Thermal-Aware Task Allocation and Scheduling for Heterogeneous Multi-core Cyber-Physical Systems

Shikang Xu, Israel Koren and C. M. Krishna
Department of Electrical and Computer Engineering University of Massachusetts Amherst, Amherst, MA, 01003

Abstract— *Over the past few years, the impact of heating on device failure rates has become increasingly important in cyber-physical systems. Such systems are often used in harsh environments and have weight, volume and power constraints which make it hard to dissipate heat economically and effectively. Thermal-aware task scheduling techniques are therefore called for to reduce heat stress on the computational platform.*

In this paper, we outline a thermal-aware task allocation and scheduling heuristic for use in computational platforms that feature heterogeneous computational cores. Such systems, consisting typically of some powerful, out-of-order, cores, together with simple, in-order, cores, allow the user a wider range of power-performance tradeoffs than traditional homogeneous multicore systems. We show how using fairly simple thermal-aware task allocation and scheduling principles results in a substantial enhancement in the expected lifetime of the system.

Index Terms*— *DVFS, Real-time systems, Cyber-physical systems, Lifetime extension, Heterogeneous multi-core

1 Introduction

Cyber-physical systems (CPSs) are proliferating in life-critical and cost-sensitive contexts today. The economical provision of high reliability in such systems is very important.

Thermal stress is an important source of accelerated device aging, resulting in premature device failure. Indeed, failure rates are often modeled as exponentially increasing with absolute temperature. Furthermore, much of the thermally-induced damage is cumulative, not transient. As technology has advanced and feature sizes have shrunk, heat density has increased, making high temperature an important factor in device degradation and failure.

We focus in this paper on how to allocate and schedule CPS tasks on heterogeneous multi-core platforms, in a thermal-aware fashion. Such platforms, such as big.LITTLE from ARM [1], typically consist of high-performance cores with advanced pipelining features for enhanced throughput, coupled with very small cores only capable of simple, in-order, instruction processing. All cores, simple and complex, on the platform share the same instruction set so that task migration across core types is possible. Such platforms are attractive for CPS since they offer an increased flexibility in scheduling dynamically varying real-time workloads [2]–[4].

In this paper, we introduce a simple task allocation and scheduling algorithm for real-time systems using a heterogeneous multi-core platform. The aim is to meet all deadlines while reducing thermal stress. Our simulations show that this algorithm significantly enhances the core Mean Time To Failure (MTTF) over a baseline algorithm.

The rest of the paper is organized as follows. Section 2 reviews related prior work and points out the thermal impact on lifetime and reliability. Section 3.2 contains our algorithm. Simulation results illustrating the performance of this algorithm are presented in Section 4. Section 5 concludes this paper.

This work was partially supported by the National Science Foundation under grant CNS 1329831.

2 Background

2.1 Related Works

Over the past decade, much work has been conducted on general-purpose, single-ISA, heterogeneous multi-core systems task scheduling [5]–[10]. In [9], Craeynest *et al.* developed a workload-to-core mapping algorithm for single-ISA heterogeneous multi-core processors to improve performance by dynamically monitoring the behavior (through measures like Cycles per Instruction and Instruction-Level Parallelism) of tasks. In [10], Zhang *et al.* proposed an algorithm based on dynamic execution behavior of workloads, to map tasks to cores to achieve higher energy-efficiency while maintaining comparable performance as in [9].

For real-time systems, recent research focuses on algorithms to guarantee an increased amount of workload to be finished before its deadline. Baruah introduced a polynomial-time feasibility analysis method that can determine if a set of real-time tasks can be scheduled on an multi-core system and tasks can migrate between cores [2]. Kim *et al.* proposed a scheduling algorithm for aperiodic hard real-time tasks executing on heterogeneous platforms that restrict tasks to be executed on certain cores according to their expected completion time [11].

