seBoost: Selective Boosting for Heterogeneous Manycores

Santiago Pagani*, Muhammad Shafique*, Heba Khdr*, Jian-Jia Chen†, and Jörg Henkel*

*Chair for Embedded Systems (CES)
Karlsruhe Institute of Technology (KIT), Germany

†Department of Informatics
TU Dortmund University, Germany

Corresponding Author: santiago.pagani@kit.edu

ABSTRACT

Boosting techniques have been widely adopted in commercial multicore and manycore systems, mainly because they provide means to satisfy performance requirements surges, for one or more cores, at run-time. Current boosting techniques select the boosting levels (for boosted cores) and the throttle-down levels (for non-boosted cores) either arbitrarily or through step-wise control approaches. These methods might result in unnecessary performance losses for the non-boosted cores, in short boosting intervals, in failing to satisfy the required performance surges, or in unnecessary high power and energy consumption. This paper presents an efficient and lightweight run-time boosting technique based on transient temperature estimation, called *seBoost*. Our technique guarantees meeting the performance requirements surges at run-time, thus maximizing the boosting time with a minimum loss of performance for the non-boosted cores.

1. INTRODUCTION

When executing a set of applications on a multicore or manycore system, it is common that at some moment one or more application threads need to increase their performance during some time, mostly due to performance requirements surges (peaks) at run-time. For example, this could happen in a video face recognition application when suddenly a crowd of people enter the frame [21]. Furthermore, many applications could need run-time performance surges concurrently, but such run-time requirements, although overlapping, might arrive at different times or have different durations. Boosting techniques provide the system with the means to satisfy these run-time performance requirements surges, and have therefore been widely adopted in commercial multicore and manycore systems. Namely, through Dynamic Voltage and Frequency Scaling (DVFS), boosting techniques, e.g., Intel's Turbo Boost [3, 10, 11, 17] and AMD's Turbo CORE [14], allow the system to run some cores in the chip at high voltage and frequency (VF) levels during short time intervals, even at the cost of exceeding standard operating power budgets, e.g., the Thermal Design Power (TDP). Given that executing some cores at high VF levels increases their power consumption, boosting normally results in an increment of the temperature of these cores through time. Due to this temperature increase, once the highest temperature among all cores reaches a predefined threshold, the system must either return to nominal operation (needing some cool-down time before another boosting interval), or use some closed-loop control to oscillate around the threshold (prolonging the boosting time).

Careful decisions must be taken when selecting the boosting levels. Otherwise, the temperature on the chip may raise very quickly and the performance requirements surges might not be satisfied. For example, when some applications receive a run-time boosting requirement, normally the applications that do not require boosting at this time are momentarily considered to have less priority. This implies that these applications' cores could be momentarily throttled-down, that is, reduce their voltage and frequency levels. Failing to throttle-down the cores of such non-boosting applications may result in not satisfying the performance requirements surges due to a rapid raise in the temperature, while excessive throttling-down will result in unnecessary overall performance penalties to the system. The following motivational example (results in Figure 1) provides further insight into the impact of boosting on temperature and performance.

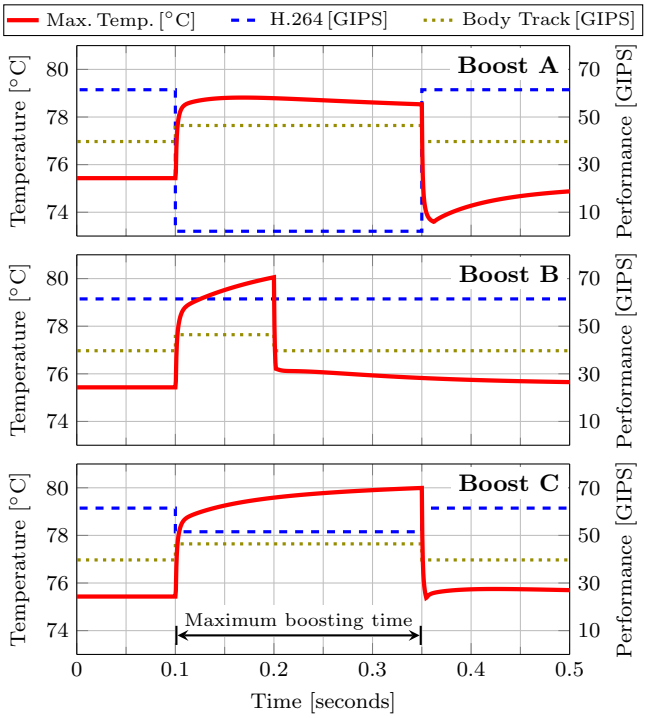


Figure 1: Motivational example for three different boosting techniques. The red line shows the maximum temperature among all cores (left axis). The performance of the applications is measured in Giga-Instruction Per Second (GIPS).

Motivational Example: For simplicity of presentation, consider a system with 16 cores[1] with size 3.2 mm × 3.0 mm,

[1]Alpha 21264 cores in 22 nm, simulated with McPAT [12].

arranged in 4 rows and 4 columns. Assume a critical temperature of 80°C, and a cooling solution from HotSpot's [9] default configuration. Consider that the system is executing two applications from the Parsec benchmark suite [1], specifically, an H.264 video encoder and a body track application, each running 8 parallel dependent threads (one thread per core). At nominal operation we assume that all cores are running at 3 GHz, and we conduct simulations with gem5 [2], McPAT [12], and HotSpot [9] to obtain performance, power, and temperature traces, respectively.

At 0.1 s after starting execution, the body track application needs to boost its performance. Particularly, all 8 cores running this application need to be boosted to 3.7 GHz. The required boosting time is not precisely given, but it is expected to last no more than 0.25 s. Figure 1 presents simulation results corresponding to the maximum temperature among all cores as a function of time, for three different boosting methods. Boosting method A simply decides to fully throttle-down the H.264 application to the slower available frequency, that is, 0.1 GHz. Thus, the body track application is able to be boosted during 0.25 s at the required frequency, but the H.264 application is only able to achieve 3% of its nominal performance. Boosting method B decides to keep running the H.264 application at its nominal values. Hence, although now the H.264 application suffers no performance losses, in this case the body track application can only be boosted to 3.7 GHz during 0.1 s, reducing the boosting time compared to method A. Contrarily, boosting method C intelligently selects to throttle-down the H.264 application to 2.5 GHz. In this way, the body track application is able to be boosted to the required 3.7 GHz during the full 0.25 s, precisely reaching the critical temperature at the end of the maximum required boosting time. Moreover, the H.264 application is able to achieve 84% of its nominal performance under boosting method C.

This motivational example shows the necessity of an efficient boosting method. Nevertheless, until now existing boosting techniques have neglected to make such careful decisions when choosing the boosting levels and the duration of the boosting interval, and this remains an open problem.

Objective: The objective of this paper is to present an efficient and lightweight *run-time* boosting technique, that guarantees meeting the run-time requirements surges, with minimum performance losses for the applications/threads running on the non-boosted cores. Furthermore, our technique, called *seBoost* (from Selective Boosting), is also capable of refining the boosting decisions when, in the middle of a boosting interval, other applications/threads receive additional (concurrent) boosting requirements. Moreover, given that the continuous increasing performance demands and the power budget constraints have lead to the emergence of heterogeneous architectures [20], *seBoost* is natively developed to handle heterogeneity.

For the run-time performance requirements surges, there are two parameters to consider: (1) the required VF levels and (2) the duration of the surges. For the VF levels of the cores requiring boosting, we consider two cases: (1a) that they are given, and (1b) that they are unknown. For the former (1a), we focus on minimizing the performance losses of the non-boosted cores while satisfying the required VF levels of the boosted cores. For the latter (1b), maximizing the VF levels of the boosted cores has higher priority.

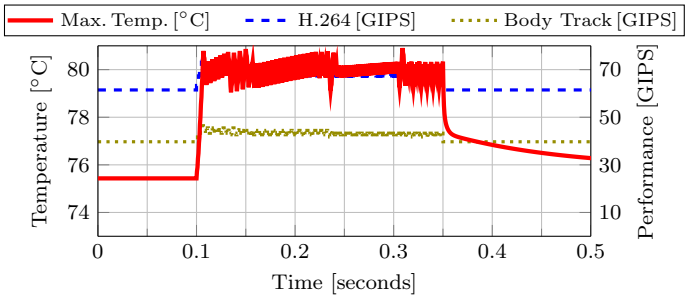


Figure 2: Motivational example for TurboBoost [3]. The red line shows the maximum temperature among all cores (left axis). The performance of the applications is measured in Giga-Instruction Per Second (GIPS).

