# Improving the Energy Efficiency of a Many-Node Heterogeneous Computing System Utilizing Application-Induced Energy Claims

Bachelorarbeit im Fach Informatik

vorgelegt von

**Maximilian Wagner**

geb. am 07. März 1990
in Ingolstadt

angefertigt am

**Lehrstuhl für Informatik 4**
**Verteilte Systeme und Betriebssysteme**

**Department Informatik**
**Friedrich-Alexander-Universität Erlangen-Nürnberg**

| | |
|---|---|
| Betreuer: | **Dipl.-Inf. Christopher Eibel**<br>**Dipl.-Inf (FH) Timo Hönig** |
| Betreuender Hochschullehrer: | **Prof. Dr.-Ing. habil. Wolfgang Schröder-Preikschat** |
| Beginn der Arbeit: | **12. Dezember 2016** |
| Abgabe der Arbeit: | **12. Mai 2017** |

## Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

## Declaration

I declare that the work is entirely my own and was produced with no assistance from third parties. I certify that the work has not been submitted in the same or any similar form for assessment to any other examining body and all references, direct and indirect, are indicated as such and have been cited accordingly.

(Maximilian Wagner)
Erlangen, 12. Mai 2017

# ABSTRACT

Current HPC systems, such as supercomputers, consume severe amounts of energy. Specialized hardware, such as GPUs, is already being included into HPC clusters. However, all of the nodes in an HPC cluster still generally feature hardware with similar performance and energy consumption. Therefore, the potential for improving the energy efficiency of HPC clusters by including low-power systems featuring processors designed for the mobile and embedded systems field was investigated in this thesis. Low-power processors, such as the ARM-Cortex-A53, are designed to be highly energy-efficient while still providing competitive computational power.

To evaluate the potential increase in energy efficiency, a small heterogeneous cluster featuring systems with both general-purpose and low-power processors was assembled. Initial tests on the cluster revealed that some workloads could be executed with increased energy efficiency on the low-power systems. To utilize the knowledge gathered in the initial set of tests, existing software designed for the scheduling and resource-management of HPC systems was modified to incorporate this knowledge into the process of node selection.

It was found that using the modified version instead of the default version reduced the cluster's energy consumption by at least 32 %, proposing that extending the heterogeneity present in HPC clusters by including low-power systems can increase the energy efficiency of HPC clusters.

# KURZFASSUNG

Aktuelle HPC-Systeme,wie beispielsweise Supercomputer, verbrauchen gravierende Mengen an Energie. Spezialisierte Hardware-Komponenten, wie beispielsweise GPUs, sind bereits Teil von HPC-Verbunden. Allerdings bestehen nach wie vor alle Systeme in einem HPC-Verbund aus Hardware mit vergleichbarer Leistung und ähnlichem Energieverbrauch.

Ziel dieser Arbeit ist daher die Untersuchung des Potentials, die Energieeffizienz eines HPC-Verbundes zu verbessern, indem Systeme mit geringem Strombedarf hinzugezogen werden, die Prozessoren aus dem Bereich der mobilen und eingebetteten Systeme aufweisen. Diese Prozessoren mit geringem Strombedarf (z. B. ARM Cortex-A53) wurden gezielt entworfen, um sowohl hohe Energieeffizienz als auch kompetitive Rechenleistung bereitzustellen.

Um die potentielle Verbesserung der Energieeffizienz zu evaluieren, wurde ein kleiner heterogener Verbund aus Systemen zusammengestellt, die sowohl Allzweck-Prozessoren als auch Prozessoren mit geringem Strombedarf beinhalten. Anfängliche Tests auf dem Verbund haben gezeigt, dass bestimmte Arbeitspakete mit besserer Energieeffizienz auf den Systemen mit geringem Strombedarf ausgeführt werden konnten. Um das in den anfänglichen Tests gesammelte Wissen verwenden zu können wurde bestehende HPC-Software, die verantwortlich für die Planung des zeitlichen Ablaufs sowie die Verwaltung der Ressourcen ist, so angepasst, dieses Wissen im Prozess der Knotenauswahl einzubeziehen.

Es wurde festgestellt, dass die Verwendung der modifizierten statt der ursprünglichen Software-Version den Energieverbrauch des Evaluationsverbundes um mindestens 32 % senken konnte. Dies impliziert, dass die Erweiterung der Heterogenität von HPC-Verbunden um Systeme mit geringem Stromverbrauch die Energieeffizienz dieser Verbunde verbessern kann.

# CONTENTS

# 1 INTRODUCTION

Supercomputers, also often referred to as high-performance computing (HPC) systems, are commonly used for large-scale simulations of all kinds, for example weather and earthquake prediction, or fluid- and aerodynamic simulation. Their main focus, as is already part of the term HPC, is performance. However, these systems also consume severe amounts of energy, and their power demand is continuously increasing at an extreme scale; for example the supercomputer Tianhe-1A (2010) demands 4.04 MW. Its successor, the Tianhe-2 (2013), already demands 17.8 MW [1]. This equals an increase of 145.21 % per year. In the light of these huge power demands, ways to improve the energy efficiency of HPC systems have to be found.

HPC clusters usually consist of a large number of systems, commonly referred to as nodes. These nodes are then split into partitions, with each partition being managed by their own dedicated control units. Some of these partitions are composed of systems featuring specialized hardware, for example graphics processing units (GPUs), but generally, all of a partition's nodes are systems with exactly the same hardware. This includes central processing unit (CPU), random-access memory (RAM), and other hardware features. Therefore, all of these nodes provide equal computational power per watt, and are optimized for performance.

Other fields of systems research, such as mobile and embedded systems, have already advanced further in the development of highly energy-efficient hardware. Their initial goal was to provide systems with sufficient computational power for the respective field of use with as little energy consumption as possible; for example the advanced reduced instruction set computing machine (ARM) processor family [2]. But recently, their focus has shifted towards increasing the computational power of these systems while at least maintaining or even further reducing their energy demand. The result of these newer efforts is the ARMv8 family of processors, such as the ARM Cortex-A53 [3]. Compared to previous ARM generations, new generations provide increased computational power while still shining with a low energy consumption; allowing microcontroller units (MCUs) featuring these processors to be introduced to novel fields of application. They are already established today in a lot of common large-scale computational grids, such as data-centers [4], reducing the grid's energy consumption when processing requests that are less time constrained. Investigations on whether using low-power processors, such as the ARM family, in HPC systems could improve energy efficiency have begun more recently, for example the Mont-Blanc project [5].

## 1.1 Motivation

General-purpose systems have been shown to not always be the most efficient execution environment for certain workloads with special attributes. Hence, the need for specialized hardware exists. Examples include graphics cards, such as the NVIDIA TITAN Xp [6], which feature GPUs with high

amounts of cores (e.g., the TITAN Xp features 3840 cores) compared to generic processors, and are therefore more efficient for workloads that benefit from extremely high parallelism, such as creation or manipulation of image data.

Modern clusters often need to provide ways of reducing their energy demand if runtime performance is less of a concern. The usual method of providing said behavior is utilizing operating systems (OSs) to manage local energy consumption of systems [7]. Possibilities include limiting core frequencies or shutting off currently idle cores completely [8]. However, generic systems still have a high power consumption per time unit compared to specialized low-power systems such as the previously mentioned MCUs featuring ARM processors, as shown in Section 4.2. Therefore, a heterogeneous cluster that features both generic and low power systems has the potential to be more energy efficient than an equivalent cluster without low-power systems.

However, managing heterogeneous clusters presents unique challenges regarding workload scheduling and target-node(s) selection. Therefore, existing scheduling and resource-management software need to also include information about runtime and energy differences between nodes to make correct decisions, as shown in Section 4.3. Hence, modifications of existing software tool chains for cluster management might be necessary, as investigated in Chapter 3.

In this thesis, the benefits and challenges of introducing heterogeneity to HPC are investigated by building a small heterogeneous cluster consisting of nodes with generic and low-power processors, employing as well as modifying existing HPC software, and evaluating energy and performance behavior of said cluster.

## 1.2 Overview of this Thesis

First, important fundamentals for the comprehension of this thesis, such as the basic setup and components of an HPC cluster, or limiting a processor's power consumption by enforcing power caps, and a selection of related work are presented in Chapter 2.

Next, the design and implementation of a modified version of the used resource manager Simple Linux Utility for Resource Management (SLURM) [9], which is better suited for the management of a heterogeneous cluster, is discussed in Chapter 3. This prototype version, named *Simple Linux Utility for Resource Management of Heterogeneous Clusters (SLURM-HC)*, aims to use predetermined information about a workload's energy-consumption and performance behavior, referred to as application-induced energy claims, depending on the node it was executed on to improve the cluster's overall energy efficiency.

Following this, the results of two sets of tests executed on a small heterogeneous evaluation cluster are presented and evaluated in Chapter 4. The evaluation setup presented in Section 4.1 features two generations of low-power development boards with ARM processors, and a general-purpose system with an Intel Xeon (Skylake architecture) server processor. The first set of tests was used to gather individual runtime and energy-consumption data for all of the investigated benchmarks being executed on each of the evaluated systems separately; the results are presented and evaluated in Section 4.2. Subsequently, the second set of tests was used to investigate whether SLURM-HC could utilize the data gathered in the first set of tests to increase the evaluation cluster's energy efficiency compared to SLURM. For this purpose, the results of executing sets consisting of multiple benchmarks in parallel on the evaluation cluster, either managed by SLURM or SLURM-HC, are presented and evaluated in Section 4.3.

Last, the observations presented in this thesis are summarized and topics of potential future work are presented in Chapter 5.

# 2 FUNDAMENTALS

This chapter provides an overview of basic concepts and terminology required for full comprehension of this thesis.

First, a basic HPC cluster and its necessary or helpful software components are described in Section 2.1. Second, different types of hardware potentially present in a heterogeneous cluster are classified and summarized in Section 2.2, concretizing the concept of heterogeneity. Next, common methods of measuring and evaluating the energy consumption of systems are detailed in Section 2.3, since measuring energy consumption is detrimental to evaluating the energy efficiency of a cluster. In Section 2.4, the method and benefits of limiting a system's maximum power consumption by enforcing power caps is discussed, which will become important once discussing modifications to existing software in Section 3.3, and three methods of enforcing power caps are detailed and compared. Last, previous scientific work that is related to the contents of this thesis is summarized and discussed in Section 2.5.

The basic layout of an HPC cluster and commonly used software components used in HPC environments are presented in the following section.

## 2.1 HPC Cluster Basics & Software Components

The basic layout of an HPC cluster is an interconnected grid of multiple cooperating systems, or nodes, bundled into partitions. Each of these partitions is then managed by dedicated control nodes, often referred to as masters; akin to the master–slave principle [10]. The masters are the connection between users and computational nodes, offering interfaces for job submission and management to the user, scheduling the submitted jobs, and selecting which nodes the jobs will be executed on. The masters also forward necessary information to their controlled slave nodes and return the results to the user. Figure 2.1 shows the basic layout of a master-slave HPC grid, including components, tasks and interactions.

Often, HPC clusters also include dedicated database systems for the provision of kernel and OS images to computational nodes, or storage and management of accounting information about jobs and users. These clusters require specialized software environments taking care of provisioning said images, scheduling jobs, and managing the available resources. The following sections describe these software components in detail.

### 2.1.1 The OpenHPC Project

The OpenHPC (OHPC) project [11] served as the entry point for the process of setting up the evaluation cluster, further discussed in Section 4.1. It is a Linux Foundation collaborative project,

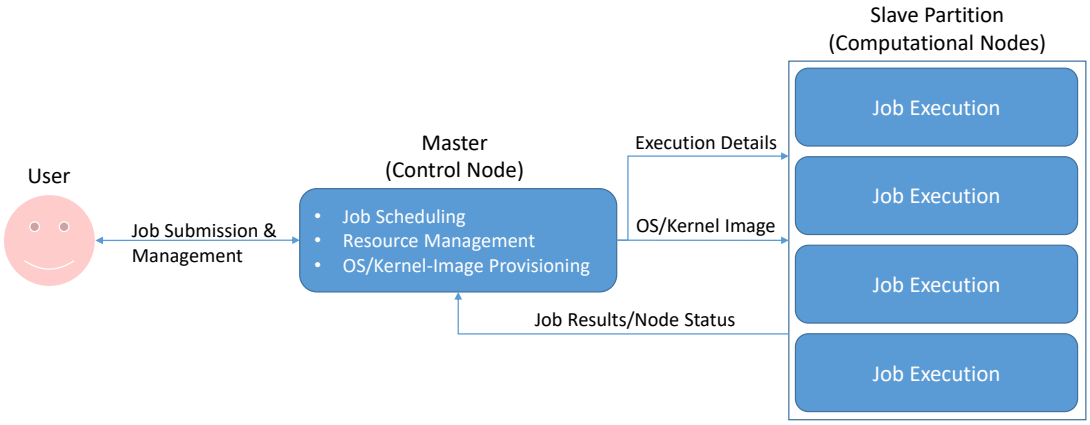


**Figure 2.1** – Components, tasks and interactions of a sample master–slave HPC grid

providing a reference collection of open-source HPC software components and full detailed install guides for Linux derivatives such as the Community Enterprise Operating System (CentOS) [12] and SUSE Linux Enterprise Server (SLES) [13]. The project's goal is to reduce the initial hurdle for administrators new to HPC wishing to set up a basic HPC cluster by providing a preconfigured software suite and a step by step install guide. The only software needed outside of the OHPC package is the respective OS. The project runs it's own repositories, providing updates and bug fixes for all components. As of November 2016, the OHPC software stack counts over 60 components [14], including versions of the OS-provisioning and scheduling/resource-management toolkits presented in the following subsections.

### 2.1.2 Operating-System–Provisioning Software

Provisioning software is the first highlighted component. It manages and distributes full images of kernel and OS to computational nodes. These bootable images are transfered to the nodes via traditional Ethernet networks. This allows for centralized management and modification of the node's full software suite without directly accessing the node or even requiring detailed knowledge of how to access it, simplifying administrative tasks like updating software and kernel drivers.

Optionally they may also provide a full virtual node file system (VNFS), allowing the nodes to operate without physical storage memory like hard disk drives (HDDs). Therefore the nodes can operate stateless, tremendously reducing the effort for restoring failed nodes after a crash.

To function correctly, provisioning software requires information about all nodes' Ethernet network interface names, Internet protocol (IP) and multimedia access control (MAC) addresses.

A sample toolkit is the WAREWULF operating-system–management toolkit [15]. Its features include, but are not limited to, stateless provisioning and monitoring of nodes, accessible via simple user interface calls. WAREWULF is part of the evaluation setup, which is fully described in Section 4.1.

### 2.1.3 Scheduling & Resource-Management Software

Scheduling and resource-management software is the second highlighted component. It acts as a gateway between the user and the actual computing nodes, usually running a control daemon on the master and worker daemons on all slave nodes.

