FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Bachelor Thesis

Task Migration Policy for Temperature-Aware Management of Many-Core Systems

by

Paul Meyer

Matrikel-Nr.: 22051025

Supervision:
Prof. Dr.-Ing. Jürgen Teich
Behnaz Pourmohseni, M.Sc.
Dr.-Ing. Stefan Wildermann

September 23, 2020

This document was produced with the typesetting system LaTeX2e.

Contents

Aufgabenstellung zur Bachelorarbeit

Fundamentals

Modern embedded systems are increasingly targeting many-core platforms to enable the concurrent execution of several applications within one system. A many-core platform comprises a large number of processing, communication, and storage resources on a single chip. In a dynamic many-core system, applications may be launched and terminated dynamically on demand, and the system is typically coordinated by a resource manager which launches applications on demand according to the current resource availability, adapts the on-chip deployment of running applications if necessary, and removes applications upon request. In this regard, task migration is widely used for rearranging the running applications to enhance the system performance.

Thermal safety is a crucial aspect that must be taken into account in the decision-making processes of the resource manager. The massive integration of resources in a many-core platform is enabled by the ongoing process technology downsizing. Such a dense integration of resources, however, leads to an increased density of power consumption on the chip which directly results in an increased on-chip temperature. To prevent chip burn-down, the resource manager must ensure that the on-chip temperature never exceeds a predefined critical threshold. This critical temperature threshold can also be expressed as a power-density constraint on the active cores in the system [1], [2]. Here, for each combination of active cores in the system, a power-density constraint can be calculated which, once enforced on each active core, guarantees that the critical temperature threshold will be respected. A constraint based on power density is substantially easier to check and react to, compared to a constraint based on temperature [1]. To respect the power-density constraint, the voltage/frequency setting of active cores is adjusted such that the power consumption density on each core remains below the given power-density constraint.

Reducing the frequency of a core adversely affects the performance of the application that is running on that core, and in general, causes a loss of performance for the whole system. In this context, task migration can be used to alleviate the system's performance loss consequent to stringent power-density constraints. Here, the goal is to use task migration to achieve a balanced distribution of temperature across the chip by changing the set of active cores. A balanced thermal profile on the chip relaxes the power-density constraint which allows active cores to operate on a higher frequency, thus, enhancing the system's performance.

Thesis Description

The goal of this bachelor thesis is to develop a task migration policy to conduct migration decisions that lead to an enhanced thermal balance on the chip and, thereby, alleviate the system performance penalty consequent to thermal constraints. The migration policy will make migration decisions in the form of moving the workload of an active core to an inactive core which inherently changes the set of active cores. Here, the goal is to develop a task migration policy that identifies a promising migration candidate which contributes most to balancing the on-chip thermal profile. A balanced thermal profile in turn relaxes the power-density constraint which then allows active cores to operate on a higher frequency, thus, increasing the system performance. The migration policy can also be used in a loop to perform several consecutive migrations to enhance the system performance iteratively.

Since migration decisions are taken at run time, the computational overhead of the decision-making step must be kept in a tolerable range. Typically, in a many-core system, the number of possible migration candidates can grow so high that an exhaustive enumeration of them to find the best migration candidate might introduce an intolerable run-time overhead. Therefore, the migration policy must resort to heuristics to find promising migration candidates with an acceptable search overhead. To that end, this thesis will target at developing a fast migration policy based on a two-step heuristic where the problem of finding an optimal migration candidate is solved by (i) finding the best migration source and (ii) finding the best migration destination, consecutively.

The following tasks must be performed as part of this thesis:

- Acquiring the necessary knowledge on the topics of many-core task migration [3] and power-density-based thermal analysis of many-core systems [1], [4].
- Getting acquainted with the framework OpenDSE [5], in particular, the post-DSE emulation of run-time platform dynamism.
- Implementing the power-density-based many-core thermal analysis from [1] in OpenDSE to be used for conducting thermal-aware migration decisions.
- Developing a heuristic to find promising active cores to be taken as the migration source and evaluating its performance wrt. a brute-force search. The result can also be provided as a sorted list of active cores ranked in the order of their contribution to a balanced temperature distribution.
- Developing a heuristic to find promising inactive cores to sever as the migration destination and evaluating its performance wrt. a brute-force search. Also here, the result can be provided as a sorted list of inactive cores ranked in the reverse order of their contribution to the on-chip temperature imbalance.
- Combining the two heuristics into a two-step migration policy and evaluating the performance gain and thermal balance achieved by the resulting policy against a brute-force search for the best migration option (pair of active and inactive cores).

- Preparation of the written thesis, documentation of all programs, and archiving of the relevant data in a repository or on a CD/DVD.
- The examination includes the written thesis and the final presentation.

1 Introduction

1.1 Motivation

For a long time, Moore's law and the Dennard scaling have dictated the change in the size and number of transistors in processors [6]. Moore's law proposes for every two year, a doubling of the number of transistors in a chip. Dennard scaling states the power density of a single transistor remains constant as the transistor size is reduced and, thus, the power consumption of a transistor decreases at the same rate as its area scales down. While decreasing the area is still possible today, the power consumption has reached its limitations, in particular, due to the limitations for supply voltage scaling. Highly packet transistors generate a lot of heat on a small space and, as a result, cause high temperatures. To prevent the high temperature from damaging the chip, the power consumption has to be limited [7]. In a single-core system, it becomes very challenging to manage the high power density and high temperatures, while also maintaining the performance demands of modern applications. Hence, chip manufactures changed their focus to the production of multi-core and many-core systems.

Many-core systems grant the possibility to run the individual cores on lower frequencies and voltages, therefore, using less power while delivering the same or even higher computational performance. On the other hand, in many-core systems, the performance demands of modern applications is satisfied by parallel computing on multiple cores instead of frequency scaling on a single core [8]. In future, the number of cores built into many-core systems is going to increase rapidly [9].

Because of the increase in the number of processing units (cores) and the power density in a modern many-core system, it is necessary to manage and control the power consumption, and in the bigger picture, the chip's temperature. The consideration of temperature most of the time introduces some restriction on the performance of the system or the amount of power it can use without getting too hot.

In a many-core systems, each application is composed of a set of tasks that can be executed in parallel so that applications can exploit the parallel processing potential of the system. So, system management is necessary to determine a suitable assignment of applications' tasks to platform's computational resources (cores). This assignment is usually done in a way to fulfill certain criteria like real-time constraints or a low power consumption. Sometimes, it becomes necessary or beneficial to modify the assignment of tasks to cores later on, for example, to move an application away from a faulty core. In these cases, usually techniques of *task migration* are used. Because task migration decisions are made at run time, it is possible to take into account the available information about the system's current state like the temperature and power consumption of each core

in the migration decisions, e.g. to improve the temperature distribution in the system.

1.2 Thesis Contribution and Outline

Contribution In this thesis, a method is proposed for thermal management of many-core systems using task migration. In the proposed method, the available information about temperature and power consumption of chip's cores is used to make task migration decisions in a way that the system achieves a better temperature balance. Moreover, combining thermal consideration with task migration into a temperature-aware task migration policy for many-core systems, is a key part of this thesis. For the proposed policy the many-core temperature analysis called Thermal Safe Power (TSP)-Analysis from [1] is used to develop a heuristic to make the needed task migration decisions. Task migrations considered in this work allow us to disable an active core and enable an idle core, which requires the migration of all tasks. A two-step procedure is used for the proposed migration policy to make the migration decisions: (1) selecting, from the active cores, the best candidate as the source for the migration, (2) selecting the best candidate among the idle cores as the migration destination. Both steps use the temperature and power-related values obtained from the TSP analysis. In the first step the active cores are ranked using their individual heat contribution to the system. The cores with higher rank contribute more heat to the hottest part of the system, hence, disabling them contributes most to the thermal balance. The first step of the proposed task migration policy returns a sorted list of active cores in which the first core is the best candidate for migration source and the last core is the worst candidate. In the second step, a sorted list of idle cores is computed using the information from the TSP analysis. In this sorted list, the cores with a higher rank are cooler and, therefore, better candidates to be switched on. Using these two lists, the user is flexible to choose the source and destination cores for a migration.

The evaluation of the heuristic and its comparison with a brute-force search confirms its viability for improving the thermal balance of a many-core system, but also for the impact of a migration on the system performance which is evaluated with the average frequency gain among active cores. The thesis also investigates whether a single TSP analysis is sufficient for selecting both source and destination cores or if two separate consecutive calculations are necessary where the second analysis (for ranking of destination cores) takes into account the selected migration source. The contribution of this thesis can be summarized as a temperature-aware task migration policy, which optimizes the thermal balance of the system while minimizing the computational overhead of the migration decision making process.

Outline The content of this thesis is organized as follows: In chapter 2, the background and fundamentals of thermal considerations is provided. In section 2.1, it is explained why thermal considerations are necessary, and the recent and current fundamental techniques for chip thermal management and control are presented. This is followed by introducing the concept of TSP analysis [1] which allows us to express temperature in terms of power consumption. This brings us to the formal analysis of thermal safe power, and

later, to the safe power density. Then, in section 2.2, an overview of task migration is provided and the general idea of the proposed temperature-aware task migration policy is introduced.

In chapter 3, the proposed temperature-aware task migration policy is presented with the TSP analysis from [1] as its core. Section 3.1 gives an overview of a black-box implementation of the TSP analysis the authors provide in [1], which is used as a reference to validate our implementation of the TSP analysis into the OpenDSE framework [5]. Next, in section 3.2, the pseudo code from [1] for TSP analysis is provided and our implementation into the OpenDSE framework is explained. Then a similar discussion is provided for the Safe Power Density (SPD) analysis in section 3.3. In section 3.4, the experiments environment and its parameters is set up, which is later used to compare the different search approaches. For task migration this brings us to section 3.5 for the first step in the task migration policy, the source core search. There, we present various baseline source-core selection approaches, e.g., extensive search and random selection, and present our proposed approach based on the TSP analysis. We also evaluate the performance of all approaches comparatively in our experiments environment to demonstrate the effectiveness of the proposed approach. Then, the second step of the policy, the destination core search, is discussed in section 3.6 where we establish the different methods to get a suitable migration destination. There, the proposed method for the destination search is in section 3.6.1 introduced, the updated implementation to the core ranking and list generation is shown, and at last, the validation for this new method is done by evaluating and comparing with the other methods.

The different search approaches for the two steps of the policy are then in section 3.7 combined and evaluated for a whole migration, then, compared for SPD gain and average frequency gain, which validated that our proposed method is cost efficient and has a optimal estimation quality. The last step for the policy is to compare different migration approaches, which are established and explained in section 3.8. There, different migration approaches including the proposed approach are then evaluated for SPD gain and average frequency gain. As the last part of our evaluation for the different migration methods, we evaluate the average frequency gain the methods can achieve using practical frequency values, i.e., a quantized frequency spectrum used in realistic systems. Chapter 4 presents the conclusions and provides some possible aspects of future work.

2 Background and Fundamentals

In this chapter, first, we look into the reasons why temperature must be considered in an embedded system. Following that, different approaches for thermal consideration in many-core systems are presented. First, the most basic approach for thermal management, a *single* and *constant* power constraint, is presented. Often times, the *Thermal Design Power (TDP)* is used as this power constraint, since it is provided by chip manufactures and states the highest expected sustainable power level the chip can run at [1]. However, TDP is not the maximum power a chip can use, especially for many-core systems, as a result, *Dynamic Thermal Management (DTM)* is introduced. Then, we present DTM which is a far more flexible approach for thermal management and can sufficiently handle many-core systems, but comes with the drawback of having a huge computational performance loss when triggered by anticipated or detected thermal violations. Thus, DTM is only used for preventing critical temperatures, and triggering it is most of the time avoided. To achieve this, modern systems use *Thermal Safe Power (TSP)* [1] which is presented afterwards. TSP enables calculating the optimal power consumption for each core based on a given critical temperature for the chip. It takes as input the list of the active and idle cores in the system and returns a power constraint. This power constraint is the maximum amount of power that each active core in the system is allowed to consume without the DTM being triggered. After the introduction of these topics in section 2.1, section 2.2 provides an overview on task migration.

2.1 Thermal Considerations

Each chip uses, for a specific load, a certain amount of power which heats up the system. If this is not controlled, the chip might reach critical temperatures and burn down. So, the first logical step to prevent damage is to direct the heat out of the system using a cooling subsystem or proper chip packaging. The highest power level which is expected to be sustainable, is called Thermal Design Power (TDP) and is provided by the chip manufacturer [1]. A chip with a certain TDP needs a cooling system which is specified to dissipate this amount of power to guarantee thermal safety. TDP is not the maximum power consumption a chip is able to use [1]. So, a higher power consumption than TDP increases the chip's temperature to a critical point, which damages the chip. DTM is used to prevent damage and chip burn-down and works based on a predefined threshold temperature. If any point of the chip reaches this threshold temperature, DTM is triggered which reduces the chip's power consumption by lowering the voltage and frequency of each core to a known safe limit or by power gating [1]. DTM, therefore, comes with a threat of loosing performance, but even more importantly, by having to consider this loss

when designing time-critical applications. With the performance drop due to DTM, or in other words, lowering the core frequency, the application is executed much slower and, as a result, its Worst-Case Execution Time (WCET) may increase. So, for an efficient system, we limit the number of times DTM is triggered by defining a power constraint. For a single core systems, a chip-level power constraint is sufficient, hence, TDP is often used, since it is by definition the highest expected safe power consumption [1]. But, with the increasing development of many-core systems, a single constant power constraint is no longer suitable [1]. For a many-core system a single constant power constraint, is split up among all cores, which results in a safe power consumption, if all cores are active. But if only a subset of cores are active, the individual active cores can consume more power and, by that, reach higher temperatures which triggers DTM. This makes single and constant power constraints inefficient for many-core systems.

Since DTM does not consider the actual system load and just operates based on the violation of the threshold temperature, we would have to keep track of the current temperature of the system to make sure we stay below the DTM and, as a result, prevent DTM counteractions. This requires a temperature analysis, which can be done with the TSP analysis [1]. The TSP analysis uses the system's current mapping (the distribution of active and inactive cores), the thermal properties of the chip's floorplan, and the properties of each thermal node to calculate a chip-wide core-level power constraint. Each active core can consume power up to this power constraint without any point on the chip exceeding the threshold temperature of DTM. Therefore, using TSP analysis, we can calculate a thermally safe power constraint for the active cores in the system and prevent DTM triggers by ensuring that each core does not exceed this threshold. This makes the thermal management of the system more robust and conservative with performance penalties.

Being able to calculate a thermally safe power constraint, we can use the TSP analysis to find a distribution of active cores in the system which results in a higher safe power constraint and, as a result, increases the amount of power that can be safely consumed by each core. This enables increasing the system performance by switching to those optimal active core distributions. Before looking into the TSP analysis, we must be familiar with the fundamentals of the other solutions first, starting with TDP as chip-level power constraint.

2.1.1 Single Constant Power Constraints

To best describe TDP when used as a single and constant power constraint, let us consider an example system with a single core. Assume the core consumes power P_{core} and produces temperature T_{core}. The connection between the power consumed by a core and the temperature resulting from it can be analyzed by a thermal analysis based on the duality of thermal and electrical circuits. The critical temperature T_{crit} marks the threshold temperature beyond which the chip will burn up or be damaged, that is, when $T_{core} > T_{crit}$. Each system has an own T_{crit}. Considering this threshold and the relation between temperature and power, we can set $T_{core} = T_{crit}$ and calculate the core power level P_{crit} that will result in this temperature. P_{crit} will be regarded as TDP.