Thus, we focus on minimizing the performance losses of the non-boosted cores only if the boosted cores can all run at maximum frequency. Otherwise, the non-boosted cores are throttled-down completely, and we run the boosted cores at the highest possible frequencies. With respect to the duration of the surges, we also consider two cases: (2a) that the maximum expected duration of the surges is known, and (2b) that they are unknown. For the former (2a), the goal is to boost for as long as possible under this maximum expected duration. For the latter (2b), we assume that the surges can last for a very long time, thus the goal is to boost indefinitely.

Our Contributions: Based on the above discussions,

- For given required VF levels and a maximum boosting time requirement, we derive an efficient and lightweight *run-time* algorithm that selects the throttle-down levels for the non-boosted cores, such that the boosted cores execute at the required VF levels during the entire maximum expected boosting time, and the non-boosted cores do not suffer unnecessary performance losses.
- We extend the previous algorithm to consider cases in which the maximum boosting time is given, but the required VF levels are unknown. This algorithm tries to maximize the VF levels of the cores requiring boosting during the entire maximum expected boosting time, again minimizing the performance losses for the non-boosted cores. For this case maximizing the VF levels of the boosted cores is with higher priority.
- Finally, we extend our two previous algorithms to consider cases in which the maximum expected boosting time is unknown. In these scenarios, the goal is to find VF levels that can be sustained indefinitely.

2. RELATED WORK

Intel's Turbo Boost [3, 4, 10, 11, 17] allows cores to run at high VF levels when there is available headroom within power, current, and temperature constraints. Namely, when the system requires a performance surge, if the measured temperature, power, and current are below the constraints, the cores boost their VF levels in single steps (within a control period) until it reaches an upper limit dictated by the number of active cores. Similarly, if the temperature, power, or current violates the constraints, the cores reduce their VF levels in single steps until the constraints are satisfied or the

nominal VF values are reached. Furthermore, when Intel's Turbo Boost is triggered, normally all active cores in the chip operate at the same VF levels, and throttling-down a few cores with low priority applications/threads is not always allowed. Figure 2 shows Turbo Boost's behavior for the motivational example in Section 1. Here, although the H.264 application suffers no losses and the body track application runs faster than its nominal frequency, the latter fails to meet its run-time requirements, only running at 3.7 GHz for merely 0.01 s out of the required 0.25 s.

Computation sprinting [16] proposes dealing with run-time surges via parallelism, by activating cores that are normally power-gated during short time bursts (typically shorter than 1 s). Boosting through DVFS is intentionally discouraged, motivated by the ideal linear relation between power and performance expected when activating several cores at equal VF levels, compared to the power consumption for achieving the same performance in less cores at higher VF values. Although this is a valid point, waking up cores from power-gated or deep-sleep modes, plus the correspondent thread migrations may result in significant overheads, particularly considering the short duration of the sprinting periods. Furthermore, [16] fails to provide methods to select the number of sprinting cores to find a compromise between the sprinting performance and the sprinting time, and ignores the possibility of power-gating cores with lower priority applications/threads to prolong the sprinting time.

The work in [5] proposes a proactive temperature management method based on temperature prediction using Autoregressive Moving Average (ARMA) modeling, which estimates temperature based on past temperature measurements. Nevertheless, although the model is updated at run-time when workload changes are detected, since ARMA relies on past temperature measurements, it is not well suited for fast temperature prediction caused by drastic power changes.

The work in [7] presents a run-time thermal management technique based on RC thermal networks that aims to minimize the latest completion time among the applications. Using convex optimization, the authors first derive an optimal solution with a too high time complexity for run-time usage. By exploiting the structure of certain matrices in their convex optimization formulation, the authors present an approximated solution which is 20 times faster than their convex optimization approach. Unfortunately, this technique still needs more than 15 ms to compute the VF levels of *each* core, which is not fast enough for dealing with short run-time performance surges in manycore systems.

From the discussed related work, we observe that there exists no *lightweight* technique that *efficiently selects* the cores' VF levels based on future temperature estimation that guarantees meeting run-time performance requirements surges.

3. SYSTEM MODEL

This section reviews the hardware, thermal, and application model adopted for the rest of the paper.

3.1 Hardware Model

We focus on a system of M heterogeneous cores. Every core has Dynamic Voltage and Frequency Scaling (DVFS) capabilities at a core level, that is, the voltage and frequency of every core is independent to the settings of other cores, and they can be changed at any given point. For a core to support stable execution at a specific frequency, the core's voltage has to be set above a minimum value. This minimum voltage is different for every frequency, and higher frequencies have higher minimum voltage levels. For a given frequency, using a voltage level higher than the minimum consumes unnecessary power. Thus, instead of considering all possible voltage and frequency combinations, we consider voltage and frequency pairs in which the voltage is set to the minimum accepted value for the specified frequency. Naturally, due to the core heterogeneity, there is a different set of VF pairs for each core, and we define them as $\left\{F_0^{c_i}, F_1^{c_i}, F_2^{c_i}, \ldots, F_{H_i}^{c_i}\right\}$ for all $i = 1, 2, \ldots, M$ cores, such that H_i is the number of VF pairs for core c_i. The specific VF pair used at a given moment by core c_i is denoted as $F_{S_i}^{c_i}$, where $1 \leq S_i \leq H_i$ if core c_i is active, and $S_i = 0$ if core c_i is inactive (in some low-power mode, e.g., power-gated).

Moreover, there is a maximum chip power constraint, $P_{\max}$, and a maximum chip current constraint, $I_{\max}$. These are electrical constraints that cannot be exceeded, for example, from the capacity of the power supply or the wire thickness.

3.2 Thermal Model

For our thermal model, based on the well-known duality between thermal and electrical circuits, we assume a given RC thermal network modeled from the chip and cooling solution. Any thermal modeling tool can be used to derive the RC thermal network, e.g., HotSpot [9]. An RC thermal network consists on N thermal nodes. Matrix $\mathbf{B} = [b_{i,j}]_{N \times N}$ represents the thermal conductances between neighboring nodes. In order to account for the effects of the transient temperatures, each thermal node is associated to a thermal capacitance, and all the capacitances are represented in matrix $\mathbf{A} = [a_{i,j}]_{N \times N}$. The ambient temperature, defined as T_{amb}, is considered to be constant and hence there is no capacitance associated to it. Column vector $\mathbf{G} = [g_i]_{N \times 1}$ represents the thermal conductance between each thermal node and the ambient temperature. The power consumptions of the nodes are considered to be heat sources, represented by column vector $\mathbf{P} = [p_i]_{N \times 1}$. By defining matrix $\mathbf{C} = -\mathbf{A}^{-1}\mathbf{B}$, the temperature on every node can be computed through a system of N first-order differential equations [15], specifically,

$$\mathbf{T}' = \mathbf{C}\mathbf{T} + \mathbf{A}^{-1}\mathbf{P} + T_{\text{amb}}\mathbf{A}^{-1}\mathbf{G}, \tag{1}$$

where column vector $\mathbf{T} = [T_i(t)]_{N \times 1}$ represents the temperature on every thermal node at time t, and column vector $\mathbf{T}' = [T_i'(t)]_{N \times 1}$ contains the first-order derivative of the temperature on every thermal node with respect to time. Without loss of generality, we assume that the N thermal nodes in the RC thermal network are ordered in such a way that the first M nodes correspond to the M heterogeneous cores.

From the work in [15], the steady-state temperature on node k, defined as T_{steady_k}, can be computed as

$$T_{\text{steady}_k} = \sum_{j=1}^{N} b^{-1}{}_{k,j} \cdot p_j + T_{\text{amb}} \cdot \sum_{j=1}^{N} b^{-1}{}_{k,j} \cdot g_j, \tag{2}$$

where $\mathbf{B}^{-1} = \left[b^{-1}{}_{k,j}\right]_{N \times N}$ is the inverse of matrix $\mathbf{B}$.

