than 2, meaning that it belonged to the Denver cluster. Otherwise, it belonged to the A57 cluster. These types of assumptions are problematic when striving for a generically applicable framework, as resources will either be neglected or falsely assumed to be available.

In order to address the problem mentioned above, a function was created, which essentially took the thread ID as an argument and returned which cluster the thread belongs to. Two variables that are set at configuration time are used within the function, namely the total number of clusters as well as an array containing the starting thread ID of each cluster. The total number of clusters are iterated through in a for-loop that checks if the thread ID is greater than or equal to the next index in the array. If it is not, the cluster is found.

Other assumption regarding static asymmetry was that the resource width of the clusters were equal to the Nvidia Jetson TX2, which was apparent when the framework wishes to assert that the lookup tables with performance and energy estimates are complete, where it only tests a statically defined number of widths for completion. This was addressed by once again using the variable containing the total number of clusters as well as an array which contains the number of available widths for each cluster. Instead of the static assumption, the widths of each cluster were traversed using their corresponding index in the width array.

For a less intricate but a far more extensive modification, many hundred lines of code could be culled by simply swapping static values for variables. The most rigorous example of this is that the source code featured a few hundred lines of code that were an exact copy of a few hundred other lines, the only difference being that the first segment referenced the Denver cluster, and the second referenced the A57 cluster. Using the previously mentioned function to determine the cluster of the current thread, a top-down search and replace operation solved this issue.

3.2.2 Task scheduling

Some modification was also needed in parts of the code that handled task scheduling, namely the case where tasks were randomly assigned to cores. The original version featured control flow where a random number was tested if it belonged to the Denver or A57 cluster, and then assigned to a random width within the bounds of the static assumptions of the respective clusters. To address this, a random number is generated within the bounds of available threads and then tested for which cluster it belongs to using the variables defined at configuration. Consequently, a random resource width is generated from the list of available widths of the corresponding cluster that is also configured before runtime.

3.2.3 Frequency scaling

Concerning dynamic asymmetry, several assumptions were made regarding the runtime platform. System paths were defined in the source code and static values were written directly to these static path definitions. To address this, a flexible and scalable method was implemented. A function was defined that takes the target cluster

and the requested frequency as arguments. With these values, an external Bash script is called. Modifying this script is needed when migrating platforms, but as a result, it is more malleable when modifying system parameters than directly doing so from C++ code.

Other than defining the aforementioned function, the framework no longer uses static values when invoking frequency scaling. Instead, it derives desired frequencies directly from the configuration parameters.

3.3 Simics

Simics will be used to simulate various hardware models that differ in amount of clusters, cluster sizes, and number of cores with assorted performance characteristics. This is mainly relevant in the later parts of the project where the generic applicability of the STEER framework will be evaluated. In practice, Simics provides the simulation environment where the user simply uses a scripting language to define various system parameters such as core count and available CPU frequencies.

3.3.1 Intel ISIM

Similarly to Simics, ISIM is managed through a script that sets various timing parameters for the profiling layer, such as cycles per read, cycles per write, and cycles for various arithmetic operations. Even though STEER estimates power costs through a predictive model based on static power measures from offline benchmarks, the virtual Simics environment may allow ISIM to provide better accuracy by measuring the power consumption at runtime, consequently providing a more accurate calculation of the optimal execution place. The parameters that are needed for ISIM to function correctly are shown in figure 3.1. Most of these parameters are not too relevant to this thesis but the *workpoint* list is used to calculate power at different frequency levels and the *thermal_throt_limit* will be expanded upon in section 3.3.6.

3.3.2 Clear Linux

The propriety operating system on Simics target systems is Clear Linux, an open source distribution of GNU/Linux managed by Intel's open source platform. The choice to use Clear Linux as a modern operating system on Simics target systems is a choice of convenience, as described by Jakob Engblom [17].

The Simics integrated version of Clear Linux does, however, feature some modifications as opposed to the distribution that is usually run on physical hardware. There are vital file system nodes featured in the default distribution that are not exposed to the target software as the proprietary CPU models are somewhat simplified. Examples of missing nodes are ones used for processor frequency and power management through DVFS. This is problematic in regards to STEER, mainly due to the fact that STEER requires the ability to scale the processor frequency at runtime. Missing power management nodes remove the possibility to generate static power profile values for the system, which STEER needs to properly calculate energy costs.

```
@thermal_R = 40.0
@leakage_parameter = 0.069
@thermal_C = 0.1
@thermal_throt_limit = 5500.0
@thermal_hysteresis = [0.0, 3.0]
@ambient = 40.0
@reference_temp = 40.0
@workpoints = [[0.90, 2e9],[0.85, 1e9]]
@cdyn_virus = 1.05e-9
@cdyn_idle = 1.6e-11
@governor = "performance"
```

Figure 3.1: Parameters that are needed for ISIM to function properly.

3.3.3 DVFS workaround

To get started with Simics, the developers have supplied a number of scripts for a new user to quickly get Simics up and running with Clear Linux as the proprietary operating system. A common way for a Linux user to regulate the frequency of their CPU is to write the desired frequency to the corresponding CPU core file. Since this functionality is not supported in Simics, some workarounds are needed.

As can be seen in figure 3.2, while the target system is active, a monitoring script is used which keeps track of the outputs on the terminal on the target system and activates once a chosen keyword is found. This can be seen on line 3, where this script wait until it notices *chfq*. Once noticed, line 4 halts the simulation and line 5 reads the output line of the console, which is the current text that has been output on the target terminal and can be seen in figure 3.3. The output is then parsed and the if statements makes the relevant changes to the targeted cores. Upon completion of the changes, the simulation is started again. This functionality mimics the way STEER changes core frequencies during runtime and is instant from the view of the target system.

A concern arose regarding the behavior of output to the target terminal since it is not guaranteed that output is handled immediately, and thus could introduce a delay before it appears on the terminal for the monitoring script to detect and handle. To measure a possible print overhead a similar script to the one mentioned above was used, which simply froze the simulation once triggered and took the timestamp of when it was triggered. The trigger phrase was issued as the final instruction before

```
script-branch {
    while TRUE {
        bp.console_string.wait-for machine0.serconsole.con "chfq"
        stop
        $cmd_output = machine0.serconsole.con->output_line
        $split = (split-string " " $cmd_output)
        echo ("Set cluster " + $split[0] + " to " + $split[1])
        if $split[0] == "S" {
            machine0.mb.cpu0.core[0][0].vtime->frequency = $split[1]
            machine0.mb.cpu0.core[1][0].vtime->frequency = $split[1]
            machine0.mb.cpu0.core[2][0].vtime->frequency = $split[1]
            machine0.mb.cpu0.core[4][0].vtime->frequency = $split[1]
        } else if $split[0] == "B" {
            machine0.mb.cpu0.core[3][0].vtime->frequency = $split[1]
            machine0.mb.cpu0.core[5][0].vtime->frequency = $split[1]
        } else {
            echo "incorrect cluster"
        }
        run
    }
}
```

Figure 3.2: Script that monitors the target terminal for *chfq* and changes frequency of the chosen cluster if found.