Using TDP as our system-wide chip-level power constraint, we can calculate T_{TDP}. Since TDP is provided by the manufacturer as the highest sustainable power [1], we can assume $T_{TDP} \leq T_{crit}$. Because we assume a single core system, this one core uses all of the available power $P_{core} = P_{TDP}$. This specific power limit is the maximum power level the example system with a single core can maintain indefinitely.

2.1.2 Dynamic Thermal Management (DTM)

If we now make the step from a single-core system to many-core systems, the usefulness of a single and constant value like TDP as a chip-level power constraint is not longer guaranteed. Since we want to prevent damage to the system, we set the chip-level power constraint to the highest safe power for the worst case. In perspective of a many-core system, the worst case is when all cores are active, running at their highest frequency and demanding the most power. This power level is then divided among all active cores, in this case among all cores in the system. By using TDP as the power constraint, the power level of each core is $P_{core} = \frac{P_{TDP}}{\#cores}$, which is safe according to [1]. Assume now a system where only a subset of cores in the system are active. There, the individual power constraint for each core is proportionally higher the less active cores we have. The individual power consumption results in $\frac{P_{TDP}}{\#active\,cores} \geq \frac{P_{TDP}}{\#cores}$ and, therefore, is unsafe, since the active cores now use more power than they are designed for. So, a single and constant power constraint is not suitable for power regulation in many-core systems.

Dynamic Thermal Management (DTM) provides a solution to this problem by powering down the system before it gets too hot. It prevents the system from being destroyed, regardless of the amount of power being used, just by using the temperature of the system as a indication of how close the system is to the critical temperature. The critical temperature is different for each system, but for our evaluation in this thesis, let us consider a temperature of 80°C as the highest safe temperature with enough discrepancy to let the system run safely. DTM uses this threshold temperature as its trigger point and, if overstepped, reacts with countermeasures to reduce the temperature back into the safe region again. One way to cool the system down is to increase the heat flow of the system to the ambient by increasing the fan speed. If this option is not available (for example, if we only use a passive cooling solution) then the other option is to regulate the produced heat. This is done by decreasing the power consumption of the system by limiting the usage of each core. The total power used by a core can be calculated using eq. (2.1) as the sum of its static power consumption P_{static} and dynamic power consumption P_{dyn}. The static power consumption is dependent on the leakage current I_q and the supply voltage V_{dd}. The dynamic power depends on the switching activity of the core $0 \leq \alpha \leq 1$, its effective capacitance $\hat{C}_{eff}$, supply voltage V_{dd} and frequency f. By reducing frequency f, we can also reduce the supply voltage V_{dd}. This allows us to control both static and dynamic power consumption by changing the frequency and voltage of a core which is called *Dynamic Voltage and Frequency Scaling (DVFS)*. It is also possible to scale only the frequency without voltage scaling—called *Dynamic Frequency Scaling (DFS)*—, but in this thesis, we focus on DVFS as it is more commonly used.

$$\begin{aligned} P_{total} &= P_{static} &+ P_{dyn} \\ &= I_q \cdot V_{dd} &+ \alpha \cdot \hat{C}_{eff} \cdot V_{dd}^2 \cdot f \end{aligned} \tag{2.1}$$

DVFS and DFS also affects the performance of a core because of the change in frequency f. The performance of a core is represented by its *Instruction Per Second (IPS)* which is calculated using eq. (2.2). IPS is a product of the core's frequency f and its *Instruction Per Cycle (IPC)* and, thus, is directly dependent on the core's frequency. Hence, using DVFS or DFS always has an impact on the performance of the system, since we are reducing the frequency. But, in the case of DTM, we need to achieve a strong decrease in temperature abruptly, hence, the scaling factor is very high. This makes DTM, as an approach for temperature control uninteresting due to the huge performance loss.

$$IPS = f \cdot IPC \tag{2.2}$$

2.1.3 Thermal Safe Power (TSP)

DTM often results in high performance loss due to a hard DVFS or DFS that comes with the very conservative safe temperature it wants to reach quickly when triggered. To get a smoother transition, we could gradually reduce power consumption, but this might be too slow to maintain the thermal safety of the chip. An alternative solution is to calculate the critical power consumption for the current distribution of active and idle cores.

The *Thermal Safe Power (TSP)* analysis from [1] enables us to calculate a core-level power consumption constraint for the current distribution of active and idle cores. This power constraint is guaranteed never to trigger DTM if satisfied by each active core. This means, it allows us to have the highest possible power consumption per core which can be maintained indefinitely and, hence, gives us a stable computational performance as long as the mapping (distribution of active cores) stays unchanged. In our proposed temperature-aware task migration policy, we use the results of intermediate calculations in the TSP analysis to find the best candidate cores for migration source and destination. Therefore, we first present the TSP analysis from [1] and then explain how we use its intermediate results for migration decisions.

Hardware and Thermal Model

To be able to come from a thermal context to a power context, we have to transition from the architecture's hardware model to its thermal model, which enables us to connect power with temperature. The hardware model consists of Z hardware blocks consisting of M cores and $M' = Z - M$ non-core blocks, e.g. caches. The thermal model consists of $N >= Z$ thermal nodes, representing each core and other elements of heat transfer, like the cooling system. With this model, we can calculate the thermal flow in a circuit. Now, we connect these models based on the duality of thermal and electrical circuits which gives us the following differential equation:

$$\mathbf{A}\mathbf{T}' + \mathbf{B}\mathbf{T} = \mathbf{P} + T_{\text{amb}}\mathbf{G} \tag{2.3}$$

In eq. (2.3) matrix $\mathbf{A} = [a_{i,j}]_{N\times N}$ contains the thermal capacitance of each node, matrix $\mathbf{B} = [b_{i,j}]_{N\times N}$ the thermal conductance of each node to each other node, column vector $\mathbf{T} = [T_i]_{N\times 1}$ represents the temperature of each node, column vector $\mathbf{T'} = [T_i']_{N\times 1}$ accounts for the first order derivative of the temperature of each node with respect to time, column vector $\mathbf{P} = [p_i]_{N\times 1}$ contains the power consumption of each node, and column vector $\mathbf{G} = [g_i]_{N\times 1}$ contains the thermal conductance between each node and the ambient. Since we are only interested in the steady state temperature, we can consider $\mathbf{T'} = 0$ and transform eq. (2.3) to eq. (2.4).

$$\mathbf{T} = \mathbf{B}^{-1}\mathbf{P} + T_{\text{amb}}\,\mathbf{B}^{-1}\mathbf{G} \tag{2.4}$$

$\mathbf{P}$ can also be split into multiple parts according to the hardware model, i.e., $\mathbf{P} = \mathbf{P}^{\text{cores}} + \mathbf{P}^{\text{blocks}} + \mathbf{P}^{\text{internal}}$ where $p_i^{\text{internal}} = 0, \forall i \in N$. As a result, we get eq. (2.5).

$$\mathbf{T} = \mathbf{B}^{-1}\mathbf{P}^{\text{cores}} + \mathbf{B}^{-1}\mathbf{P}^{\text{blocks}} + \mathbf{B}^{-1}T_{\text{amb}}\mathbf{G} \tag{2.5}$$

This model gives us the ability to calculate the temperature of a system as a function of its power usage. We can calculate each temperature element T_i of the $\mathbf{T}$ matrix. T_i gives us the temperature of thermal node i and is calculated using eq. (2.6).

$$T_i = \sum_{j=1}^{N} b_{i,j}^{-1} \cdot p_j^{\text{cores}} + \sum_{j=1}^{N} b_{i,j}^{-1} \left(p_j^{\text{blocks}} + T_{\text{amb}} \cdot g_j \right) \tag{2.6}$$

With this, we have a function to calculate the temperature of each node in the system based on the power consumption of different nodes in the system and the constant thermal parameters of the chip.

Thermal Safe Power (TSP) Calculation

Now we have the fundamental formula to calculate the steady state temperature for each core and non-core in the architecture by using its defined variables $\mathbf{B}$, $\mathbf{A}$, $\mathbf{G}$, $\mathbf{P}^{\text{blocks}}$, $\mathbf{P}^{\text{cores}}$ and T_{amb}. With this, we can describe the problem of safe power constraint as introduced in [1]. TSP is a system-wide core power constraint for a given mapping of active cores, which can be used by each core without triggering DTM. The mapping of the system is defined as the binary vector $\mathbf{Q} = [q_i]_{M\times 1}$, where $q_i = 1$ if core i is active whereas $q_i = 0$ if core i is idle. Using the mapping vector $\mathbf{Q}$, [1] calculates core-level power constraint P_{TSP} using eqs. (2.7) to (2.9).

$$P_{\text{TSP}}(\mathbf{Q}) = \begin{cases} P_{\text{TSP}}^{\star}(\mathbf{Q}) & \text{if} \quad P_{\text{TSP}}^{\star}(\mathbf{Q}) \leq R\left(\sum_{i=1}^{M} q_i\right) \\ R\left(\sum_{i=1}^{M} q_i\right) & \text{otherwise.} \end{cases} \tag{2.7}$$

$$P^{\star}_{\mathrm{TSP}}(\mathbf{Q}) = \min_{\forall i \in L} \left\{ \frac{T_{\mathrm{DTM}} - P^{\mathrm{core}}_{\mathrm{inact}} \cdot \sum_{j=1}^{M} b^{-1}_{i,k_j}(1-q_j)}{\sum_{j=1}^{M} b^{-1}_{i,k_j} \cdot q_j} + \frac{-\sum_{j=1}^{N} b^{-1}_{i,j}(p^{\mathrm{blocks}}_{j} + T_{\mathrm{amb}} \cdot g_j)}{\sum_{j=1}^{M} b^{-1}_{i,k_j} \cdot q_j} \right\} \tag{2.8}$$

$$R(m) = P^{\mathrm{core}}_{\mathrm{inact}} + \frac{P_{\max} - \sum_{i=1}^{N} p^{\mathrm{blocks}}_{i} - P^{\mathrm{core}}_{\mathrm{inact}} \cdot M}{m} \tag{2.9}$$

The proof in [1] shows that eqs. (2.7) to (2.9) result in a system-wide core-power limit P_{TSP} which guarantees that the many-core system can run indefinitely, with the highest possible power consumption and, thus, performance, without triggering DTM.

2.1.4 Thermal Safe Power Density

The previous formulas only use a fixed power value for all inactive cores $P^{\mathrm{core}}_{\mathrm{inact}}$. But, in most scenarios the system is heterogeneous and contains multiple types of cores which are suitable for different types of applications. Cores from different types have different power consumption levels when they are inactive, but they also have a different die area. Therefore, the previous formulas have to be adapted to support heterogeneous systems. The inactive power of different core types can be integrated by using $p^{\mathrm{core}}_{\mathrm{inact},\,j}$ for the inactive power of core j where $j = 1,\ldots,M$. Also the area has to be integrated into the formulas by switching context from power to power density. Hence, we use the notation P^{ρ}_{TSP} as the safe power-density constraint and use $area^{\mathrm{core}}_{j}$, for the area of core j where $j = 1,\ldots,M$. This leads to eqs. (2.10) to (2.12), which result in a thermal *Safe Power Density (SPD)*. The proof that eqs. (2.10) to (2.12) fulfilling the thermal constraints is provided in [1]. We can now use these formulas to calculate $P^{\rho}_{\mathrm{TSP}}(\mathbf{Q})$ for a given mapping $\mathbf{Q}$, which can be multiplied by the area of a specific core to get its thermal safe power level.

$$P^{\rho}_{\mathrm{TSP}}(\mathbf{Q}) = \begin{cases} P^{\rho\star}_{\mathrm{TSP}}(\mathbf{Q}) & \text{if} \quad P^{\rho\star}_{\mathrm{TSP}}(\mathbf{Q}) \leq R^{\rho}(\mathbf{Q}) \\ R^{\rho}(\mathbf{Q}) & \text{otherwise.} \end{cases} \tag{2.10}$$

$$P^{\rho\star}_{\mathrm{TSP}}(\mathbf{Q}) = \min_{\forall i \in L} \left\{ \frac{T_{\mathrm{DTM}} - \sum_{j=1}^{M} b^{-1}_{i,k_j} \cdot p^{\mathrm{core}}_{\mathrm{inact},\,j}(1-q_j)}{\sum_{j=1}^{M} b^{-1}_{i,k_j} \cdot area^{\mathrm{core}}_{j} \cdot q_j} + \frac{-\sum_{j=1}^{N} b^{-1}_{i,j}(p^{\mathrm{blocks}}_{j} + T_{\mathrm{amb}} \cdot g_j)}{\sum_{j=1}^{M} b^{-1}_{i,k_j} \cdot area^{\mathrm{core}}_{j} \cdot q_j} \right\} \tag{2.11}$$

$$R^{\rho}(\mathbf{Q}) = \frac{P_{\max} - \sum_{i=1}^{N} p^{\mathrm{blocks}}_{i} - \sum_{j=1}^{M} p^{\mathrm{core}}_{\mathrm{inact},\,j}(1-q_j)}{\sum_{j=1}^{M} area^{\mathrm{core}}_{j} \cdot q_j} \tag{2.12}$$

2.2 Task Migration for Thermal Management

In this thesis, the TSP analysis presented in sections 2.1.3 and 2.1.4 is used to develop a temperature-aware task migration policy. Thus, in the following, an overview of task migration and its application for thermal management is given before we present the main contribution of the thesis in chapter 3.

Task migration is the procedure of moving a task running on one core to another core. The task itself is part of an application, which is mapped to a many-core architecture. Because of the continuous changes in the workload of many-core systems, the initial mapping of an application might not be the optimal mapping for all situations, and, as a result, might have to be altered. If the need for a migration is there (e.g., an overheating situation) and the system allows a reconfiguration (e.g., no real-time constraints are violated), the current state has to be analyzed and an optimal migration destination has to be found. If a destination has been found and the processor has enough resources for the migration process, the task can be migrated. After selecting the migrating task and its destination, the current state of the task must be saved in the memory and an optimal route to the destination core must be chosen on which the data is transferred from the source memory to the destination memory. Then, the task state is loaded from the memory to the new core and all future references to this task can be redirected to the new position. The procedure explained above is provided just for completeness of presentation. The migration policy presented in this thesis focuses on the decision making part of the migration process and is independent of how the migration is performed.

Task migration can be used to improve the system with respect to different aspects, for example, to increase the system's performance, or to reduce the power consumption by moving all tasks to low power cores. In this thesis, the main goal is to redistribute active cores in the system for a better thermal balance. In general, a core can be used to execute multiple tasks. Because a change in the thermal balance of the system depends on a change in the location of active cores, for thermal management, we must perform migrations in a way that the distribution of active cores is changed. So, we consider migrations in which all tasks running on one active core are moved to an idle core so that the first core (migration source) can become idle and the second core (migration destination) becomes active. Based on this scheme, we study the effectiveness of task migration for thermal management and propose a temperature-aware task migration policy for the thermal management of many-core systems based on the TSP analysis.

3 Temperature-Aware Task Migration Policy

In this chapter, we propose a temperate-aware task migration policy for thermal management of many-core. In the proposed method, the available information about temperature and power consumption of chip's cores is used to make task migration decisions in a way that the system achieves a better temperature balance. For this purpose, we modify the TSP/SPD analysis from [1]. Therefore, in the following, we look into the implementation of the TSP analysis with the goal to obtain enough information from it to calculate, for a given mapping, a nearly optimal pair of source and destination cores for migration. The first step of this process is to implement the TSP analysis and validate the correctness of our implementation. We use the OpenDSE framework [5] for our implementation. "OpenDSE is a design space exploration framework for embedded systems, written in Java. It follows the Y-chart approach where an application consisting of data-dependent tasks is mapped to an architecture consisting of resources" [5]. Particularly, we use OpenDSE for its system model and to perform experiments emulating the run time of a many-core system for evaluation of our techniques and for the validation of our assumptions. Thus, as part of this thesis, the TSP analysis from [1] is implemented in Java. In the following, the pseudo-code of TSP from [1] is presented, and the implementation of it in OpenDSE is explained using small snippets of the code.