Based on [13], the work in [15] also presents an analytical method to compute the transient temperature on node k as

a function of time, defined as $T_k(t)$, and computed as

$$T_k(t) = T_{\text{steady}_k} + \sum_{i=1}^{N} e^{\lambda_i \cdot t} \cdot v_{k,i} \cdot \sum_{j=1}^{N} z_{i,j} \left(T_{\text{init}_j} - T_{\text{steady}_j} \right), \quad (3)$$

where column vector $\mathbf{T}_{\text{init}} = [T_{\text{init}_k}]_{N \times 1}$ holds the initial temperatures on all thermal nodes at $t_0 = 0$, $\{\lambda_1, \lambda_2, \ldots, \lambda_N\}$ are the eigenvalues of matrix $\mathbf{C}$, $v_{k,i}$ corresponds to an element in matrix $\mathbf{V} = [v_{k,i}]_{N \times N}$ which contains the eigenvectors for matrix $\mathbf{C}$, and $z_{i,j}$ corresponds an element in matrix $\mathbf{V}^{-1} = [z_{i,j}]_{N \times N}$, which is the inverse of matrix $\mathbf{V}$.

3.3 Application Model

We consider L multi-threaded applications, where every application can be mapped to a single core (single thread) or to several cores (parallel dependent threads). Every core can be mapped with one or more application threads (multi-tasking). Through experimental evaluation, we assume to have given input tables with power profiles for every thread of every application, for all levels of thread parallelism and for all available voltage and frequency pairs of each different core (due to core heterogeneity). Hence, for a given mapping of applications to physical cores, we define function $P_{c_i}(S_i)$ which returns the power consumption for core c_i running at $F_{S_i}^{c_i}$. When S_i is 0, function $P_{c_i}(S_i)$ returns the power consumption for the corresponding low-power mode. If more than one application/thread is mapped to core c_i, function $P_{c_i}(S_i)$ returns the highest power among the power values of all the applications/threads mapped to c_i. Similarly, for the same mapping, function $I_{c_i}(S_i)$ returns the current consumed by core c_i running at $F_{S_i}^{c_i}$.

Furthermore, with respect to the effects of temperature on leakage currents and leakage power consumption, we consider that the current and power profile tables are obtained (or modeled) at the critical temperature T_{crit}. In this way, our algorithms are able to make decisions considering safe margins with respect to leakage power and leakage currents.

When an application requires a run-time surge in performance, we assume that there are certain scenarios in which the maximum expected duration of the surges is known or it can be estimated. This information can be very useful when deciding the VF boosting levels, thus we assume it to be a system-level abstraction, and we consider it as input to our algorithms in Section 5.1 and Section 5.2. To specify the maximum expected boosting time requirement, historic profiling under different workloads could be used. That is, it might not be possible to predict when a certain kind of workload will arrive, however, the system can potentially estimate the duration of a performance surge based on the workload type once the workload has arrived. For example, for a signal processing application, the application might not be able to predict when a new signal will arrive, however, the application could estimate how much time it will require to process a new signal based on the size of the data. Another potential way is through application phase-classification [19], that is, critical workload phases might need maximum boosting levels during the phase and the duration of each phase can be profiled offline. Nevertheless, although the duration of a phase can be estimated, it might not be possible to predict when each phase will need to be executed. Moreover, in Section 5.3, we provide extensions to consider cases in which the maximum expected boosting times cannot be specified.

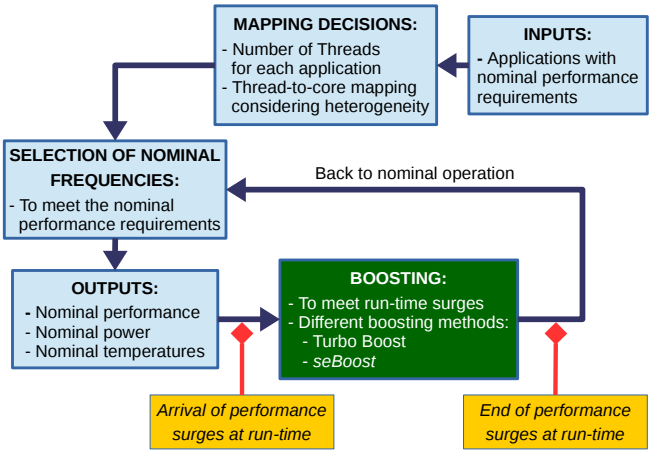


Figure 3: System and problem overview.

4. PROBLEM DEFINITION

We assume a given mapping of applications to physical cores, and given nominal voltage/frequency levels for every core $\{F_{\text{nom}_1}^{c_1}, F_{\text{nom}_2}^{c_2}, \ldots, F_{\text{nom}_M}^{c_M}\}$, where $1 \leq \text{nom}_i \leq H_i$ for all $i = 1, 2, \ldots, M$. We assume that running the system at the nominal voltage/frequency levels for the given mapping meets the nominal performance requirements of all applications, and the critical temperature T_{crit} is not exceeded. That is,

$$T_{\text{steady}_k} \leq T_{\text{crit}} \qquad \text{for all } k = 1, 2, \ldots, M.$$

The given mapping and nominal voltage/frequency levels can be derived with any of the existing solutions in the literature, e.g., [7].

At a given time t_0, for which we know the temperatures on all cores, that is, column vector $\mathbf{T}_{\text{init}} = [T_{\text{init}_k}]_{N \times 1}$ is known, one or more application threads require a *run-time* increase in their performance, achieved by increasing the voltage/frequency levels of their corresponding cores, that is, boosting. The indexes of the cores requiring boosting are defined in set W, that is, core c_i requires boosting if $i \in W$. Similarly, the indexes of the active cores that do not require boosting are defined in set $\overline{W}$, and the indexes of the inactive (power-gated or sleeping cores) are defined in set $\not{W}$. The required boosting time is expected to last until no more than t_1. During this time, the cores not requiring boosting, that is, all c_i for which $i \in \overline{W}$, are considered to have less priority, and can therefore be momentarily throttled-down by reducing their voltage/frequency levels. All cores return to nominal operation after the run-time requirements expire. An overview of this description is presented in Figure 3.

First, we assume that the run-time VF level requirements are given. That is, for all c_i cores for which $i \in W$, there is a given VF level that *must* be satisfied, defined as $F_{R_i}^{c_i}$ such that $1 \leq R_i \leq H_i$. The problem then focuses on selecting the throttle-down levels for the non-boosted cores, such that the maximum temperature among all cores reaches T_{crit} precisely at t_1, without exceeding $P_{\max}$ and $I_{\max}$. In this way, the non-boosted application threads do not suffer from unnecessary performance losses, like for example adopting a trivial solution that throttles-down the non-boosted cores to their minimum voltage and frequency settings. We deal with this problem in Section 5.1.

Secondly, we assume that the run-time VF level requirements are unknown. Thus, the new problem focuses on selecting the VF levels of the boosted cores in order to maximize the performance of their application threads, and also selecting the throttle-down levels for the non-boosted cores, such that the maximum temperature among all cores reaches T_{crit} precisely at t_1, without exceeding $P_{\max}$ and $I_{\max}$. Here, maximizing the performance of the boosted cores is with higher priority. This is presented in Section 5.2.

Finally, we consider cases in which the maximum expected boosting time is unknown. Hence, this last problem focuses on finding VF levels for the previous two cases, but that can be sustained indefinitely (not only until t_1) without exceeding T_{crit}, $P_{\max}$, and $I_{\max}$. This is presented in Section 5.3.

5. SEBOOST: SELECTIVE BOOSTING

5.1 Given Required Boosting Levels

For this case, the required VF levels of the cores that need boosting are given, together with the duration of the maximum expected boosting time. From the power profiles, according to the threads and applications mapped to each core, we know the power consumptions of every core for all possible VF settings. Thus, using Equation (2) and Equation (3), we can estimate the temperature behavior on all cores after selecting their VF levels. Specifically, we can compute the temperature on every core at time t_1 after selecting the throttle-down levels of all non-boosted cores.

Our proposed *seBoost* solution, presented in Algorithm 1, is based on a binary search like approach. Namely, *seBoost* sets the VF levels of the boosted cores for the given $F_{R_i}^{c_i}$ requirements for all $i \in W$. Then, applying binary search proportional to the nominal VF values on each core, *seBoost* tests a limited number of combinations of VF levels for the non-boosted cores. Estimating the temperatures at t_1 through Equation (2) and Equation (3), *seBoost* selects the combination that resulted in the highest VF pairs such that $\max_{k=1,2,\ldots,M}\{T_k(t_1)\} \leq T_{\text{crit}}$, and both $P_{\max}$ and $I_{\max}$ are not exceeded.