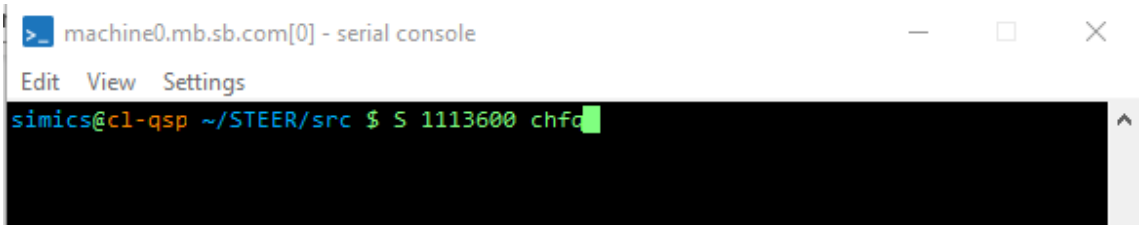


Figure 3.3: Change frequency output, **S** selects the small cluster, **1113600** is the new frequency and *chfq* is the keyword that makes the script stop the simulation and change the frequency of the chosen cores.

the parallel section in the NAS benchmarks. The ISIM data then showed that there was only one core actively working on the active workload, indicating that printing occured before the parallel section started. Once a visual inspection of the data was made and it seemed like everything was in order, the benchmark was allowed to continue to completion. Once the benchmark was finished the timestamp of the freeze was compared to when the parallel section started, which showed that the freeze was issued the timestep right before the workload began. To analyze further, a similar verification was added to STEER, which produced the same result and indicates that the output is processed and output immediately to the terminal.

3.3.4 Pre-built CPU models

Another feature of Simics that showed promise was that several CPU models built for different Intel architectures already resided within Simics, such as Alder Lake, Haswell, Goldmont to name a few. The Intel Alder Lake, for instance, is an asymmetric platform which features a performant as well as an energy efficient core cluster. This could prove especially useful as the thesis emphasizes task scheduling on asymmetric platforms. However, these models do not seem to change anything in regards to performance and energy consumption when applied in a Simics script. A number of tests were run with some of the different pre-built models and no changes were experienced while running the benchmarks on their own, nor when running the benchmarks with ISIM applied to gather performance data. Even across energy efficient and performance cores on asymmetric CPU models, no significant difference was found.

Instead of relying on the pre-built CPU models which attempted to emulate real world platforms, the *Simics Quick Start Platform* (QSP) was used. QSP is a made up system that does not exist in physical hardware which purpose is to have an easy system for software development or architectural studies. To facilitate this ease of development Simics comes with a couple of pre-written scripts that can be used to get started with modeling a target system running a QSP core. The accompanying *qsp-isim-clear-linux.simics* script was used as a basis to build the tested target systems mentioned in section 3.3.7.

As a fully virtual platform, it proved to introduce less simulation overhead as it didn't attempt to emulate any real world hardware techniques. By default, QSP is a single cluster platform with symmetric cores. With some simple scripting, however, an asymmetric behaviour could be implemented by changing the performance and energy characteristics of individual cores.

3.3.5 Measurements using ISIM

In order to affirm that Simics was working properly and could be deemed fit as the selected hardware simulation tool for the thesis, benchmark data was collected using Intel ISIM. The benchmarks consisted of matrix multiplication and memory copying. These two benchmarks were chosen to observe the results of a compute intensive workload, as well as a highly memory-bound workload.

As the machine simulated in Simics will not be a perfect copy of the physical hardware these benchmarks are done to affirm whether Simics performs sufficiently similar to a physical setup to be able to be used as a substitution for continued work with the simulator. The simulated setup will strive to resemble the Nvidia Jetson TX2 for comparison purposes and is explained in more detail in section 3.3.7.

ISIM collects multiple data points from the simulated system at an interval of 2 ms, capturing the power consumed by each core as well as the current processor frequency, temperature and number of *instructions per cycle* (IPC). The benchmarks all contain a startup period where there is only one core working, this startup looks the same for 1, 2, and 4 cores on the same frequency setting and differs slightly between frequency settings. This period is disregarded in the motivational data presented in the results section since there is only one core working, and all the work that is being done is in preparation for the actual workload, creating data that is later used in the benchmarking sections. Once the parallel section starts and calculations begin, the data starts being measured to focus on the workload of different numbers of cores. An example of the power data can be found in Appendix B.1.

3.3.6 Thermal throttling workaround

As previously mentioned, ISIM generates thermal data during the simulation to more accurately simulate the target system during runtime. This is where a problem arises in terms of the means of the thesis. To retrieve accurate power data for the benchmarks, the CPU frequency needs to be constant during the entirety of the workload. At higher frequencies this leads to a rapid temperature escalation where, with the default configuration, the system governor throttles the CPU after just a couple of seconds and finishes the workload at the lowest available frequency. To get around this behaviour, a relatively simple solution was found that worked on the frequencies being measured. Essentially, the governor that regulated frequencies was removed by setting the throttling limit to about 5000 °C. This number was not chosen for any specific reason other than to ensure that no switching occurred while a workload was active. This method made the simulation unstable and prone to crash after some time, around when the simulated hardware hit 120-130 °C. This had the consequence that the size of the workloads had to be restricted so that the simulation did not crash before the benchmarks finished.

Due to the solution being unstable, attempts were made to find a way to increase the cooling capabilities of the target system. However, no configuration was found that changed the way the system heated up. As such, the aforementioned solution was utilized in order to generate initial motivational data.

3.3.7 Modeling a target machine in Simics

The simulated system is configured to mimic the Nvidia Jetson TX2, an asymmetric cluster-based platform as described in section 2.1. There is no available Simics model for the Nvidia Jetson TX2 and one needs to be emulated to generate data which

```
# Borrow all parameters from the standard file, with some small
# changes to default parameters.
params from "%simics%/targets/qsp-x86/qsp-clear-linux.simics"
    default cpu_comp_class       = "x86QSP1"
    default show_con0            = TRUE
    default connect_real_network = "no"
    default auto_login_root      = FALSE
    default auto_login           = TRUE
    default num_cores            = 6
    default freq_mhz             = 2000
```

Figure 3.4: Start parameters for the Jetson TX2 emulation.

```
machine0.mb.cpu0.core[0][0].set-step-rate 0.5
machine0.mb.cpu0.core[1][0].set-step-rate 0.5
machine0.mb.cpu0.core[2][0].set-step-rate 0.5
machine0.mb.cpu0.core[3][0].set-step-rate 1.9
machine0.mb.cpu0.core[4][0].set-step-rate 0.5
machine0.mb.cpu0.core[5][0].set-step-rate 1.9
```

Figure 3.5: IPC settings for individual cores.

will be used as a comparison to the motivational data generated by the physical hardware. The parameters presented in figure 3.4 were used as a baseline for the simulation, with 6 cores operating at 2 GHz. As previously mentioned, none of the pre-built CPU models performed differently in early tests. As such, all following testing utilizes the QSP model due to its higher simulation performance compared to other platform models that were tested. On the physical Jetson TX2 the single core performance of a Denver core is roughly 3.8 timer faster than a single A57 core. As such, the emulated Denver cores had their IPC set to 1.9 while the A57 cores were set to 0.5 and can be seen in figure 3.5. These numbers were chosen due to Simics not being able to handle an IPC of 1.0 and 3.8 for the A57 and Denver cores. The IPC difference makes the core types have distinct performance results, as well as consume different amounts of energy.

When using STEER one has to specify the layout of the cores and what configurations they will work with. For example if the system has a cluster of four cores that should only run all four cores at a time on a workload we set the layout to:

0 ,1 ,2 ,3
4

which means that only the 0:th core can be the leader, and if they are, then the

workload should be run on all four cores. As a result of how the individual cores map in Simics, core 3 and 4 are swapped when using the setup shown in figure 3.5 and would result in three small cores and one big core being run on the layout above. And as such needs to "swap places" with each other to get the correct behavior.