3.1 Reference TSP Analysis

A black box implementation of the TSP analysis from [1] is available. This black-box implementation calculates the thermal safe power value for any given mapping (distribution of active and inactive cores in the system). The necessary inputs describing the thermal characteristics of the many-core chip are provided by input files for the TSP analysis. The following inputs are provided:

1. Matrix $\mathbf{B} = [b_{i,j}]_{N\times N}$ gives the thermal conductance of each node to each other node in the system and is provided in input file `52core.out`.

2. Vector $\mathbf{G} = [g_i]_{N\times 1}$ gives the thermal conductance of each node in the system to the ambient and is provided in input file `52core.out`.

3. Vector $\mathbf{P}_{\text{inact}}^{\text{types}} = [p_{\text{inact},i}^{\text{types}}]_{T\times 1}$, with $T =$ #core-types gives the power consumption of inactive cores from a certain core type and is provided in file `inactivePowers.trace`.

4. Vector $\mathbf{Area}_{\text{core}}^{\text{types}} = [area_{\text{core},i}^{\text{types}}]_{T\times 1}$, with T = #core-types gives the die area of a core from a certain core type and is provided in file `areas.trace`.

5. Vector $\mathbf{Count}_{core}^{\text{type}} = [count_{\text{core},i}^{\text{type}}]_{T\times 1}$, with T = #core-types gives the number of cores for a specific core type and is provided in file

6. Vector $\mathbf{Q} = [q_i]_{M\times 1}$, with M = #cores which gives the state of a core when active as $q_i = 1$ and idle as $q_i = 0$ and is provided in input file `givenMapping.map`.

We used this pre-compiled executable as the golden reference for the validation of our implementation of the TSP analysis in OpenDSE in Java. A partial implementation of the TSP analysis is provided as open-source code by the authors on their website. We use this partial implementation as a reference to see how the necessary inputs of the analysis must be constructed from the content of the files.

3.2 Homogeneous TSP Calculation

For the implementation of the TSP analysis, the Pseudo code shown in algorithm 1 and the equations presented in section 2.1.3 are used which are adopted from [1]. The implementation of this pseudo-code in OpenDSE is straightforward. For that, the `class TSPAnalyzer` is created, which implements the method `double getThermalSafePower(Mapping mapping)`. This method gets as input a mapping defined as the binary vector $\mathbf{Q} = [q_i]_{M\times 1}$, where $q_i = 1$ if core i is active whereas $q_i = 0$ if core i is idle. The method calculates and returns the thermal safe power as a double value. In the first step of our implementation, all variables and necessary inputs are parsed from their files and, then, are validated using method `checkInitialized()`. To ensure consistency, we also needed a method to construct a correct binary vector $\mathbf{Q}$ with correct location for each core from a given set of active cores provided from OpenDSE. The main part of the TSP analysis is the calculation of the auxiliary power value *auxP*, lines 3–5 in algorithm 1. This value represents, for each core, the power level it would be able to maintain thermally safe, if all the other active cores use at most the same amount of power. The value auxP is calculated by adding up all of the temperature contribution from the inactive cores, other non-core blocks, and the ambient to the core under analysis and, then, subtract this value from the maximum allowed temperature, in this case T_{DTM}. This gives us a temperature constraint which we can translate into a power constraint by dividing it by the heat contribution from all active cores as shown in line 5 of algorithm 1. The resulting power constraint ensures that the current core remains under the threshold temperature. The calculation of auxP is done as shown in lines 2–15 in listing 3.1. Listing 3.1 has three loops, the first, in line 2, iterating over all blocks in the system, the second, in line 5, iterating over all cores in the system and the third, in line 10, iterating over all thermal nodes in the system. Due to N = #nodes $\geq Z$ = #blocks $\geq M$ = #cores is the third loop more important to complexity and covers the second loop. As a result, our calculation of auxP has a polynomial complexity of $\mathcal{O}(ZN)$, with Z = #blocks and N = #nodes which is also provided in [1].

Algorithm 1: Pseudo code from [1] for Thermal Safe Power (TSP) analysis returning a TSP constraint for a given mapping $\mathbf{Q}$

Input: $\mathbf{Q}, \mathbf{P}^{\text{blocks}}, T_{\text{amb}}, T_{\text{DTM}}, P^{\text{core}}_{\text{inact}}, P_{\max}$, and floorplan;
Output: Uniform TSP power constraint for mapping $\mathbf{Q}$;

1: $P^{\star}_{\text{TSP}}(\mathbf{Q}) \leftarrow \infty$;
2: **for** *all* $i \in L$ **do** {Computes $P^{\star}_{\text{TSP}}(\mathbf{Q})$ according to eq. (2.8)}
{Subtract temperature generated by inactive cores from threshold temperature}
3: $auxP \leftarrow T_{\text{DTM}} - P^{\text{core}}_{\text{inact}} \cdot \sum_{j=1}^{M} b^{-1}_{i,k_j}(1 - q_j)$;
{Subtract temperature generated by other non-core blocks in the system and the ambient temperature from auxP}
4: $auxP \leftarrow auxP - \sum_{j=1}^{N} b^{-1}_{i,j}(p^{\text{blocks}}_{j} + T_{\text{amb}} \cdot g_j)$;
{Divide auxP by the heat-contribution of active cores}
5: $auxP \leftarrow \frac{auxP}{\sum_{j=1}^{M} b^{-1}_{i,k_j} \cdot q_j}$;
6: **if** $auxP < P^{\star}_{\text{TSP}}(\mathbf{Q})$ **then** {Search for the minimum auxP and set $P^{\star}_{\text{TSP}}(\mathbf{Q})$ to the result}
7: $P^{\star}_{\text{TSP}}(\mathbf{Q}) \leftarrow auxP$;
8: **end**
9: **end**
10: **if** $P^{\star}_{\text{TSP}}(\mathbf{Q}) \leq R\left(\sum_{i=1}^{M} q_i\right)$ **then** {Compute $P_{\text{TSP}}(\mathbf{Q})$ according to eq. (2.7)}
11: $P_{\text{TSP}}(\mathbf{Q}) \leftarrow P^{\star}_{\text{TSP}}(\mathbf{Q})$;
12: **else**
13: $P_{\text{TSP}}(\mathbf{Q}) \leftarrow R\left(\sum_{i=1}^{M} q_i\right)$;
14: **end**
15: **return** $P_{\text{TSP}}(\mathbf{Q})$;

To find a uniform thermally safe power constraint for all active cores in the system, we must find the smallest auxP value among all blocks in the system, lines 6 – 9 in algorithm 1. This auxP value gives us a core power level which all active cores can maintain without exceeding the critical temperature so that DTM is not triggered. Finding this critical auxP value is done as shown in line 16 – 18 in listing 3.1 inside of the first loop, by saving only the minimum value of auxP among all cores.

Each system has an upper bound on the amount of power that can be consumed by the whole chip, represented by $P_{\max}$ and dictated by the power delivery subsystem. To make sure that we always stay within the power range permitted by $P_{\max}$, we use eq. (2.9), which calculates the sum of all the upper bound of the power that can be consumed by each block in the system and compares it with $P_{\max}$. If the calculated power constraint can result in a violation of $P_{\max}$, the constraint is adjusted to respect $P_{\max}$. Once a safe power constraint is obtained, we convert the unit from W/mm^2 to W/m^2 for compatibility with the units in OpenDSE, because in OpenDSE area is assumed to be given in m^2 while the calculation is TSP analysis are based on mm^2. The implementation of this refinement is shown in listing 3.2. The result of this calculation will be the thermal safe power for the given mapping, i.e., $P_{\text{TSP}}(\mathbf{Q})$, in W/m^2 which is returned as output.

```
2   for (int i = 0; i < n_blocks; i++) {
3     double heatContributionInactiveCores = 0;
4     double heatContributionActiveCores = 0;
5     for (int j = 0; j < n_cores; j++) {
6       heatContributionActiveCores += bInvMatrix[i][j] *
            mapping.isActive(j);
7       heatContributionInactiveCores += bInvMatrix[i][j] * (1 -
            mapping.isActive(j));
8     }
9     double heatBlocksAndAmbient = 0;
10    for (int j = 0; j < numberThermalNodes; j++) {
11      heatBlocksAndAmbient += bInvMatrix[i][j] * (pBlocks[j] + t_amb
            * gVector[j]);
12    }
13    double auxP = t_dtm - p_inact_core *
          heatContributionInactiveCores;
14    auxP = auxP - heatBlocksAndAmbient;
15    auxP = auxP / heatContributionActiveCores;
16    if (auxP < PworstStar) {
17      PworstStar = auxP;
18    }
19  }
```

Listing 3.1: The code snippet for calculating *auxP* and $P^{\star}_{\mathrm{TSP}}(\mathbf{Q})$ in our implementation of the TSP analysis for a homogeneous many-core system. The snippet corresponds to lines 2 – 9 in algorithm 1.

```
20  maxTSP = p_inact_core + (p_max - totalPowerBlocks - p_inact_core *
        M) / m;
21  if (PworstStar <= maxTSP) {
22    tspValue = PworstStar;
23  } else {
24    tspValue = maxTSP;
25  }
26  tspValue *= 1000 * 1000;
27  return tspValue;
```

Listing 3.2: The code snippet for calculating $R\left(\sum_{i=1}^{M} q_i\right)$ and $P_{\mathrm{TSP}}(\mathbf{Q})$ in our implementation of the TSP analysis for a homogeneous many-core system. The snippet corresponds to lines 10 – 15 in algorithm 1.

The TSP analysis implementation above enables us to calculate, for any given mapping, a thermally safe core power constraint. This value can also be used as a measure of the *thermal balance* in the system. Given two systems with the same number of active cores but different safe power constraints, the active cores in the system with a lower power constraint are located close to each other so that at least one point in that region reaches the critical temperature with a lower power consumption per core, compared to the other system. Whereas, in the system with a higher power constraint, the active cores

Algorithm 2: Pseudo code from [1] for a Safe Power Density (SPD) analysis returning SPD constraint for a given mapping $\mathbf{Q}$

Input: $\mathbf{Q}, \mathbf{P}^{\text{blocks}}, T_{\text{amb}}, T_{\text{DTM}}, p^{\text{core}}_{\text{inact},\,j}$ for $j = 1, 2, \ldots, M, P_{\max}$, and floorplan;
Output: Uniform SPD constraint for mapping $\mathbf{Q}$;

1: $P^{\rho\star}_{\text{TSP}}(\mathbf{Q}) \leftarrow \infty$;
2: **for** *all* $i \in L$ **do** {Computes $P^{\rho\star}_{\text{TSP}}(\mathbf{Q})$ according to eq. (2.8)}
{Subtract temperature generated by inactive cores from threshold temperature}
3: $auxP \leftarrow T_{\text{DTM}} - \sum_{j=1}^{M} b^{-1}_{i,k_j} \cdot p^{\text{core}}_{\text{inact},\,j}(1 - q_j)$;
{Subtract temperature generated by other non-core blocks in the system and the ambient temperature from auxP}
4: $auxP \leftarrow auxP - \sum_{j=1}^{N} b^{-1}_{i,j}(p^{\text{blocks}}_{j} + T_{\text{amb}} \cdot g_j)$;
{Divide auxP by the heat-area-contribution of active cores}
5: $auxP \leftarrow \frac{auxP}{\sum_{j=1}^{M} b^{-1}_{i,k_j} \cdot area^{\text{core}}_{j} \cdot q_j}$;
6: **if** $auxP < P^{\rho\star}_{\text{TSP}}(\mathbf{Q})$ **then** {Searches for the minimum auxP and sets $P^{\rho\star}_{\text{TSP}}(\mathbf{Q})$ to the result}
7: $P^{\rho\star}_{\text{TSP}}(\mathbf{Q}) \leftarrow auxP$;
8: **end**
9: **end**
10: **if** $P^{\rho\star}_{\text{TSP}}(\mathbf{Q}) \leq R^{\rho}(\mathbf{Q})$ **then** {Computes $P^{\rho}_{\text{TSP}}(\mathbf{Q})$ according to eq. (2.10)}
11: $P^{\rho}_{\text{TSP}}(\mathbf{Q}) \leftarrow P^{\rho\star}_{\text{TSP}}(\mathbf{Q})$;
12: **else**
13: $P^{\rho}_{\text{TSP}}(\mathbf{Q}) \leftarrow R^{\rho}(\mathbf{Q})$;
14: **end**
15: **return** $P^{\rho}_{\text{TSP}}(\mathbf{Q})$;

are distributed more evenly over the chip which minimizes their thermal impact on each other and allows each of them to use more power before the system reaches the critical temperature. So, given two systems with the same number of active cores, the system with a higher safe power constraint can be considered to have a better thermal balance.

3.3 Heterogeneous TSP Calculation

The TSP analysis implementation from section 3.2 provides an algorithm to calculate a thermally safe core power constraint for any given mapping (distribution of active and inactive cores). The TSP analysis is based on algorithm 1 and eqs. (2.7) to (2.9) from [1]. Equations (2.7) to (2.9) and algorithm 1 calculate for a homogeneous system (system with a single core type) the thermal safe power. For heterogeneous systems (systems with different core types), we adapt the previous algorithm. [1] provides eqs. (2.10) to (2.12) and algorithm 2 to calculate a Safe Power Density (SPD) for any given mapping $\mathbf{Q}$. Equations (2.10) to (2.12) consider the inactive power consumption and the area of the different core types. This translates the thermal safe power to a thermal safe power density and, therefore, leads to a core-level power density constraint. Algorithm 2 shows in line 3 the consideration of the inactive power for different core types and in line 5 the consideration of the die area for different core types. Similarly to algorithm 1, algorithm 2 has a complexity of $\mathcal{O}(ZN)$ with $Z = \#\text{blocks}$ and $N = \#\text{nodes}$.

We add the area and inactive power for different core types to our implementation. Listing 3.3 shows in line 6 and 9 with the method `double getCoreArea(int index)` the usage of the die areas for different core types. Listing 3.3 also shows in line 7 and 10 with the method `double getCoreStaticPower(int index)` the usage of the inactive power for different core types. Variable `double areaContribution` in line 9 and variable `inactivePowerContribution` in line 10 from listing 3.3 are used in line 24 in listing 3.4 to calculate $R^{\rho}(\mathbf{Q})$ as in eq. (2.12). The data for the variables $p^{\text{core}}_{\text{inact},j}$ and $area^{\text{core}}_{j}$ are loaded via the methods `double getCoreStaticPower(int index)` and `double getCoreArea(int index)`. The two methods use the index of a core in the floorplan, convert the index into the corresponding resource of OpenDSE and then return the respective value for inactive power or area for this resource. The values gathered from OpenDSE with the two methods are validated against the values loaded from the files. In our listing 3.3, the variable `double heatContributionActiveCores` uses the method `getCoreArea(j)` and the variable `double heatContributionInactiveCores` uses the method `getCoreStaticPower(j)`, hence, the unit of the variable `double auxP` changes from power to power density. Keeping track of the minimum auxP value is done in the same way as the previous listing 3.1. Calculating the upper limit $R^{\rho}(\mathbf{Q})$ to $P^{\rho}_{\text{TSP}}(\mathbf{Q})$ also changed to power density, therefore, we use `totalPowerBlocks` and `inactivePowerContribution`. We apply the limit similar to the previous algorithm and ensure not to violate $P_{\max}$. The unit of the safe power density constraint is also affected by the additional division by the area, but, since we have to multiply the area again to get a explicit power value for a core, the areas cross each other out if we use the same source for the area values. Thus, we can use the same unit conversion as in the previous listing 3.2.