Algorithm 1 is described in detail as follows. Lines 1 to 12 set the initial conditions. That is, the search area for each core is initialized proportional to its nominal frequency, index h keeps track of the core with the largest search area, and the selected frequencies indexes S_i are set to: R_i for the boosted cores, to the minimum frequencies (that is, 1) for the non-boosted cores (in case all combinations violate $P_{\max}$, $I_{\max}$, or T_{crit} before t_1), and to 0 for the inactive cores. Lines 13 to 30 correspond to proportional binary search loop, whose exit condition depends on the core with the largest search area. Lines 14 to 16 set the midpoints u_i^{mid} according to the current search area for every core c_i with $i \in \overline{W}$. Here, a maximum is taken between the midpoint and the minimum frequency index 1, because as different cores have search areas with different sizes depending on their nominal frequencies, some cores might finish their search before core c_h. Lines 17 to 19 compute the total current, the total power, and the maximum estimated temperature among all cores at t_1 using Equations (2) and (3), for the current midpoints of the search. That is, the power values used in Equation (2) come from function $P_{c_i}\left(u_i^{\text{mid}}\right)$. Then, the next search area is chosen depending on whether all cores remain below or above T_{crit} at t_1, and whether the current and power constraints are satisfied. Line 22 keeps track of the selected indexes S_i that resulted in the highest VF levels such that T_{crit}, $P_{\max}$, and $I_{\max}$ were not exceeded. The conditions in lines 23 and 27 guarantee correct search indexes, accounting for the different search area sizes on the cores.

Algorithm 1 *seBoost*: Given Boosting Requirements

Input: $\{F_{\text{nom}_1}^{c_1}, \ldots, F_{\text{nom}_M}^{c_M}\}$, $\mathbf{T}_{\text{init}}$, t_1, W, $F_{R_i}^{c_i}$ for all $i \in W$;
Output: Voltage/frequency pairs $F_{S_i}^{c_i}$ for all $i = 1, 2, \ldots, M$;

1: $h \leftarrow$ First element in $\overline{W}$;
2: **for all** $i \in W$ **do**
3: $\quad S_i \leftarrow u_i^{\text{mid}} \leftarrow R_i$;
4: **end for**
5: **for all** $i \in \not{W}$ **do**
6: $\quad S_i \leftarrow u_i^{\text{mid}} \leftarrow 0$;
7: **end for**
8: **for all** $i \in \overline{W}$ **do**
9: $\quad S_i \leftarrow u_i^{\min} \leftarrow 1$;
10: $\quad u_i^{\max} \leftarrow \text{nom}_i$;
11: $\quad$ **if** $\text{nom}_h < \text{nom}_i$ **then** $h \leftarrow i$; **end if**
12: **end for**
13: **while** $u_h^{\min} \leq u_h^{\max}$ **do**
14: $\quad$ **for all** $i \in \overline{W}$ **do**
15: $\quad\quad u_i^{\text{mid}} \leftarrow \max\left\{\left\lfloor \frac{u_i^{\max}+u_i^{\min}}{2} \right\rfloor, 1\right\}$;
16: $\quad$ **end for**
17: $\quad \sum \text{I} \leftarrow \sum_{i=1}^{M} I_{c_i}\left(u_i^{\text{mid}}\right)$;
18: $\quad \sum \text{P} \leftarrow \sum_{i=1}^{M} P_{c_i}\left(u_i^{\text{mid}}\right)$;
19: $\quad \text{maxT} \leftarrow \max_{k=1,\ldots,M}\{T_k(t_1)\}$; {Eq (2) & (3), for $P_{c_i}\left(u_i^{\text{mid}}\right)$}
20: $\quad$ **if** $\text{maxT} \leq T_{\text{crit}}$ **and** $\sum \text{P} \leq P_{\max}$ **and** $\sum \text{I} \leq I_{\max}$ **then**
21: $\quad\quad$ **for all** $i \in \overline{W}$ **do**
22: $\quad\quad\quad S_i \leftarrow u_i^{\text{mid}}$;
23: $\quad\quad\quad$ **if** $u_i^{\min} \leq u_i^{\max}$ **then** $u_i^{\min} \leftarrow u_i^{\text{mid}} + 1$; **end if**
24: $\quad\quad$ **end for**
25: $\quad$ **else**
26: $\quad\quad$ **for all** $i \in \overline{W}$ **do**
27: $\quad\quad\quad$ **if** $u_i^{\min} \leq u_i^{\max}$ **then** $u_i^{\max} \leftarrow u_i^{\text{mid}} - 1$; **end if**
28: $\quad\quad$ **end for**
29: $\quad$ **end if**
30: **end while**
31: **return** $F_{S_1}^{c_1}, F_{S_2}^{c_2}, \ldots, F_{S_M}^{c_M}$;

The number of combinations tested by *seBoost* is merely $\log\left(\max_{\forall i \in \overline{W}}\{\text{nom}_i\}\right)$. Contrarily, a brute force algorithm would have tested $\prod_{\forall i \in \overline{W}} \text{nom}_i$ combinations. If running all non-boosted cores at their minimum frequencies still exceeds $P_{\max}$, $I_{\max}$, or T_{crit} at time t_1, then clearly the boosting time interval will be shorter than the maximum expected under the given VF level requirements and initial temperatures.

5.2 Unknown Required Boosting Levels

In this subsection, we consider that the required boosting levels of the cores that need boosting are unknown, but the duration of the maximum expected boosting time is known. For this case, the pseudo-code of *seBoost* is presented in Algorithm 2. The core computation of Algorithm 2 is based on Algorithm 1 (called as a function), but it requires some additional logic. Namely, by assuming that the boosted cores are set to the maximum VF levels and the non-boosted cores are set to the minimum VF levels, *seBoost* first verifies if the temperature at t_1 remains below T_{crit}, and whether the current and power constraints are satisfied. If this condition holds, this means there exists at least one solution in which all boosted cores can be executed at maximum performance. Thus, since maximizing the performance of the boosted cores is the priority, R_i is set to H_i for all $i \in W$ and we execute

Algorithm 2 *seBoost*: Unknown Boosting Requirements

Input: $\{F^{c_1}_{\text{nom}_1}, \ldots, F^{c_M}_{\text{nom}_M}\}$, $\mathbf{T}_{\text{init}}$, t_1, and W;
Output: Voltage/frequency pairs $F^{c_i}_{S_i}$ for all $i = 1, 2, \ldots, M$;
1: **for all** $i \in W$ **do**
2: $S_i \leftarrow H_i$;
3: **end for**
4: **for all** $i \in \cancel{W}$ **do**
5: $S_i \leftarrow 0$;
6: **end for**
7: **for all** $i \in \overline{W}$ **do**
8: $S_i \leftarrow 1$;
9: **end for**
10: $\sum \text{I} \leftarrow \sum_{i=1}^{M} I_{c_i}\left(u_i^{\text{mid}}\right)$;
11: $\sum \text{P} \leftarrow \sum_{i=1}^{M} P_{c_i}\left(u_i^{\text{mid}}\right)$;
12: $\text{maxT} \leftarrow \max_{k=1,\ldots,M} \{T_k(t_1)\}$; {Eq (2) & (3), for $P_{c_i}(S_i)$}
13: **if** $\text{maxT} \leq T_{\text{crit}}$ **and** $\sum \text{P} \leq P_{\max}$ **and** $\sum \text{I} \leq I_{\max}$ **then**
14: Call Algorithm 1 with W, and $F^{c_i}_{R_i} = F^{c_i}_{H_i}$ for all $i \in W$;
15: **else**
16: Call Algorithm 1 with $\overline{W}$, $F^{c_i}_{R_i} = F^{c_i}_{1}$ for all $i \in \overline{W}$, and $F^{c_i}_{\text{nom}_i} = F^{c_i}_{H_i}$ for all $i = 1, 2, \ldots, M$;
17: **end if**
18: **return** $F^{c_1}_{S_1}, F^{c_2}_{S_2}, \ldots, F^{c_M}_{S_M}$;

Algorithm 1 with set W to find the VF levels for the non-boosted cores.