Raravi *et al.* showed that, if the tasks can only migrate between the cores of the same type, an optimal algorithm needs cores that are $1 + \frac{\beta}{2}$ times faster ($0 < \beta < 1$) than in a system where tasks can migrate among multiple core types [3].

Recently, Chwa *et al.* proposed a task scheduling algorithm for real-time systems with two core types [4]. In their algorithm, the cores are assumed to share the same instruction set architecture (ISA) so that tasks can migrate between core types. A two-phase approach is followed: first, assignment of tasks to the appropriate core type, and second, a modified DP(deadline partitioning)-fair [12] method is used to schedule the workload on each type.

2.2 Thermal Related Reliability and Lifetime

The reliability of VLSI circuits is affected by multiple mechanisms. Modeling these has been an active research topic for decades. Electromigration (EM) and Oxide Breakdown are reported to be dominant permanent failure mechanisms of VLSI circuits as CMOS technology scales [13] and consequently, these are the modes that we focus on.

According to [14], the Mean-Time-To-Failure (MTTF) due to the oxide breakdown process is given by:

$$MTTF_{bd} = A_{bd} * V^{-(a-bT)} * e^{\frac{X+(Y/T)+ZT}{kT}} \quad (1)$$

where V is the voltage applied to the gate oxide, T is the absolute temperature in Kelvin, k is Boltzmann's constant and A_{bd} is a scale factor. The values of the other parameters are [14]: $a = 78$, $b = -0.0081$, $X = 0.759eV$, $Y = -66.8eV * K$ and $Z = -8.37e4eV/K$. Note that these parameters, or even the MTTF model, can be different as the CMOS manufacture technology developing.

The mechanism behind EM has been studied extensively. One of the early results that is still widely used for estimating the MTTF due to EM, was proposed by Black [15]:

$$MTTF_{em} = A_{em} * J^{-n} e^{\frac{E_a}{kT}} \quad (2)$$

where A_{em} is a scale factor, J is the current density, E_a is activation energy and n is a material based constant. For copper, these values are $J = 1e^6 A/cm$ [16], $E_a = 0.9eV$ and $n = 1.1$ [17].

The failure of a system is a random process and the reliability of a system at time t is the probability that the system is functional throughout the time interval $[0, t]$. The probability of a device failure occurrence if often modeled by by the Weibull distribution [18]:

$$F(t) = 1 - R(t) = 1 - e^{-(t/\alpha)^{\beta}} \quad (3)$$

where $F(t)$ is the failure occurrence probability, $R(t)$ is the reliability function, β is the Weibull slope parameter (β=2, [19]), and α is a scale parameter satisfying $\alpha = MTTF/\Gamma(1+1/\beta)$.

The reliability expressions shown above were obtained from a combination of experimental results and proposed physical models. In this paper, the approach of [18] is adopted to calculate the reliability in a dynamic thermal environment (i.e., under varying temperatures). Time is divided into k time frames, [0,Δt],[Δt, 2Δt], ..., [(k-1)Δt, $k\Delta$t]. Each time frame is short enough so that the temperature and voltage are roughly constant within it. The resulting reliability of a functional block in an IC (e.g., an integer execution unit), denoted by $R_{blk}(t)$, is:

$$R_{blk}(t) = R(k\Delta t) = \prod_{i=1}^{i=k} [1 - (R_i((i-1)\Delta t) - R_i(i\Delta t))] \quad (4)$$

where $R_i(i\Delta t)) = R_{i_{em}}(i\Delta t)) * R_{i_{bd}}(i\Delta t))$; $R_{i_{em}}(i\Delta t))$ and $R_{i_{bd}}(i\Delta t))$ are the reliabilities due to electromigration and oxide breakdown, respectively, and are calculated by Equation 3 using the MTTFs derived from the reliability model using the temperature of the ith time interval.

The reliability of a core at time t is the product of the reliability of all the functional blocks of the core at time t.