With this, we have implemented an algorithm to calculate the thermal safe power density constraint for any mapping in a heterogeneous many-core system. This algorithm can be used to compare different mappings by their safe power density. A mapping with a higher safe power density constraint offers a more balanced distribution of heat in the system compared to one with a lower constraint and is, as a result, preferred in a comparison. Listing 3.3 has three loops, the first, in line 2, iterating over all blocks in the system, the second, in line 5, iterating over all cores in the system and the third, in line 14, iterating over all thermal nodes in the system. Due to $N = \#\text{nodes} \geq Z = \#\text{blocks} \geq M = \#\text{cores}$ is the third loop more important to complexity and covers the second loop. This results in a complexity of $\mathcal{O}(ZN)$, which is the same complexity as algorithm 2 provides, so, we have a polynomial implementation of SPD from [1] in the OpenDSE framework.

```
2   for (int i = 0; i < n_blocks; i++) {
3     double heatContributionInactiveCores = 0;
4     double heatContributionActiveCores = 0;
5     for (int j = 0; j < n_cores; j++) {
6       heatContributionActiveCores += bInvMatrix[i][j] *
            mapping.isActive(j) * getCoreArea(j);
7       heatContributionInactiveCores += bInvMatrix[i][j] * (1 -
            mapping.isActive(j)) * getCoreStaticPower(j);
8       if (i == 0) {
9         areaContribution += mapping.isActive(j) * getCoreArea(j);
10        inactivePowerContribution += (1 - mapping.isActive(j)) *
              getCoreStaticPower(j);
11      }
12    }
13    double heatBlocksAndAmbient = 0;
14    for (int j = 0; j < numberThermalNodes; j++) {
15      heatBlocksAndAmbient += bInvMatrix[i][j] * (pBlocks[j] + t_amb
            * gVector[j]);
16    }
17    double auxP = t_dtm - heatContributionInactiveCores;
18    auxP = auxP - heatBlocksAndAmbient;
19    auxP = auxP / heatContributionActiveCores;
20    if (auxP < PworstStar) {
21      PworstStar = auxP;
22    }
23  }
```

Listing 3.3: The code snippet for calculating *auxP* and $P_{\text{TSP}}^{\rho\star}(\mathbf{Q})$ in our implementation of the SPD analysis for a heterogeneous many-core system. The snippet corresponds to lines 2 – 9 in algorithm 2.

```
24  maxSPD = (p_max - totalPowerBlocks - inactivePowerContribution) /
        areaContribution;
25  if (PworstStar <= maxSPD) {
26    spdValue = PworstStar;
27  } else {
28    spdValue = maxSPD;
29  }
30  spdValue *= 1000 * 1000;
31  return maxSPD;
```

Listing 3.4: The code snippet for calculating $R^{\rho}(\mathbf{Q})$ and $P_{\text{TSP}}^{\rho}(\mathbf{Q})$ in our implementation of the SPD analysis for a heterogeneous many-core system. The snippet corresponds to lines 10 – 15 in algorithm 2.

Figure 3.1: The heterogeneous 52 core system, which is used as evaluation environment. The cores are sorted by their core type, from cores with a big die area to cores with a small die area. Then they are sorted from left to right and from bottom to the top. This order is used by all system defining matrices and vectors.

3.4 Experiment and Evaluation Environment

The evaluation of the different approaches for the upcoming heuristic of our temperature-aware task migration policy is based on a heterogeneous many-core chip with 52 cores of three different core types. The evaluation environment has a floorplan as shown in fig. 3.1. The floorplan shows the different core types (AMD K6-III, IBM PowerPC and AMD K6-2), which are clustered through sorting from big to small cores, from left to right and from bottom to top. This sorting ensures correct indices for the algorithms. In our experiments we assume $T_{\mathrm{DTM}} = 80°\mathrm{C}$, $P_{\max} = 400\mathrm{W}$ and $T_{\mathrm{amb}} = 45°\mathrm{C}$. The variables for core area and inactive power are received from calls to `double getCoreArea(int index)` and `double getCoreStaticPower(int index)`. The matrices and vectors $\mathbf{B}$, $\mathbf{G}$ and $\mathbf{P}^{\mathrm{blocks}}$ are loaded from files, which were sorted according to the floorplan. The mapping $\mathbf{Q}$ is provided through a parameter as a collection of OpenDSE `Resources`, consisting only of the active cores in the mapping this collection is then used to construct a binary vector $\mathbf{Q}$ of all cores in the system where $q_i = 0$ if core i is idle and $q_i = 1$ if core i is active.

The different approaches for the upcoming source core search and destination core search are all dependent on the provided mapping $\mathbf{Q}$, especially the number of active cores in this mapping or, in other words, the system load with the current mapping. Since our floorplan has 52 cores, the range of the system load is from 0 to 52 active cores, but for a system with 50% load or 26 active cores and 26 idle cores we have $\binom{52}{26} \approx 495 \cdot 10^{12}$ different mappings. Because the amount of possible mappings is too big to cover, we decided to look at a subset of 5000 mappings. The mappings are selected randomly from the set of possible mappings with a specific number of active cores. The amount of possible mappings with a low or high number of active cores is smaller than the amount of possible mappings with 26 active cores, since the amount of possible mappings is

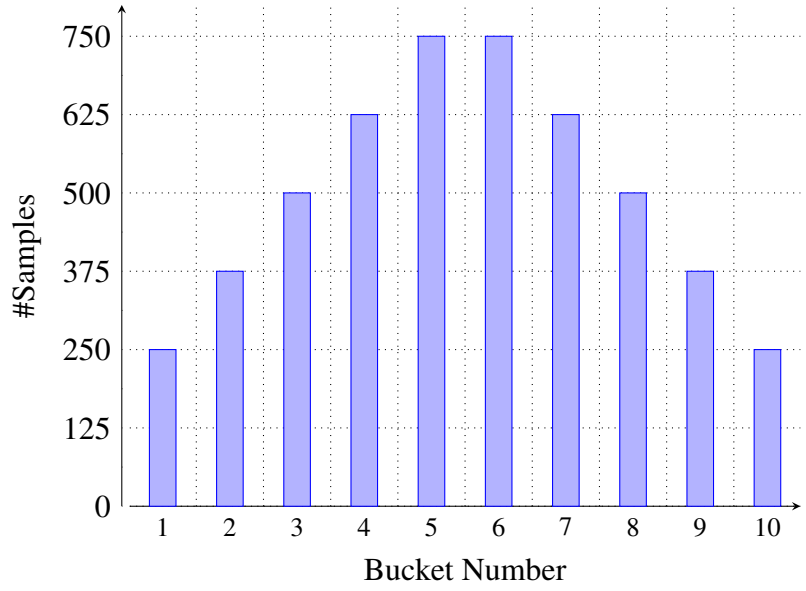

Figure 3.2: The distribution of 5000 samples in total, where the base sample amount is $\frac{5000/2}{\#buckets} = 250$ and an additional sample amount of $\sum_{i=0}^{I} \frac{5000/2 \cdot 2}{\#buckets} \cdot i$ for the bucket with index I.

given by $\binom{52}{\#\text{active cores}}$. We, therefore, define a pyramid scheme for the distribution of the amount of samples gathered. For a better readability of the plots for different approaches, we decided to have 10 sub-ranges of system load called *buckets* for example, one bucket corresponds to system loads of up to six cores, the next bucket to system loads of 7-11 cores, and so on. A single bucket collects for its sub-range a number of mapping samples, where the system load of a sample is randomly selected from the sub-range. We apply the pyramid scheme for sample distribution to the concept of buckets, which leads to a distribution as shown in fig. 3.2. Figure 3.2 divides the 5000 total samples by two, where the first half is used as a base amount of samples equally divided among all buckets and the second half stacked on the buckets in a pyramid scheme. This sample distribution provides more samples where more mappings are possible, but still has a base amount for each bucket.

3.5 Source Core Selection for Task Migration

Our proposed task-migration policy needs, for the selection of a suitable source core, a heuristic with the goal to increase the thermal balance of the system. We use the SPD analysis from [1] as the heuristic. For the source core search, our goal is to find the active core that results in the biggest increase in the safe power density if it is disabled. So, the possible candidates are the cores which can be disabled, so the active cores for the current mapping. We rank the available candidates based on the resulting power constraint from the SPD analysis. Assuming we choose two different core candidates from the same mapping, we generate, for each candidate, a new mapping where the respective

candidate is disabled. We calculate the two safe power density constraints for the two different resulting mappings. If the power density constraint of the first resulting mapping is greater than the power density constrain of the second mapping, we consider the first power density constraint to be better, because disabling the first core in the mapping increases the safe power density more than disabling the second core. Thus, the first core is a better candidate for the migration source. The gain in safe power density achieved by disabling the selected core is the gain we get in thermal balance.

We call the approach of comparing all core candidates with the heuristic above and selecting the best option *extensive search*. Extensive search does for every source core candidate one SPD analysis. We can reduce the number of SPD analyses by using *random selection*. Random selection considers only a random subset of the core candidates. The selection of the best candidate from this random subset is done similarly to the extensive search. The *proposed method* we propose uses the internal results of the SPD analysis to find the best candidate. In the following, we explain the three different methods and evaluate the quality of the proposed method and the random selection with respect to the extensive search.

3.5.1 Extensive Search

The extensive search is the brute-force approach among the three different methods. The possible migration source core candidates are the active cores in the system. We calculate for each core candidate a new mapping where the respective core candidate is disabled and use the SPD analysis from [1] to get the new safe power density constraints for each resulting mapping. The generated mappings are ranked by the safe power density and the mapping with the highest safe power density is the mapping where the best core candidate was disabled. This best core candidate is selected for the migration source. This thorough analysis gives us the possibility of finding the best candidate, but also the worst candidate. We later use the information of the worst candidate as a reference for the evaluation. By utilizing the SPD analysis for each core candidate, the extensive search results in a high resource and time overhead and, hence, is not practical to be used in a real chip. From [1], we know that the complexity of the SPD analysis is $\mathcal{O}(ZN)$, with $Z = \#\text{blocks}$ and $N = \#\text{nodes}$. The extensive search uses the SPD analysis $A = \#\text{active cores}$ times, as a result, the complexity of the extensive search will be $\mathcal{O}(ZNA)$.

3.5.2 Random Selection

The random selection approach uses a random subset of the possible core candidates and searches for the core which provides the highest safe power density gain when disabled. The size of the subset indicates the number of times we perform the SPD analysis and, therefore, dictates the cost and quality of our estimation. Assuming we choose a size of five, we are looking at five randomly selected cores from the set of core candidates. The process of selecting the best candidate is the same as the selection process of the extensive search. Assuming we have a mapping with five active cores and we use a subset size of

five candidates, the random selection provides the same result as the extensive search. For a mapping with more active cores than five, the quality of the random selection decreases.

The random selection offers a trade-off between the quality of the estimation and the cost in computation overhead and execution time. The complexity of a random selection approach with a subset size of K is $\mathcal{O}(ZNK)$.

3.5.3 Proposed Method

Our *proposed method* utilizes the internal results SPD analysis to create a heuristic using a single SPD analysis. This drops the complexity of the source core search to the complexity of the SPD analysis $\mathcal{O}(ZN)$. By utilizing the internal results of the SPD analysis, we use the temperature distribution in system to find the core which results in the highest safe power density gain when disabled. To present the idea of our proposed method, we generate symbolic heatmaps of systems with a specific mapping. Figure 3.3 represents such a symbolic heatmap for a 8×8 many-core system and its corresponding mapping vector $\mathbf{Q}_A$ (the distribution of active and idle cores, where $q_i = 1$ if core i is active and $q_i = 0$ if core i is idle). The vector $\mathbf{Q}_A$ has seven active cores represented in red. The active cores are the heat dissipating cores in the heatmap, which are marked with a border. Figure 3.3 is a symbolic system utilizing the safe power density constraint obtained by the SPD analysis, thus, reaching the highest safe temperature. The hottest core in the system is the one in the middle of the cluster, which is marked with **S**. It is also the core, which reaches the threshold temperature and, as a result, dictates the safe power density constraint of the system. For our proposed method, we select the hottest core as our migration source, since the hottest core provides the limit to the safe power density constraint. The SPD analysis calculates the auxP value for every core and chooses the minimum of those as constraint. Hence, we can use the same selection as our source core for task migration. We implement this in our algorithm for SPD in line 28 in listing 3.5. The variable `int coreIndexPworstStar` keeps track of the index of the core with the worst auxP value. The index can later be translated back into a OpenDSE `Resource`.

```
26        if (auxP < PworstStar) {
27          PworstStar = auxP;
28          coreIndexPworstStar = i;
29        }
```

Listing 3.5: The code snippet for keeping track of the index of the core responsible for the smallest *auxP* in our adapted implementation of the SPD analysis.

3.5.4 auxP List

Our proposed method provides the single best estimation for the migration source, but the selected core might be unavailable for task migration e.g., due to timing constraints of the tasks running on it. So, we provide a sorted list of alternative candidates. A candidate at the front of the list is a better candidate for migration source than a candidate at the

$$\mathbf{Q}_A = [0,0,0,0,0,0,0,0 \\ 0,0,0,0,0,0,0,0 \\ 0,0,1,1,0,0,0,0 \\ 0,1,1,1,0,0,0,0 \\ 0,1,1,0,0,0,0,0 \\ 0,0,0,0,0,0,0,0 \\ 0,0,0,0,0,0,0,0 \\ 0,0,0,0,0,0,0,0]$$

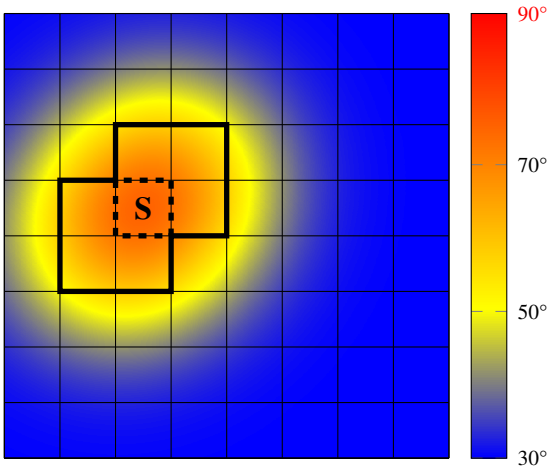

Figure 3.3: The symbolic heatmap for a homogeneous 8×8 many-core system, with the mapping $\mathbf{Q}_A$ resembling a worst-case mapping with a cluster of seven active cores. The middle core, marked with an *s* is the hottest core in mapping $\mathbf{Q}_A$ and, therefore, provides the TSP/SPD constraint. Disabling this core results in the highest TSP/SPD constraint gain. The proposed source core search approach uses the auxP values calculated in the SPD analysis to identify the hottest core.

end. The best core has the smallest auxP value and is, as a result, the first entry in the list. In our algorithm for SPD analysis, the cores are referenced by an index, so we set up a `ArrayList<AuxPCoreIndexPair> srcList` to keep track of the core's index and the respective auxP value. We fill the list with the auxP-core pair like line 22 in listing 3.6. Line 21 in listing 3.6 makes sure we only add the active cores into the list, which becomes important later with the destination core search. The `class AuxPCoreIndexPair` provides a `compare()` method, ranking the pair according to the ascending order of the auxP values. This ranking scheme is then used in line 39 in listing 3.7 to sort the `srcList` from a low auxP to a high auxP. The resulting sorted list can be traversed from start to end to get the next best core for the migration source. As a result, the list provides alternative solutions if the best solution is not available for task migration.

```
21        if (mapping.isActive(i) == 1){
22          srcList.add(new AuxPCoreIndexPair(auxP, i));
23        }
```

Listing 3.6: The code snippet for adding an *auxP-coreIndex* pair to the source-candidate list, resembling the list of active cores with their respective *auxP* value in our adapted implementation of the SPD analysis.

```
39    Collections.sort(srcList);
```

Listing 3.7: The code snippet for sorting the source-candidate list by the natural order of the *auxP* value in the *auxP-coreIndes* pair in our adapted implementation of the SPD analysis.