The control daemon is responsible for scheduling jobs, managing available resources, and forwarding job execution requests to suitable nodes. It provides the user with an interface for job submission and distribution, allocating resources based on user requirements or default per-job values. If not enough resources are available at the point of submission, the job request will be queued and scheduled for a later time. Resources usually only include processor cores and memory, but can be extended to include less general-purpose hardware features like the availability of GPUs.

The worker daemons act similar to a remote shell environment: they wait for job-submission details from the control daemon, execute the specified workloads, and return the results. Also they frequently report their current state and allocated resources back to the control daemon, ensuring the data used by the controller to make scheduling and resource-allocation decisions is correct.

One example of this software is SLURM [9], an open-source cluster-management system for Linux clusters. It is part of our evaluation setup presented in Section 4.1, and the design and implementation of a modified version are presented in Chapter 3. Since the evaluation setup features hardware heterogeneity, existing hardware is classified into categories in the following section, presenting the individual attributes and specialties of different hardware types.

## 2.2 Hardware Classification

The term hardware heterogeneity implicitly states that existing hardware can strongly differ in both design and field of use. Hence, classifying existing hardware into categories called hardware types is beneficial, highlighting their attributes and specialties, and comparing their field of use. This allows for fixed terms to be used when discussing heterogeneous hardware. The following subsections clarify the meaning of the terms used in this thesis when referring to hardware types. In Table 2.1, the introduced hardware types and their attributes are summarized.

### 2.2.1 General-Purpose Hardware

General-purpose hardware systems are designed for the highest variety of tasks. They consist of hardware found commonly in private and enterprise environments. A sample configuration nowadays: a quad-core general-purpose processor with 64-bit support; 8 to 16 GiB of RAM; physical storage memory such as solid-state drives (SSDs); and either Ethernet or Wi-Fi network connection. General-purpose processors are usually designed following the complex instruction set computing

**Table 2.1** – Hardware types and attributes

| Hardware type | Attributes |
|---|---|
| General-purpose hardware | common, flexible, possibly less time-/energy-efficient than specialized hardware |
| Server processor | more cores with less power than general-purpose processors, multi-CPU support, ECC memory support |
| Graphics processing unit | high number of cores, highly parallel, usually dedicated to computer graphics tasks, sometimes additionally supports utilization for general-purpose tasks (e.g., CUDA [16]) |
| Low-power processor | low power consumption, RISC architecture, adequate performance, require adapted software |

(CISC) design principle, differentiating them from reduced instruction set computing (RISC) based low-power processors such as the ARM family.

The main advantage of general-purpose systems is their flexibility, being able to compute all kinds of workloads. However, this flexibility comes at the cost of being slower or less energy-efficient than specialized hardware for certain workloads. An argument favoring clusters composed of only general-purpose systems is the flexibility in scenarios where the nature of incoming workloads is unpredictable and diverse. However, a cluster of nodes with enough heterogeneous specialized hardware might be able to match the efficiency of a general-purpose–systems cluster in those scenarios, and surpass it in all others. The next section presents the general design concept of specialized hardware and three of the most common categories.

### 2.2.2 Specialized Hardware

Specialized hardware is the result of the search for a way to increase time and/or energy efficiency for a narrow field of use compared to general-purpose hardware.

The term hardware acceleration is used to describe one subvariant of these cases. It refers to specialized hardware performing certain tasks strictly faster than general-purpose hardware. This increased speed of execution is possible for example by exploiting greater concurrency, or by reducing the instruction overhead via provision of unique, specialized interfaces.

Three common categories of specialized hardware are presented in the following sections: server processors, GPUs, and low-power processors.

#### Server Processors

Server processors feature a higher amount of cores with less computational power per core compared to general-purpose CPUs. Their design is based on the need for high parallelism when processing common server-environment workloads. Different to standard desktop processors, server processors also often support setups with multiple CPUs per board, enabled by additional I/O links being part of their architecture. They also support unique features commonly present in server environments, for example error-correcting (ECC) memory.

Recent examples include the Intel Xeon E3-12xx v6 series [17], based on the Kaby Lake microarchitecture. Compared to its predecessor, the Skylake microarchitecture, Kaby Lake processors feature clock speeds increased by up to 300 MHz, and faster clock speed changes, reducing the time required for transitioning between, for example, the default clock speed and the higher clock speed realized by the Intel Turbo Boost technology. The Xeon E3-12xx v6 series of processors features thermal design powers (TDPs) of 72 to 73 W [17]. CPUs from different Xeon families are featured in five out of the current top ten supercomputers worldwide [1].

#### Graphics Processing Units

GPUs are processors specialized on creating and modifying image data rapidly. They have a huge number of cores, allowing for efficient processing of workloads with good parallel scaling. Traditionally, they are dedicated to computer-graphics–related tasks only. But recently the concept of so-called general-purpose GPUs (GPGPUs) has become increasingly common. This concept allows GPUs to be used by tasks not related to graphics calculations that also benefit from high parallelism, like computational fluid dynamics or large-scale numerical simulations [18]. They obtain speedups of few orders of magnitude compared to optimized multicore CPU implementations [18]. Hence, it is no surprise to see GPGPUs being part of multiple supercomputers in the top 10 worldwide [1].

One of the currently most powerful standalone graphics cards, the NVIDIA TITAN Xp [6], features 3840 cores with a frequency of 1582 MHz and 12 GB of video RAM (VRAM). It supports the NVIDIA CUDA [16] interface, allowing the use as a GPGPU.

#### Low-Power Processors

Low-power processors, such as all CPUs of the ARM family, are specialized on maximum power efficiency. These low-power processors are based on the concept of RISC, which is a CPU design strategy that aims to provide a simplified instruction set that is highly optimized for its architecture, resulting in a high performance per watt value. This allows the low-power processors to consume very little energy while still providing adequate performance for a lot of tasks. However, they face a problem: RISC-designed CPUs require adapted software, since they do not provide the full set of instructions as CISC-designed general-purpose CPUs would.

They are commonly used on cheap small development boards like the Hardkernel ODROID series, such as the ODROID-C2 [19], which features an ARM Cortex-A53 [3] (ARMv8 architecture) quad-core processor. This board and its direct predecessor, the ODROID-C1+ [20], are part of the evaluation setup detailed in Section 4.1. The evaluation setup also includes an external energy measurement device, which was used to record the nodes' energy consumption on-line. The following section details how energy consumption can either be recorded with measurements performed on-line, or estimated by creating energy profiles.

## 2.3 Energy-Measurement Approaches

Measuring the actual energy consumption of workloads executed on hardware can be done in multiple ways. Indirect approaches allow estimation of consumed energy by creating energy models that correlate expected or recorded resource usage with energy consumption data. Often, the resource usage caused by executing a workload is determined by evaluating hardware-provided internal performance counters. The direct approach is to perform measurements on-line over the full execution time of a workload with the help of an energy measurement device (e.g., a multimeter).

The following sections describe these approaches, their requirements, advantages and problems in more detail.

### 2.3.1 Energy Models

Energy models aim to estimate the energy consumption of a software component by utilizing knowledge about the code and its surrounding infrastructure to derive functions allowing calculation of estimated results. For example, the scenario of estimating a Java application's energy consumption running in a virtual machine (VM) was translated to the following formula [21]:

$$E_{component} = E_{computational} + E_{communication} + E_{infrastructure} \tag{2.1}$$

The formula combines estimates of the energy costs of computation, communication, and the surrounding infrastructure. $E_{computational}$ represents costs of execution of byte codes, native methods and thread synchronization if applicable. $E_{communication}$ represents costs of all occurring network communication, based on the size of received/transmitted data and the cost of single units of data. $E_{infrastructure}$ represents the cost of the surrounding OS, such as process scheduling or context switching, and the runtime platform (in this case the Java VM). The estimates of the energy costs are often created by evaluating hardware-provided performance counters, which track the hardware activity events that occur during the execution of a workload.

Hardware-provided performance counters allow estimating current and total power consumption of processes by utilizing internal performance counters and translating their values into power consumption. These performance counters are a set of special-purpose registers which store the counts of related hardware activity events; for example memory read/write instructions or amount of total used CPU cycles. By monitoring the use of system components during workload execution and correlating the resulting performance-counter values with the total power consumed by the system, energy models can be derived [22]. However, evaluating the relation between performance-counter values and energy consumption requires initial energy measurements.

Energy models provide the advantage of re-usability once created and verified, allowing for the energy consumption of code to be estimated without measuring the actual energy consumption of the code being executed. However, these models also have disadvantages: being estimations, they do not represent actual reliable data when faced with unpredicted behavior, such as OS background processes accessing the same hardware resources as the workload. Additionally, the models rely on the software being strictly deterministic, which might not be the case, for example if parallelism is involved. Portability is another problem: changing the execution environment, for example by modifying the hardware, requires overhauling the formulas to account for the new costs. Therefore, always recording the workload's actual energy consumption by utilizing an external measurement device can be desirable. The functionality, advantages and problems of these measurement devices are presented in the following subsection.

### 2.3.2 External Measurement Devices

With the help of external measurement devices, energy consumption data of the execution of workloads on any system can be recorded on-line, provided that the system is compatible with the measurement tool. There is a wide variety of tool variants available, ranging from simple multimeters to full-scale specialized current mirrors with live digital data transmission interfaces [23], for example a universal serial bus (USB) interface.

In case of the sampling frequency of measurements being high enough, and the to-be-measured energy values being within the device's measurement range, external devices are the most precise of all energy-measurement approaches [21], which is their main advantage. However, achieving precision for a large range of voltage and electric current is difficult. Therefore, most external tools have only narrow ranges of precision or even supported incoming voltage/current. Hence, knowledge about the expected values to be investigated is required to choose the correct tool a priori to actual measurements. Another argument against measurements being performed on-line is that, generally, they have to always be performed for every single execution. Also, they require isolated testing environments or a large amount of iterations to gain correct average values. Otherwise, outliers can distort the results, triggered by for example OS background processes. Despite these requirements, it was chosen to record the energy-consumption data presented in Chapter 4 by utilizing an external measurement device, ensuring the best possible accuracy for the results. The utilized device is presented in Section 4.1.3.

A tool that used to include energy modeling into its functionality is Intel's running average power limit (RAPL) [24] interface, which enforces power caps. Power capping and common methods of enforcing power caps, including RAPL, are presented in the following subsection.

## 2.4 Power Capping

Many current systems provide power-management mechanisms that allow dynamically scaling the amount of power consumed by the system. Therefore, the system's energy efficiency can be improved by only consuming as much power as required for the current tasks executed on it. Utilizing these power management mechanisms, limits for the maximum power consumed by the system, regardless of current system load, can be enforced. Enforcing these limits is commonly referred to as power capping.

An example of the benefits of power capping are data centers, as presented in [25]: Only in rare occasions, the entirety of systems present in data centers have to operate at maximum performance. However, the data center facility still must be able to support these peaks in power consumption. Therefore, the entire infrastructure, such as power supplies and cooling systems, often end up being overdesigned for the typical loads expected, wasting resources. With the help of power capping, the data center's peak power consumption can be limited, allowing for a downscaled infrastructure.

How power capping is realized depends on the system. Common methods include dynamic frequency scaling (DFS), dynamic voltage and frequency Scaling (DVFS), and Intel's RAPL [24] interface. These three methods are presented in the following subsections, starting with DFS.

### 2.4.1 Dynamic Frequency Scaling

DFS enables dynamic adjustments of a processor's clock speed, depending on the required performance. By reducing the clock speed, and therefore reducing the processor's performance, the processor's power consumption can be reduced. By enforcing upper limits for the clock speed, power caps can effectively be realized. However, the reduced power consumption does not scale linearly with the resulting decrease in performance, therefore often leading to increased energy consumption [25]. Hence, DFS is an inefficient way of reducing a system's power consumption. Additionally, a processor is not the only power consumer present in systems. Other system components also consume power, for example RAM. However, DFS only affects the processor, and therefore is not in all cases the most efficient way to limit a system's power consumption regarding the resulting loss of performance. Implementations of DFS include Intel's SpeedStep technology, commonly used in Intel's mobile processor line, and AMD's Cool'n'Quiet and PowerNow! technologies, used in AMD's desktop and server processor line or mobile processor line, respectively. Another approach to limiting frequency is DVFS, which is presented in the following subsection.

### 2.4.2 Dynamic Voltage-Frequency Scaling

Contrary to DFS, DVFS dynamically adjusts the voltage supplied to many hardware components present in a system, for example CPU and RAM, and subsequently their frequency. Voltage has a quadratic impact on a component's power consumption according to Ohm's law [26]:

$$P = C * V^2 * f \tag{2.2}$$

$P$ represents the power consumption, $C$ the component's capacity, $V$ the voltage, and $f$ the frequency. Hence, by reducing the voltage, more power is saved than if only the frequency is reduced, resulting in a better relation between saved power consumption and loss of performance. Hence, enforcing the same power cap with DVFS instead of DFS, a higher remaining performance is achieved, and subsequently better execution times. Therefore, DVFS represents a more energy-efficient way to adjust a system's power consumption. Initially, DVFS was only utilized to reduce energy consumption, but in the last years, enforcing power caps with DVFS was also researched [25]. For example, Gandhi

et al. investigated enforcing power caps by utilizing DFS, DVFS, and even a combination of both at the same time, in [27]. Another power-capping mechanism that combines DFS and DVFS is Intel's RAPL interface, which is presented in the following subsection.

### 2.4.3 Intel's Running Average Power Limit Interface

RAPL combines automatic DVFS and clock throttling to enforce user-defined power-consumption limits. DVFS is used to approximate the exact power limit, whereas clock throttling is used for further required fine-tuning to keep the system's power consumption as close to the limit as possible [25]. Therefore, RAPL enables finer granularity of power-consumption states than DVFS, while still benefiting from the improved energy efficiency that DVFS provides in comparison to DFS. Furthermore, RAPL is directly integrated into the processor, improving the speed of identifying load changes and adapting voltage/frequency limits to the new level of load [25]. Since RAPL was developed by Intel, it is only available on Intel's own family of processors. The hardware components managed by RAPL are distributed into three to four domains, which can each be managed separately [28]:

- the CPU's cores
- if present: the Intel HD graphics chip, integrated into the CPU
- the dynamic RAM (DRAM)
- the entire CPU (or CPUs in the case of multi-processor systems), including all cores and caches, and, if present, the graphics chip

Since part of the systems in the evaluation setup, described in Section 4.1, feature Intel's Xeon E3-1275 v5 [29] supporting RAPL, the power caps discussed in Chapter 3 were enforced with the help of RAPL. Power capping is the last fundamental concept introduced in this thesis, leaving only related work to be presented and discussed in Chapter 2, which is covered in the following section.

## 2.5 Related Work

In this section, a selection of previous scientific works related to the contents of this thesis are presented, discussing their relations to this thesis.