3 Thermal Aware Heterogeneous Multi-core Scheduling Algorithm

3.1 System Models

The heterogeneous multi-core system we consider consists of two core types: out-of-order ("big") and in-order ("small") cores. The former are much more complicated, with sophisticated pipelining techniques to improve performance (at the cost of power consumption); the latter are simple and slow. All cores use the same instruction set architecture, meaning that tasks can, deadlines permitting, execute on any core. All cores of the same type are identical to one another. As already mentioned, this structure is commercially realized these days, one example being the ARM big.LITTLE architecture.

The task set is periodic, with deadlines equal to the period. This is a common model for real-time systems in practice [20]; we will later extend it to the case where deadlines are constrained to be less than or equal to their period. Each task τ_i is characterized by the following three parameters: period (p_i), Worst-Case Execution Time (WCET) on a big core (e_i^b) and WCET on a small core (e_i^s). In this paper, it is assumed that for all τ_i, $e_i^b \leq e_i^s$ and the utilization on the big core satisfies $u_i^b = \frac{e_i^b}{p_i} \leq 1$. If the utilization on small core, $u_i^s = \frac{e_i^s}{p_i}$, is larger than 1, only using small cores is insufficient to meet the worst-case computational demands of the workload. Tasks can be migrated from one core to another during execution.

We also assume that the cores share the main memory and last level cache (LLC). With this assumption, the task migration overhead is negligible [9]. For systems with many-cores, the assumption may not be true. However, cores are usually clustered into groups that share LLC and memory. The proposed algorithm can be applied to the cores that are lie in the same group.

The reliability figure-of-merit we use is the product of the individual reliabilities of all the cores. The expected lifetime of the system is defined as the point at which this figure of merit declines to a predetermined level.

3.2 Task Assignment and Scheduling

3.2.1 Baseline Algorithm

To provide appropriate context for our work, we start with a brief description of a baseline multicore scheduling algorithm. As already mentioned, Chwa, *et al.*, propose an optimal task scheduling algorithm for heterogeneous multi-core real-time systems. Two core types are assumed [4]. Optimality is demonstrated by proving that as long as a task set can be feasibly scheduled on a heterogeneous multi-core system, it can be scheduled by their algorithm. The algorithm proposed in [4] includes two phases: workload assignment (Hetero-Split) and schedule generation (Hetero-Wrap). A high-level description is shown in Figure 1. In [4], it is assumed that some tasks will execute faster on type-I cores while other tasks will execute faster on type-II cores. If the utilization of a task τ_i on the cores that executes it slower can be greater than 1, there is a minimum fraction of such a task that must be executed on

Notation:
e_i^1/e_i^2: WCET of τ_i on type-I/II core
Γ^1/Γ^2: list of tasks on type-I/II core
x_i^1/x_i^2: portion of τ_i on type-I/II core
M_1: list of tasks $u_i^1 * x_i^1 + u_i^2 * x_i^2 = 1$ and $x_i^1 > 0, x_i^2 > 0$
M_2: list of tasks $u_i^1 * x_i^1 + u_i^2 * x_i^2 < 1$ and $x_i^1 > 0, x_i^2 > 0$
P_1/P_2: list of tasks only assigned to type-I/II cores
Hetero-Split:
calculate minimum fraction of each task i on the faster core using equation (5) and (6)
$\Gamma^1 \leftarrow \{\tau_i | e_i^1 < e_i^2\}$, $\Gamma^2 \leftarrow \{\tau_i | e_i^2 < e_i^1\}$
IF workload on type-I (type-II) cores is larger than its capacity
repeat
find the τ_k with the smallest (largest) $\frac{e_i^1}{e_i^2}$
move τ_k to type-II (type-I) cores (lo_k^1 (lo_k^2) cannot be moved)
Until workload on type-I (type-II) cores is no larger than its capacity

Hetero-Wrap:
For tasks in M_1, M_2 and P_1
fill the type-I cores' slices from the beginning of slice
For tasks in M_1, M_2 and P_2
fill the type-II cores' slices from the end of slice