4

Evaluation

This chapter presents the results obtained from the efforts made to make STEER more energy-performance balanced using the EDP metric, as well as testing its general applicability. Both physical and simulated hardware were used in the attempts to assert the general applicability of the framework.

4.1 EDP alterations

The first of two main thesis objectives was to alter the STEER scheduler with the aim of minimizing the energy-delay product in order to attain a balance between energy and performance. By modifying the source code of the framework and testing the changes to a reasonable extent, this goal was fulfilled.

4.1.1 Framework modifications

The resulting modifications to the STEER framework turned out to be quite trivial in regard to lines of code. Since a proper system for selecting the optimal execution setting was already in place, what remained was to calculate the estimated EDP of each setting in order to evaluate which setting yielded the lowest value. The newly implemented search function is presented in Algorithm 1.

Compared to the 'Search for Best Configuration' algorithm presented by Jing Chen et al., there are two modifications. Firstly, the EDP value of the current iteration is calculated by multiplying the estimated energy cost by the estimated execution time. Secondly, the iteratively updated optimal configuration is based on this EDP value instead of the total energy cost.

4.1.2 New TX2 data

In order to test the capability of the STEER modification targeting EDP, the altered framework was used to generate motivational data similar to that of Jing Chen et al. (displayed in Appendix A.1). The very same hardware platform (Nvidia Jetson TX2) was used by running matrix multiplication and memory copy benchmarks designed for the framework. The new data is shown in Appendix A.2 and the more important EDP values compiled from the data are displayed in Figure 4.1.

Algorithm 1 Search for optimal EDP configuration

```
1: for f in possible frequencies do
2:    for c in cluster indexes do
3:       for w in resource widths of cluster do
4:          time ← get_timetable[f][c][w]
5:          runP ← get_powertable[f][c][w]
6:          idleP ← idleP_cluster[f] × w ÷ active_cores[c]
7:          energy ← time × (idleP + runP)
8:          edp ← time × energy
9:          if edp < min_edp then
10:            min_edp ← edp
11:            best_freq ← f
12:            best_cluster ← c
13:            best_width ← w
14:         end if
15:      end for
16:   end for
17: end for
```

The new data reveals that the optimal EDP execution place is (D,2) at a frequency of 1.57 GHz for both matrix multiplication and memory copy. Compared to the execution time optimal setting of (D,2) at 2.04 GHz, execution time is increased by 28% while energy consumption is reduced by 30% for matrix multiplication. Similarly, execution time is increased by 29% and energy consumption is reduced by 30% for memory copy compared to the optimal setting for execution time.

For reference, the optimal execution time settings can be compared to running the benchmarks on (D,2) at 0.35 GHz. For matrix multiplication, execution time would increase by a colossal 473% while the energy consumption would decrease by 37%. For memory copy, the execution time would increase at a relatively larger 489% while the energy consumption would decrease by 35%. It is clear to see that the quality of the trade-offs differ heavily between execution place and processor frequency.

Despite being run at the same settings on the same platform, there are some differences between the new data and the motivational data presented by Jing Chen et al. The possible reason(s) behind this are discussed in Section 5.1.1.

4.2 General applicability

The second objective of the thesis was to investigate the generic applicability of STEER by using the framework on platforms other than the Nvidia Jetson TX2. The framework was run on Intel Alder Lake in order to verify a reasonable generic applicability on another physical platform. STEER was also tested in Simics with hopes of proving that the framework may prove to be useful on a myriad of virtual hardware platforms.

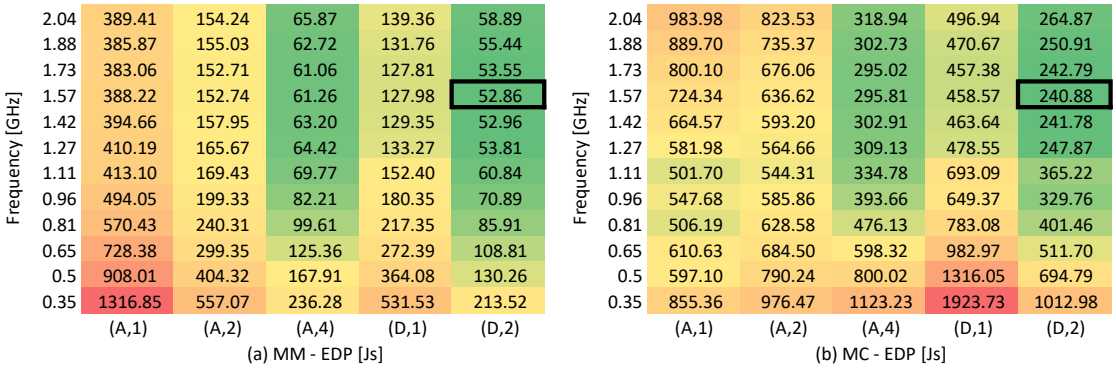


Figure 4.1: EDP values of matrix multiplication and memory copy on Nvidia Jetson TX2 generated by algorithm 1. The columns are named on the form (cluster, width), where 'A' designates the A57 cluster and 'D' designates the Denver cluster. The cells marked with thick borders indicate the optimal configurations.

4.2.1 Framework modifications

Before using STEER, the user must define some configuration parameters that allows the framework to use the platform properly. However, some parts of STEER were statically designed for Nvidia Jetson TX2, which meant that some changes to the source code were required, as described in Section 3.2 above. The modified framework was able to run on Intel Alder Lake practically seamlessly, which indicates that the endeavours to generalize STEER were successful.

4.2.2 Power modeling

As STEER utilizes a predictive power model in order to minimize execution time, a unique one has to be generated for each new platform. For this purpose, a Python script was created together with supporting Bash scripts that will utilize the NAS benchmark suite together with the Intel RAPL technology in order to generate power models for each 10% interval of memory-boundedness. After some configuration similar to what is needed before using STEER, the power modeling script will map out the memory-boundedness of each NAS benchmark, a mapping that may differ between platforms. Then, an average power consumption estimate is generated by running the select benchmarks at each available processor frequency and execution place while RAPL measures the actual power consumption.

4.2.3 Benchmarking STEER on Intel Alder Lake

In order to benchmark the framework on the physical Intel Alder Lake platform, the STEER benchmarks for matrix multiplication and memory copy were used. The very same runtime settings were used as when previously generating results on the TX2 platform as described in Section 4.1. The resulting EDP tables from the Alder Lake benchmarking can be found in Figure 4.2 together with the original execution time and energy consumption estimations in Appendix A.3.

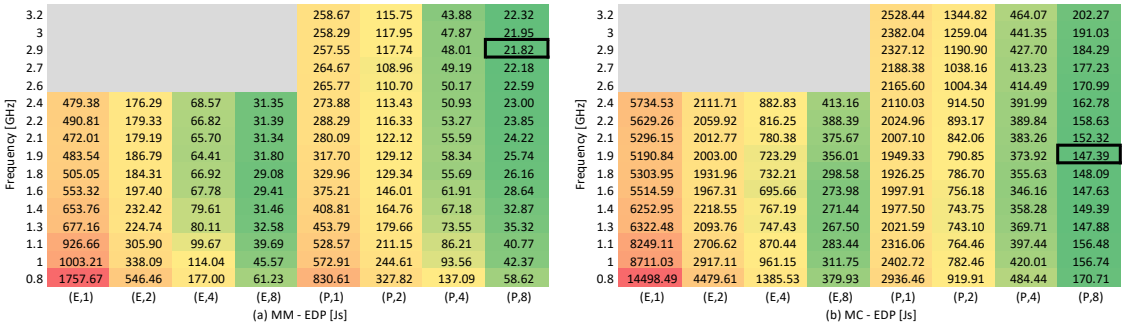


Figure 4.2: Benchmark EDP values of matrix multiplication and memory copy on Intel Alder Lake. The columns are named on the form (cluster, width), where 'E' designates the energy efficient cluster and 'P' designates the performant cluster. The cells marked with thick borders indicate the optimal configurations. The frequency of the E-cluster cannot surpass 2.4 GHz, hence the grey areas.