The first two presented publications offer further insight into the energy-measurement and power-capping approaches presented in this chapter. Noureddine et al. [21] highlight different energy-measurement approaches, as discussed in Section 2.3, in detail, investigate and compare examples for these approaches, and provide recommendations on how to efficiently measure energy consumption of devices and software. Petoumenos et al. [25] compare existing power-capping mechanisms, such as the approaches mentioned in Section 2.4, investigating which is the optimal mechanism for any given computing environment.

The publications presented next all discuss components that are part of the heterogeneous cluster evaluated in this thesis. Jensen et al. [30] discuss similarities and overlaps of design challenges for future exascale and embedded systems, highlighting the actual relation between two seemingly very different system-design approaches. It is related to this thesis because the evaluated heterogeneous cluster combines systems commonly used in HPC clusters and low-power boards that originated from the research on embedded systems. Blem et al. [31] compare RISC- and CISC-designed processors, investigating whether the used instruction set architecture (ISA) influences performance and energy-efficiency behavior of CPUs. The evaluation setup used in this thesis features systems with RISC- and

CISC-designed processors. Azimi et al. [32] propose a new method of managing power capping in server clusters, decentralizing the logic to reduce response delays and actuation latency by avoiding hierarchical power-management systems. They use many software components that are part of this project's evaluation environment as well, for example the resource manager SLURM, as presented in Section 4.1.

The following publications are part of the Mont-Blanc Project [5], which has the aim to design a new type of computer architecture capable of setting future global HPC standards, built from energy-efficient solutions used in embedded and mobile devices. It was incited by the findings of Rajovic et al. [33] in 2013. In another publication, Rajovic et al. [34] advocate building an HPC system from low-power embedded and mobile parts and evaluate a prototype consisting of ARM Cortex-A9 chips, investigating and estimating the potential energy savings brought by future iterations of ARM family processors. In contrast to their work, this thesis features a heterogeneous cluster combining general-purpose systems with systems featuring ARM processors. Weloli et al. [35] describe available tools and platforms for the exploration of using 64-bit ARM processors in many-node clusters and propose evaluation methodologies. The ODROID-C2 that is part of the cluster evaluated in this thesis features a 64-bit ARM processor as well.

## 2.6 Summary

This chapter provided an overview of important concepts, terms, and methodologies needed for the following chapters.

First, basic terminology and software components of HPC clusters were presented, allowing to understand the interactions and tasks happening on such clusters. Next, the concept of heterogeneous hardware was clarified by classifying existing hardware, presenting key design principles and differences between the types. Following that, two different approaches of obtaining energy-consumption data were highlighted, introducing profile-based energy models and external measurement devices, and specifying their advantages and problems. Next, the concept of power capping was presented, and three common methods of enforcing power caps were detailed and compared. Last, potentially interesting related work was referenced, providing locations of concepts and discussions on topics similar or related to the goals of this thesis.

The next chapter presents the design and implementation of modifications adapting existing scheduling and resource-management software to a heterogeneous cluster to improve the cluster's energy efficiency.

# 3 DESIGN & IMPLEMENTATION

This chapter describes the design and implementation of modifications adapting the scheduling and resource-management software SLURM to support a heterogeneous cluster. In its default version, SLURM does not consider the varying energy and performance between nodes included in a heterogeneous cluster. Therefore, a modified version was designed and implemented, that can consistently benefit from the selection of specialized hardware present in a heterogeneous cluster.

First, the basic functionality of SLURM is summarized in Section 3.1, allowing easier understanding of the next section. Following this, Section 3.2 presents the architecture of SLURM, detailing important components and commands, possibilities for configuration, and its present plug-in infrastructure. Knowledge about SLURM's default architecture is important to understand the implemented modifications necessary to adapt SLURM to heterogeneous clusters, as discussed in Section 3.3. This section comprises a discussion of the goals that were achieved with these modifications, how the modifications were integrated into the existing architecture, and how they were implemented in detail.

## 3.1 Basic SLURM Functionality

As presented in Section 2.1.3, scheduling and resource-management software is a vital part of every HPC cluster. One example of this software is SLURM [9]. Its basic functionality and tasks are summarized in this section.

SLURM is tasked with automatizing the selection of resources to be reserved for a submitted job, and when to execute the job. For this, the following information is required: the amount of resources available in total per node; the current status of resource usage; and the job's resource requirements and, if specified, deadline. This information can either be hard-coded into SLURM or supplied by the user(e.g., the cluster administrator). Based on this data, SLURM then decides which node or nodes will execute the job. If the required resources are currently available, SLURM reserves them, transmits necessary job information to the computational nodes, and returns the results to the user after the execution is complete. If not enough resources are currently available, SLURM saves the job's details and queues the job until enough resources are available. SLURM provides three key functions: allocation of access to the compute nodes' resources; provision of a framework for submission, execution and monitoring of work on the allocated nodes; and management of a queue of pending work to arbitrate the contention for resources.

The following section provides an overview of SLURM's architecture.

## 3.2 SLURM's Architecture

SLURM is an open-source, fault-tolerant, and highly scalable cluster-management and job-scheduling system for large and small Linux clusters [9], written in the C programming language.

It consists of three main components: a control and management daemon running on the master node; execution-handling daemons running on all slave nodes; and command binaries for interaction with the user or administrator. It also provides an optional database daemon which supports accounting tasks, but is not relevant for this thesis as discussed in Section 3.2.1. Figure 3.1 depicts SLURM's daemon components, a sample list of common user commands, and the relation between these daemons and commands.

SLURM's framework also supports an optional plug-in infrastructure, allowing for further customization of existing behavior or introduction of additional novel functionality. Examples include backfill scheduling, enforcement of resource limits by user account, or multi-factor–job-prioritization algorithms.

In the following sections, SLURM's important components and commands and its plug-in infrastructure are described in further detail, and the intended way to configure SLURM outside of providing/modifying actual code is presented.

### 3.2.1 Important Components & Commands

Among all available SLURM daemon components and commands, only three are utilized for this thesis: the controller daemon slurmctld; the compute node daemon slurmd; and the user command srun. Both daemons are required for the basic functionality of SLURM, and the command srun is the most direct way to submit jobs to the cluster. All other components and commands provide

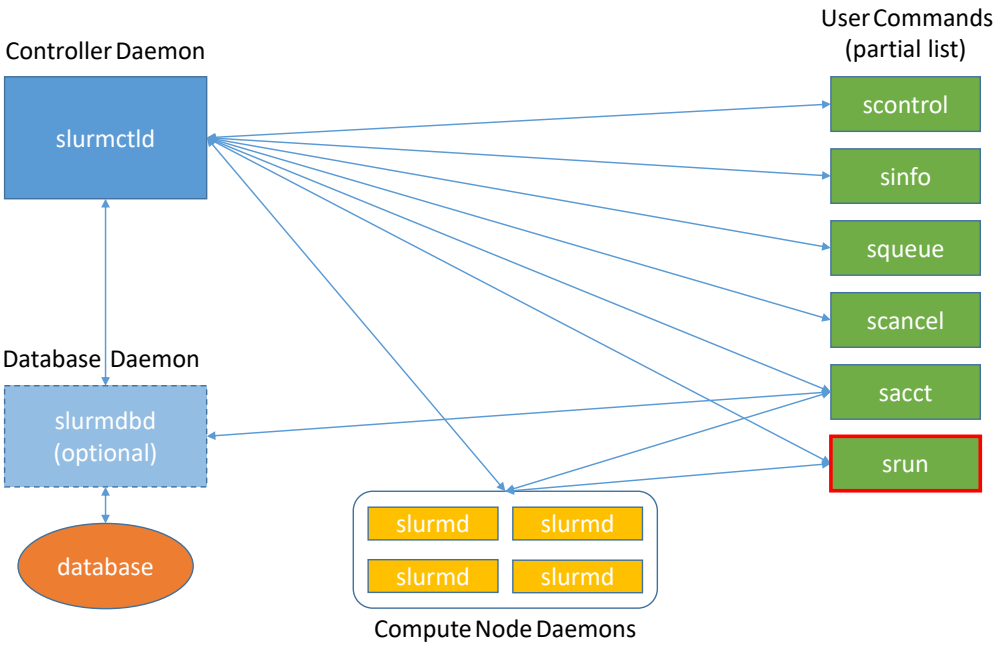


**Figure 3.1** – SLURM components and their relation [9]

additional functionality for managing and using the cluster that was not used in the adaptation of SLURM to heterogeneous clusters, and are therefore not further discussed. Why and how SLURM needs to be modified to fully take advantage of a heterogeneous cluster's specialized hardware is discussed in Section 3.3.

#### slurmctld

The controller daemon slurmctld is the centerpiece of SLURM, permanently running on the master node. It is responsible for monitoring all other SLURM daemons and cluster resources, accepting job submissions, and allocating resources to submitted jobs or scheduling them for later execution. All user commands interact in some way with slurmctld, as partially shown in Figure 3.1. Hence, SLURM cannot function without a running instance of slurmctld. To prevent temporary loss of functionality if slurmctld or the entire master node crashes, SLURM supports optionally running a backup instance of the controller daemon on a second master node. This can be activated by editing the configuration file mentioned in Section 3.2.3.

#### slurmd

The compute node daemon slurmd is the connection between the master node's control daemon slurmctld and its assigned partition of slave nodes. All on-line slave nodes need to run an instance of slurmd, or they cannot be provided with job information by the master node. Besides receiving job information and returning execution results, slurmd is also responsible for launching, monitoring, and, if requested, killing jobs on its respective compute node.

#### srun

The command srun allows the user to request resource allocation and job execution on the cluster managed by SLURM. Usage is as follows:

```
1  srun [options] executable <args>
```

A wide amount of options is available, allowing for e.g. specification of exact required resources, transmission of additional environment variables, or output redirection to a certain file path. A full list of options can be found in SLURM's official documentation for srun [36]. In the course of this thesis, only the option to export environment variables to the target node will be used. Its syntax looks like this:

```
1  --export=<environment variables>
```

#### Component Interaction

The roles of all important components and commands were described, but not how they interact with each other. Therefore, a sample execution of a user-submitted job now serves to explain the individual roles in further detail. It is visualized in Figure 3.2. First, the user has to submit the job via the command srun. The sample job binary to be executed is called `testjob`. It is assumed that one compute node is sufficient and will be completely allocated to the job. Parallel execution of one job on multiple nodes or of more than one job on a single node would unnecessarily complicate SLURM's behavior and is therefore not further investigated in the course of this thesis, albeit supported and common practice in the scope of HPC clusters. To highlight the process of exporting environment

variables by using the `export` option, a sample environment variable called `ev1` and its value of 1 is also submitted. This is the resulting call of srun:

```
1   srun --export="ev1=1" testjob
```

Upon submission, the srun binary parses the input for options, the job-binary name to be executed, and, if present, its parameters. The resulting information is saved into an internal data structure, which gets passed on to the control daemon slurmctld.

slurmctld then checks whether required resources have been specified. In this example, the user has not specified any resources as required. Therefore, slurmctld is free to select any of the currently available nodes. It is assumed that there is exactly one idle compute node available at the time of job submission, called `cnode1`. After the node selection and allocation of `cnode1` is complete, slurmctld transfers information about which node has been allocated back to srun. `Srun` then attempts to establish communication with `cnode1`'s instance of slurmd.

If communication was established successfully, srun passes all available job execution information to `cnode1`'s slurmd. In this example's case, the transfered information is the job-binary name `testjob` and the environment variable `env1=1`. Next, slurmd sets the transmitted environment variables locally at `cnode1`, and afterwards initiates execution of the binary `testjob`. Upon termination of `testjob`, whether successful or not, `cnode1`'s slurmd returns the results back to the control node's srun.

srun notifies slurmctld that the job's execution is finished, and slurmctld subsequently marks `cnode1` as free for future allocations. `Srun` then displays the returned results to the user either by console output, or, if configured that way, by writing to an output file. This marks the end of the sample execution.

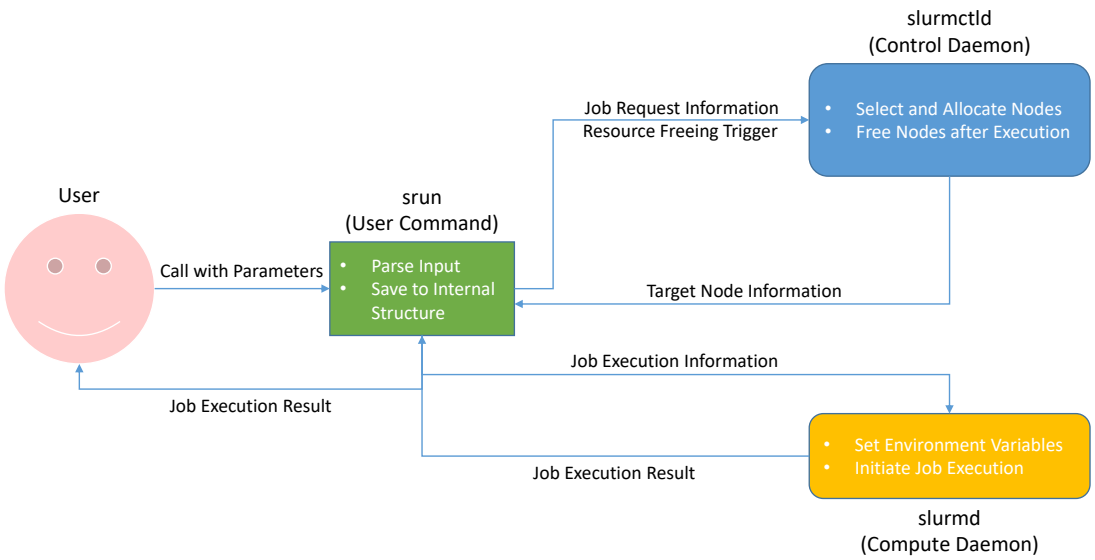


**Figure 3.2** – Sample job execution

### 3.2.2 Plug-In Infrastructure

With individual components and commands and their tasks and interactions discussed, the next major part of SLURM's architecture is the available plug-in infrastructure. Plug-ins are an easy way to extend or modify SLURM's functionality beyond the default. A SLURM plug-in is a dynamically linked code object which is loaded explicitly at run-time by the SLURM libraries [37].

Each plug-in has to select one of many available application programming interfaces (APIs) to implement, for example cryptography, network topology, or custom scheduling. Common C and SLURM-specific libraries are both available for use. Additionally, each API provides access to built-in methods tailored to the specific field of use.

Therefore, plug-ins represent a simple way to integrate custom code into SLURM without significantly modifying its full architecture. This is taken advantage of to implement the modifications to SLURM which are necessary to adapt it to heterogeneous clusters, as described in Section 3.3. Selecting which plug-ins are to be loaded is set in SLURM's configuration file, which is the subject of the next sub-section.