3.5.5 Experimental Results: Safe Power Density Gain

We evaluate the previously explained three different source core search methods and compare them against each other. For the extensive search, we collect the best source core and the worst source core to give a range for the possible results we get from the other methods. The random selection is done with a subset size of five randomly selected cores and a subset size of one randomly selected core. The random selection with a size of one provides a reference for the simple case of choosing a random core from the available core candidates. We use as evaluation environment the setup from section 3.4 and collect for each search method 5000 samples in total, which are distributed as shown in fig. 3.2 into the 10 different buckets. For each search approach, the experimental results for each bucket are displayed using a boxplot in the following. The boxplots of each approach are designated with a specific color. The following list shows the order and the color for the boxplots of each search method:

1. *Extensive-Best (EB)* in green, is the extensive search for the best possible core candidate.

2. *Proposed (P)* in blue, is our proposed method utilizing the internal results of the SPD analysis from [1].

3. *Random-5 (R5)* in light gray, is the random selection with a random subset size of five and resulting in the best possible core candidate among the cores in the random subset.

4. *Random-1 (R1)* in dark gray, is the random selection with a single randomly selected core from the possible core candidates.

5. *Extensive-Worst (EW)* in red, is the extensive search for the worst possible core candidate.

The following plots are generated with *BoxPlotR* [10], an online boxplot generator based on *R*, which is avaliable at `http://boxplot.tyerslab.com/` and described in [11].

$$G_{P^{\rho}_{\mathrm{TSP}}} = \frac{P^{\rho}_{\mathrm{TSP}}(\mathbf{Q}_{\mathrm{after}})}{P^{\rho}_{\mathrm{TSP}}(\mathbf{Q}_{\mathrm{before}})} - 1 \tag{3.1}$$

The first set of results provide, for each search approach, the gain in safe power density obtained by disabling the core found by each approach. The results are shown in fig. 3.4 for various system load levels where the gain is shown in percentage, calculated using eq. (3.1). The system load in fig. 3.4 represents the number of active cores for the current mapping. We are dividing the range of system load into ten buckets, with a subrange of five active cores. We start the system load at two active cores. The task migration needs at least one inactive core to choose from, so we exclude the amount of 52 active cores from the plot. In the first bucket, the mean safe power density gain for *EB*, *P* and *R5* are 9.47%, 9.36% and 9.47% respectively. *EB* and *R5* share the same value and the

deviation of the boxplots are the same. This proves the point, that for a equal or less number of active cores than the size of the subset, the random method performs similarly to the extensive method.

Our proposed method performs similarly, with only a slight decrease in the mean safe power density gain compared to *EB* and *R5*. The difference from the previous results to the mean safe power density gain of *R1* shows that a randomly selected core has, for a low system load, a poor estimation quality. We can gain on average 7.00% in safe power density with a randomly selected core. *R1* performs similarly to *EW* for higher system loads. *EW* is here and for the next plots a reference for the worst possible gain and serves as a visual support, but is not further discussed. For the second bucket, we have average SPD gains of 7.18% for *EB*, 6.98% for *P* and 6.81% for *R5*. Our proposed method is again only slightly below the *EB* and reassures the very good estimation quality. *R5* performs poor compared to our proposed method which also holds for higher system loads, which shows that our proposed method has a better estimation quality while being less resource and time expensive than the *R5* selection. Thus, *R5* is uninteresting for the purpose of task migration. *EW* shows that even when selecting the worst core, we gain safe power density, but only due to disabling one core. For the highest system load of 51 active cores in the mapping, the extensive search utilizes the SPD analysis 51 times, by providing the best possible core selection. In contrast to this high overhead, our proposed method uses for different system loads always only a single SPD analysis, while providing a nearly as good source core selection as the extensive search. Hence, our proposed method is a cost efficient and near-optimal source core search for the goal of increasing the thermal balance in the system, with a complexity of $\mathcal{O}(ZN)$, while providing a sorted list of alternative solutions.

3.5.6 Experimental Results: Average Frequency Gain

Increasing the safe power density of a system allows active cores to run on a higher frequency which increases the performance of the system. Therefore, we also evaluate the average frequency gain among active cores as a rough estimation of performance gain. The average frequency is calculated across all active cores in the system. When looking at the average frequency gain before and after the source core selection, the selected core transitions from an active core into an idle core. So, we exclude the selected core from the average frequency gain.

Calculating the frequency of a core with the result of SPD analysis, we have to consider DVFS by using the basic principles of power modeling, represented by eqs. (2.1) and (3.2) to (3.6). Equation (2.1) was already explained in section 2.1.2, but we now look especially on the relationship between P_{dyn} and f shown by eq. (3.4). Using DVFS, V_{dd} and f are increased with the same rate. We want to calculate the frequency gain before *(b)* and after (a) we disable a core in the system. Looking at the gain in dynamic power and using DVFS, we get the relationship as in eq. (3.5), which shows, that the dynamic power gain is equal to the cubic frequency gain. P_{dyn} can be expressed with $P_{density}$ by reconfiguring eq. (2.1), replacing P_{total} using eq. (3.2) leading to eq. (3.3). To calculate the average frequency for the active cores without the selected core, we set C in eq. (3.6)

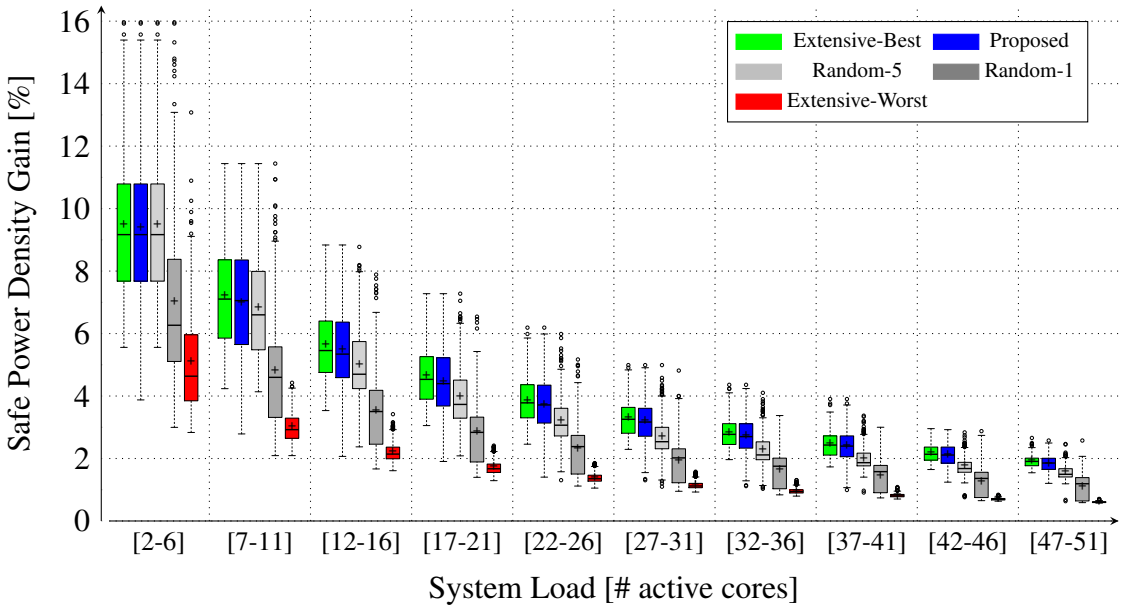

Figure 3.4: The boxplot for the safe power density gain evaluation in percent against the system load, divided into 10 buckets, for the source core search methods *Extensive-Best (EB)*, *Proposed (P)*, *Random-5 (R5)*, *Random-1 (R1)* and *Extensive-Worst (EW)*. *P* utilizes the value *auxP* to rank the core candidates.

accordingly. The average frequency gain per core takes the sum of all frequency gains and divides it by the number of cores. The frequency gain can be replaced with the cubic root of the dynamic power gain, which then can be replaced by eq. (3.3). The resulting eq. (3.6) gives us a way to calculate the average frequency gain with the safe power density we obtain by the SPD analysis.

$$\begin{aligned} P_{total} &= P_{static} &&+ P_{dyn} \\ &= I_q \cdot V_{dd} &&+ \alpha \cdot \hat{C}_{eff} \cdot V_{dd}^2 \cdot f \end{aligned} \qquad \text{(Equation (2.1))}$$

$$P_{density} = \frac{P_{total}}{A} \tag{3.2}$$

$$P_{dyn} = A_i \cdot P_{density} - P_{stat,i} \tag{3.3}$$

$$P_{dyn} = \alpha \cdot \hat{C}_{eff} \cdot V_{dd}^2 \cdot f \tag{3.4}$$

$$\frac{P_{dyn}^{(a)}}{P_{dyn}^{(b)}} = \frac{\cancel{\alpha \cdot \hat{C}_{eff}} \cdot V_{dd}^{(a)} \cdot V_{dd}^{(a)} \cdot f^{(a)}}{\cancel{\alpha \cdot \hat{C}_{eff}} \cdot V_{dd}^{(b)} \cdot V_{dd}^{(b)} \cdot f^{(b)}} = \left(\frac{f^{(a)}}{f^{(b)}} \right)^3 \tag{3.5}$$

$$\begin{aligned}\text{avg freq gain per core} &= \left(\frac{1}{|C|}\cdot\sum_{i\in C}\frac{f_i^{(a)}}{f_i^{(b)}}\right)-1\\ &= \left(\frac{1}{|C|}\cdot\sum_{i\in C}\frac{{P_{dyn,i}^{(a)}}^{\frac{1}{3}}}{{P_{dyn,i}^{(b)}}^{\frac{1}{3}}}\right)-1\\ &= \left(\frac{1}{|C|}\cdot\sum_{i\in C}\left(\frac{A_i\cdot P_{density}^{(a)}-P_{stat,i}}{A_i\cdot P_{density}^{(b)}-P_{stat,i}}\right)^{\frac{1}{3}}\right)-1\end{aligned} \tag{3.6}$$

The results of the average frequency gain for different approaches are shown in fig. 3.5 Here we calculated using eq. (3.6) the average frequency gain when a selected core is disabled, and convert the resulting values to percent. We have the same plot formatting setup from section 3.5.5. Figure 3.5 has for the first bucket a mean average frequency gain of 3.46%, 3.39% and 3.46% for *EB*, *P* and *R5*, respectively. The results for our method show, that we are selecting most of the time the core who gives us the highest average frequency gain when disabled. For the following buckets *R5* drops below *P*, which makes our proposed method more attractive. The overall trend is similar to the trend from fig. 3.4.

3.6 Destination Core Selection for Task Migration

In the second step of our temperature-aware task migration policy, we identify the best migration destination, with the destination core search. The task migration activates the selected destination core, thus, the core candidates have to be the idle cores in the system. With the help of the SPD analysis, we calculate for each core candidate, the safe power density constraint and use it to rank the core candidates. The ranking scheme of the destination search is the same scheme from the source core search from section 3.5, though the goal is different. For thermal management, disabling the migration source should result in the highest gain in safe power density possible. Enabling the migration destination, on the other hand, should result in the lowest loss in safe power density, so the systems thermal balance is only decreased as low as possible. Hence, for destination core search we call the resulting safe power density, after enabling a core candidate, better, the higher it is. This is the same ranking scheme the source core search uses. Assuming we have a list of the idle cores and sort this list with the ranking above, we get in the first position the core, which decreases the thermal-balance of the system the least when disabled. The ranking scheme lets us evaluate the following different destination core search approaches.

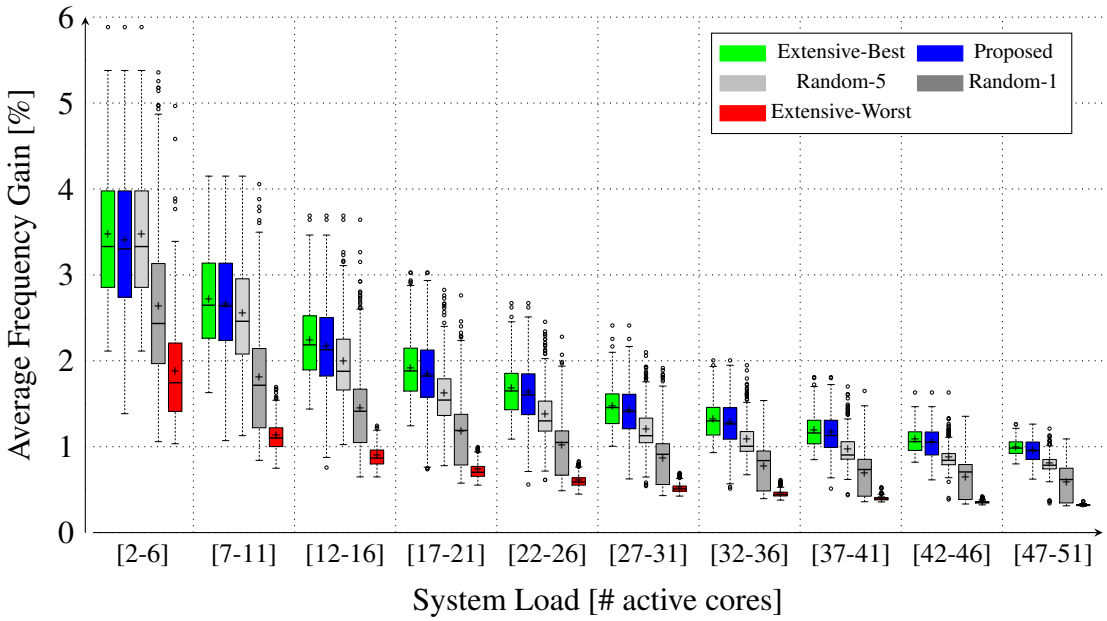

Figure 3.5: The boxplot for the average frequency gain evaluation(across active cores, excluding the migration source), in percent against the system load, divided into 10 buckets, for the source core search methods *Extensive-Best (EB)*, *Proposed (P)*, *Random-5 (R5)*, *Random-1 (R1)* and *Extensive-Worst (EW)*. *P* utilizes the value *auxP* to rank the core candidates.

- Extensive search, is the brute force method for getting the best and worst migration destination, by ranking all core candidates.
- Random selection, is the extensive search applied on a random subset of core candidates to reduce the resource and time cost of the search.
- Proposed method, is our proposed destination core search, which utilizes the internal results of the SPD analysis from [1].