As with the previously presented TX2 measurement, the performant cluster at maximum resource width yields the shortest execution time. Interestingly enough, the most energy efficient settings are very similar between the benchmarks, with (E,8) at 1.3 GHz for matrix multiplication and (E,8) at 1.1 GHz for memory copy. Although both benchmarks yield similar results, the optimal settings for EDP differ by some margin, with (P,8) at 2.9 GHz for matrix multiplication and (P,8) at 1.9 GHz for memory copy.

In terms of performance-energy trade-offs, the optimal EDP settings may be compared to (P,8) at 3.2 GHz, which is the optimal setting for minimizing the execution time of both benchmarks. The optimal EDP setting for matrix multiplication increases execution time by 9% while reducing the energy consumption by 10%. For memory copy, the optimal EDP setting increases execution time by 21% and reduces the energy consumption by 40%. As with the TX2 measurement, the optimal EDP settings are both in the column of the maximum resource width of the performant cluster. These results produced on Alder Lake are discussed in Section 5.1.2.

4.2.4 Benchmarking STEER in Simics

All benchmarks in Simics were run on an emulated Jetson TX2, as described in section 3.3.7. Figure 4.3 shows the results of running a benchmark workload that consists of 100 tasks, each task computes matrix multiplication on matrices of size 128x128, in Simics. The greener portions are better than the redder and the bold cell is the best result in each respective table. Since the simulation is done with arbitrary cores they have been named '**S**' for "small" and '**B**' for "big" in the table. The number that follows is the width of the benchmark, the number of cores that were working on the task at once.

Figure 4.4 shows an incomplete table of benchmarks that consists of 2000 tasks, each task copying 2048 bytes of data in memory, in Simics. Similarly as in the matrix multiplication greener cells are better and red cells are not to be prefered. The

GHZ	128x100 MM				
2,04	7,15287	3,6025203	1,824198	1,956298	0,975193
1,88	7,77497	3,796489	2,036266	2,150467	1,029676
1,73	8,46518	4,2646782	2,16447	2,288537	1,167375
1,57	9,4695	4,8488215	2,56078	2,557836	1,295788
1,42	10,6919	5,5369915	2,852805	2,767414	1,385914
1,27	12,0623	6,18132	3,230957	3,071731	1,670403
1,11	14,2088	7,2718256	3,761107	3,670548	1,848677
0,96	17,0914	8,5102946	4,447371	4,504373	2,261394
0,81	21,1752	10,67453	5,494027	5,522239	2,72163
0,65	28,4683	14,532201	7,327403	7,466319	3,82579
0,5	42,3159	21,315902	10,81682	10,86095	5,259031
0,35	82,1378	41,376301	20,93898	20,7794	10,68727
TypeCore# ->	S1	S2	S4	B1	B2
	Performance [s]				

128x100 MM				
5,963502	5,357807	5,112529	5,182913	4,968183
6,1516	5,383921	5,385698	5,489129	5,034374
6,279622	5,579692	5,279752	5,387417	5,253126
6,534525	5,809836	5,56115	5,589883	5,341665
6,839788	6,056211	5,690203	5,506153	5,206179
7,156241	6,21838	5,901421	5,521842	5,524464
7,652062	6,540984	6,079938	5,793192	5,469337
8,237394	6,738672	6,171295	6,185184	5,754207
9,119573	7,383147	6,567562	6,555378	5,96716
10,70507	8,504651	7,24001	7,332444	6,736537
13,67649	10,21196	8,554217	8,630799	7,294115
22,09345	15,2547	11,93938	12,32178	10,55294
S1	S2	S4	B1	B2
Energy [J]				

Figure 4.3: Simics matrix multiplication benchmarks

GHZ	2048x2000 MC				
2.04	14.65951	7.522973	4.088587	4.729355	2.486917
1.88	16.03834	8.2953404	4.55684	4.775172	2.534915
1.73					
1.57					
1.42					
1.27					
1.11	29.52505	14.92487	7.77045	7.702776	4.057846
0.96					
0.81					
0.65					
0.5					
0.35	169.3708	85.424819	43.89585	43.1252	21.56645
TypeCore# ->	S1	S2	S4	B1	B2
	Performance [s]				

2048x2000 MC				
16.44299	15.27703	15.38386	15.31886	14.94918
16.75537	15.59569	15.82659	15.40586	15.12565
19.91985	17.43322	16.46787	16.62172	16.23056
52.0171	37.99101	31.12759	32.78098	28.26048
S1	S2	S4	B1	B2
Energy [J]				

Figure 4.4: Simics memory copy benchmarks, incomplete due to unwanted simulation behavior

table has not been completed due to problems arising from the simulator beyond the authors control, which will be discussed in section 5.1.3.

5

Conclusion

This chapter will contain discussion and conclusions based on the previously presented results in regards to the modifications of the STEER framework and its applicability in Simics. There will also be some final deliberations concerning future work with STEER and how Simics may yet prove to be a useful tool in efforts to test the framework.

5.1 Discussion

Although the effort put into the thesis didn't amount to an overwhelming amount of results, there is still plenty of discussions to be held. Both of the benchmark suites run on Nvidia Jetson TX2 and Intel Alder Lake provided interesting data, and there is still a lot to be explored about the intricacies of Simics.

5.1.1 EDP modifications

As base for the discussion, benchmark data from running STEER on Nvidia Jetson TX2 was presented by Jing Chen et al. in the introductory paper on the STEER framework. The source data can be found in Appendix A.1. The benchmark data was compiled into tables where the total EDP cost of each setting is presented, found in Figure 2.2. For this thesis, a new benchmark suite with the same settings was run on the very same platform. The source data can be found in Appendix A.2 and the compiled EDP data can be found in Figure 4.1.

Considering performance, even if both sets of data display that the optimal setting for minimizing the execution time of both benchmarks is (D,2) at 2.04 GHz, the pattern of the results vary slightly. The new data displays a more pronounced difference in execution time between (A,4) and (D,1), where the former is consistently faster. Another important note concerns the notion of memory-boundedness. The old data showed clear signs of memory-boundedness between resource widths in the memory copy benchmark. The new data only shows this correlation at lower frequency levels, while the higher frequency levels display execution time differences similar to those of the non-memory-bound matrix multiplication benchmark between resource widths. Two possible reasons for these performance differences are discussed below.

One explanation for the performance differences could be the fact that the older data was measured about two years ago. Various factors of hardware degradation

can be considered, such as aging of electronic components, thermal stress due to continuous operation under high temperatures, or physical wear and tear of mechanical components. While unlikely that hardware degradation is the sole reason that the performance differs, it's not unreasonable to consider it as a factor.

Another possible reason behind the performance contrast could be the various changes made to the framework in efforts to generalize it. Albeit that generalizing the framework was the main purpose of modifying the framework, some changes may inadvertently have affected the performance. The modifications within the efforts of this thesis kept every bit of functionality and should not be the reason for such drastic performance differences. Previous framework modifications may, however, be the perpetrator, as the framework has certainly underwent changes between two years ago and the start of this thesis.