### 3.2.3 Configuration

Configuring SLURM is achieved by editing an external configuration file, usually called `slurm.conf`. `slurm.conf` is an ASCII file which describes general SLURM configuration information, the nodes to be managed, information about how these nodes are grouped into partitions, and various scheduling parameters associated with those partitions [38]. General configuration information includes which particular plug-ins are to be loaded and allows setting values for custom variables specific to certain plug-ins. The configuration file needs to be present on, and should be consistent across, all nodes in the cluster, including the master nodes.

Besides general setup like node and partition configuration, the field `SchedulerType` is important for this thesis. It determines which scheduler plug-in is to be used. The default plug-in is called `sched/builtin`. Therefore, the specific line in `slurm.conf` looks like this:

```
1 [...]
2 SchedulerType=sched/builtin
3 [...]
```

The `sched/builtin`–plug-in represents a first-in–first-out (FIFO) scheduling queue that additionally supports job priority sorting. It serves as the code base for most of the modifications applied to SLURM in this thesis. The following Section 3.3 presents and discusses those modifications.

## 3.3 Design & Implementation of SLURM-HC

Most scheduling and resource-management software, including SLURM, is conceptualized for clusters with partitions that consist of mostly equal systems. Usually, these systems only differ in small details like support for certain software or hardware features, or amount of RAM. Therefore, similar execution-times and energy-consumption values for any single workload are expected for all nodes bundled into a partition. However, a highly heterogeneous partition consists of systems whose execution times and energy consumption may drastically differ for any single workload, as shown in Chapter 4.

Existing node selection and scheduling algorithms do not incorporate drastically varying energy and speed performance between nodes for different jobs, and therefore can not consistently benefit

from the selection of specialized hardware in a heterogeneous many-node cluster. Thus, modifications to the node selection and scheduling behavior are likely to improve the energy efficiency.

The following subsections discuss which information needs to be taken into consideration, where the modifications were made, and a detailed presentation of what modifications were made. The resulting modified version of SLURM was named SLURM-HC, as in *Simple Linux Utility for the Resource-Management of Heterogeneous Clusters*.

### 3.3.1 Modification Goals

The goal of this thesis is to improve the energy efficiency of a many-node heterogeneous computational grid by utilizing application-induced energy claims. Considering these claims within SLURM's internal logic was achieved by modifying SLURM. The specific goals pursued with the implemented modifications are presented in this section.

#### 1st Modification Goal: Influence Node Selection

Heterogeneous clusters consist of nodes with hardware varying in processor speed, amount of RAM, and general processor design like the RISC or CISC architecture. Therefore, any given job will consume different total energy and time amounts based on which nodes are assigned to it. If a workload consisting of multiple jobs is distributed to nodes without knowledge of expected execution time and energy consumption per job, the resulting total consumed time and energy values are unlikely to be optimal. Therefore, acquiring said knowledge and incorporating it to the process of node selection is the first goal of modifying SLURM for heterogeneous clusters.

**Example for the Impact of Correct Node Selection:** To showcase the impact of correct node selection, the following simple scenario is assumed: A workload consisting of two different jobs is submitted, called $j_1$ and $j_2$. The partition assigned to the execution of the workload consists of only two nodes. One system that features a general-purpose processor is called gpn and the other system, which features a low-power processor, is called lpn. The processor terminology refers to the classifications introduced in Section 2.2. Since the low-power processor has a reduced clock speed compared to the general-purpose processor, it is likely to take longer for the execution of either $j_1$ or $j_2$, but consume less energy per second. Previous execution of both jobs on single nodes resulted in the energy and time values shown in Table 3.1.

**Table 3.1** – Execution-time and energy-consumption values of two sample jobs on different node types

| Node Type | Job | Execution Time [s] | Energy Value [J] | Idle Consumption [W] |
|---|---|---|---|---|
| General-purpose | $j_1$ | 2 | 20 | 5 |
| General-purpose | $j_2$ | 1 | 30 | 5 |
| Low-power | $j_1$ | 2 | 10 | 1 |
| Low-power | $j_2$ | 3 | 15 | 1 |

**Table 3.2** – Total energy consumption based on node allocation with values from Table 3.1

| Job Assigned to gpn | Job Assigned to lpn | Total Energy (idle) | Total Energy (load) |
|---|---|---|---|
| $j_1$ | $j_2$ | 40 J | 35 J |
| $j_2$ | $j_1$ | 45 J | 40 J |

Two possibilities exist for allocating the submitted jobs to the available nodes, assuming both nodes will be allocated: either $j_1$ or $j_2$ will be executed on gpn, and the remaining job on lpn. The cluster's resulting total energy consumption $E_{total}$ is calculated with the following formula:

$$E_{total} = E_{j_1} + E_{j_2} + |t_{j_1} - t_{j_2}| * P_{idle_{j_{1/2}}} \tag{3.1}$$

$E_{j_{\{1,2\}}}$ is the energy consumed by the allocated node during the execution of $j_{\{1,2\}}$, and $t_{j_{\{1,2\}}}$ the time needed for the execution. $P_{idle_{j_{1/2}}}$ is the idle consumption of whichever node finished first and idly waits for the other node to complete execution. The time it has to wait for is the timely distance between $t_{j_1}$ and $t_{j_2}$.

Assuming the previously recorded values will reoccur, allocating $j_1$ to gpn results in 40 J total energy consumption, whereas the other option results in 45 J total energy consumption, as seen in Table 3.2. Hence, correct allocation saves 5 J in this example. Even under the assumption that nodes in an HPC cluster should never be idle, and thus disregarding idle consumption in Equation (3.1), Table 3.2 shows that 5 J can again be saved by correct allocation. Therefore, correct allocation can improve the energy efficiency of a heterogeneous cluster.

#### 2nd Modification Goal: Adapt Job Configuration

Another possibility for saving energy based on previous knowledge is correctly adapting the jobs to their allocated node's specification. An example: Assuming a job which supports multi-threading, by manipulating the number of threads the job will be executed with, based on the allocated node's hardware specifications, may again improve performance or energy efficiency, as shown in Chapter 4. This can, in some cases, be achieved by manipulating environment variables on the target nodes. Since the user has no knowledge which nodes will be assigned to his job submission unless he specifies them, he cannot set the optimal number of threads himself. Hence, adding environment variable manipulation to the node-selection logic can be beneficial to improving the energy efficiency of a heterogeneous many-node cluster as well.

#### 3rd Modification Goal: Introduce Power Caps

The final option to adapt scheduling and resource-management software to heterogeneous clusters is enforcing custom power caps, introduced in Section 2.4, on the nodes based on the submitted job. Power caps allow the limitation of the maximum power consumed by a node's CPU, limiting CPU clock speed. This affects both execution time and energy consumption of a job, as shown in Chapter 4. Even though execution time is generally increased by enforcing power caps, the resulting total energy consumed can often be reduced, as shown in [23]. This is known as the "race-to-sleep vs. crawl-to-sleep" debate [23]. But the optimal value for power caps can differ for each job, as shown in Chapter 4 as well. Hence, pre-determining the exact optimal values for power caps for all combinations of known jobs and nodes and enforcing them after node selection is potentially beneficial to the goal of improving energy efficiency of a heterogeneous many-node cluster.

To conclude, pre-recording performance values and setup specifics, such as number of threads or power caps, for jobs on a per-node base, and incorporating this knowledge into the node-selection logic of scheduling and resource-management software shows promising results. Therefore, adapting existing software to the task of managing heterogeneous clusters is likely to increase the cluster's energy efficiency.

### 3.3.2 Integration into SLURM

With three goals (i.e., node selection, configuration of thread numbers, enforcement of power caps) for modifications introduced, the next task is to find the right spot within the existing software architecture of SLURM to implement these modifications. The modifications require alteration of existing behavior and introduction of novel logic. Therefore, editing or expanding the code base is necessary. As presented in Section 3.2, three components are available: slurmctld, slurmd, and srun. Since most of the modifications alter the process of node selection, which happens on the master node, slurmd is automatically excluded because it runs on the slave nodes. SLURM's plug-in infrastructure, introduced in Section 3.2, represents the most simple way to alter existing and introduce new behavior. However, since srun is compiled as a separate stand-alone binary, plug-ins do not directly affect it. Hence, slurmctld remains as the component to be modified.

Since it was determined that slurmctld will be modified, the appropriate plug-in API has to be found. The modifications need to alter node selection, and therefore presumably have to happen before or during that process. However, SLURM's node selection is a rather complex algorithm and is therefore hard to alter without causing unexpected behavior. Hence, the modifications should, if possible, happen before node selection is actually started. But all plug-in APIs supported by SLURM will only be triggered after node selection is complete. Therefore, the only possible spot left for the proposed modifications is `directly` after node selection, by implementing the modifications and re-initiating the process of node selection. The next-in-line plug-in API is the scheduler. Two scheduler plug-ins are part of default SLURM, `sched/builtin` and `sched/backfill`. With the former being less complex, and neither being suitable for heterogeneous clusters, an altered version of the existing `sched/builtin` plug-in has been created.

However, not all modifications can be implemented in the altered scheduler plug-in. Information about the environment variables to be exported to the compute nodes is only available within srun and cannot be modified directly from the outside. Therefore, srun needs to be slightly modified as well, even though the plug-in cannot directly affect it. The detailed implementation of the modifications in both the scheduler plug-in and srun is discussed in the next section.

### 3.3.3 Detailed Implementation

Modifications are implemented in two distinct parts of SLURM's architecture. The resulting modified version was named SLURM-HC. An altered version of the default scheduling plug-in `sched/builtin` modifies the node-selection process and manages power caps. It is called `sched/slurm_hc`, following the introduced name of the modified SLURM version. The other part is a slightly modified version of srun, called `srun_hc`, which modifies the to be exported environment variables to adjust the number of threads a job will be executed with.

#### API Modifications

First, the API and important parts of `sched/builtin`'s code structure are presented. The plug-in consists of three separate code files: `builtin.c` contains all methods and therefore the main logic

of the scheduling plug-in; `builtin.h` is the respective header file to be included in other C files wishing to interact with the plug-in; and `builtin_wrapper.c`, which implements the mandatory interface, connecting method calls of external code with the correct internal methods implemented in `builtin.c`. The only interface method that was modified is the following:

```
1  int
2  slurm_sched_p_newalloc( struct job_record *job_ptr )
3  {
4    return SLURM_SUCCESS;
5  }
```

It is called whenever a new allocation of resources for a job is completed by the node-selection process. The struct `job_record` contains all internally available information regarding the job that was allocated, including the name of the job and which nodes where allocated to it. This is exactly the information which needs to be modified before the node selection is manually re-initiated. Therefore, the modified version of this interface method, now implemented in `slurm_hc_wrapper.c`, redirects the code sequence to a new method within `slurm_hc.c`:

```
1  int
2  slurm_sched_p_newalloc( struct job_record *job_ptr )
3  {
4    return slurm_hc_newalloc(job_ptr);
5  }
```

This new method, named `slurm_hc_newalloc`, contains the logic which implements the proposed modifications.

#### Scheduling Plug-In Modifications

Next, the logic implemented in `slurm_hc_newalloc` is detailed. To function correctly, the modified method requires knowledge about detailed data recorded during previous executions of jobs on nodes: specifically, a combination of node name, job name, enforced power caps, and resulting execution time and energy consumption. This information could for example be stored in the form of lists sorted by the optimization criteria. Based on this information, the optimal available node for the submitted job will be allocated and, if supported, the power cap enforced on the respective node. “Optimal” in this case refers to whichever node consumed the lowest amount of energy. Selecting the optimal node based on different criteria, for example execution time or energy–delay product (EDP), introduced in Section 4.2, has not been implemented. This leaves potential for future work, as discussed in Chapter 5. The implemented logic is detailed in Algorithm 3.1. Therefore, two key functionalities are implemented in `slurm_hc_newalloc`: overwriting the default node selection to ensure the optimal available node is selected; and configuring both the workload and the node’s execution environment to ensure the best energy efficiency.

Besides altering the interface method and introducing the new modification method (and subsequent helper methods), no further changes to the existing `sched/builtin` plug-in are made in `sched/slurm_hc`, except for the renaming of methods and variables where needed. Therefore, the full scheduling logic implemented in `sched/builtin` remains unchanged.

#### srun Modifications

As mentioned previously, modifying the number of threads by accessing the environment variables to be exported is not possible from within `slurm_hc_newalloc`. Therefore, srun needs to be slightly

**Require:** Details of the submitted job: $I_{job}$, pre-recorded execution information: $I_{exec}$
Store name of previously allocated node in $N_{alloc}$
Find names of all currently available nodes and store in $N_{av}$
Iterate over $I_{exec}$ by node name $N_i$ to find optimal node $N_{opt}$
**if** $N_i \in N_{av}$ **then**
    $N_{opt} \leftarrow N_i$
    Break iteration
**else**
    Continue iteration
**end if**
End of iteration process
**if** $\nexists N_{opt}$ **then**
    Exit
**end if**
**if** job supports multi-threading **then**
    Store number of threads in $I_{job}$
**end if**
**if** $N_{opt}$ supports power capping **then**
    Initiate power cap enforcement on $N_{opt}$
**end if**
**if** $N_{opt} != N_{alloc}$ **then**
    Free previous allocation of $N_{alloc}$
    Update $I_{job}$ to request node $N_{opt}$
    Re-initiate node-selection process with updated $I_{job}$
**end if**

**Algorithm 3.1** – Logic implemented in the method `slurm_hc_newalloc`

modified as well. Part of default srun's internal logic is depicted in Algorithm 3.2. The modified version `srun_hc` features an extended logic, now including modification of environment variables after setting up the job's environment. Prerequisite for this is slurmctld providing the necessary information. The extended logic featured in `srun_hc` is depicted in Algorithm 3.3.

Concluding this section, all modification goals mentioned in Section 3.3.1 were able to be implemented without significant changes to SLURM. Node selection and power-cap enforcement is handled in a modified version of the shipped scheduler plug-in `sched/builtin`. It is called `sched/slurm_hc`. Environment-variable manipulation is incorporated into an extended version of srun, called `srun_hc`. This extended version of srun receives and utilizes information processed in and forwarded by `sched/slurm_hc`.

Parse user input, including environment variables to be exported
Transmit job information to slurmctld
Wait for node allocation results returned by slurmctld
Set up environment for job: $E_{job}$
Communicate with allocated node

**Algorithm 3.2** – partial srun default behavior

1: Parse user input, including environment variables to be exported
2: Transmit job information to slurmctld
3: Wait for node allocation results returned by slurmctld
4: Set up environment for job: $E_{job}$
5: Look for environment variable values transmitted by slurmctld
6: **if** environment variable values are found **then**
7: Add to or overwrite existing environment values in $E_{job}$
8: **end if**
9: Communicate with allocated node

**Algorithm 3.3** – partial `srun_hc` extended behavior

## 3.4 Summary

This chapter presented the design and implementation of modifications beneficial to adapting existing scheduling and resource-management to the specialized hardware used in heterogeneous clusters.