Contrarily, if all boosted cores cannot be executed at their maximum VF levels, then we need to find different boosting levels. For this purpose, R_i is set to 1 for all $i \in \overline{W}$ and we execute Algorithm 1 with set $\overline{W}$, instead of set W. In this way, the non-boosted cores are forced to run as slow as possible, and Algorithm 1 now selects the VF levels of the cores that *do* require boosting.

5.3 Unknown Maximum Expected Boosting Time

In this subsection, we extend our two previous algorithms to consider cases in which the maximum expected boosting time is unknown. Here, we assume that the surges can last for a very long time, and thus the goal is to find VF levels that can be sustained indefinitely.

Basically, instead of obtaining VF levels that result in transient temperatures for which the maximum temperature among all cores reaches T_{crit} precisely at t_1, now we focus on staying just below T_{crit} when $t \to \infty$, that is, in the steady state, without exceeding $P_{\max}$ and $I_{\max}$. Therefore, we only need to make two changes in order to extend our algorithms. First, in Line 19 and Line 12 of Algorithm 1 and Algorithm 2, respectively, we now compute T_{steady_k} through Equation (2) instead of computing $T_k(t_1)$ through Equation (3), such that "$\text{maxT} \leftarrow \max_{k=1,\ldots,M} \{T_k(t_1)\}$" becomes "$\text{maxT} \leftarrow \max_{k=1,\ldots,M} \{T_{\text{steady}_k}\}$". Secondly, in Line 20 and Line 13 of Algorithm 1 and Algorithm 2, respectively, since now we want to stay just below T_{crit}, we change "$\text{maxT} \leq T_{\text{crit}}$" to "$\text{maxT} < T_{\text{crit}}$".

6. CONCURRENCY AND CONTROL LOOP

Algorithms 1 and 2 implicitly assume that all boosted cores have the same maximum expected boosting time. When this is not the case, time t_1 is set to the maximum value among all boosting times. Furthermore, Algorithm 1 (or 2) should be re-executed every time the boosting requirements change. That is, once one or more cores finish their boosted execution, Algorithm 1 (or 2) should be re-executed (now with less cores requiring boosting), in order to improve the overall performance of the system. Similarly, if during a boosting period one or more additional cores require boosting, time t_1 is set to the maximum expected absolute boosting time among all boosted cores, and Algorithm 1 (or 2) is re-executed to adjust the VF levels in order to feasibly boost all required cores: the cores that were already boosted plus the new cores that also require boosting. Figure 4 shows an example with such concurrent boosting requirements.

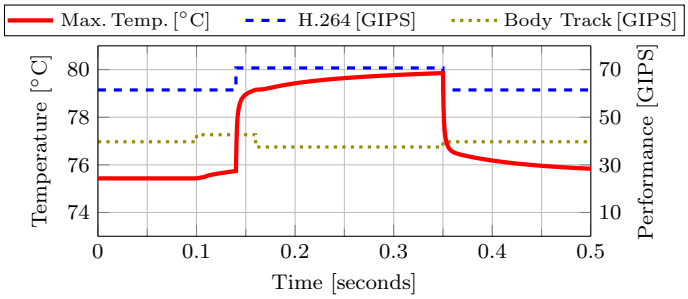


Figure 4: Concurrent boosting example. Body track is boosted to 3.3 GHz from 0.10 s to 0.16 s, and H.264 is boosted to 3.4 GHz from 0.14 s to 0.35 s. The red line shows the maximum temperature among all cores (left axis). The performance of the applications is measured in Giga-Instruction Per Second (GIPS).

Furthermore, although the simple closed-loop control system used by Turbo Boost does not guarantee meeting the run-time performance surges, it does provide a simple method to prolong the boosting intervals *after* the temperature, power, or current constraints were exceeded. For example, even running the non-boosted cores at their minimum frequencies, it is not always possible to satisfy the run-time surges during the entire time requirement, particularly for high initial temperatures. For such cases, it would be very pessimistic to directly return to nominal operation after the temperature, power, or current constraints are exceeded. Thus, *seBoost* can incorporate a simple closed-loop control system like the one used by Turbo Boost which is triggered *after* any of these constraints is exceeded. This simple control system can then reduce the VF levels of the boosted cores, not longer meeting the requirements surges, but achieving higher performance than at nominal operation.

7. EVALUATIONS

This section presents experimental evaluations conducted with gem5 [2], McPAT [12], and HotSpot [9], plus traces from measurements on an Odroid-XU3 [8] mobile platform with an Exynos 5 Octa (5422) [18] chip. We compare *seBoost* against Turbo Boost [3], and against a simple boosting method that throttles down the non-boosted cores to the slowest frequencies.

7.1 Setup

We consider a heterogeneous 72 core system, shown in Figure 5. The system consists of 24 *out-of-order* (OOO) Alpha 21264 cores and 16 *simple* Alpha 21264 cores, based on simulations conducted on gem5 [2] and McPAT [12] for 22 nm, and 16 *in-order* Cortex-A7 cores and 16 OOO Cortex-A15

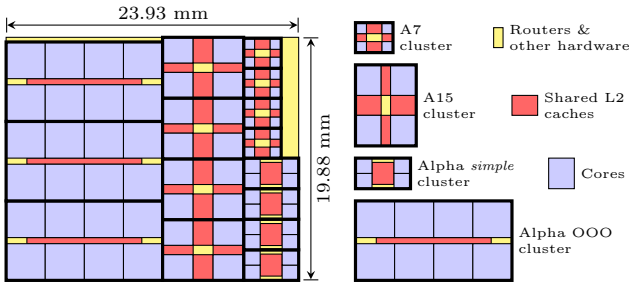


Figure 5: Floorplan for the heterogeneous 72 core system considered in our experimental evaluations. For a given cluster type, each cluster is identified as a, b, c, and d, from top to bottom.

cores, based on real measurements on an Odroid-XU3 [8] mobile platform with an Exynos 5 Octa (5422) [18] chip with ARM's "big.LITTLE" architecture. According to our simulations, each OOO Alpha core has an area of 9.6 mm^2, and there is a shared 2 MB L2 cache every eight cores. Moreover, each *simple* Alpha core has an area of 1.6 mm^2, and there is a shared 2 MB L2 cache every four cores. The areas of the A7 and A15 cores are estimated from die figures of the Exynos 5 Octa (5422) [18] chip as 0.8 mm^2 and 5.2 mm^2, respectively. There is a shared 512 KB L2 cache every four A7 cores, and a shared 2 MB L2 cache every four A15 cores. For such an architecture and its corresponding floorplan (Figure 5), we use HotSpot [9] (with its default configuration) to model the RC thermal network, which is required by *seBoost* to estimate the temperatures through Equations (2) and (3). For the OOO and *simple* Alpha cores, we assume available frequencies $\{0.2, 0.4, \ldots, 4.0\}$ GHz, and the voltage settings for each frequency are taken from the work in [6]. For the A7 and A15 cores, the available frequencies in the Odroid-XU3 platform are $\{0.2, 0.3, \ldots, 1.4\}$ GHz and $\{1.2, 1.3, \ldots, 2.0\}$ GHz, respectively, and the voltage values are automatically selected by the platform.

For benchmarks, we consider the Parsec benchmark suite [1], where each application can run $1, 2, \ldots, 8$ parallel dependent threads on the OOO Alpha cores, and $1, 2, \ldots, 4$ parallel dependent threads on the other three types of cores. The ambient temperature is set to 45°C, and we consider a critical temperature of 80°C.