One can observe that not only the performance but also the estimated energy consumption of both benchmarks differ, rendering new optimal settings for minimizing energy consumption. Since the same power model that was used for the old data also was used to generate the new. However, this is most likely due to the previously discussed performance differences which should be the only factor that separates the two data sets.

Despite the differences in the estimated performance and energy consumption, the newly compiled EDP data displays that the optimal EDP setting for both matrix multiplication and memory copy is (D,2) at 1.57 GHz. This is noteworthy, as the optimal EDP setting for both benchmarks in the older data is (D,2) at 1.73 GHz, i.e., the same setting at one frequency level higher. This ought to be an indication that the energy-performance balancing modifications were successful and that the framework is able to select an estimated optimal trade-off setting. In order to confirm this properly, more data is needed. The data should preferably be retrieved from various platforms and be coupled with an extensive analysis of the framework, as described in Section 5.3.3.

5.1.2 Affirming generic applicability on Intel Alder Lake

The STEER benchmark suite of matrix multiplication and memory copy was run on Intel Alder Lake for the purpose of affirming the generic applicability of the framework. The source data is found in Appendix A.3 and the compiled EDP tables are found in Figure 4.2.

As anticipated, the estimated best setting for minimizing execution time for both benchmarks is (D,8) at 3.2 GHz. It is only reasonable to assume that the maximum resource width of the performant cluster at the highest available frequency should yield the shortest execution time as long as the workload is parallelizable. Similarly to the new Nvidia Jetson TX2 benchmark results, the (memory-bound) memory copy benchmark displays a proportional execution time decrease when doubling the resource width on a cluster. Various factors regarding the benchmark in question and the characteristics of the platform decide if this is reasonable or not. An indication that the memory-boundedness affect the results, however, is the difference between

frequency levels. The non-memory-bound matrix multiplication benchmark displays a 49% decrease in execution time when scaling from 1.6 GHz to 3.2 GHz (D,8), which is fully reasonable. For memory copy, the execution time is reduced by a mere 23% when doubling the frequency from 1.6 GHz to 3.2 GHz (D,8), which indicates that the benchmark indeed is memory-bound.

The power model for the Intel Alder Lake was generated using the Python script developed for the purpose of generalizing the framework. The procedure of generating a power model for a new platform and using it in STEER was successfully seamless. Surprisingly enough, the optimal settings for minimizing energy consumption were very similar, with (E,8) at 1.3 GHz and (E,8) at 1.1 GHz for matrix multiplication and memory copy, respectively. The same cannot be said for the benchmark results from Nvidia Jetson TX2, which indicates that there is no generally optimal setting for minimizing energy consumption across workload characteristics and platforms. All of the new benchmark results display that the energy efficient clusters purposefully provide the most energy efficient execution places, which confirms that asymmetric platforms are versatile in balancing performance and energy efficiency.

Even though the Intel Alder Lake benchmarks were specifically intended to confirm generic applicability, the optimal EDP settings are still interesting to discuss. As for the Nvidia Jetson TX2 benchmarks, both matrix multiplication and memory copy display that the performant cluster at maximum width between the midmost and highest frequency provides the best trade-off. This raises the question if this is the default case for asymmetric platforms. The likely answer is that it depends on the benchmark application. Both matrix multiplication and memory copy is highly parallelizable, which allows the execution time to scale well with increased resource width. A naively implemented Fibonacci benchmark, however, would likely not scale at all with an increased resource width, and thus display a width of one as its optimal EDP setting. The limited amount of test benchmarks doesn't display the possibly large extent of reasonable optimal EDP settings across various platforms and benchmarks.

To summarize, the objective to affirm the generic applicability of a slightly modified STEER framework was a success. The power profiling and consequent use of the new power model in STEER was seamless, and the benchmarks could be run on the new platform after some minor modifications. As no other physical asymmetric platform than the Nvidia Jetson TX2 was available there was no incentive to develop and test a power modeling method that didn't use Intel RAPL. In Simics, this may be solved by using the built-in power and thermal management tools. On physical platforms that do not feature Intel RAPL, however, no power modeling method is available for use within the framework.

5.1.3 Simics motivational data

Compared to the results from the physical Jetson TX2 the performance of the simulated benchmarks follow the same general pattern. More cores (higher throughput) on a higher frequency is the best. This result is expected and is the same as the results by Jing et. al. However, the simulated results differ in how the pattern

for energy consumption culminates. Whereas the physical benchmark shows that staying around the middle of the pack at 1.11 GHz is the most effective energy wise for all core types and all workload widths, the simulation data shows that the setup with the highest performance also is the best power wise.

While there is no concrete explanation as to why the power consumption of matrix multiplication is different, we can make some assumptions. First and foremost, there needs to be more accurately defined power scaling at different workpoints that does not scale linearly. At the moment all of the power calculations are made from a single workpoint with a static provided voltage that scales linearly with the frequency. At 2.04 GHz a small core require 0.7 W while executing the workload when the same core only need 0.12 W at 0.35 GHz. The performance difference between 2.04 and 0.35 GHz is around 5.83 times, while the power consumed is almost the same around 5.35 times. The voltage would usually need to be increased to provide stability when running cores at a higher frequency. As mentioned in section 2.2 the formula for calculating dynamic power is $P = \frac{1}{2} * C * V^2 * frequency$, and shows us that a change in voltage results in a non-linear power change and could therefore explain the difference in the simulated results.

Figure 4.4 in section 4.2.4 was left mostly blank. This was due to a shortcoming in Simics when running memory-bound benchmarks. As can be seen from the table, benchmarks on S1 at 2.04 GHz are finished around 14.6 seconds, while S2 takes 7.5 seconds. These performance results indicate that something is wrong with either the benchmarks or with Simics. The memory copy benchmarks are supposed to be highly memory-bound, which means that there should be a limited performance increase from executing the workload on multiple cores. A performance increase from 14.6 to 7.5 seconds is close to 100% and indicates that the executed workload behaves more like the matrix multiplication benchmark, which has a very low level of memory-boundness. When running the same benchmarks on a physical machine we get a slight performance increase when running the workload on multiple cores, which is the behavior that is expected. Therefore we can rule out that there are problems with the benchmarks themselves. To assert that memory bound workloads behave incorrectly in Simics we also ran two of the more memory-bound NAS benchmarks. When adding additional cores to the new benchmarks they also exhibited a linear performance increase, leading us to the conclusion that our target machine does not simulate memory-operations correctly.

After some investigation we realized that in order to improve the performance of the simulation, most of the memory handling is abstracted away, and a consequence of the abstraction seems to be that memory operations are handled right away. Therefore the workload can utilize the full effect of another core, which is not the case on a physical machine, resulting in the linear performance increase. A consideration was made that the problem could originate from the choice of core type. As all of the tests were done with the QSP core type, these problems could originate from the QSP model due to it being a simpler model. However tests with other core types such as a coffee lake core end up with the same results.

In the documentation there exists a workshop on adding a cache hierarchy to the

target system. An attempt was made to run the benchmarks on a modified target machine that had a three level cache added to simulate cache hits and misses. To easier spot a performance difference with the added caches, a cache miss was set to induce a cycle stall of 10.000 cycles and hopefully remove the linear performance increase from adding cores to the workload. Unfortunately the added cache hierarchy did not seem to have any functional impact on the target as the benchmarks on one and two cores still followed a linear performance increase. Furthermore the addition of these caches impacted the simulation performance noticeably which led to benchmarks that took around 5 minutes to finish in real time now were done after around 25 minutes. As such, the long simulation times did not encourage further exploration of other benchmark workloads.