First, the basic functionality of SLURM was summarized in Section 3.1. Following this, Section 3.2 presented the architecture of SLURM, detailing important components and commands, possibilities for configuration, and its present plug-in infrastructure. Last, modifications necessary to adapt SLURM to heterogeneous clusters were discussed in Section 3.3. This section included the goals that were achieved by implementing modifications, how the modifications were integrated into the existing architecture, and how they were implemented in detail. The resulting modified version of SLURM was named SLURM-HC.

In the next chapter, the potential for improving the energy efficiency of a heterogeneous many-node cluster by modifying existing scheduling and resource-management software is investigated by evaluating the results of common benchmarks executed on a small heterogeneous cluster with both SLURM and SLURM-HC.

# 4 EVALUATION

In this chapter, the results of tests are presented and evaluated, with the goal of investigating the effect of utilizing the modifications implemented in SLURM-HC on the energy efficiency of a heterogeneous cluster.

First, the used evaluation setup, including hardware and software components, is presented in Section 4.1, detailing hardware specifications, benchmark selection and how time and energy consumption was recorded. Presenting the setup in detail allows reconstructing the testing environment if the results are to be reproduced or extended in future work. Next, the results of the initial set of tests are presented in Section 4.2. The goal of the initial set of tests was to find the most energy-efficient configuration and node for each of the investigated benchmarks by measuring all possible combinations. Based on the results, the modifications detailed in Chapter 3 were designed. Last, the impact on energy efficiency of the modifications implemented in SLURM-HC is investigated by comparing and evaluating the results of executing multiple benchmarks in parallel on the evaluation cluster managed by either SLURM or SLURM-HC, presented in Section 4.3.

## 4.1 Evaluation Setup

The setup used for evaluating the difference in energy efficiency between SLURM and SLURM-HC, as detailed in Chapter 3, consists of three key components: a heterogeneous cluster, composed of various employed hardware and software, serving as the execution environment for the evaluation; benchmarks that allow comparing the performance of the cluster nodes they are executed on; and an external energy measurement device used to gather on-line energy consumption values of the executing nodes. In the following subsections, the employed components are presented and detailed, starting with the cluster itself.

### 4.1.1 Heterogeneous Cluster Components & Setup

The evaluation cluster consists of four distinct nodes: Two identical general-purpose systems featuring current Intel Xeon quad-core server processors, and two different generations of the same small development board featuring low-power ARM processors, the ODROID-C1+ [20] and the ODROID-C2 [19]. The hardware specifications are summarized in Table 4.1.

#### General-Purpose Systems

The general-purpose systems both feature the following hardware: an Intel Xeon E3-1275 v5 (Skylake architecture) server processor [29], with four cores clocked at 3.6 GHz base frequency and

**Table 4.1** – Hardware specifications and OSs of evaluation-cluster nodes

| Node | CPU | RAM | Storage | OS |
|---|---|---|---|---|
| ODROID-C1+ | ARM Cortex-A5 4x1.5 GHz | 1 GiB DDR3 792 MHz | 16 GiB MicroSDHC | Ubuntu 14.04.5 |
| ODROID-C2 | ARM Cortex-A53 4x1.5 GHz | 2 GiB DDR3 912 MHz | 16 GiB MicroSDHC | Ubuntu 16.04.1 LTS |
| Xeon (master) | Intel Xeon E3-1275 v5 4x3.6 GHz | 16 GiB DDR4 2,133 MHz | 256 GiB SSD | CentOS Linux 7.2.1511 |
| Xeon (slave) | Intel Xeon E3-1275 v5 4x3.6 GHz | 16 GiB DDR4 2,133 MHz | none (stateless) | CentOS Linux 7.2.1511 (image provisioned) |

up to 4.0 GHz in turbo mode, 8 MiB cache, support for up to eight threads due to hyper-threading, and 64-bit computation; 16 GiB of DDR4-RAM with 2,133 MHz clock frequency; and a 500 W power supply. Additionally, one of them, which was configured as the master node, is equipped with a 256 GiB SSD used as physical storage for the installed OS and other necessary software. The other system is operating stateless by utilizing provisioning software, see Section 2.1.2, and therefore needs no physical storage, reducing its energy consumption. Both are operated by the same CentOS Linux 7 (release 7.2.1511) distribution for the x86-64 architecture, including the most recent Linux kernel (Linux 3.10.0-514.16.1.el7.x86_64).

#### Low-Power Boards

Two different generations of the ODROID-C series of low-power development boards were used. The older version is the ODROID-C1+, released in July 2015. It features: an AmLogic S805 ARM Cortex-A5 [39] (ARMv7-A architecture) quad core low-power processor with 1.5 GHz clock frequency and only 32-bit support; 1 GiB DDR3 SDRAM with 792 MHz clock frequency; and physical storage memory in the form of a 16 GiB MicroSDHC flash card plugged into a UHS-1 SDR50 slot. The installed OS is Ubuntu 14.04.5 with the Linux 3.10.96-151 ARMv7l kernel, the official version supplied by the board manufacturer.

The ODROID-C2 represents the succeeding generation of ODROID boards, released in spring 2016. It features an Amlogic S905 ARM Cortex-A53 [3] (ARMv8-A architecture) quad core low-power processor with 1.5 GHz clock frequency and 32/64-bit support; 2 GiB DDR3 SDRAM with 912 MHz clock frequency; and physical storage memory in the form of a 16 GiB SDHC flash card – the same model as used for the ODROID-C1+ – plugged into a UHS-1 SDR50 slot. The employed OS again is the official version supplied by the board manufacturer: Ubuntu 16.04.1 LTS with the matching Linux 3.14 LTS kernel.

#### Cluster Setup and Software Components

The presented systems were connected with Gigabit Ethernet to form a heterogeneous cluster. One of the Xeon systems was configured to act as the master node, whereas the remaining Xeon and both ODROID systems served as slave nodes. The same version of scheduling and resource-management software was installed on all nodes: SLURM 15.08.7. The installed version of SLURM-HC, the

modified SLURM version presented in Chapter 3, is based on SLURM 15.08.7 as well. In addition, an open-source implementation of the Message Passing Interface (MPI), Open MPI v1.10.2 [40], was installed on all nodes, with the distributions matching the specific installed OSs. MPI is required for the execution of the benchmarks detailed in Section 4.1.2. Furthermore, the WAREWULF [15] v3.6 toolkit was installed on the master node to enable OS-image provisioning, as described in Section 2.1.2. This allowed the slave Xeon system to operate without physical storage memory, reducing its energy consumption. No other software toolkits were installed on any of the cluster nodes. In the next section, the selection of benchmarks used to compare performance across the cluster nodes is presented.

### 4.1.2 Selected Benchmarks

Benchmarks allow the comparison of the performance of systems with different hardware specifications and/or software configurations. To evaluate the performance of highly parallel clusters, such as HPC clusters, benchmarks need to mimic the computation and data movement characteristics of workloads commonly submitted to such clusters. One group of benchmarks designed specifically for the evaluation of highly parallel systems are the NAS parallel benchmarks (NPB) [41]. They are derived from computational fluid dynamics (CFD) applications [42], a typical field of use for many-node computational grids. Originally, the NPB suite consisted of eight different benchmarks: five kernels and three pseudo-applications. The kernels are called `IS`, `EP`, `CG`, `MG`, and `FT`; and the pseudo-applications `BT`, `SP`, and `LU`. Full documentation on represented algorithms and signature behavior is available in [41] or at [42]. The initial suite has been extended to include multi-zone versions of `BT`, `SP` and `LU`, consistently called `BT-MZ`, `SP-MZ`, and `LU-MZ`. These multi-zone versions support multiple levels of parallelism by dividing the problems into zones distributed to a number of outer threads, which then redistribute them to their own set of inner threads.

All eleven benchmarks can be adjusted by two parameters: number of threads and problem size. In the case of `BT-MZ`, `SP-MZ` and `LU-MZ`, the number of inner and outer threads can be configured separately. An implementation of the NPB using the OpenMP programming model [43], which is supported by Open MPI, is available. Using the OpenMP version of the NPB, configuring thread numbers is achieved by setting values for the environment variables `OMP_NUM_THREADS` (general/outer threads) and `OMP_NUM_THREADS2` (MZ-only: inner threads).

The benchmark problem sizes are predefined and referred to as classes S, W, and A to F [42]. The letters represent the following sizes:

- Class S: a small size designed for quick tests
- Class W: a slightly larger size designed for workstations used in the ’90s [42]
- Classes A, B, C: the standard test sizes, increasing by roughly 4x in size from class to class
- Classes D, E, F: significantly larger test sizes, increasing by roughly 16x in size from class to class

Detailed information on problem sizes and parameters for each benchmark and class is available at [44]. Selecting a benchmark’s problem size is achieved by setting a parameter before compilation. Hence, there are separate benchmark executables for each class. In the upcoming sections 4.2 and 4.3, the results of tests including the presented benchmarks – configured with different thread numbers or combinations, and problem-size classes – will be detailed and evaluated. Problem sizes of class D or higher require more RAM than available on the ODROID boards and were therefore

excluded from the tests. For the remainder of this thesis, problem-size classes will be directly annotated to the benchmarks' names in the following way: $\texttt{name}_{\texttt{class}}$, for example $\texttt{EP}_{\texttt{A}}$.

To evaluate the cluster's energy consumption during the tests, on-line energy measurements were performed utilizing an external energy measurement device. The used device is presented in the following subsection.

### 4.1.3 Energy Measurement Device

The external energy measurement device used for recording the energy consumption of the systems evaluated in this chapter is the Microchip MCP39F511 Power Monitor Demonstration Board [45], a single-phase power and energy monitoring system designed for real-time measurements of input power for AC/DC power supplies. It is depicted in Figure 4.1. The device measures electric current based on the voltage drop across a 2 mΩ shunt resistor, and voltage across a 1000:1 voltage divider. The measured current and voltage values are then used to internally calculate power and energy with a maximum error of 0.1 % across a dynamic range of 4000:1. Therefore, the results are highly accurate for both low and high input voltage and current values. Recording the results is achieved by connecting the board via USB to a system, and utilizing specialized software, such as the supplied Power Monitor Utility [45].

The results for total energy consumption presented in the following sections were calculated by recording power-consumption values with a resolution of 2 ms, multiplying the current power-consumption value with the difference in system time between the current and previous time steps, and adding the resulting energy value to the sum of previous values until the last recorded time step has been included into the calculation. The total recorded time is calculated by subtracting the system time of the first recorded step from the system time of the last recorded step. The results of the initial set of tests on the evaluation setup are presented and evaluated in the following Section 4.2.


**Figure 4.1** – The Microchip MCP39F511 Power Monitor Demonstration Board [45]

## 4.2 Single Benchmark Execution Tests

The initial set of tests gathered energy-consumption and execution-time data for all available NPBs, executed on single nodes of the cluster. All test results are averaged over at least five separate runs, excluding outliers. The energy consumption was measured utilizing the external device described in Section 4.1.3, with the measurement started directly before job submission via srun and stopped directly after the call of srun returns. `IS`, `EP`, `CG`, `MG`, and `FT` were executed as single-zone workloads with the maximum number of threads supported simultaneously by the nodes: four on the ODROIDs and eight on the Xeon system. From the remaining pseudo-application benchmarks, the multi-zone versions `BT-MZ`, `SP-MZ`, and `LU-MZ` were chosen, configured with varying combinations of inner and outer thread numbers. Therefore, the effect of varying thread combinations on the systems' energy consumption could be observed as well.

In addition to execution time and energy consumption, a third metric is commonly used to compare system execution performance: the energy–delay product (EDP). The default EDP simply multiplies consumed energy and time values, weighting both parameters equally. Therefore, smaller EDP values equal better system performance. Custom weights an increased impact of either value, for example squaring the time value. Utilizing the EDP allows judging whether an increased energy efficiency is worth the higher execution time.

The goal of this initial set of tests was to find benchmarks that could be executed more energy-efficient on the employed ODROID low-power nodes compared to the Xeon system, which would allow for the cluster's energy efficiency to be improved by including low-power nodes into the cluster setup. In addition to investigating the default Xeon node, forcing various power caps on the Xeon node with the help of the Intel RAPL interface and the resulting change in energy efficiency was investigated. The best energy efficiency results were observed at power caps of either 10 W or 15 W. Therefore, whenever power caps of the Xeon system are mentioned in the following sections of Chapter 4, one of these two power caps was enforced, depending on which had better results for the executed benchmark. The first tests investigated the execution of the single-zone workloads `IS`, `EP`, `CG`, `MG`, and `FT`, and their results(energy consumption, execution time, and EDP) are presented and evaluated in the following subsection.

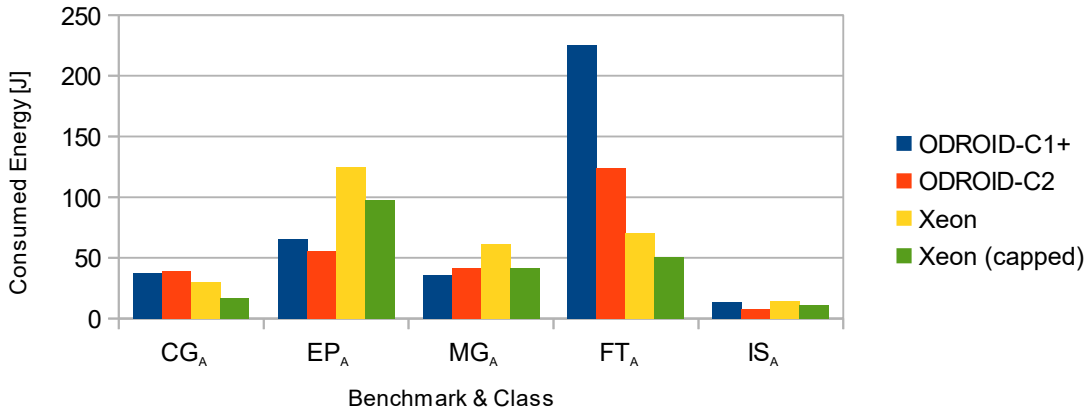


**Figure 4.2** – Energy consumption of nodes executing single-zone class A NPB

**Table 4.2** – Execution time, energy consumption, and EDP of single-zone class A NPB on different nodes

| Benchmark | Node | Execution Time | Energy Consumption | EDP |
|---|---|---|---|---|
| $CG_A$ | ODROID-C1+ | 9.10 s | 37.30 J | 339.43 Js |
| | ODROID-C2 | 8.00 s | 39.45 J | 315.60 Js |
| | Xeon | 0.58 s | 29.91 J | 17.33 Js |
| | Xeon (capped) | 0.63 s | 17.11 J | 10.71 Js |
| $EP_A$ | ODROID-C1+ | 27.92 s | 65.73 J | 1,835.04 Js |
| | ODROID-C2 | 12.58 s | 55.54 J | 697.56 Js |
| | Xeon | 1.86 s | 124.83 J | 232.68 Js |
| | Xeon (capped) | 3.05 s | 97.18 J | 296.42 Js |
| $MG_A$ | ODROID-C1+ | 14.80 s | 35.79 J | 529.62 Js |
| | ODROID-C2 | 7.76 s | 41.66 J | 323.24 Js |
| | Xeon | 1.46 s | 61.24 J | 89.14 Js |
| | Xeon (capped) | 1.52 s | 41.29 J | 62.76 Js |
| $FT_A$ | ODROID-C1+ | 55.12 s | 225.63 J | 12.44 kJs |
| | ODROID-C2 | 22.80s | 123.50 J | 2,815.80 Js |
| | Xeon | 1.23 s | 70.71 J | 87.26 Js |
| | Xeon (capped) | 1.87 s | 50.19 J | 93.82 Js |
| $IS_A$ | ODROID-C1+ | 3.41 s | 13.25 J | 45.17 Js |
| | ODROID-C2 | 1.51 s | 8.00 J | 12.08 Js |
| | Xeon | 0.30 s | 14.77 J | 4.41 Js |
| | Xeon (capped) | 0.41 s | 10.74 J | 4.39 Js |

### 4.2.1 NAS Parallel Benchmarks Single-Zone Tests

In this section, a selection of test results of separately executing `IS`, `EP`, `CG`, `MG`, and `FT` on single nodes is presented and evaluated. It was observed, that the problem class size had no significant impact on how the nodes performed compared to each other. Therefore, only the results of single-zone benchmarks configured with problem-size class A are presented in this section. All of the results referenced in this section are listed in Table 4.2.