The extensive search and random selection for the destination core search have the same structure as their respective counterparts in the source core search from sections 3.5.1 and 3.5.2. The only difference is that the core candidates are the idle cores in the system, which get ranked with the scheme clarified above. We propose our method for destination core search in the following.

3.6.1 Coolest Core Problem

The source core search utilizes the internal results from the SPD analysis from [1] and we do the same for the destination core search. The source core search used the auxP values of the core candidates and selects the core with the smallest auxP, since this core provides the safe power density constraint of the specific mapping. The core candidates

changed from the active cores in the system for the source search, to the idle cores in the system for the destination search. Hence, we utilize the auxP value of the idle cores for the destination search. We use the same sorting of the auxP values and select the last entry as the destination core. The last entry gives us the core which is the farthest away from any active core and, therefore, is the coolest core in the system.

We evaluate the same environment setup used in section 3.5.5 and section 3.5.6 for the destination core search. A destination core search without any active cores can not be performed, so we exclude this option. On the other hand, destination core search for a mapping with only a single inactive core or in other words 51 active cores, has only the trivial solution, so we exclude this option too, which leaves us with a range of 1 to 50 active cores for the system load. The following destination search approaches are evaluated and compared for safe power density gain and average frequency gain.

1. *Extensive-Best (EB)* in green, is the extensive search for the best possible core candidate.
2. *Proposed (P)* in blue, is our proposed method utilizing the internal results of the SPD analysis from [1].
3. *Random-5 (R5)* in light gray, is the random selection with a random subset with a size of five and resulting in the best possible core candidate among the cores in the random subset.
4. *Random-1 (R1)* in dark gray, is the random selection with a single randomly selected core from the possible core candidates.
5. *Extensive-Worst (EW)* in red, is the extensive search for the worst possible core candidate.

The values for the safe power density gain plot are calculated using eq. (3.1) and the values for the average frequency gain among the active cores, excluding the selected core, are calculated using eq. (3.6).

We first look at the plot for safe power density shown in fig. 3.6. For the first bucket, the mean safe power density gain of *EB*, *P* and *R5* are -4.33%, -5.72% and -4.74% respectively. *R5* is not like in fig. 3.4 equal to *EB*, because we select the best candidate of five random cores from the 45 to 50 idle cores. *P* is a lot lower than *EB* and *R5*, which means that our proposed method has a bad estimation quality selecting the best migration destination for thermal balance. For higher system loads, *P* drops further down, next to *EW* which again shows how bad our source core search approach will perform when applied for destination search. The plot for average frequency gain shown in fig. 3.7 has the same overall result. For the first bucket, *EB*, *P* and *R5* result in an average frequency gain of -1.60%, -2.11% and -1.77% respectively. The drop for higher system loads of *P* further down, next to *EW* is here too, reinforcing that our source core selection approach is not suitable for destination selection.

The coolest core, therefore, is not the choice we should make for a migration destination. Moreover, the raw auxP value from the SPD analysis does not provide a suitable

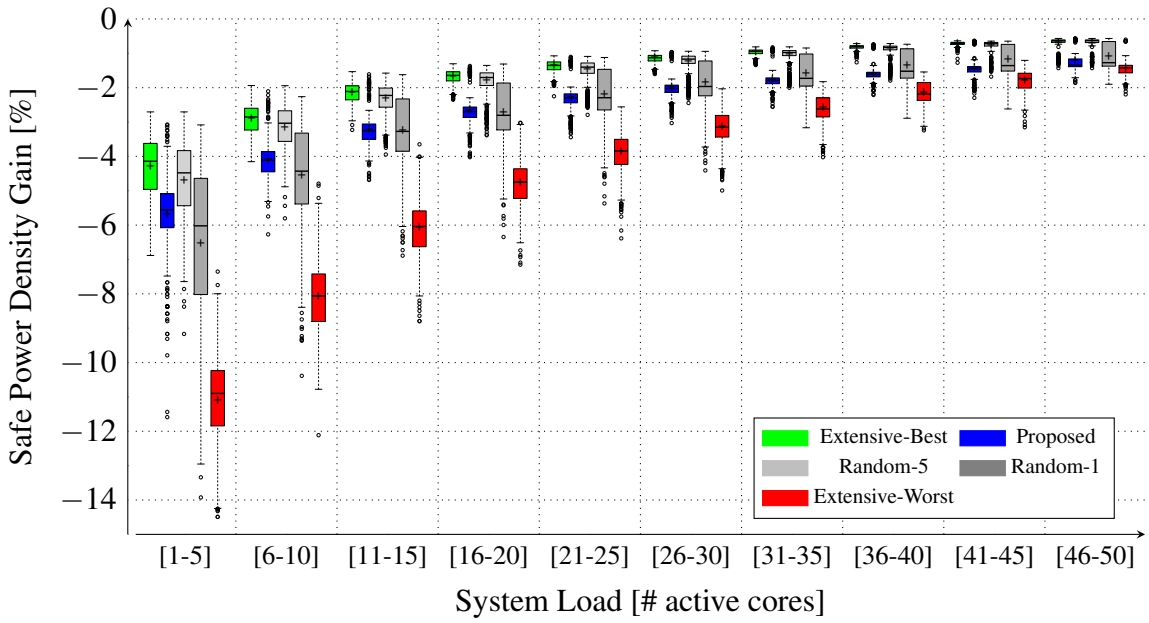

Figure 3.6: The boxplot for the safe power density gain evaluation in percent against the system load, divided into 10 buckets, for the destination core search methods *Extensive-Best (EB)*, *Proposed (P)*, *Random-5 (R5)*, *Random-1 (R1)* and *Extensive-Worst (EW)*. *P* utilizes the value *auxP* to rank the core candidates.

heuristic for ranking idle cores for their safe power density loss when activated. Equation (2.11) provides the compatibility to heterogeneous systems by considering the area of the active cores. The area of the inactive cores is never considered in our search above, so, when selecting the coolest core, we bias the search to choose the largest core farthest away from any active core. Because the selected core is a large high power core, it is introducing a lot of heat.

3.6.2 Modifying auxP List

Assuming we select the smallest core farthest away from any active core, we introduce a lot less heat than the high power core would introduce and, as a result, decrease the safe power density less than the high power core. The small auxP values from the SPD analysis provide the idle cores which are farthest away from any active core. By dividing the auxP with their respective core area, we get a bias towards small low power cores. Hence, the rank of an idle core is dependent on $\frac{auxP_i}{area_i^{core}}$, where big results for a specific core make the core be considered a better migration destination. We introduce a new `ArrayList<AuxPCoreIndexPair> destList` which is filled with the result of $\frac{auxP_i}{area_i^{core}}$ for a specific idle core and its index as in listing 3.8. The List `destList` is then sorted as in line 40 in listing 3.9, so that it starts with the best core for migration destination.

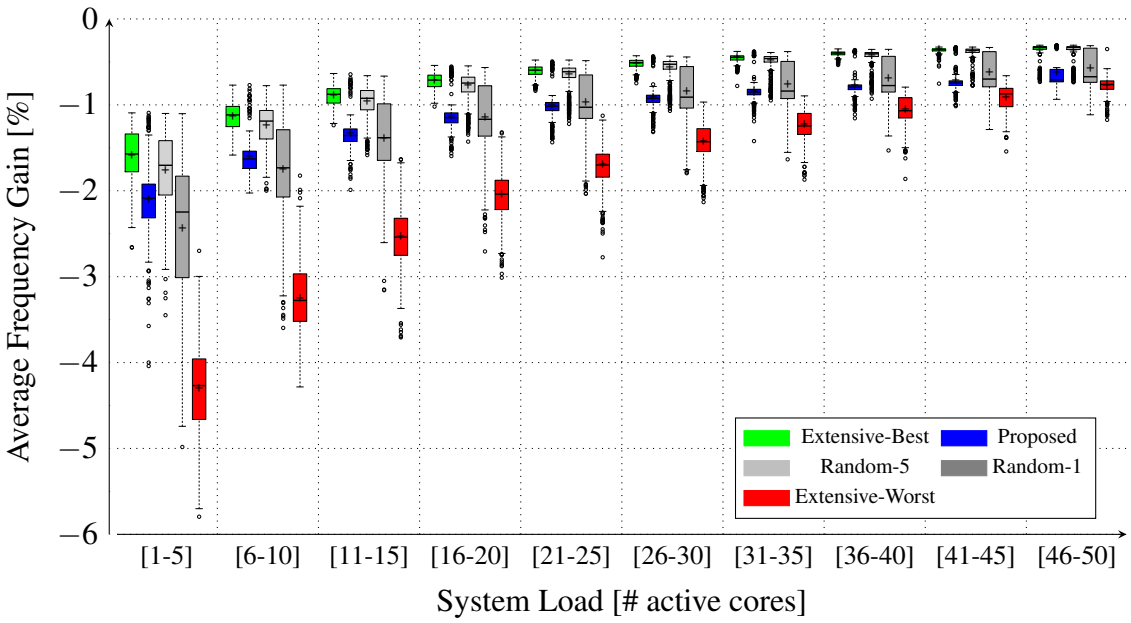

Figure 3.7: The boxplot for the average frequency gain evaluation(across active cores, excluding the migration destination), in percent against the system load, divided into 10 buckets, for the destination core search methods *Extensive-Best (EB)*, *Proposed (P)*, *Random-5 (R5)*, *Random-1 (R1)* and *Extensive-Worst (EW)*. *P* utilizes the value *auxP* to rank the core candidates.

```
30        if (mapping.isActive(i) == 0) {
31          double destSearchValue = auxP / getCoreArea(i);
32          if (destSearchValue > PbestStar) {
33            PbestStar = destSearchValue;
34            coreIndexPbestStar = i;
35          }
36        }
```

Listing 3.8: The code snippet for keeping track of the best core candidate and adding an *auxP-coreIndex* pair to the destination-candidate list, resembling the list of idle cores with their respective $\frac{auxP}{area_i^{core}}$ value in our adapted implementation of the SPD analysis.

```
40    Collections.reverse(destList);
```

Listing 3.9: The code snippet for sorting the destination-candidate list by the natural order of the $\frac{auxP}{area_i^{core}}$ value in the *auxP-coreIndes* pair in our adapted implementation of the SPD analysis.

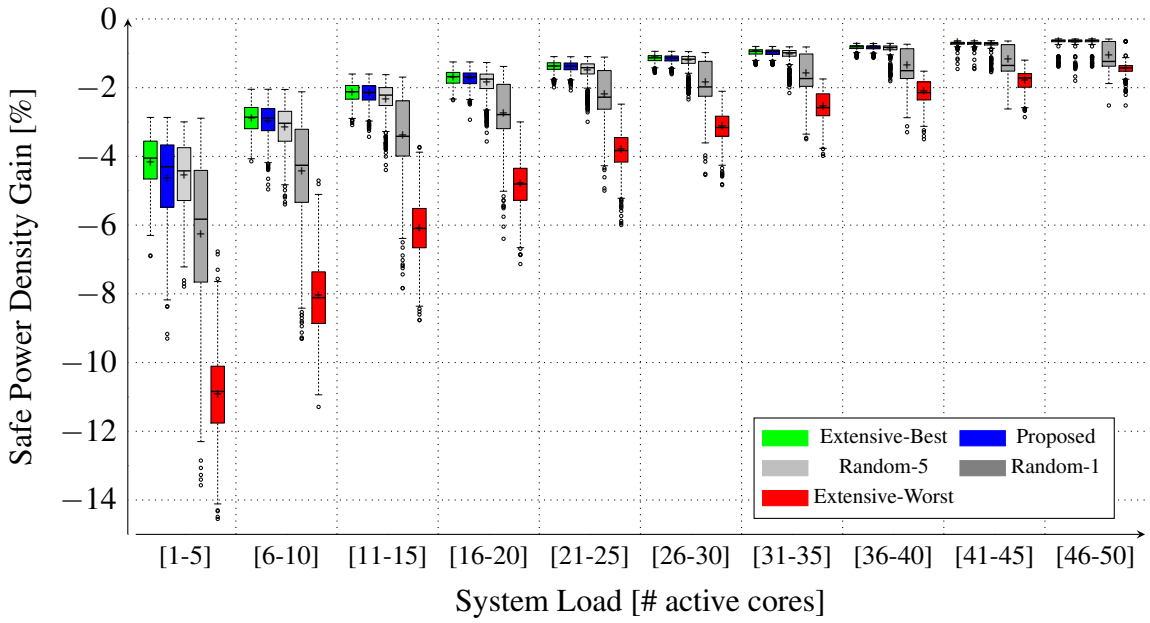

Figure 3.8: The boxplot for the safe power density gain evaluation in percent against the system load, divided into 10 buckets, for the destination core search methods *Extensive-Best (EB)*, *Proposed (P)*, *Random-5 (R5)*, *Random-1 (R1)* and *Extensive-Worst (EW)*. *P* utilizes the value $\frac{auxP}{area_j^{\text{core}}}$ to rank the core candidates.

3.6.3 Experimental Results

We evaluate the new proposed method for destination core search with the same setup as above for safe power density gain and average frequency gain. Figure 3.8 shows the plot for safe power density gain. For the first bucket *EB*, *P* and *R5* result in mean safe power density gains of -4.32%, -4.66% and -4.58% respectively. *P* is a lot closer to *EB* and *R5* than with the method before. In the next bucket, the mean for *P* is greater than the mean for *R5*, and continues to be greater for higher system loads, which shows that our new method has a good estimation quality. A similar picture can be seen in the fig. 3.9 for the average frequency gain. The mean average frequency gains for *EB*, *P* and *R5* are -1.61%, -1.80% and -1.77% respectively, showing the increase in estimation quality for our new method. For higher system loads the mean average frequency gain for *P* is greater than mean for *R5*, again validating our estimation quality for the destination core search.

The new proposed method utilizes $\frac{auxP_i}{area_i^{\text{core}}}$ calculated with the intermediate results of the SPD from [1]. The complexity for calculating the migration destination is equal to the complexity of the SPD analysis $\mathcal{O}(ZN)$.