A concern arose in regards to the workaround shown in section 3.3.6 concerning disabling the automatic frequency throttling, and if the results could be affected by the high thermal limit. In the startup phase of the thesis higher frequencies were used, from 1 to 4 Ghz, which overheated very quickly and subsequently crashed the simulation. Running ISIM while one of these early crashes occured showed that the power consumption did increase a bit during the final moments before the crash. This could have become a problem if the part of the code that crashed were changed and it was possible to run workloads while the cores overheated. Fortunately the frequencies chosen were lowered to match the frequencies available on the Jetson TX2, that operate between 0.35 to 2.04 GHz, and as such no further crashes occured during the final data gathering.

5.2 Practicality of Simics

From the start, Simics seemed to have the potential to work well with STEER. Being able to emulate machines and get accurate performance and energy data from these simulations would make it possible to build a power model without access to the physical hardware. During the progression of the thesis it became more and more apparent that in order to retrieve accurate results, a lot of details of hardware behavior was needed, most of which were not readily available. For example, voltage values at different frequency points are needed to calculate power levels. As this information was not able to be found, testing was done using a value that came with the ISIM example and then scaled proportionately with the frequency.

As previously mentioned in both the Methods section and the Results section Simics has some inherent flaws when considered from the point of view of this thesis. The fact that the system nodes controlling CPU frequency on a physical machine are missing on the simulated target makes the baseline STEER implementation incompatible with Simics. On the other hand, being able to utilize ISIM to analyze the power consumption of individual cores and from that data build the power model that STEER uses to schedule tasks is a smoother process than going with the RAPL route discussed in section 3.1.2.

A modified version of Clear Linux is the proprietary operating system on Simics target machines. Could another operating system be used that features the necessary

system nodes on which STEER relies? Probably not. Firstly, there is no guarantee that, even though another operating system would feature the nodes, the virtual machine would properly be able to use them. There is a reason that the culled system nodes are missing. Adding them through an arbitrary operating system would likely not work at all, or cause the system to be unstable. Secondly, a more practical reason that another operating system would probably not function for the purpose of the thesis is that it increases the simulation time extraordinarily, taking the better part of an hour to let alone boot the system.

Most of the problems were able to be solved by different workarounds which seemed to not impact the behavior of the simulated system negatively. Being able to change frequency, even if not by intended means, show that the functionality is present in Simics. If some function or support to change frequency is added baseline then it would be easier to experiment with dynamic frequency scaling in systems.

5.3 Future work

Sustainability is a growing societal concern for many and sustainable computing is no exception. This section contains a sample of objectives that may be targeted in future efforts to develop the project even further.

5.3.1 Simplified configuration process

STEER, as presented by Jing Chen et al., required some amount of configuration to function properly, mainly in regards to defining the static and dynamic asymmetry of the system. The modifications introduced in this thesis added even more manual configuration, especially in regards to power model generation. Some static definitions are even needed in multiple locations. Should the user simply desire to use a default configuration of all clusters and cores, this could be largely automized using various utilities that retrieve kernel information.

5.3.2 Local versus global minimization of EDP

When defining the goal of what minimizing EDP implies, there are two approaches to the problem. Trying to solve the problem of minimizing on a local or global level. This thesis has focused on the local definition.

The local solution looks at each task individually and determines where each task should be executed by utilizing the lookup-table that was constructed in the prediction phase. This approach is the same as when STEER optimizes for energy. But unfortunately, this method can not guarantee that EDP is kept at a minimum the same way that it works for energy when considering the whole task set if there are dependencies between tasks. The global solution would therefore look at the entire task set and minimize the energy consumption by keeping most of the available resources working while at their most EDP efficient levels.

There are several approaches one can consider when trying to solve the problem of

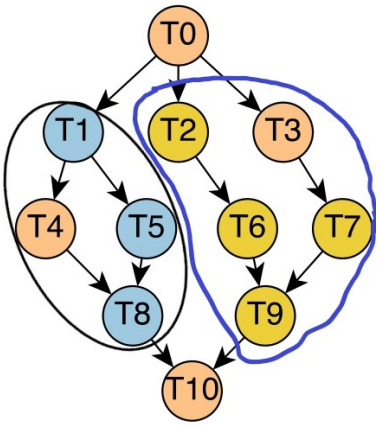


Figure 5.1: Graph of tasks with dependencies. Tasks in the black and blue sections have no dependencies between each other.

Image source: Jing Chen et al. [10]

minimizing the global EDP of a task set with inter-task dependencies. One approach that could be explored is to analyze the task graph and identify where some tasks can be scheduled independent from each other. Looking at figure 5.1, we can see that **T0** and **T10** are tasks that cannot be executed in parallel to any other tasks, due to every other task being dependent on **T0** to finish before they can start their execution and **T10** needing every other task to finish before it starts its execution. The black and blue sections of the graph are then completely independent from each other and can be compared EDP-wise to each other. There might be some tasks that can be rescheduled on another cluster or resource width to reduce the overall EDP of the section it resides in, and in some cases it might be possible to include tasks over the section borders as well. This approach might even be able to recursively work inwards, for example, in the blue section **T2** and **T6** could be split into their own section and the same for **T3** and **T7**. Unfortunately we were not able to do any quantitative testing concerning this approach due to time constraints.

5.3.3 Asserting generic functionality

STEER relies heavily on predictive models, namely a power model generated beforehand as well as an extrapolated performance model during runtime. This is great for keeping operational overhead at a minimum, but may also introduce inaccuracies that aren't addressed. Especially for new platforms, this may pose an issue. Accurately measuring execution time is trivial, but measuring the power consumption could be implemented as an optional mode where real-time measurements are used instead of the predictive power model. Being able to prove that the framework functions properly would reassure the user that energy (or EDP) will in fact be minimized.

5.3.4 Simulated setups utilizing Simics

As previously discussed in section 5.1.3, memory handling in Simics seems to be abstracted away to such a degree that benchmarks measuring the performance of memory operations work incorrectly. While looking through the Simics documentation and included lab and workshop material, we could not find a way to easily introduce more accurate memory handling. However there exists documentation on how to implement a physical memory while building a hardware model. This could mean that if a physical memory model is implemented in detail and a cache hierarchy is used, memory operations and corresponding benchmarks might be simulated more correctly.

5.4 Conclusion

The generated results indicate that the outset thesis objectives were successfully reached. STEER was modified, allowing the option to prioritize minimizing energy-delay product which meant that workloads could be balanced to increase energy efficiency while still acknowledging performance. With the modifications, STEER could successfully run on a greater variety of platforms.

Simics has a lot going for it as a tool while working with these kinds of architectural experiments. There are some shortcomings concerning functionality that were able to be worked around and could probably get added as a defined function by the developers. Either by re-introducing the system node files which controls CPU frequency on a physical machine and make the simulator react to changes in those files, or by adding a formal method into the simulation scripting language to facilitate frequency changes. A larger concern is the behavior concerning memory benchmarks in the simulation. From the documentation it seems that with a more detailed simulated model of the hardware there could be a possibility of getting the simulation to act more like its physical counterpart. Building a model was deemed outside of the scope of this thesis and as such was not able to be further explored.

In conclusion, potential future developments of STEER entail promising prospects for practical use, reducing the energy consumption of real-world applications. With future work, the framework may be generalized further, simplifying the process of running arbitrary applications through STEER, which could pave way for further adoption, promoting sustainability within the field of computer science.

Bibliography

[1] M. P. Mills, "The cloud begins with coal big data, big networks, big infrastructure, and big power an overview of the electricity used by the global digital ecosystem sponsored by: National mining association american coalition for clean coal electricity," 2013.