#### Energy Consumption

The initial goal was to find benchmarks that can be executed on the ODROIDs consuming less energy than being executed on the Xeon system, disregarding the difference in execution time. This search was successful: certain problem sizes of benchmarks could be executed on the ODROIDs with reduced energy consumption. In particular, the execution of $EP_A$, $MG_A$, and $IS_A$ consumed 42–56 % less energy on the ODROID boards than on the Xeon. The energy consumption results of tests on single-zone NPBs with class A are visualized in Figure 4.2. Enforcing power caps on the Xeon always improved its energy efficiency, reducing the difference to 13–43 %, allowing the Xeon to catch up to and sometimes (e.g., in the case of $MG_A$) even match the energy efficiency of one of the ODROIDS – but never both. Therefore, focusing only on energy efficiency, a heterogeneous cluster could potentially benefit from employing low-power boards, such as the ODROID family.

### Execution Time

Pure energy efficiency at the cost of execution time might not be desired, especially in the context of HPC. The ODROIDs required significantly more time to execute all of the investigated benchmarks. In the case of $EP_A$, $MG_A$, and $IS_A$, execution on the ODROIDs took between 5x ($IS_A$) and 10.1x ($MG_A$) longer than on the uncapped Xeon, comparing only the execution time of whichever ODROID consumed less energy. Enforcing power caps on the Xeon reduced the gap in execution time: now execution on the ODROIDs only took between 3.7x ($IS_A$) and 9.7x ($MG_A$) longer than on the capped Xeon. Still, a significant difference in execution time remains, favoring the Xeon. Therefore, the ODROIDs are only valid choices for a cluster if the increased execution time is acceptable.

### Energy–Delay Product

As listed in Table 4.2, the difference in EDP values between the ODROIDs and the Xeon changes greatly for each of the investigated benchmarks. In general, the ODROID-C2's increased processing power resulted in EDPs significantly closer to the Xeon's results compared to the ODROID-C1+'s; the only exception is $CG_A$, where the ODROID-C1+'s EDP was only 7.6 % higher than the ODROID-C2's. However, even the ODROID-C2's EDPs – ignoring benchmarks where the ODROIDs were less energy-efficient – are 2.75x ($IS_A$) to 5.15x ($MG_A$) higher than the Xeon's EDPs. Therefore, the ODROIDs can only be valued higher than the Xeon if energy efficiency is weighted significantly higher than time. The formula for the EDP with energy weightings is the following:

$$EDP_{w_E} = t_{execution} * E_{consumed}^{w_E} \tag{4.1}$$

$t_{execution}$ represents the execution time, $E_{consumed}$ represents the consumed energy, and $w_E$ represents the energy's weighting. To highlight the difference in weightings required: even for the results of $IS_A$ – which has the smallest difference in EDPs between ODROID-C2 and the capped Xeon – at least weighting the energy value to the power of five is required to favor the ODROID-C2. However, comparing the EDPs of the ODROID-C2 and the uncapped Xeon, weighting the energy value to the

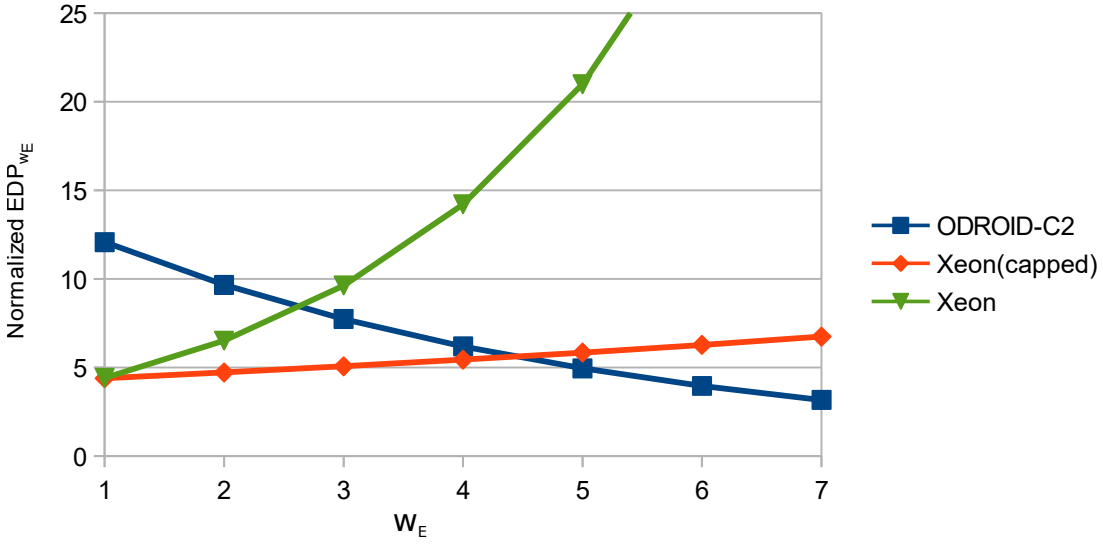


**Figure 4.3** – Normalized EDP values of two nodes for $IS_A$ and increasing energy weights

power of three is sufficient to favor the ODROID-C2. The difference in EDP values for increasing energy weightings is visualized in Figure 4.3.

To summarize, it was discovered that $EP_A$, $MG_A$, and $IS_A$ could be executed more energy-efficiently on at least one of the ODROID boards compared with the Xeon system. However, both the execution time and the EDP resulting from executing the listed benchmarks on the ODROIDs are significantly worse than on the Xeon system. Therefore, energy efficiency needs to be weighted significantly higher than performance to benefit from extending a cluster with the ODROID boards. The results of tests on the remaining multi-zone benchmarks BT-MZ, SP-MZ, and LU-MZ are presented and evaluated in the next subsection.

### 4.2.2 NAS Parallel Benchmarks Multi-Zone Tests

In this section, a selection of test results of separately executing BT-MZ, LU-MZ, and SP-MZ on single nodes is presented and evaluated. All of the results referenced in this section are listed in Table 4.3. The goal was the same as in the previous section: finding benchmarks that can be executed on the

**Table 4.3** – Execution time, energy consumption, and EDP of multi-zone class W/A NPB on different nodes

| Benchmark | Node | Threads | Execution Time | Energy Consumption | EDP |
|---|---|---|---|---|---|
| $BT\text{-}MZ_W$ | ODROID-C1+ | (4 1) | 15.51 s | 64.55 J | 1000.98 Js |
| | ODROID-C2 | (4 1) | 7.22 s | 34.24 J | 247.33 Js |
| | Xeon | (4 1) | 0.88 s | 58.65 J | 51.75 Js |
| | Xeon (capped) | (4 1) | 1.44 s | 46.06 J | 66.33 Js |
| $LU\text{-}MZ_W$ | ODROID-C1+ | (2 2) | 17.81 s | 72.91 J | 1298.56 Js |
| | ODROID-C2 | (2 2) | 7.81 s | 41.99 J | 327.78 Js |
| | Xeon | (8 1) | 0.48 s | 32.24 J | 15.35 Js |
| | Xeon (capped) | (8 1) | 0.81 s | 26.25 J | 21.27 Js |
| $SP\text{-}MZ_W$ | ODROID-C1+ | (2 2) | 12.71 s | 50.96 J | 647.67 Js |
| | ODROID-C2 | (2 2) | 6.84 s | 36.25 J | 247.85 Js |
| | Xeon | (8 1) | 0.38 s | 26.96 J | 10.27 Js |
| | Xeon (capped) | (8 1) | 0.63 s | 20.55 J | 12.95 Js |
| $BT\text{-}MZ_A$ | ODROID-C1+ | (4 1) | 192.13 s | 719.90 J | 138.31 kJs |
| | ODROID-C2 | (4 1) | 78.06 | 393.64 J | 30.73 kJs |
| | Xeon | (4 2) | 8.68 s | 654.83 J | 5.68 kJs |
| | Xeon (capped) | (4 2) | 14.71 s | 477.97 J | 7.03 kJs |
| $LU\text{-}MZ_A$ | ODROID-C1+ | (4 1) | 197.73 s | 827.81 J | 163.69 kJs |
| | ODROID-C2 | (1 4) | 119.81 s | 673.52 J | 80.69 kJs |
| | Xeon | (2 4) | 5.33 s | 362.22 J | 1.93 kJs |
| | Xeon (capped) | (2 4) | 8.14 s | 271.32 J | 2.21 kJs |
| $SP\text{-}MZ_A$ | ODROID-C1+ | (2 2) | 174.91 s | 746.32 J | 130.54 kJs |
| | ODROID-C2 | (2 2) | 99.29 s | 545.92 J | 54.20 kJs |
| | Xeon | (4 2) | 7.67 s | 367.95 J | 2.82 kJs |
| | Xeon (capped) | (4 2) | 10.92 s | 299.00 J | 3.27 kJs |

ODROIDS with increased energy efficiency compared to execution on the Xeon system. However, the multi-zone versions of the NPB introduce another pair of configuration parameters that can be adjusted individually for each node and benchmark: the number of outer and inner threads. For the remainder of this thesis, combinations of outer and inner threads will be referred to in the following way: (#outer threads #inner threads), for example (4 2).

### Effect of Thread Combinations

The exact combination of outer/inner threads noticeably affected the nodes' energy consumption. An example: When executing LU-MZ$_A$ on the Xeon without power caps, the thread combination with the best result (2 4) consumed 24 % less energy than the combination with the worst result (2 1). Limiting the thread combinations to a total sum of exactly eight threads, the best combination (2 4) still resulted in 10 % less energy consumption than the worst combination (8 1). The consumed energy per investigated thread combination is visualized in Figure 4.4. However, the thread combination producing the best results on any of the given nodes changed for each benchmark, and in some cases even for each problem class. For example, the most energy-efficient thread combination for the execution of SP-MZ$_W$ on the Xeon was found to be (8 1), but for the execution of SP-MZ$_A$ it was found to be (4 2), as shown in Table 4.3. Therefore, determining the optimal number of outer and inner threads is necessary to achieve the most energy-efficient execution for each combination of benchmark, problem-size class, and executing node. All results presented after this point were selected to feature the most energy-efficient combination of outer and inner threads, as noted in the column "Threads" in Table 4.3.