3.7 Search Results Migration Construction

Now that we have the two steps for our temperature-aware task migration policy, we can analyze its thermal management capabilities, by evaluating the safe power density gain

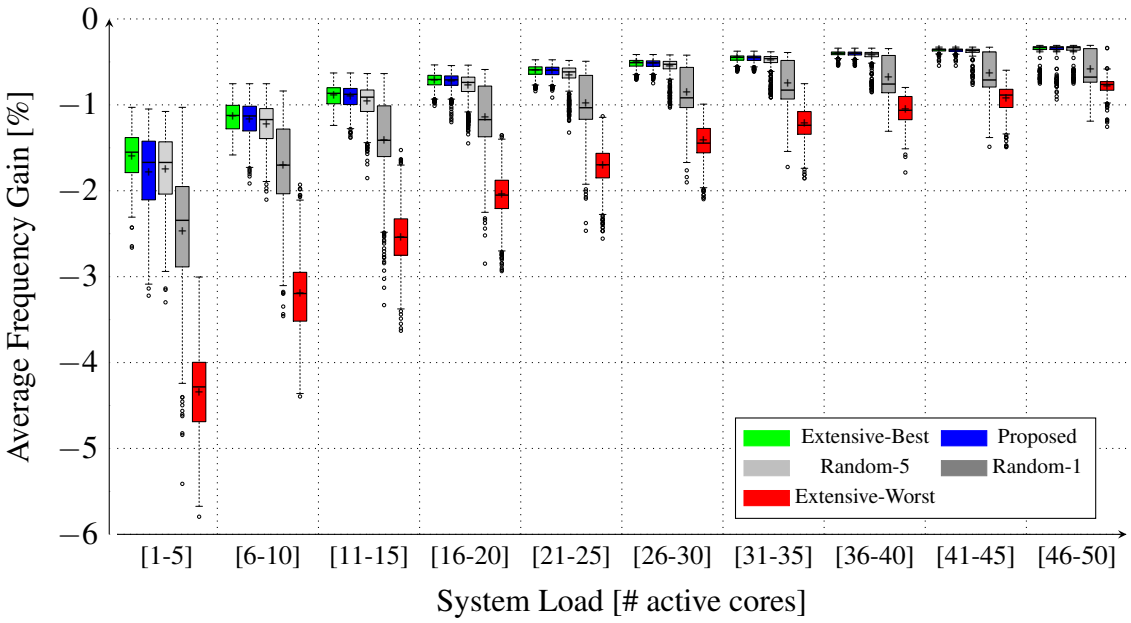

Figure 3.9: The boxplot for the average frequency gain evaluation(across active cores, excluding the migration destination), in percent against the system load, divided into 10 buckets, for the destination core search methods *Extensive-Best (EB)*, *Proposed (P)*, *Random-5 (R5)*, *Random-1 (R1)* and *Extensive-Worst (EW)*. *P* utilizes the value $\frac{auxP}{area_j^{\text{core}}}$ to rank the core candidates.

and the average frequency gain across a whole migration. The migration has two steps: (1) selecting a source core, and disabling this core and (2) selecting a destination core for the altered mapping in which the source core is disabled. This migration approach is called *two-step migration* and is used for the following evaluation. We use the sample distribution shown in fig. 3.2 and collect the data for the different search approaches as in the plots before. In our experiment, we evaluate the safe power density gain and the average frequency gain we get from a migration which selects migration source and destination with the source core search and the destination core search for the different selection methods. Given that a two-step migration for a mapping with a single active core has the intermediate mapping where no core is active, which violates the mapping constraint for the destination core search. So, we exclude mappings with a single core active. Given a mapping where all cores are active, we would have no option to increase the safe power density, because no migration is possible, so we exclude mappings with 52 active cores. Source search and destination search have both the same three methods for core selection: (1) extensive search, (2) random selection and (3) the proposed method. For the destination search, we choose the same method we used for the source search. This means we have three different approaches for selecting a core pair for the task migration. We consider the following methods for a whole task migration:

1. *Extensive-Best (EB)* in green, is the extensive search for the best possible migration source and best possible migration destination.

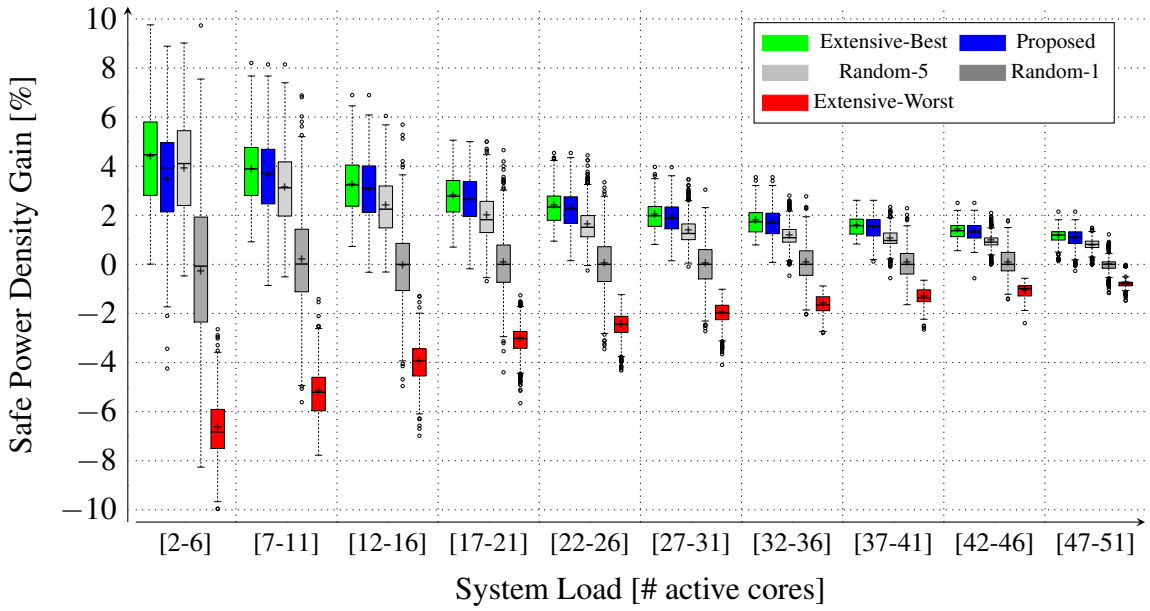

Figure 3.10: The boxplot for the safe power density gain evaluation in percent against the system load, divided into 10 buckets, for a whole migration using source core search methods and destination cores search methods *Extensive-Best (EB)*, *Proposed (P)*, *Random-5 (R5)*, *Random-1 (R1)* and *Extensive-Worst (EW)*.

2. *Proposed (P)* in blue, is our proposed method utilizing the internal results of the SPD analysis from [1].

3. *Random-5 (R5)* in light gray, is the random selection with a random subset with a size of five and resulting in the best possible core candidate for source and destination search among the cores in the random subset.

4. *Random-1 (R1)* in dark gray, is the random selection with two random cores, one for migration source and the other for migration destination.

5. *Extensive-Worst (EW)* in red, is the extensive search for the worst possible migration source and worst possible migration destination.

The safe power density gain for a whole migration is shown by fig. 3.10. The first bucket has for *EB*, *P* and *R5* the mean safe power density gains of 4.35%, 3.43% and 3.87% respectively. *P* is slightly below *R5* for the number of active cores between one and six, but for higher system loads, *P* performs better than *R5* and, as a result, is more attractive as a method for core selection for task migration. *P* also is, for higher system loads, very close to *EB*, which validates that, for a whole migration, our proposed method for selecting a migration source and migration destination is very good and stable.

The same can be said by looking at the plot for the average frequency gain shown in fig. 3.11. The first bucket has for *EB*, *P* and *R5* the mean safe power density gains of 1.58%, 1.12% and 1.37% respectively. The patterns for safe power density gain and

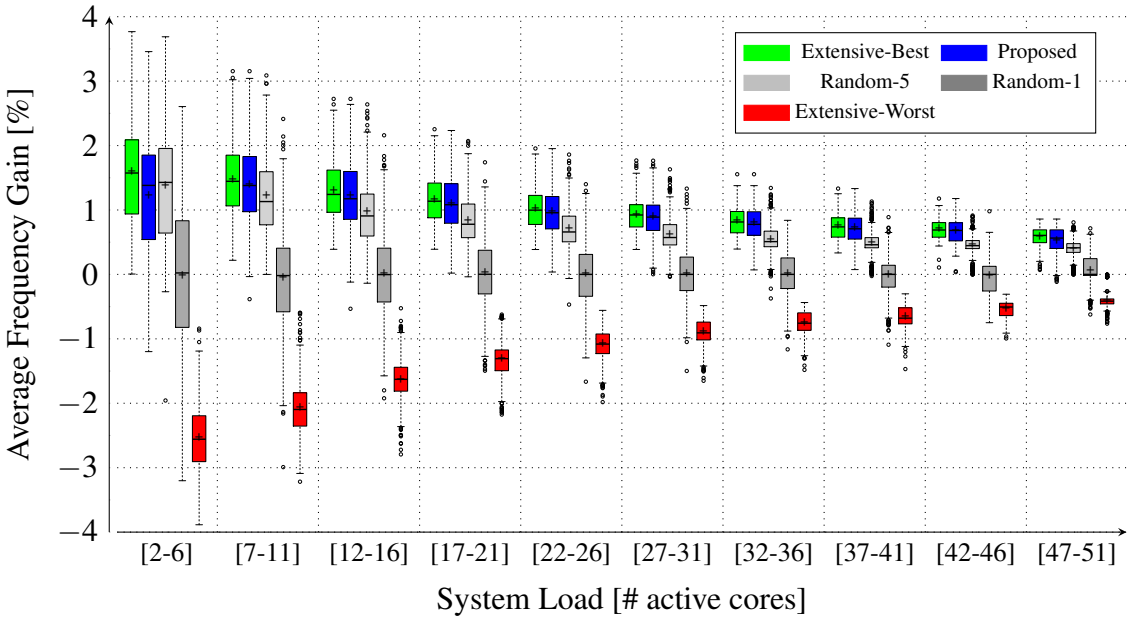

Figure 3.11: The boxplot for the average frequency gain evaluation(across active cores, excluding the migration source and the migration destination), in percent against the system load, divided into 10 buckets, for a whole migration using source core search methods and destination cores search methods *Extensive-Best (EB)*, *Proposed (P)*, *Random-5 (R5)*, *Random-1 (R1)* and *Extensive-Worst (EW)*.

average frequency gain are similar and verify the quality of our estimation. Utilizing our proposed method of the source core search and the destination core search for a whole migration, we get a complexity of $\mathcal{O}(ZN)$ and, as a result, have a fast and good task-migration policy which is temperature aware and can be used for thermal management.

3.8 Comparison of Migration Approaches

The SPD analysis's complexity is $\mathcal{O}(ZN)$ with $Z = \#\text{blocks}$ and $N = \#\text{nodes}$ (polynomial) and, therefore, we want to limit its usage to a minimum. Extensive search and random selection cannot lower their usage of the SPD analysis, because they use it to rank all cores in their subset. Our proposed method could lower the number of times the SPD analysis is used, by using a single-step migration instead of a two-step migration. The two-step migration was introduced in section 3.7 and was used because we split in section 2.2 the task migration into the two logical parts of migration source and migration destination selection. Combining the source core search and destination core search into a single core-pair search, we get the *single-step migration*. The single-step migration selects the migration source and the migration destination without altering the mapping in between, like the two-step approach does. Thus, we loose the information how the system's thermal-balance changed when the migration source is disabled. But for our proposed method, the single-step migration comes with the benefit of reducing the usage

of the SPD analysis to one or, in other words, cutting the execution time in half. The other selection methods do not have the benefit of reducing the usage of the SPD analysis, because they use it to rank the cores, so, using it once for every core. But, when extensive search and random selection use the single-step migration, it has an effect on the quality of their selected cores.

For the evaluation of the difference in quality between a single-step and a two-step migration, we utilize the *extensive migration* to get the upper limit of the possible gains. The extensive migration is the brute-force approach for finding the core-pair, which increases the safe power density most. To find the best migration core-pair, the extensive migration calculates, for every migration core-pair possible, the safe power density with the SPD analysis. A mapping with A active cores has A possible migration sources and $(M-A)$ possible migration destinations. Since we go through all possible permutation for a mapping we have $A \cdot (M-A)$ possible pairs. This results in a complexity of $\mathcal{O}(MA)$ for going through all pairs, multiplied by the complexity of the SPD analysis $\mathcal{O}(ZN)$, resulting in a combined complexity of $\mathcal{O}(ZNMA)$ for the complete extensive migration. We can assume through testing a single SPD analysis to take 3 ms including the time for sorting the auxP-lists. For a system with 26 active cores, we have 26^2 SPD analyses for the extensive migration, which takes roughly 2.028 s. Hence, we limited the amount of samples for the upcoming evaluations to 5000 for a faster evaluation (3 hours).

3.8.1 Compared Approaches

We evaluate the different migration approaches as follows:

1. *Extensive Migration (EM)* in red, is the brute force method for finding the core-pair, which generates the greates safe power density gain.

2. *Single-Step Extensive (SS-E)* in light green, is the extensive source and destination search, where both use the same core mapping to select the migration source and migration destination in a single-step.

3. *Two-Step Extensive (TS-E)* in dark green, is the extensive source and destination search with a intermediate mapping to consider the change in the system by disabling the migration source.

4. *Single-Step Proposed (SS-P)* in light blue, is the proposed source and destination search, where we use a single SPD analysis to select migration source and migration destination in a single-step.

5. *Two-Step Proposed (TS-P)* in dark blue, is our proposed source and destination search with a intermediate mapping to consider the change in the system by disabling the migration source.

6. *Single-Step Random-1 (SS-R1)* in gray, is the random source and destination selection with a subset size of one, where both cores are selected from the same mapping.

The plot formatting is the same, except that we now have six boxplots per bucket. The safe power density gain among the different migration methods is shown in fig. 3.12. Overall, we can see that *EM* and *TS-E* result in the highest mean safe power density gain. For the first three buckets, we can see the difference to the other methods, but for higher system loads, the protrusion is negligible. The difference between *EM* and *TS-E* is for the whole system load range negligible, with the highest difference of 4.26% to 4.22% in the first bucket respectively. This shows, that the *TS-E* is as good as the *EM*, while loosing an order of complexity compared to *EM*. *TS-E* and *SS-E* have a big difference of 4.22% to 3.57% in the mean safe power density gain for the first bucket respectively. The *SS-E* has, as a result, a big quality loss due to not considering the change of the thermal balance in the system, when the migration source is disabled. Moreover, *SS-E* does not have any cost benefit over *TS-E* which renders *Single-Step Extensive (SS-E)* as unattractive, except as reference for other methods. *SS-R1* shows for all buckets a mean average SPD gain close to zero, therefore, uninteresting and only a visual support for the next plots, but not further discussed. The cost benefit of *SS-P* against *TS-P*, on the other hand, can compensate the slight drop in mean safe power density gain for the first bucket from 3.36% for *TS-P* to 3.30% for *SS-P*. For higher system loads, the difference between *SS-P* and *TS-P* drops to near zero, thus, making the cost benefit of *SS-P* compensate the small drop in estimation quality for the first bucket. This sets the *SS-P* before *TS-P* when it comes to choosing which proposed method is better for the temperature-aware task migration policy. Comparing *EM* with our *SS-P* shows that, except for the first bucket, we only have a marginal drop in estimation quality, which confirms the estimation quality of *SS-P* to be near optimal.

To validate our assumption, we look at the evaluation for the average frequency gain, which is shown in fig. 3.13. The first bucket has for *EM*, *SS-E*, *TS-E*, *SS-P* and *TS-P* the mean average frequency gains of 1.60%, 1.31%, 1.54%, 1.21% and 1.24% respectively. The patterns are the same as in the previous SPD plot. *SS-P* here too performs near equal to the two-step counterpart *TS-P*. This reassures us that combining the source and destination search into a single SPD analysis is viable for our proposed method. Except for the first bucket, *SS-P* performs similarly to *EM*, reassuring, that the estimation quality is near optimal.

3.8.2 Impact of Quantized Frequency Steps

The calculation of the average frequency gain uses the power modeling eqs. (2.1) and (3.2) to (3.6), which calculate frequency in a continuous form, but real cores operates on discrete frequency steps. Assuming a core has infinite frequency steps, the previous average frequency evaluation would satisfy our need. Calculating the real average frequency gain requires the knowledge of the cores frequency steps. OpenDSE provides us with the frequency steps in form of a normalized and quantized frequency factor for a given total power. The product of the frequency factor and the maximum frequency a core can operate at, results in the real frequency the core is running on. The frequency gain can only be calculated for the same core, so the maximum frequency is canceled out and the frequency gain is equal to the frequency factor gain, shown in eq. (3.7).