[2] P. J. Kuehn and M. Mashaly, "Dvfs-power management and performance engineering of data center server clusters," IEEE, Jan. 2019, pp. 91–98, ISBN: 978-3-903176-13-3. DOI: 10.23919/WONS.2019.8795470.

[3] M. Zahran, "Heterogeneous computing: Here to stay," *Communications of the ACM*, vol. 60, pp. 42–45, 3 Mar. 2017, ISSN: 15577317. DOI: 10.1145/3024918.

[4] L. T. Su, "Architecting the future through heterogeneous computing," vol. 56, 2013, pp. 8–11, ISBN: 9781467345132. DOI: 10.1109/ISSCC.2013.6487618.

[5] J. Chen, M. Manivannan, B. Goel, M. Abduljabbar, and M. Pericàs, "Steer: Asymmetry-aware energy efficient task scheduler for cluster-based multicore architectures," IEEE Computer Society, 2022, pp. 326–335, ISBN: 9781665451550. DOI: 10.1109/SBAC-PAD55451.2022.00043.

[6] M. Pericàs, "Elastic places: An adaptive resource manager for scalable and portable performance," *ACM Transactions on Architecture and Code Optimization*, vol. 15, 2 Apr. 2018, ISSN: 15443973. DOI: 10.1145/3185458.

[7] A. A. Suzen, B. Duman, and B. Sen, "Benchmark analysis of jetson tx2, jetson nano and raspberry pi using deep-cnn," IEEE, Jun. 2020, pp. 1–5, ISBN: 978-1-7281-9352-6. DOI: 10.1109/HORA49412.2020.9152915.

[8] E. Rotem, A. Yoaz, L. Rappoport, *et al.*, "Intel alder lake cpu architectures," *IEEE Micro*, vol. 42, pp. 13–19, 3 May 2022, ISSN: 0272-1732. DOI: 10.1109/MM.2022.3164338.

[9] *Dynamic power equation*, Accessed: 2023-05-31. [Online]. Available: https://www.intel.com/content/www/us/en/docs/programmable/683461/current/dynamic-power-equation.html.

[10] J. Chen, M. Manivannan, M. Abduljabbar, and M. Pericàs, "Erase: Energy efficient task mapping and resource management for work stealing runtimes," *ACM Transactions on Architecture and Code Optimization*, vol. 19, 2 Jun. 2022, ISSN: 15443973. DOI: 10.1145/3510422.

[11] B. Steinbach, *Recent Progress in the Boolean Domain*. Cambridge Scholars Publishing, 2014, ISBN: 9781443859677. [Online]. Available: http://ebookcentral.proquest.com/lib/chalmers/detail.action?docID=1683200.

[12] J. Löff, D. Griebler, G. Mencagli, *et al.*, "The nas parallel benchmarks for evaluating c++ parallel programming frameworks on shared-memory archi-

tectures," *Future Generation Computer Systems*, vol. 125, pp. 743–757, Dec. 2021, ISSN: 0167739X. DOI: 10.1016/j.future.2021.07.021.

[13] J. Engblom and D. Ekblom, "Simics: A commercially proven full-system simulation framework," 2006. [Online]. Available: https://www.researchgate.net/publication/228557661.

[14] J. Engblom, *The public release of intel simics (and more)*, Mar. 2022. [Online]. Available: https://community.intel.com/t5/Blogs/Products-and-Solutions/Software/The-Public-Release-of-Intel-Simics-and-More/post/1372402.

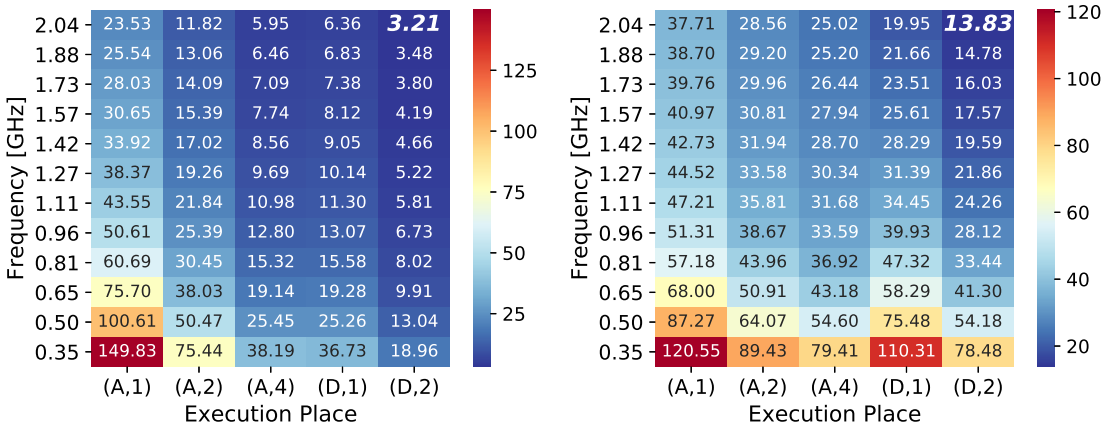
[15] *Intel integrated simulation infrastructure with modeling*, Accessed: 2023-01-30. [Online]. Available: https://www.intel.com/content/www/us/en/developer/tools/integrated-simulation-infrastructure.html.

[16] Intel, "Running average power limit energy reporting / cve-2020-8694 , cve-2020-8695 / intel-sa-00389," Nov. 2020. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/running-average-power-limit-energy-reporting.html.

[17] J. Engblom, *Using clear linux* for teaching virtual platforms*, Jun. 2019. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/using-clear-linux-for-teaching-virtual-platforms.html.

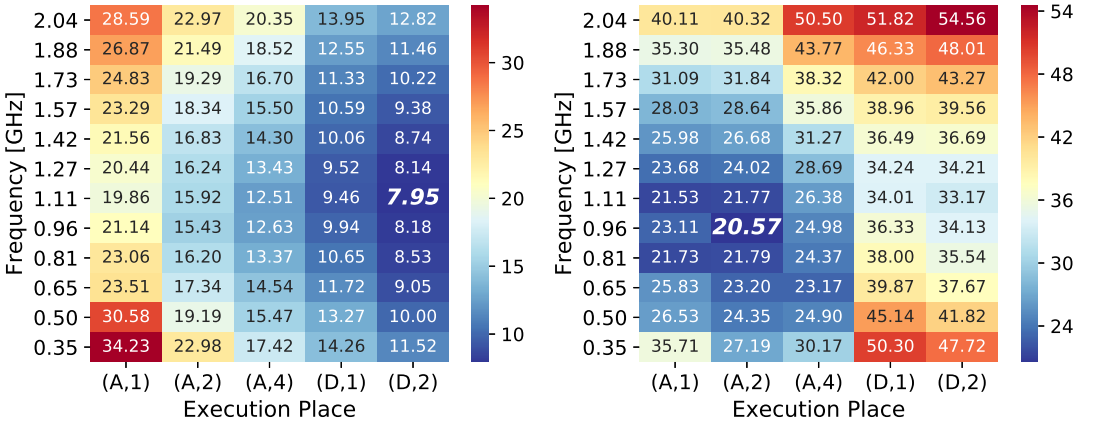
A

STEER benchmark data



(c) MM - Execution Time

(d) MC - Execution Time



(a) MM - Energy

(b) MC - Energy

Figure A.1: Motivational data for why static asymmetry, dynamic asymmetry, and task heterogeneity is important to consider, as presented by Jing Chen et al.