7.2 Results

We run the applications from the Parsec benchmark suite under different scenarios. First, we focus on different applications individually, considering multiple instances of the same application, with different number of threads per instance and also different thread-to-core mappings. For each scenario we also consider different arrival periods for the run-time performance surges and different maximum expected boosting times. Details can be found in Table 1. Secondly, we focus on mixed application scenarios, considering several cases with different applications, number of threads and mappings. We consider 10 different scenarios, detailed in Table 2. For each scenario in Table 1 and Table 2, we conduct closed-loop evaluations involving simulations with gem5 [2] and McPAT [12] for the Alpha cores, power and performance traces from real measurements in the Odroid-XU3 platform for the A7 and A15 cores, and thermal simulations

Scenario	Alpha OOO	Alpha *simple*	A15	A7
I1 Boost: 10s Period: 30s	a: $4^{[3.6]}$ threads $4^{[3.6]}$ threads b: $4^{[3.6]}$ threads $4^{[3.6]}$ threads c: 4 threads 4 threads	a: $4^{[4.0]}$ threads b: $4^{[4.0]}$ threads c: $4^{[4.0]}$ threads d: $4^{[4.0]}$ threads	a: 4 threads b: 4 threads c: 4 threads d: 4 threads	a: 4 threads b: 4 threads c: 4 threads d: 4 threads
I2 Boost: 20s Period: 25s	a: 8 threads b: 8 threads c: $2^{[4.0]}$ threads $6^{[4.0]}$ threads	a: - b: - c: 3 threads d: 2 threads	a: $4^{[2.0]}$ threads b: $2^{[2.0]}$ threads c: $2^{[2.0]}$ threads d: -	a: - b: $2^{[1.4]}$ threads c: $2^{[1.4]}$ threads d: $4^{[1.4]}$ threads
I3 Boost: ∞ Period: -	a: $5^{[4.0]}$ threads $3^{[4.0]}$ threads b: - c: 7 threads 1 threads	a: $4^{[4.0]}$ threads b: 1 threads c: 2 threads d: -	a: $2^{[2.0]}$ threads b: - c: 1 threads d: 4 threads	a: $4^{[1.4]}$ threads b: - c: - d: 2 threads
I4 Boost: 7s Period: 10s	a: $6^{[4.0]}$ threads $2^{[4.0]}$ threads b: 8 threads c: 4 threads 4 threads	a: 3 threads b: 4 threads c: - d: -	a: - b: - c: 4 threads d: 2 threads	a: - b: 1 threads c: $4^{[1.4]}$ threads d: -

Table 1: Details of the application mapping scenarios for the experiment with individual applications. Indexes a, b, c, and d represent the cluster ID as explained in Figure 5. Every line corresponds to an application instance executed in the corresponding cluster with the indicated number of threads, where "-" means that a cluster is not executing any application. A super-index enclosed in brackets next to the number of threads implies that the specific application instance will have run-time performance surges, where the target frequency is the number between the brackets (in GHz) and the duration of the surges is detailed in the scenario number column. Here, *Boost* details the duration of the run-time performance surges, and *Period* details how often such surges arrive.

with HotSpot [9]. We consider that the nominal frequency for the OOO and *simple* Alpha cores is 2.0 GHz, and the nominal frequencies for the A7 and A15 cores are 0.8 GHz and 1.5 GHz, respectively. Moreover, for every application scenario, we consider that the system was running for long enough to achieve the steady-state temperatures at nominal frequencies, and we assume these temperatures to be the initial temperature in each case. Given that the Odroid-XU3 platform has no performance counters to measure the total number executed instructions, we use throughput as our performance metric, where throughput is defined as the total number of application instances finished every second. We consider that every time an application instance finishes, another instance is immediately executed under the same mapping and frequency settings. Figure 6 shows an overview of our simulation framework.

Figure 7 shows the timing behavior of each policy for the mixed application scenario M6, as detailed in Table 2. The overheads incurred by *seBoost* to decide the boosting levels at run-time are considered in the experiments, resulting in 7.5 ms for this specific case as seen in the figure. With respect to performance, we can see that both the simple throttling-down method and *seBoost* satisfy the run-time requirements surges, but *seBoost* manages this with higher performance for the non-boosted applications. As for Turbo Boost, we see that the performance of the non-boosted applications is in fact much higher than that of *seBoost*. The average performance of the boosted applications is also slightly higher than that of *seBoost*. However, although the average boosted performance is slightly higher (only 3%), Turbo Boost fails to *constantly* meet the minimum run-time performance requirements for the boosted applications by a very small amount (around 2 frequency steps), specifically, only satisfying the

Scenario	Alpha OOO	Alpha *simple*	A15	A7
M1 Boost: ∞ Period: -	a: 2 blacks. 6 x264 b: $4^{[3.6]}$ ferret $4^{[3.6]}$ bodytr. c: 8 dedup	a: - b: $2^{[3.8]}$ dedup c: 1 ferret d: $3^{[3.8]}$ bodytr.	a: - b: 1 facesim c: $4^{[1.9]}$ fluidan. d: 2 blacks.	a: 2 ferret b: - c: $2^{[1.0]}$ vips d: 4 stream.
M2 Boost: 3s Period: 45s	a: $8^{[4.0]}$ bodytr. b: $1^{[4.0]}$ x264 c: $8^{[4.0]}$ dedup	a: 1 dedup b: 4 canneal c: 3 x264 d: 2 blacks.	a: 4 bodytr. b: $2^{[1.8]}$ facesim c: $1^{[1.8]}$ swaptn. d: 1 x264	a: 4 swaptn. b: 2 vips c: 2 freqm. d: 2 x264
M3 Boost: 27s Period: 35s	a: $5^{[4.0]}$ dedup $3^{[4.0]}$ blacks. b: - c: $7^{[3.4]}$ ferret $1^{[3.4]}$ dedup	a: - b: - c: - d: $4^{[4.0]}$ dedup	a: - b: - c: - d: 1 streamcl.	a: 2 blacks. b: 1 bodytr. c: 4 x264 d: 4 fluidan.
M4 Boost: 30s Period: 50s	a: 8 bodytr. b: 7 x264 1 ferret c: 8 blacks.	a: - b: - c: $4^{[4.0]}$ x264 d: $4^{[4.0]}$ canneal	a: $2^{[1.9]}$ stream. b: $2^{[1.8]}$ swaptn. c: $1^{[2.0]}$ bodytr. d: $1^{[2.0]}$ ferret	a: $4^{[1.2]}$ x264 b: $2^{[1.2]}$ freqm. c: - d: -
M5 Boost: 23s Period: 40s	a: 1 bodytr. 7 x264 b: $2^{[4.0]}$ blacks. $6^{[4.0]}$ dedup c: -	a: 2 ferret b: - c: - d: -	a: 1 bodytr. b: 2 blacks. c: 1 facesim d: 2 swaptn.	a: 4 ferret b: 2 facesim c: $4^{[1.4]}$ fluidan. d: 1 stream.
M6 Boost: 16s Period: 45s	a: $3^{[4.0]}$ canneal $2^{[4.0]}$ x264 b: 8 ferret c: 8 bodytr.	a: $1^{[3.4]}$ blacks. b: $2^{[3.2]}$ bodytr. c: 3 dedup d: $4^{[4.0]}$ x264	a: 4 freqm. b: 1 ferret c: 1 stream. d: 4 vips	a: 1 swaptn. b: 2 x264 c: 4 blacks. d: 2 bodytr.
M7 Boost: 10s Period: 30s	a: $4^{[3.6]}$ x264 $4^{[3.6]}$ canneal b: $4^{[3.6]}$ blacks. $4^{[3.6]}$ swaptn. c: 4 dedup 4 x264	a: 4 swaptn. b: 4 ferret c: $4^{[4.0]}$ ferret d: $4^{[4.0]}$ blacks.	a: $4^{[2.0]}$ fluidan. b: 4 freqm. c: 4 facesim d: $4^{[1.9]}$ stream.	a: 4 vips b: $4^{[1.4]}$ freqm. c: $4^{[1.2]}$ x264 d: 4 bodytr.
M8 Boost: 32s Period: 40s	a: - b: 8 dedup c: $4^{[4.0]}$ ferret $4^{[4.0]}$ x264	a: - b: - c: 3 x264 d: 3 dedup	a: 1 x264 b: 4 bodytr. c: 4 stream. d: 1 ferret	a: $1^{[1.4]}$ blacks. b: $2^{[1.3]}$ vips c: $2^{[1.0]}$ swaptn. d: $4^{[1.4]}$ facesim
M9 Boost: ∞ Period: -	a: $1^{[3.8]}$ ferret $7^{[3.8]}$ blacks. b: $4^{[4.0]}$ canneal c: 8 dedup	a: 1 dedup b: 3 bodytr. c: 2 ferret d: 4 swaptn.	a: 1 swaptn. b: $4^{[2.0]}$ blacks. c: - d: -	a: $1^{[1.3]}$ vips b: 2 ferret c: 4 stream. d: $2^{[1.3]}$ facesim
M10 Boost: ∞ Period: -	a: $5^{[4.0]}$ ferret $3^{[4.0]}$ bodytr. b: - c: 7 canneal 1 swaptn.	a: $4^{[3.4]}$ x264 b: 1 dedup c: 2 blacks. d: -	a: $2^{[1.9]}$ freqm. b: - c: 1 x264 d: 4 vips	a: $4^{[1.4]}$ facesim b: - c: - d: 2 stream.