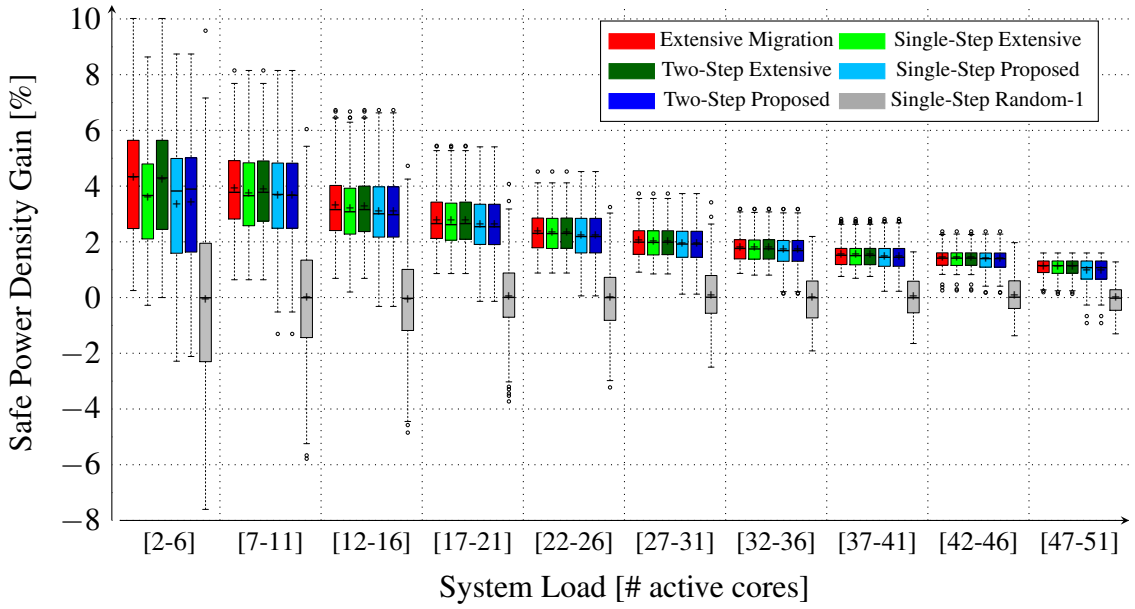

Figure 3.12: The boxplot for the safe power density gain evaluation in percent against the system load, divided into 10 buckets, for a whole migration using source core search methods and destination cores search methods *Extensive Migration (EM)*, *Single-Step Extensive (SS-E)*, *Two-Step Extensive (TS-E)*, *Single-Step Proposed (SS-P)*, *Two-Step Proposed (TS-P)* and *Random-1 (R1)*.

$$
\begin{aligned}
G_f &= \frac{f(P_{total}^{(a)})}{f(P_{total}^{(b)})} - 1 \\
&= \frac{\cancel{f_{\max}} \cdot f_{factor}(P_{total}^{(a)})}{\cancel{f_{\max}} \cdot f_{factor}(P_{total}^{(b)})} - 1
\end{aligned}
\tag{3.7}
$$

The following plot is in addition to the plots of section 3.8.1 an average frequency gain plot, which considers the quantization of the frequency steps. Because we are looking at the quantized frequency, the average frequency is lower or equal to the theoretical frequency. This puts the gain for a jump from a lower to the next higher frequency step at quantized values and the overall gain is higher. This situation is visualized in fig. 3.14. Here, we have a theoretical frequency increase from 1.9 MHz to 2.1 MHz, so, the gain would be $\frac{2.1\,\text{MHz}}{1.9\,\text{MHz}} \approx 10.5\,\%$. Assuming we have practical frequency steps every 0.5 MHz, the quantization pulls the 1.9 MHz to 1.5 MHz and the 2.1 MHz to 2 MHz. We are always rounding to the next lower frequency step. The practical frequency gain is $\frac{2\,\text{MHz}}{1.5\,\text{MHz}} \approx 33.3\,\%$. Due to the problem exemplified above, a comparison between the theoretical frequency gain and practical frequency gain is pointless and must be avoided.

The plot for average frequency gain in a practical view is shown in fig. 3.15. For the

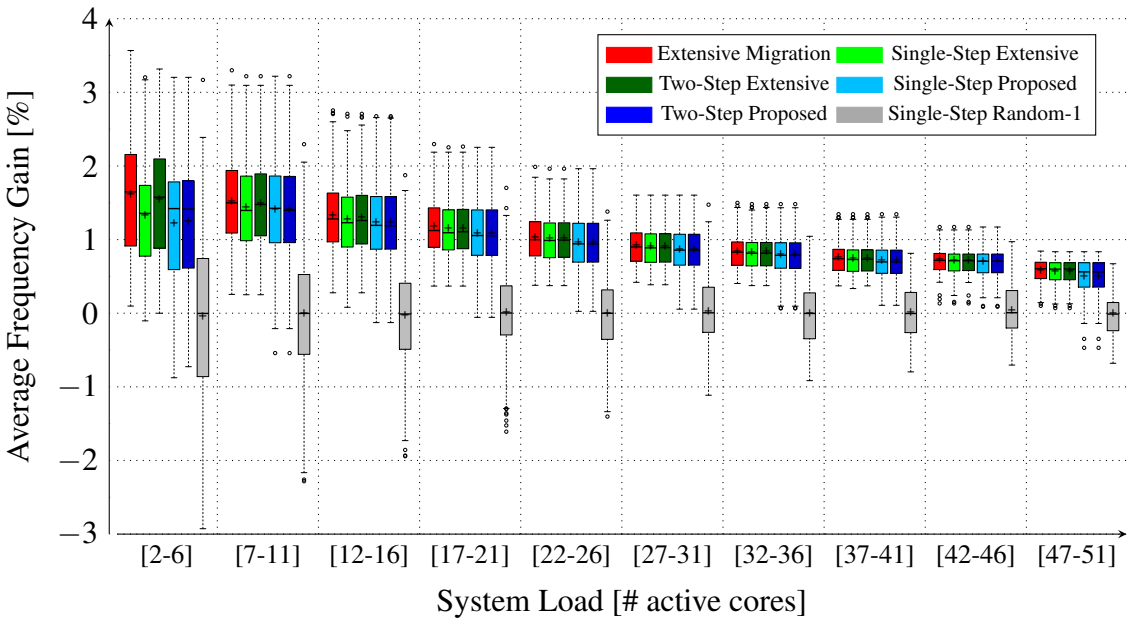

Figure 3.13: The boxplot for the average frequency gain evaluation(across active cores, excluding the migration source and the migration destination), in percent against the system load, divided into 10 buckets, for a whole migration using source core search methods and destination cores search methods *Extensive Migration (EM)*, *Single-Step Extensive (SS-E)*, *Two-Step Extensive (TS-E)*, *Single-Step Proposed (SS-P)*, *Two-Step Proposed (TS-P)* and *Random-1 (R1)*.

first, second, and ninth bucket, we have a mean average frequency gain of zero. For a small number of active cores, most of the cores are running already at their maximum possible frequency step and cannot gain any more frequency. This is why the first and second bucket have, except for some outlier no frequency gain. For a high number of active cores, the height of the next frequency step can prevent the cores from gaining frequency. As a result, we have with our frequency rate step height of 0.04 (from OpenDSE) a mean of zero for the ninth bucket. The lower the step height, the farther right the bucket with a mean of zero is, which means we could specify an optimal step height, which results in no bucket with a mean of zero. For the other buckets we can again see, that *EM* and *TS-E* are the best possible solutions, but we can also see, that our *SS-P* can hold up for higher system loads in this practical frequency context. With the results from above we validate that the efficiency and estimation quality of our *Single-Step Proposed (SS-P) Method* is superior to the other methods presented in this thesis.

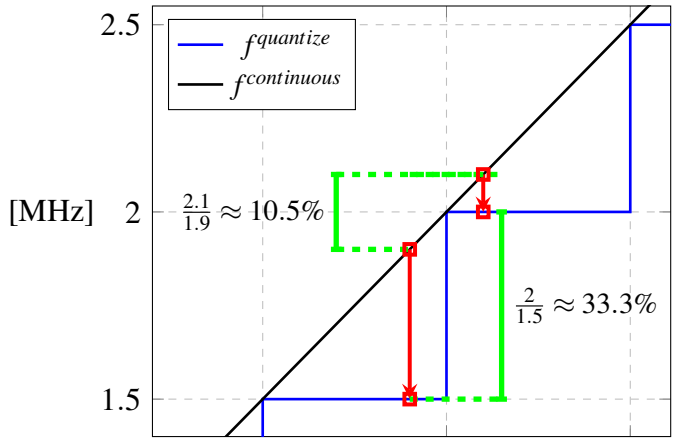

Figure 3.14: The symbolic line graph showing the difference in frequency gain between practical view($f^{quantize}$) and theoretical view($f^{continuous}$).

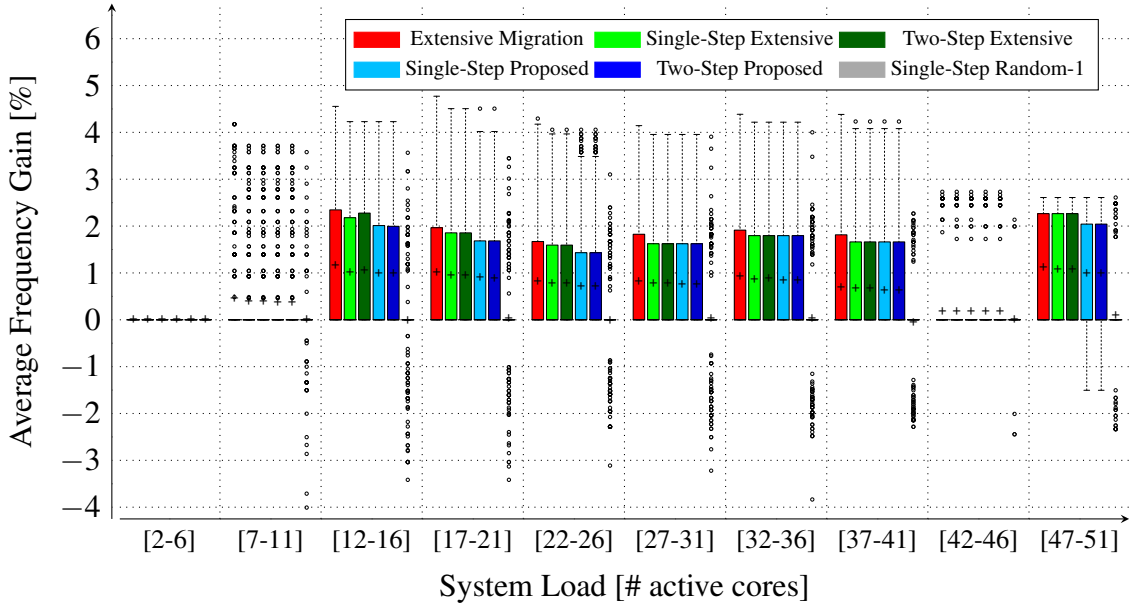

Figure 3.15: The boxplot for the practical average frequency gain evaluation (across active cores, excluding the migration source and the migration destination) in percent against the system load, divided into 10 buckets, for a whole migration using source core search methods and destination cores search methods *Extensive Migration (EM)*, *Single-Step Extensive (SS-E)*, *Two-Step Extensive (TS-E)*, *Single-Step Proposed (SS-P)*, *Two-Step Proposed (TS-P)* and *Random-1 (R1)*.

4 Conclusion

This thesis proposed a temperature-aware task migration policy that can be used for thermal management of many-core systems. As part of this work, an efficient algorithm for the temperatur-aware task migration policy is implemented, validated and integrated into the OpenDSE framework. The task migration policy utilizes the intermediate results of the Safe Power Density (SPD) analysis from [1]. The proposed algorithm is optimized to use a single SPD analysis while providing a near optimal selection of a core-pair for migration (source and destination), which benefits the thermal balance of the system the most. The algorithm for the SPD is validated against a black box program provided by the authors and documented. The effectiveness of the proposed algorithm is evaluated in terms of safe power density gain (indicating an improvement of thermal balance) and average frequency gain (indicating performance improvement). The evaluation includes several approaches including exhaustive search which is used as a reference of comparison. The results of the evaluations confirm the superiority of the proposed approach as it performs nearly similarly to the exhaustive search, but with a significantly lower overhead.

Future Work The proposed task migration policy is optimized to find the core-pair which provides the highest safe power density gain. For future work, this optimization can become multidimensional, by introducing the performance gain as another dimension we can optimize for. This requires a fast and precise method of calculating the performance for each core in the system and having an algorithm for ranking the core-pairs by the performance gain they can generate when used for task migration. Moreover, the evaluation of performance in this thesis is simplified to the evaluation of frequency gain. This evaluation can be inaccurate especially for heterogeneous systems where different core types also have different instruction sets. Therefore, an accurate evaluation of performance gain is also another aspect that can be investigated further.

Bibliography

[1] S. Pagani, H. Khdr, J. Chen, M. Shafique, M. Li, and J. Henkel, "Thermal safe power (tsp): Efficient power budgeting for heterogeneous manycore systems in dark silicon", *IEEE Transactions on Computers*, vol. 66, no. 1, pp. 147–162, 2017.

[2] H. Khdr, S. Pagani, E. Sousa, V. Lari, A. Pathania, F. Hannig, M. Shafique, J. Teich, and J. Henkel, "Power density-aware resource management for heterogeneous tiled multicores", *IEEE TC*, vol. 66, no. 3, pp. 488–501, 2017.

[3] B. Pourmohseni, F. Smirnov, S. Wildermann, and J. Teich, "Real-time task migration for dynamic resource management in many-core systems", in *Next Generation Real-Time Embedded Systems (NG-RES)*, 2020, 5:1–5:14.

[4] S. Pagani, H. Khdr, W. Munawar, J. Chen, M. Shafique, M. Li, and J. Henkel, "Tsp: Thermal safe power: Efficient power budgeting for many-core systems in dark silicon", in *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis*, 2014, pp. 1–10.

[5] F. Reimann, M. Lukasiewycz, M. Glaß, and F. Smirnov, *OpenDSE – open design space exploration framework*, 2019. [Online]. Available: `http://opendse.sourceforge.net/`.

[6] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling", in *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, 2011, pp. 365–376.

[7] H. Khdr, "Resource management for multicores to optimize performance under temperature and aging constraints", PhD thesis, Karlsruher Institut für Technologie (KIT), 2019, 155 pp. DOI: `10.5445/IR/1000095963`.

[8] G. Martin, "Overview of the mpsoc design challenge", in *Proceedings of the 43rd Annual Design Automation Conference*, ser. DAC '06, San Francisco, CA, USA: Association for Computing Machinery, 2006, pp. 274–279, ISBN: 1595933816. DOI: `10.1145/1146909.1146980`. [Online]. Available: `https://doi.org/10.1145/1146909.1146980`.

[9] S. Pagani, "Power, energy, and thermal management for clustered manycores", PhD thesis, Karlsruher Institut für Technologie (KIT), 2016, 190 pp. DOI: `10.5445/IR/1000063307`.

[10] "Kick the bar chart habit", *Nature Methods*, vol. 11, no. 2, pp. 113–113, Feb. 2014, ISSN: 1548-7105. DOI: `10.1038/nmeth.2837`. [Online]. Available: `https://doi.org/10.1038/nmeth.2837`.

[11] M. Spitzer, J. Wildenhain, J. Rappsilber, and M. Tyers, "Boxplotr: A web tool for generation of box plots", *Nature Methods*, vol. 11, no. 2, pp. 121–122, Feb. 2014, ISSN: 1548-7105. DOI: 10.1038/nmeth.2811. [Online]. Available: https://doi.org/10.1038/nmeth.2811.

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.

Erlangen, September 23, 2020

_____

Paul Meyer