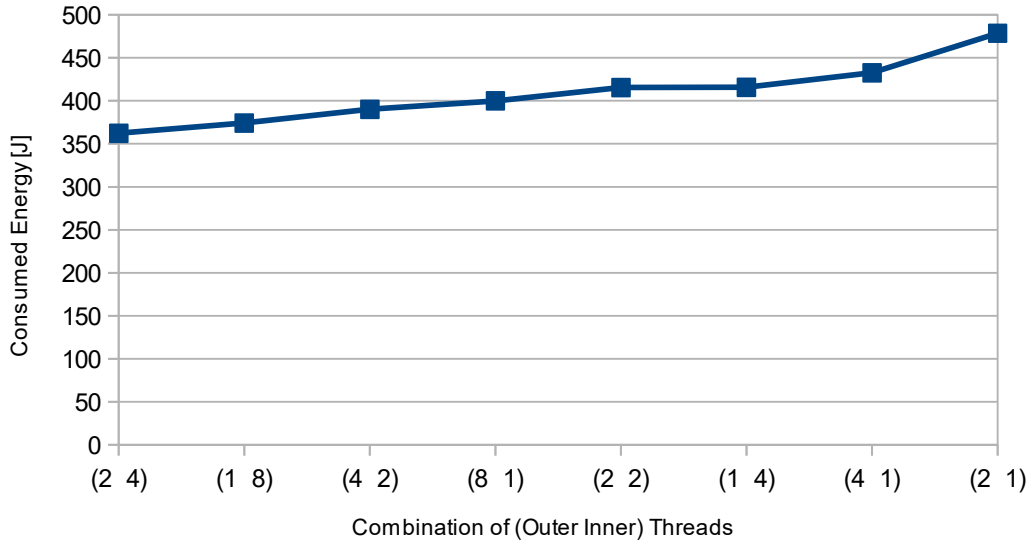


**Figure 4.4** – Influence of thread numbers on energy consumption for LU-MZ$_A$, which was executed on the Intel Xeon

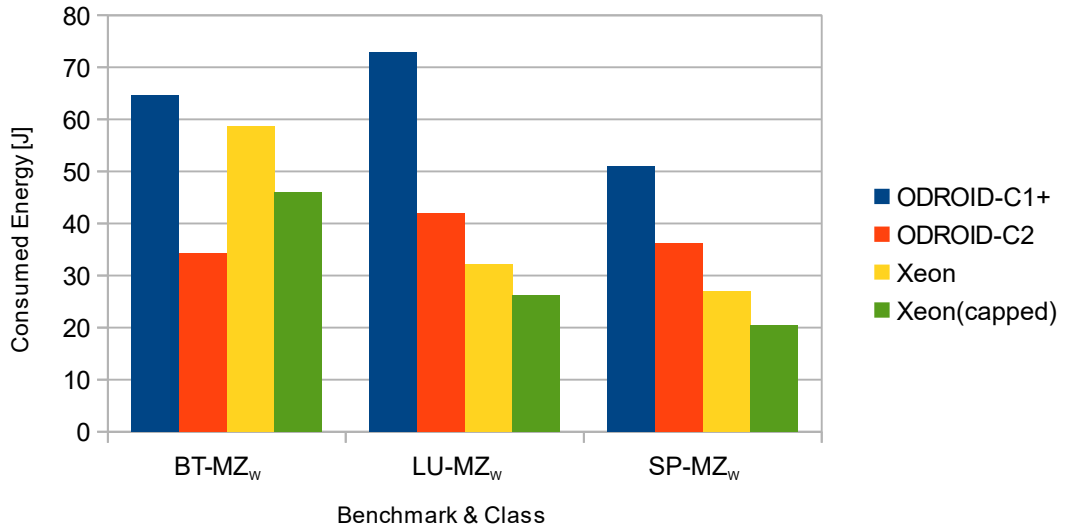


**Figure 4.5** – Energy consumption of nodes executing multi-zone class W NPB

#### Comparison of Energy Consumption, Run-Time, and Energy–Delay Product

The specific energy-consumption values for the execution of $\texttt{BT-MZ}_{W/A}$, $\texttt{LU-MZ}_{W/A}$, and $\texttt{SP-MZ}_{W/A}$ are compared in Figure 4.5 and Figure 4.6, respectively. It was observed that, even with the most optimal configuration, the ODROID-C1+ always consumed more energy and required significantly more time than the Xeon – both with and without active power caps. Therefore, the ODROID-C1+'s test results are not further evaluated in this section. However, the ODROID-C2 was more energy-efficient than the Xeon – even with the most energy-efficient power cap enforced – for the execution of $\texttt{BT-MZ}_{W}$ and $\texttt{BT-MZ}_{A}$. In the case of $\texttt{BT-MZ}_{W}$, the ODROID-C2 consumed 41.6 % less energy than the uncapped Xeon, and 25.7 % less than the power-capped Xeon. For the execution of $\texttt{BT-MZ}_{A}$, the ODROID-C2 required 39.9 %/17.6 % less energy than the uncapped/power-capped Xeon. However, considering execution-time differences and the resulting EDPs, results similarly unfavorable for the ODROID-C2 as in Section 4.2.1 were observed. Even for the execution of $\texttt{BT-MZ}_{W/A}$, the ODROID-C2's EDPs were between 3.7x and 5.4x higher than the Xeon's. Therefore, the same condition as in Section 4.2.1 needs to be fulfilled: the energy efficiency needs to either be the only criteria or, at least, weighted significantly higher than the execution time to prefer the ODROID-C2 over the Xeon. No further benchmarks/problem-size classes were found where the ODROID-C2 was more energy-efficient.

### 4.2.3 Summary of Single-Benchmark–Execution Test Results

Concluding the presentation and evaluation of test results in the previous subsections, it was observed that $\texttt{EP}_{A}$, $\texttt{MG}_{A}$, $\texttt{IS}_{A}$, and $\texttt{BT-MZ}_{W/A}$ could be executed between 13 % and 56 % more energy-efficient on at least one of the ODROID boards than on the Xeon system. Enforcing power caps on the Xeon resulted in reduced energy consumption, but did not allow the Xeon system to match the respective ODROID board's energy efficiency. However, the improved energy efficiency of the ODROID boards comes at the cost of significantly increased execution time compared to the Xeon system. Comparing the nodes' resulting EDPs, it was observed that the ODROIDs' EDPs were 2.75x to 5.4x higher than the Xeon's EDPs for the benchmarks that could be executed on the ODROIDs with reduced energy consumption. Therefore, energy efficiency needs to be valued significantly higher than execution

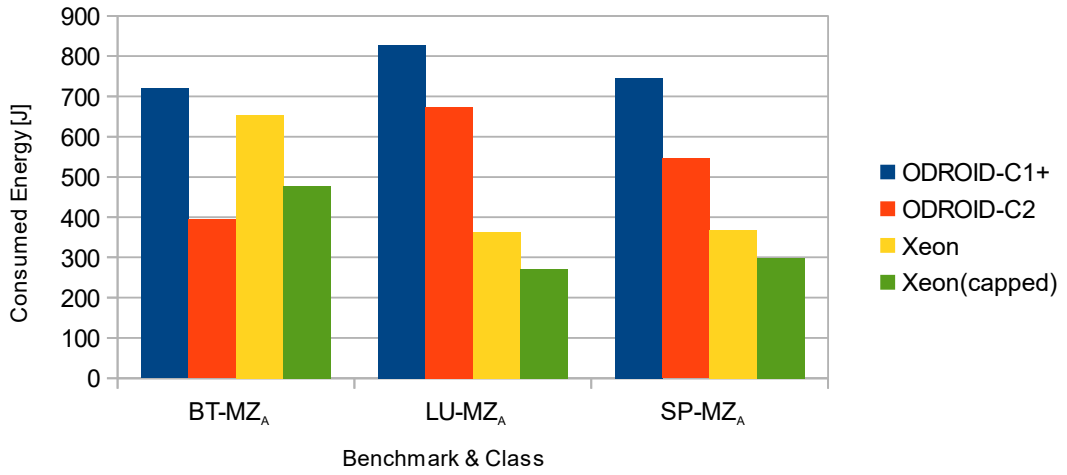


**Figure 4.6** – Energy consumption of nodes executing multi-zone class A NPB

time to favor the ODROIDs over the Xeon. It was observed that weighting the consumed energy at least to the power of five favors the ODROID-C2 over to the capped Xeon, whereas weighting the consumed energy to the power of three is sufficient to favor the ODROID-C2 over the uncapped Xeon. Additionally, in case of the multi-zone NPB, selecting the correct combination of outer and inner threads for each benchmark and node was found to improve the energy efficiency as well.

With the help of the observations presented in this chapter, the modified scheduling and resource manager SLURM-HC was able to improve the node selection and benchmark configuration processes. How this improved the energy efficiency of the evaluated cluster for the execution of jobs consisting of multiple benchmarks is evaluated in the following section.

## 4.3 Multi-Benchmark Parallel Execution Tests

In this section, the results of tests in which jobs consisting of multiple benchmarks were executed on the evaluation cluster are presented and evaluated. Specifically, it was investigated if utilizing the knowledge gained through the tests detailed in Section 4.2 could improve the cluster's overall energy efficiency. Which benchmarks were chosen and how they were submitted to the cluster is detailed in the following subsection.

### 4.3.1 Benchmark Set Components & Submission Method

Multiple benchmarks were submitted to the cluster in parallel using a number of threads, each requesting the execution of their own set of benchmarks in a consistent, deterministic order. The

**Table 4.4** – Distribution of evaluated benchmarks to three parallel submission threads

| Benchmark-Set | Thread$_1$ | Thread$_2$ | Thread$_3$ |
|---|---|---|---|
| sz-load | 1x EP$_A$ | 1x MG$_A$ | 16x CG$_A$ |
| mz-load | 2x BT-MZ$_W$ | 2x LU-MZ$_A$ | 1x BT-MZ$_W$ |

parallel submission was realized by utilizing the GNU parallel shell tool [46] to distribute the set of to-be-submitted benchmarks to three distinct threads. The number of threads was selected to equal the amount of available nodes, preserving the deterministic order of submission by preventing race-conditions between multiple threads after one of the benchmarks finished execution. Utilizing GNU parallel, the order of executed commands per thread is guaranteed to be deterministic. Therefore, the results of multiple test runs can be combined to calculate averages for execution time and energy consumption.

The execution of two different sets of benchmarks was tested: a single-zone NPB set called `sz-load` and a multi-zone NPB set called `mz-load`. `sz-load` consists of one $EP_A$, one $MG_A$ and sixteen $CG_A$ benchmarks; `mz-load` of three $BT\text{-}MZ_W$ and two $LU\text{-}MZ_A$ benchmarks. The exact distribution of benchmarks to GNU parallel threads is listed in Table 4.4. The specific benchmarks and problem-size classes forming these sets and the distribution to the threads were chosen to minimize the delay between the first and last thread finishing if SLURM-HC was responsible for the node selection. In addition, the specific order of submission was chosen to guarantee that the most energy-efficient node for each benchmark, based on the results presented in Section 4.2, would be allocated to the benchmarks by SLURM-HC – with the exception of the single instance of $BT\text{-}MZ_W$ submitted by $Thread_3$, as listed in Table 4.4, which was executed by the ODROID-C1+. This exception was made since none of the investigated multi-zone NPB could be executed on the ODROID-C1+ with better energy efficiency than on the Xeon system – therefore, a benchmark was chosen that required an execution time on the ODROID-C1+ as close as possible to the total time required for the execution of the entire `mz-load` benchmark set.

In the following subsection, the energy consumption, time, and EDP (with the default energy weighting of 1) results of the evaluation cluster managed by SLURM's default implementation are compared to the results of the cluster being managed by the modified version SLURM-HC, which was presented in Chapter 3, for executing both presented benchmark sets: `sz-load` and `mz-load`.

### 4.3.2 Comparison of Test Results: SLURM vs. SLURM-HC

Before the test results were available, it was assumed that managing the cluster with the help of SLURM-HC would be more energy-efficient than with the help of SLURM, since SLURM does not consider the vastly different execution time and energy consumption values resulting from allocating different nodes to submitted jobs. However, it was unclear how significant the impact of the modifications implemented in SLURM-HC on both execution time and energy consumption of the evaluation cluster would be. The results referenced in the following discussion are listed in Table 4.5 and visualized in Figure 4.7.

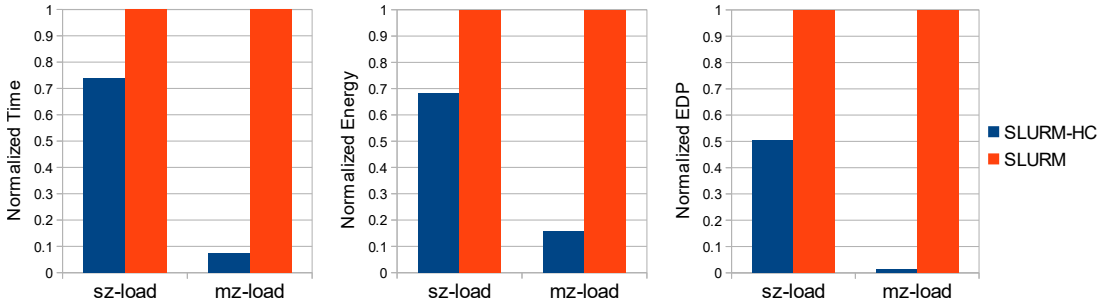


**Figure 4.7** – Comparison of single/multi-zone–test execution time, energy consumption, and EDP for SLURM and SLURM-HC, normalized to SLURM's results

**Table 4.5** – Average execution time, energy consumption, and EDP results of benchmark set tests comparing SLURM and SLURM-HC

| Benchmark Set | Management Software | Execution Time [s] | Energy Consumption [J] | EDP [kJs] |
|---|---|---|---|---|
| sz-load | SLURM | 20.68 | 797.04 | 16.48 |
| | SLURM-HC | 15.28 | 544.72 | 8.32 |
| mz-load | SLURM | 260.3 | 4,912.99 | 1278.85 |
| | SLURM-HC | 19.2 | 771.35 | 14.81 |

The results of a series of tests on `sz-load` managed by either SLURM or SLURM-HC show that by utilizing SLURM-HC, on average 26 % time and 32 % energy could be saved compared to utilizing SLURM. Therefore, SLURM-HC's EDP was only half as high as SLURM's EDP, representing a significant gain of performance and energy efficiency.

However, comparing this gain to the difference in EDP resulting from tests on `mz-load` showcases how inadequate SLURM can – under certain conditions – be for the management of a heterogeneous cluster: on average, using SLURM-HC resulted in 93 % less execution time and 84 % less energy consumption than utilizing SLURM. As a result of these immense differences in both time and energy consumption, SLURM-HC's EDP was 98.84 % smaller than SLURM's EDP, completely overshadowing the EDP gain observed for `sz-load`.

### 4.3.3 Discussion

As presented in the previous subsection, SLURM produced significantly worse results for `mz-load` than for `sz-load`. Therefore, the potential reasons for the difference between the results are investigated in this subsection.

#### 1st Reason: Configuration of Thread Numbers

Based on the fact that SLURM's results for `mz-load` are significantly worse than the results for `sz-load`, the assumption that executing the multi-zone versions instead of the single-zone versions of the NPB is inducing this significant difference in results seems natural. Since it was unclear which node would be allocated by SLURM to each of the executed benchmarks, ensuring that the optimal number of outer and inner threads is added to the benchmarks' configurations was impossible. Therefore, the combination found to be most commonly optimal for $\texttt{BT-MZ}_\texttt{W}$ and $\texttt{LU-MZ}_\texttt{A}$, which – based on the results presented in Section 4.2.2 – is (4 1), was added to the benchmarks' configurations before they were submitted to SLURM. However, executing $\texttt{LU-MZ}_\texttt{A}$ with the thread combination (4 1) consumed only about 10 % more time and energy than with the optimal combination on both the Xeon and the ODROID-C1+, and was found to be the most optimal combination for the ODROID-C2 during the tests in Section 4.2. Hence, less-than-optimal thread combinations cannot be the only reason for an almost 99 % smaller EDP achieved by utilizing SLURM-HC instead of SLURM.

#### 2nd Reason: Difference in Execution-Time–Increase

However, there is another key difference between the benchmarks used in `sz-load` and `mz-load`: results in Table 4.3 show that the ODROID-C1+ took 24x longer than the uncapped Xeon for the execution of $\texttt{LU-MZ}_\texttt{A}$, a total of 189.56 s. In comparison to that, results in Table 4.2 show that the

ODROID-C1+ in the worst case ($CG_A$) took only 14x, or 8.56 s, longer than the uncapped Xeon. Therefore, the difference of the increase in execution time – caused by selecting the ODROID-C1+ for the wrong workload – between `sz-load` and `mz-load` is only 42 % – still not enough to induce the immense difference in EDP improvement between `sz-load` and `mz-load`.

### 3rd Reason: Node Selection

Hence, at least one more cause for the difference in EDP improvement had to be found. Investigating how SLURM allocates nodes to incoming jobs was found to be priority-based. The Xeon always was the first to be allocated, followed by the ODROID-C1+, and last the ODROID-C2. Comparing this fixed order to the optimal order of allocation realized by SLURM-HC, and the resulting difference in allocation of nodes to the single components of the benchmark sets between SLURM and SLURM-HC, another cause for the difference in EDP improvement was found. A comparison of node allocation to single-benchmark components between SLURM and SLURM-HC is listed in Table 4.6. In the case of `sz-load`, SLURM only mistakenly allocated the Xeon once for $EP_A$ and, as a consequence, the ODROID-C2 once for $CG_A$, resulting in the observed negative impact on the EDP compared to SLURM-HC. However, in the case of `mz-load`, SLURM not only mistakenly allocated $BT\text{-}MZ_W$ twice to the Xeon and once to the ODROID-C2, but also allocated $LU\text{-}MZ_A$ once to the ODROID-C1+. This wrong allocation alone increased the expected total consumed time for executing `mz-load` by the previously calculated 189.56 s, almost 10x as much as the total time consumed utilizing SLURM-HC. Since the energy consumption of all nodes was recorded for the entire duration of the set's execution, suddenly both the Xeon's and the ODROID-C2's idle energy consumption immensely impacted the total energy value.