Table 2: Details of the application mapping scenarios for the experiment with mixed applications. This table is very similar to Table 1. The main difference is that here we detail *which* application type is executed in each cluster, and the word *threads* is omitted.

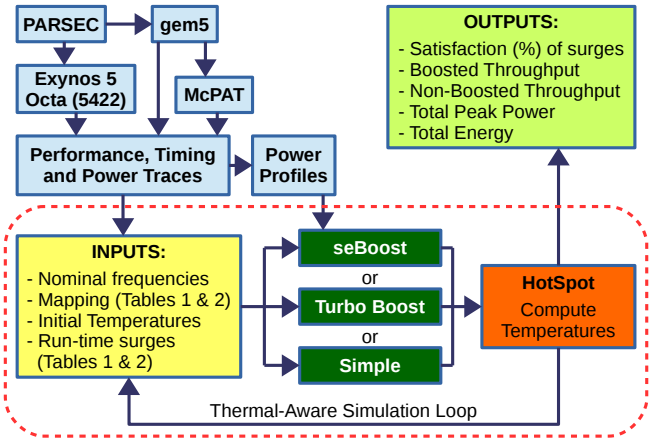


Figure 6: Overview of our simulation framework.

surges during 49% of the total boosting time.

Figure 8 shows results for 100 s of execution for all tested application scenarios detailed in Table 1 (individual application experiment). Similarly, Figure 9 shows results for 100 s

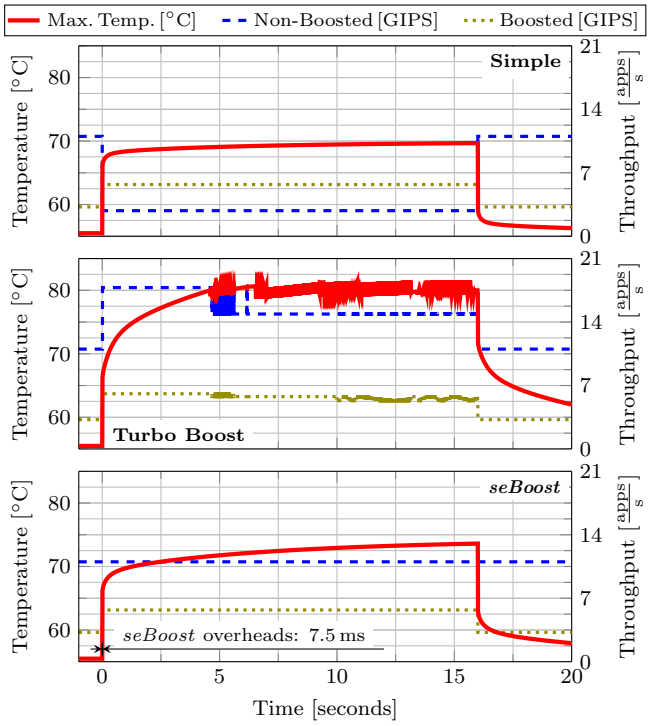
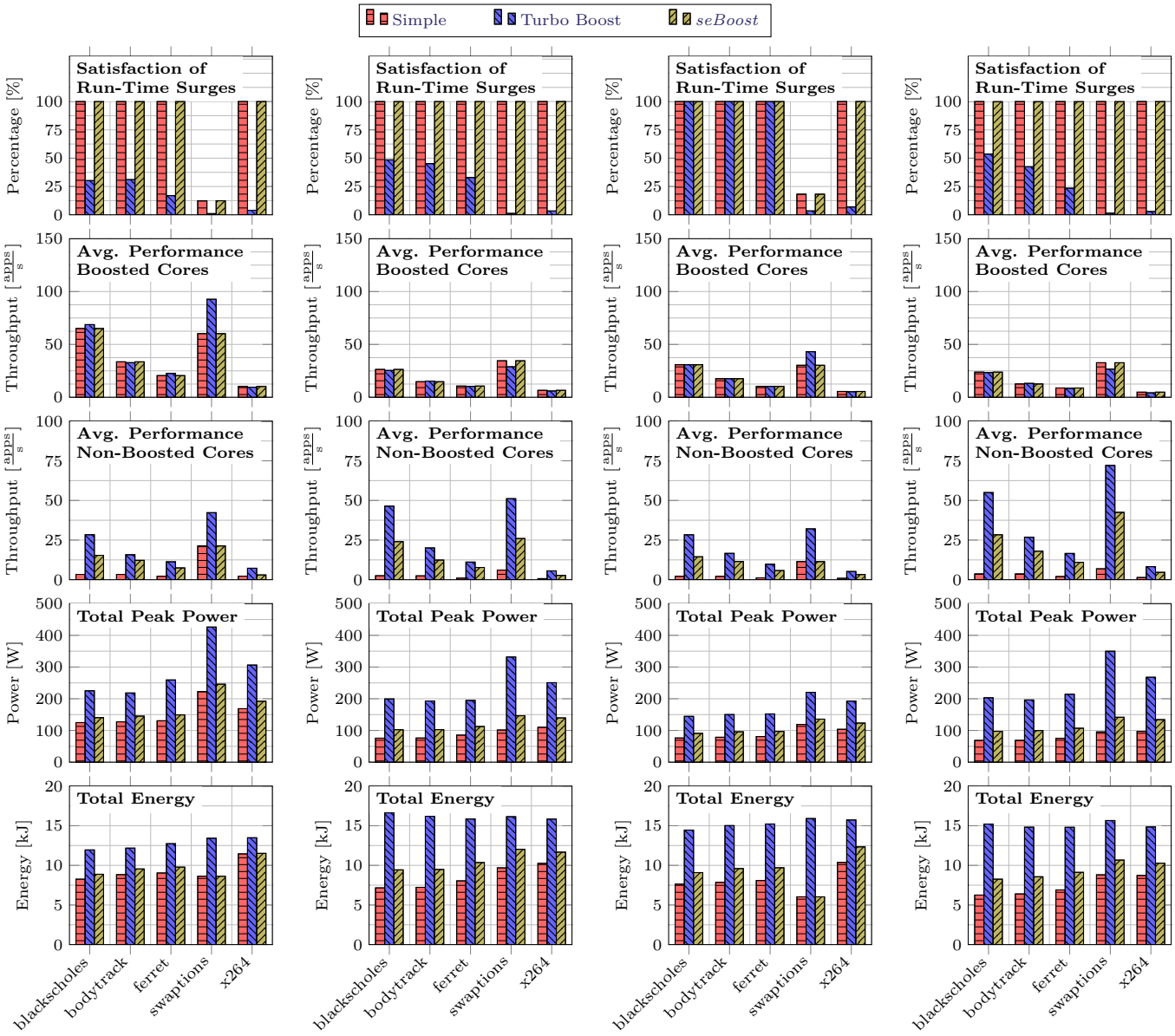


Figure 7: Timing simulation results for mixed application scenario M6. The red line shows the maximum temperature among all cores (left axis). The added performance of the boosted and non-boosted applications is measured in application instances per second, that is, throughput.

of execution for all scenarios detailed in Table 2 (mixed applications experiment). Specifically, in each figure we can see the percentage of time that the run-time performance requirements were satisfied for each boosting policy, the total average performance for the boosted applications/cores, the total average performance for the non-boosted applications/cores, the total peak power consumption, and the total energy consumption. There are a few cases in which, for the given initial temperatures, it is not possible to satisfy the run-time requirements, and thus neither *seBoost* or the simple throttling-down method manages to satisfy the requirements 100% of the time. For the rest of the tested cases, *seBoost* and the simple throttling-down method satisfy the requirements during the entire boosting interval (except for *seBoost*'s small computation overheads of a few milliseconds), as seen in Figures 8 and 9. Nevertheless, the simple boosting does so while incurring unnecessary losses of performance for the non-boosted applications.