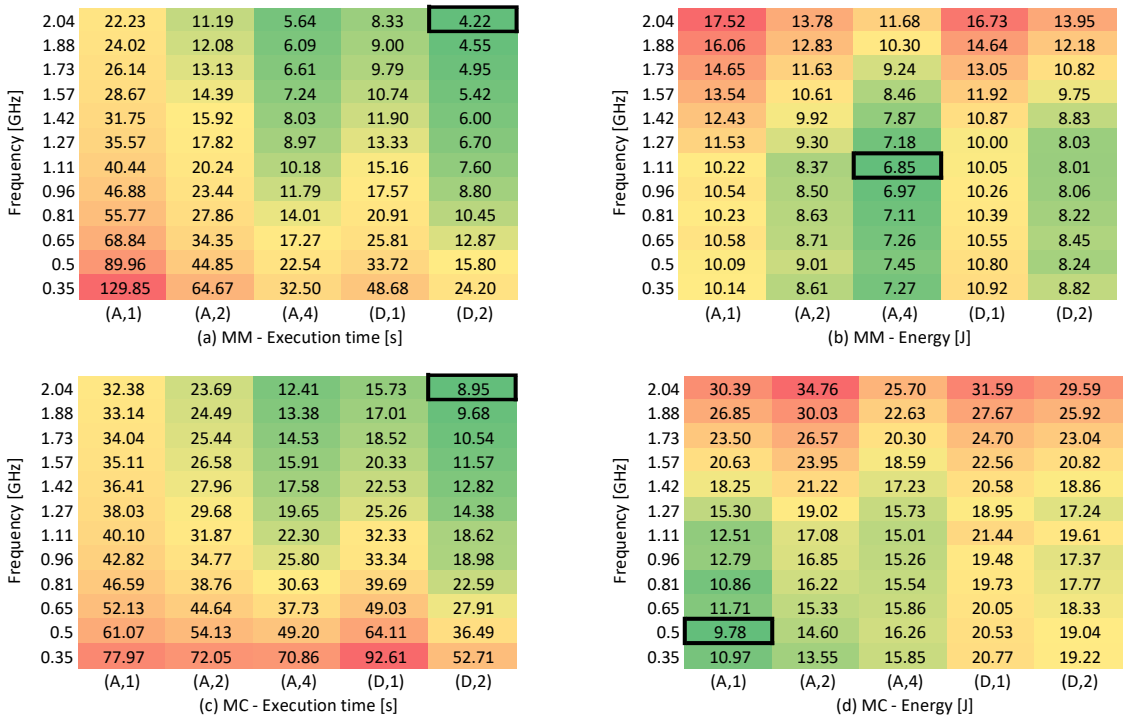


Figure A.2: New execution time and energy measurements on the Nvidia Jetson TX2 for comparison to the values presented by Jing Chen et al.

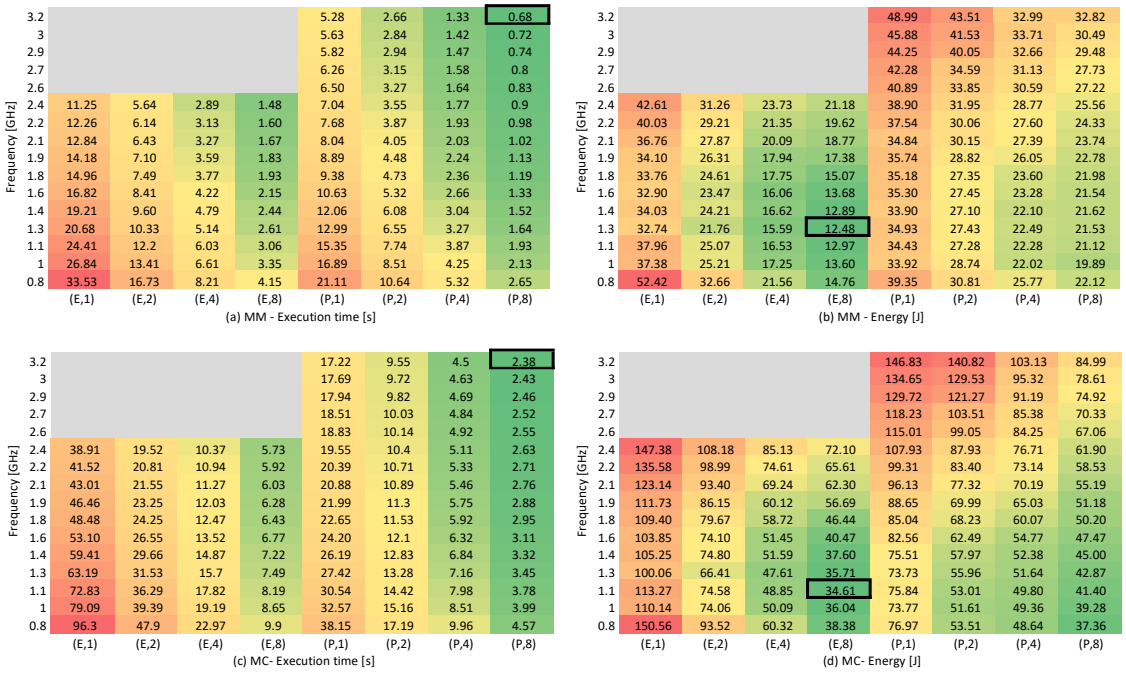


Figure A.3: Execution time and energy measurements on the Intel Alder Lake.

B

ISIM measurements

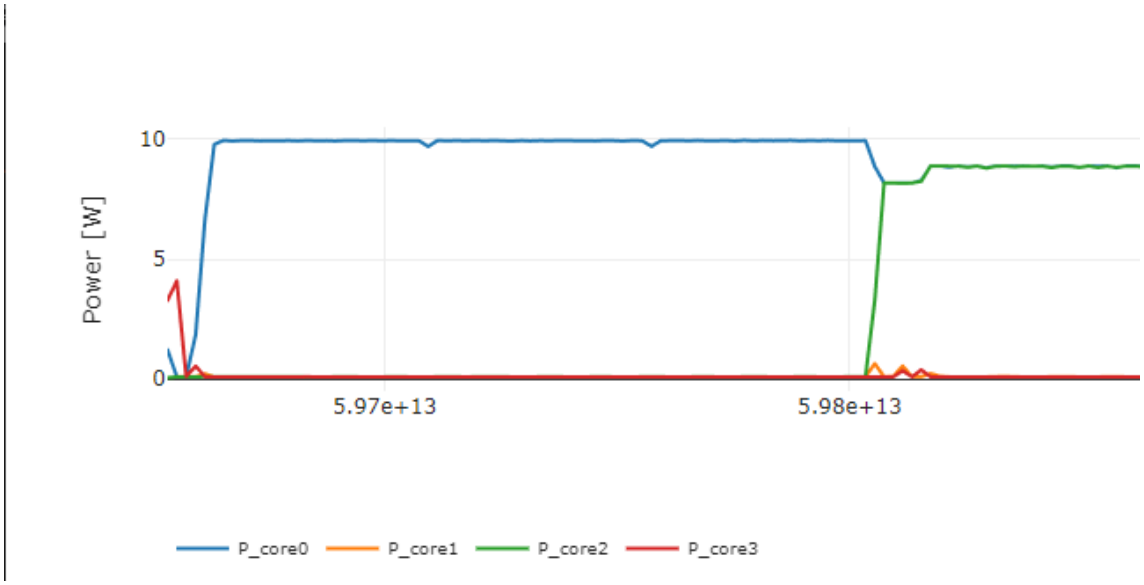


Figure B.1: Plot of power consumed by 2 cores running at 4 GHZ showcasing the startup period. P_core0 and P_core2 are overlapped once P_core2 starts working.