### Concluding Discussion

However, in the context of realistic HPC scenarios, leaving a node idle for prolonged periods of time appears to be a waste of performance. Therefore, it is assumed that an HPC cluster's nodes will not remain idle for long. Hence, ignoring the additional energy consumption of idle nodes caused by increased total execution time, the improvement of energy efficiency observed during the tests on `sz-load` appears to be more realistic. However, the benchmarks were submitted in an order which

**Table 4.6** – Comparison of node allocation to components of benchmark sets between SLURM and SLURM-HC

| Benchmark-Set | $Thread_1$ | $Thread_2$ | $Thread_3$ |
|---|---|---|---|
| Manager | Allocated Node | Allocated Node | Allocated Node |
| sz-load | 1x $EP_A$ | 1x $MG_A$ | 16x $CG_A$ |
| SLURM-HC | ODROID-C2 | ODROID-C1+ | 16x Xeon |
| SLURM | Xeon | ODROID-C1+ | 1x ODROID-C2, 15x Xeon |
| mz-load | 2x $BT\text{-}MZ_W$ | 2x $LU\text{-}MZ_A$ | 1x $BT\text{-}MZ_W$ |
| SLURM-HC | 2x ODROID-C2 | 2x Xeon | ODROID-C1+ |
| SLURM | 2x Xeon | 1x ODROID-C1+, 1x Xeon | 1x ODROID-C2 |

guaranteed that SLURM-HC could always allocate the optimal node to each benchmark. Considering that it seems improbable that the order of submission will always be optimal for SLURM-HC, the observed increase in energy efficiency is likely the best-case result achievable by utilizing the version of SLURM-HC presented in this thesis. To showcase the benefits of utilizing application-induced energy claims, more realistic job batches consisting of many more workloads should be evaluated. Considering the energy claims during the process of scheduling, SLURM-HC should be able to improve the energy efficiency of heterogeneous clusters for larger job batches based on the results observed in this thesis. However, modifying SLURM's scheduling logic was not investigated in this thesis, and therefore remains as a potential topic for future work, as mentioned in Chapter 5.

## 4.4 Summary

In this chapter, the results of two sets of tests were presented and evaluated, with the goal of investigating the effect of utilizing the modifications implemented in SLURM-HC on the energy efficiency of a heterogeneous cluster.

First, the used evaluation setup, including hardware and software components, was presented in Section 4.1, detailing hardware specifications, benchmark selection and how time and energy consumption was recorded. Next, the results of the initial set of tests were presented in Section 4.2, which allowed finding the most energy-efficient configuration and node for each of the investigated benchmarks. Last, the impact of the modifications implemented in SLURM-HC was presented by evaluating the results of tests in which multiple benchmarks were submitted to the evaluation cluster in parallel, and the cluster was either managed by SLURM or SLURM-HC. It was found that for executing the investigated sets of benchmarks, utilizing SLURM-HC instead of SLURM improved the cluster's energy efficiency by at least 32 %.

# 5 CONCLUSION & FUTURE WORK

The goal of this thesis was to improve the energy efficiency of a many-node heterogeneous computing system by utilizing application-induced energy claims. It was discovered that software designed for HPC environments has to be adapted to the specifics of a heterogeneous cluster. In particular, the scheduling and resource-management software SLURM does not incorporate the difference in energy efficiency and run-time between the different nodes present in a heterogeneous cluster. Therefore, a modified version of SLURM, named SLURM-HC (as in Simple Linux Utility for the Resource-Management of Heterogeneous Clusters), was designed and implemented. SLURM-HC incorporated application-induced energy claims, such as run-time, energy consumption and node configuration, into the node selection process. The specific energy claims were acquired by evaluating the results of tests in which a series of benchmarks was executed on all of the nodes present in the cluster individually. With the help of a subsequent series of tests featuring parallel execution of multiple benchmarks on the entire cluster, it was observed that utilizing SLURM-HC instead of SLURM reduced the evaluated cluster's energy consumption by at least 32 %, and the execution time by at least 26 %. Therefore, the goal of this thesis was fulfilled.

Future work could further improve the presented software for the management of a many-node heterogeneous computing system. The energy claims were only used during the selection of available nodes, but not for the management of the scheduling queue. Utilizing the energy claims, a more sophisticated scheduling algorithm could be developed that considers the gathered energy claims to optimize its decisions. Additionally, the energy claims were only recorded a priori. The software could be extended to verify these claims continuously against data recorded during subsequent executions of the applications known to the management software. This would also allow energy claims induced by previously unknown applications to be considered. Supplementing the energy claims presented in this thesis by adding additional criteria to the node-selection process could further improve the cluster's energy efficiency. These criteria could, for example, include the accuracy of the available data on energy claims. Another possibility would be to extend the heterogeneity of the cluster by including systems featuring other specialized hardware, such as GPUs. Subsequently, the effect on energy efficiency induced by extending the heterogeneity could be investigated and if, respectively how, the scheduling and resource-management software would have to be adapted to the new hardware types. Last, the workloads investigated in this thesis were executed solely on a single node. Hence, expanding the variety of investigated workloads by including jobs that are executed in parallel on many nodes at the same time, and varying the nodes allocated to the job's parts could be another topic for future work.

# LIST OF ACRONYMS

| | |
|---|---|
| **API** | application programming interface |
| **ARM** | advanced reduced instruction set computing machine |
| **CentOS** | Community Enterprise Operating System |
| **CFD** | computational fluid dynamics |
| **CISC** | complex instruction set computing |
| **CPU** | central processing unit |
| **DFS** | dynamic frequency scaling |
| **DRAM** | dynamic RAM |
| **DVFS** | dynamic voltage and frequency Scaling |
| **ECC** | error-correcting |
| **EDP** | energy–delay product |
| **FIFO** | first-in–first-out |
| **GPGPU** | general-purpose GPU |
| **GPU** | graphics processing unit |
| **HDD** | hard disk drive |
| **HPC** | high-performance computing |
| **IP** | Internet protocol |
| **ISA** | instruction set architecture |
| **MAC** | multimedia access control |
| **MCU** | microcontroller unit |
| **MPI** | Message Passing Interface |
| **NPB** | NAS parallel benchmarks |
| **OHPC** | OpenHPC |

| | |
|---|---|
| **OS** | operating system |
| **RAM** | random-access memory |
| **RAPL** | running average power limit |
| **RISC** | reduced instruction set computing |
| **SSD** | solid-state drive |
| **SLES** | SUSE Linux Enterprise Server |
| **SLURM** | Simple Linux Utility for Resource Management |
| **SLURM-HC** | Simple Linux Utility for Resource Management of Heterogeneous Clusters |
| **TDP** | thermal design power |
| **USB** | universal serial bus |
| **VM** | virtual machine |
| **VNFS** | virtual node file system |
| **VRAM** | video RAM |

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF ALGORITHMS

# REFERENCES


[1] *List of Top500 Supercomputers Worldwide - November 2016*. URL: https://www.top500.org/list/2016/11/ (visited on 04/2017).

[2] D. Jaggar et al. "ARM Architecture and Systems." In: *IEEE micro* 17.4 (1997), pp. 9–11.

[3] *The ARM Cortex-A53 Processor*. URL: https://www.arm.com/products/processors/cortex-a/cortex-a53-processor.php (visited on 05/2017).

[4] M. Canuto et al. "A Methodology for Full-system Power Modeling in Heterogeneous Data Centers." In: *Proceedings of the 9th International Conference on Utility and Cloud Computing*. ACM, 2016, pp. 20–29.

[5] *The Mont-Blanc Project*. URL: http://www.montblanc-project.eu/ (visited on 04/2017).

[6] *NVIDIA TITAN Xp Product Page*. URL: https://www.nvidia.com/en-us/geforce/products/10series/titan-xp/ (visited on 04/2017).

[7] C. Eibel, T. Hönig, and W. Schröder-Preikschat. "Energy Claims at Scale: Decreasing the Energy Demand of HPC Workloads at OS Level." In: *Proceedings of the 12th Parallel and Distributed Processing Symposium Workshops*. IEEE. 2016, pp. 1114–1117.

[8] C. Imes et al. "POET: A Portable Approach to Minimizing Energy Under Soft Real-Time Constraints." In: *Proceedings of the 21st IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE. 2015, pp. 75–86.

[9] *Simple Linux Utility for Resource Management (SLURM)*. URL: https://slurm.schedmd.com/ (visited on 04/2017).

[10] G. Shao, F. Berman, and R. Wolski. "Master/Slave Computing on the Grid." In: *Proceedings of the 9th Heterogeneous Computing Workshop (HCW 2000)*. 2000, pp. 3–16.

[11] *The OpenHPC Project*. URL: http://www.openhpc.community/ (visited on 04/2017).

[12] *The Community Enterprise Operating System (CentOS)*. URL: https://www.centos.org/ (visited on 04/2017).

[13] *The SUSE Linux Enterprise Server (SLES) Operating System*. URL: https://www.suse.com/products/server/ (visited on 04/2017).

[14] K. W. Schulz et al. "Cluster Computing with OpenHPC." In: *Inaugural HPC Systems Professionals Workshop (HPCSYSPROS) 16*. 2016.

[15] *The WAREWULF Project*. URL: http://warewulf.lbl.gov/trac (visited on 04/2017).

[16] *NVIDIA CUDA*. URL: http://www.nvidia.com/object/cuda_home_new.html (visited on 04/2017).

[17] *The Intel Xeon E3-v6 Server Processors*. URL: https://ark.intel.com/products/family/97141/Intel-Xeon-Processor-E3-v6-Family#@server (visited on 04/2017).

[18] B. Oancea, T. Andrei, and R.M. Dragoescu. *"GPGPU Computing"*. 2014. URL: http://arxiv.org/abs/1408.6923v1 (visited on 05/2017).

[19] *The Hardkernel ODROID-C2 Board*. URL: http://www.hardkernel.com/main/products/prdt_info.php?g_code=G145457216438 (visited on 04/2017).

[20] *The Hardkernel ODROID-C1+ Board*. URL: http://www.hardkernel.com/main/products/prdt_info.php?g_code=G143703355573 (visited on 04/2017).

[21] A. Noureddine, R. Rouvoy, and L. Seinturier. "A Review of Energy Measurement Approaches." In: *ACM SIGOPS Operating System Review* 47.3 (2013), pp. 42–49.

[22] G.L.T. Chetsa et al. "Exploiting Performance Counters to Predict and Improve Energy Performance of HPC Systems." In: *Future Generation Computer Systems* 36 (2014), pp. 287–298.

[23] T. Hönig et al. "Playing Hare and Tortoise: The FigarOS Kernel for Fine-Grained System-Level Energy Optimizations." In: *Proceedings of the 2015 Brazilian Symposium on Computing Systems Engineering (SBESC)*. 2015, pp. 80–83.

[24] Intel. "Intel® 64 and IA-32 Architectures Software Developer's Manual." In: *Volume 3B: System Programming Guide, Part* 2 (2011).

[25] P. Petoumenos et al. "Power Capping: What Works, What Does Not." In: *Proceedings of the 21st International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE. 2015, pp. 525–534.

[26] K. Kwon, S. Chae, and K. G. Woo. "An Application-Level Energy-Efficient Scheduling for Dynamic Voltage and Frequency Scaling." In: *International Conference on Consumer Electronics (ICCE)*. IEEE. 2013, pp. 3–6.

[27] A. Gandhi et al. "Optimal Power Allocation in Server Farms." In: *SIGMETRICS Performance Evaluation Review* 37.1 (2009), pp. 157–168.

[28] *The Intel Power Governor Domains*. URL: https://software.intel.com/en-us/articles/intel-power-governor (visited on 05/2017).

[29] *The Intel Xeon E3-1275 v5 Server Processor*. URL: http://ark.intel.com/products/88177/Intel-Xeon-Processor-E3-1275-v5-8M-Cache-3_60-GHz (visited on 05/2017).

[30] D. Jensen and A. Rodrigues. "Embedded Systems and Exascale Computing." In: *Computing in Science & Engineering* 12.6 (2010), pp. 20–29.

[31] E. Blem, J. Menon, and K. Sankaralingam. "Power Struggles: Revisiting the RISC vs. CISC Debate on Contemporary ARM and x86 Architectures." In: *Proceedings of the 19th IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE. 2013, pp. 1–12.

[32] R. Azimi et al. "Fast Decentralized Power Capping for Server Clusters." In: *Proceedings of the 23rd IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE. 2017, pp. 181–192.

[33] N. Rajovic et al. "Supercomputing with Commodity CPUs: Are Mobile SoCs Ready for HPC?" In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. 2013, 40:1–40:12.

[34] N. Rajovic et al. “Tibidabo: Making the Case for an ARM-Based HPC System.” In: *Future Generation Computer Systems* 36 (2014), pp. 322–334.

[35] J. W. Weloli et al. “Efficiency Modeling and Analysis of 64-bit ARM Clusters for HPC.” In: *Euromicro Conference on Digital System Design (DSD)*. 2016, pp. 342–347.

[36] *SLURM srun documentation*. URL: https://slurm.schedmd.com/srun.html (visited on 05/2017).

[37] *SLURM Plug-In Programmer Guide*. URL: https://slurm.schedmd.com/plugins.html (visited on 05/2017).

[38] *SLURM configuration file documentation*. URL: https://slurm.schedmd.com/slurm.conf.html (visited on 05/2017).

[39] *The ARM Cortex-A5 Processor*. URL: https://www.arm.com/products/processors/cortex-a/cortex-a5.php (visited on 05/2017).

[40] *Open MPI: Open Source High Performance Computing*. URL: https://www.open-mpi.org/ (visited on 05/2017).

[41] D. H. Bailey et al. “The NAS Parallel Benchmarks.” In: *The International Journal of Supercomputing Applications* 5.3 (1991), pp. 63–73.

[42] *The NAS Parallel Benchmarks*. URL: https://www.nas.nasa.gov/publications/npb.html (visited on 05/2017).

[43] *The OpenMP API Specification for Parallel Programming*. URL: http://www.openmp.org/ (visited on 05/2017).

[44] *Problem Sizes and Parameters in NAS Parallel Benchmarks*. URL: https://www.nas.nasa.gov/publications/npb_problem_sizes.html (visited on 05/2017).

[45] *The Microchip MCP39F511 Power Monitor Demonstration Board*. URL: http://www.microchip.com/DevelopmentTools/ProductDetails.aspx?PartNO=ADM00667 (visited on 05/2017).

[46] *The GNU Parallel Shell Tool*. URL: https://www.gnu.org/software/parallel/ (visited on 05/2017).