With respect to Turbo Boost, for all tested cases, although it achieves higher average performance than *seBoost* for the non-boosted cores, Turbo Boost *rarely* manages to satisfy the run-time requirements during the entire boosting interval, and the specific percentages vary drastically depending on the application scenarios. Furthermore, since Turbo Boost is not aware of the performance the applications require, sometimes it fails to satisfy the run-time requirements during the entire boosting interval because it boosts to VF levels higher than necessary. As seen in Figures 8 and 9, compared to *seBoost*, the *over-boosting* incurred by Turbo



(a) Scenario I1 (b) Scenario I2 (c) Scenario I3 (d) Scenario I4

Figure 8: Evaluation results for individual applications from Parsec. We consider multiple instances of the same application, with different number of threads per instance, different thread-to-core mappings, different arrival periods for the run-time performance surges, and different maximum expected boosting times. Details can be found in Table 1.

Boost can sometimes result in a higher average performance for the boosted cores. However, the performance gains for using Turbo Boost are relatively very small and arguably unjustified when considering the big increments to the total peak power and energy consumption. For example, Turbo Boost achieves 3% higher boosted average performance for scenario M6, while resulting in a peak power and energy consumption of 105% and 51%, respectively, higher than *seBoost*.

With respect to the overheads incurred by *seBoost*, the worst-case measured execution time for *seBoost* for all tested cases was below 7.5 ms, implemented in software (C++) as a single threaded application. This clearly shows that *seBoost* is in fact a *lightweight* technique suitable for run-time usage. Implementing *seBoost* as a dedicated hardware controller would result in negligible time overheads, specially given that the computations in Algorithms 1 and 2 can be fully parallelized.

8. CONCLUSIONS

This paper presented *seBoost*, an efficient and lightweight boosting technique based on transient temperature estimation. This technique guarantees meeting run-time performance requirements surges, by executing the boosted cores at the required frequencies for the entire boosting intervals, while throttling down the non-boosted cores. In order to

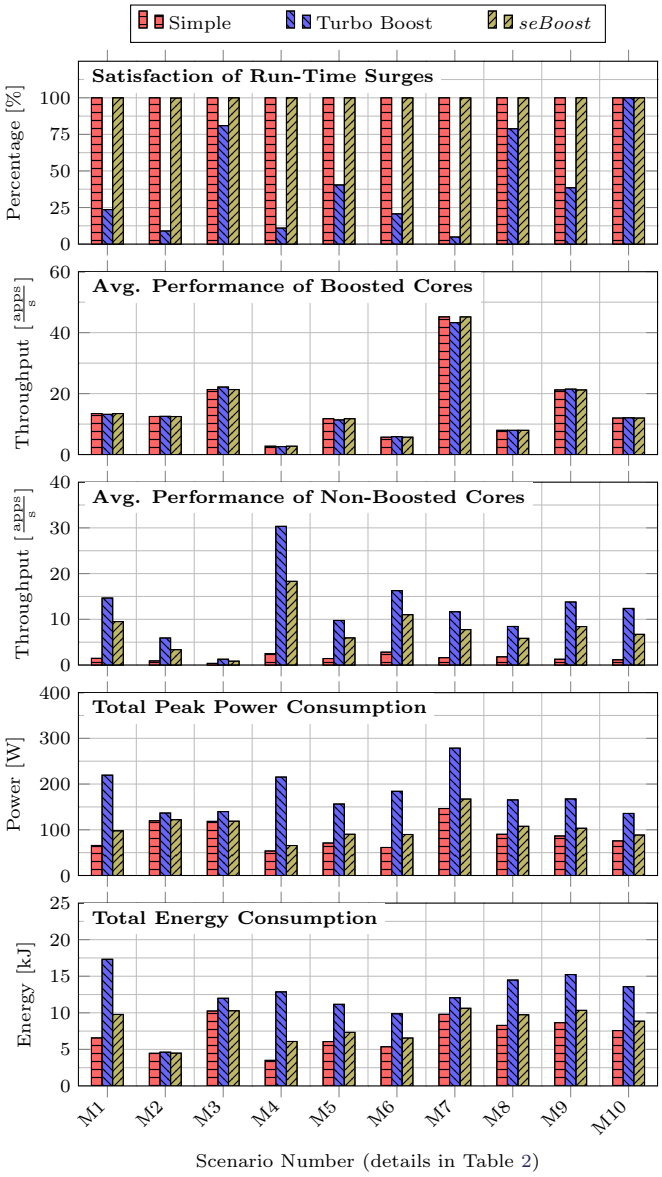


Figure 9: Evaluation results for mixed applications from Parsec. We consider different applications, with different number of threads per application instance, different thread-to-core mappings, different arrival periods for the run-time performance surges, and different maximum expected boosting times. Details can be found in Table 2.

minimize the performance losses for the non-boosted cores, the throttling down levels are chosen such that the maximum frequencies among all cores reaches the critical temperature precisely when the boosting is expected to expire. Our experiments show that *seBoost* can in fact guarantee the required performance surges, while the state-of-the-art control techniques fail to constantly satisfy the run-time requirements.

9. ACKNOWLEDGEMENTS

This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Centre *Invasive Computing* [SFB/TR 89].

10. REFERENCES

[1] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 72–81, 2008.

[2] N. Binkert, B. Beckmann, and others. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, Aug. 2011.

[3] J. Casazza. First the tick, now the tock: Intel microarchitecture (nehalem). White paper, Intel Corporation, 2009.

[4] J. Charles, P. Jassi, N. S. Ananth, A. Sadat, and A. Fedorova. Evaluation of the intel core i7 turbo boost feature. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, pages 188–197, 2009.

[5] A. K. Coskun, T. v. Rosing, and K. C. Gross. Utilizing predictors for efficient thermal management in multiprocessor socs. *Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 28(10):1503–1516, Oct. 2009.

[6] A. Grenat, S. Pant, R. Rachala, and S. Naffziger. 5.6 adaptive clocking system for improved power efficiency in a 28nm x86-64 microprocessor. In *IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pages 106–107, 2014.

[7] V. Hanumaiah, S. Vrudhula, and K. S. Chatha. Performance optimal online dvfs and task migration techniques for thermally constrained multi-core processors. *Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 30(11):1677–1690, Nov. 2011.

[8] Hardkernel Co., Ltd. Odroid-XU3. www.hardkernel.com.

[9] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. Stan. HotSpot: a compact thermal modeling methodology for early-stage VLSI design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(5):501–513, May 2006.

[10] Intel Corporation. Dual-core intel xeon processor 5100 series datasheet, revision 003, August 2007.

[11] Intel Corporation. Intel turbo boost technology in intel CoreTM microarchitecture (nehalem) based processors. White paper, November 2008.

[12] S. Li, J.-H. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 469–480, 2009.

[13] H. M. Moya-Cessa and F. Soto-Eguibar. *Differential Equations: An Operational Approach*. Rinton Press, 2011.

[14] S. Nussbaum. AMD trinity APU. In *Hot Chips*, 2012.

[15] S. Pagani, J.-J. Chen, M. Shafique, and J. Henkel. MatEx: Efficient transient and peak temperature computation for compact thermal models. In *Proceedings of the 18th Design, Automation and Test in Europe (DATE)*, March 2015.

[16] A. Raghavan, Y. Luo, A. Chandawalla, M. Papaefthymiou, K. P. Pipe, T. F. Wenisch, and M. M. K. Martin. Computational sprinting. In *Proceedings of the IEEE 18th International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1–12, 2012.

[17] E. Rotem, A. Naveh, D. Rajwan, A. Ananthakrishnan, and E. Weissmann. Power-management architecture of the intel microarchitecture code-named sandy bridge. *IEEE Micro*, 32(2):20–27, March 2012.

[18] Samsung Electronics Co., Ltd. Exynos 5 Octa (5422). www.samsung.com/exynos.

[19] T. Sherwood, E. Perelman, G. Hamerly, S. Sair, and B. Calder. Discovering and exploiting program phases. *IEEE Micro*, 23(6):84–93, Nov. 2003.

[20] C. Tan, T. Muthukaruppan, T. Mitra, and L. Ju. Approximation-aware scheduling on heterogeneous multi-core architectures. In *20th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 618–623, Jan 2015.

[21] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld. Face recognition: A literature survey. *Computing Surveys (CSUR)*, 35(4):399–458, December 2003.