Fig. 1. General flow of the algorithm in [4]

the other core type. This minimum fraction (denoted by lo_i) is calculated in the following way:

$$lo_i^1 = \begin{cases} \frac{u_i^2 - 1}{u_i^2 - u_i^1}, & \text{if } u_i^2 > 1 \\ 0, & \text{otherwise} \end{cases} \quad (5)$$

$$lo_i^2 = \begin{cases} \frac{u_i^1 - 1}{u_i^1 - u_i^2}, & \text{if } u_i^1 > 1 \\ 0, & \text{otherwise} \end{cases} \quad (6)$$

where $u_i^1(u_i^2)$ is the utilization on Type I (Type II) core.

In the task assignment phase, the workload is assigned to the different core types. The first step is to determine the minimum portion of each task that needs to be executed on the type of core that executes it faster and allocate that portion to such a type. If the workload assigned to any type of core is more than its ability to schedule after such an assignment, then the algorithm cannot schedule the task set. The second step will initially assign the rest of the workload (tasks with utilization smaller than 1 on both types cores and tasks with utilization larger than 1 on one type of core excluding the portion dealt with in first step) to the type of core that can execute it faster. If there is a type of core that cannot finish its assigned workload on time, the task will then be moved to the other core type. The order at which tasks are moved is determined by the ratio of execution times on the two core types for each task, e.g., when moving from type-I to type-II core, the task with the largest $\frac{e_i^1}{e_i^2}$ will be moved first. But if the moving task has a portion that must be executed on current core (from equation (5) and (6)), this portion will not be moved.

In the phase of schedule generation, the algorithm is a modification of DP-fair [12]. The execution is divided into slices and each task will be executed for an interval that equals the product of its utilization on the current core and slice length, e.g., τ_i will be executed for 5 seconds on a type-I core if the slice is 10 seconds and τ_i has a utilization of 0.5 on type-I core. The situation where a task is scheduled to be executed on two types of cores at the same time needs to be avoided. Tasks are divided into four groups: M_1 are the tasks where the sum of utilization on the two core types equals 1; M_2 are tasks where the sum of the utilization on the two core types is less than 1; P_1 are tasks that are only assigned to type-I cores and P_2 are tasks that are only assigned to type-II cores. The time-slices of type-I cores will be assigned to tasks in the order of M_1, M_2 and P_1 from the *beginning* of each slice. The time-slices of type-II cores will be assigned to tasks in the order of M_1, M_2 and P_2 from the *end* of each slice.

3.2.2 Thermal-Aware Task Allocation and Scheduling

The baseline algorithm described above does not take thermal considerations into account. We describe here a thermal-aware task allocation and scheduling algorithm, which we will then show has significant lifetime-enhancement benefits. The system model assumed was described in Section IIIA, namely, we assume "smaller" and "bigger" cores. Smaller cores do not have performance-enhancing (but power-hungry) features like out-of-order execution.

The general principle behind this algorithm is to preferentially assign load to the smaller cores; only that portion of a task that cannot feasibly be executed on the smaller cores is assigned to the bigger core. Second, we use offline task profiling to obtain the average temperature (T_{avg}^i) when running a task on a big core. When some tasks have to be assigned to big cores, instead of choosing the task with smallest $\frac{e_i^b}{e_i^s}$ as in [4], we choose the task with smallest $T_{avg}^i * \frac{e_i^b}{e_i^s}$. (Note: this modification will potentially sacrifice optimality.)

The pseudo code of the thermal-aware task assignment is shown in Figure 2. In stage 1, the minimum fraction of each task that needs to be executed on big cores is allocated to big cores. If the big cores cannot execute the allocated minimum workload, the given task sets cannot be scheduled, and the algorithm returns failure. In stage 2, all the remaining workload is assigned to small cores. In stage 3, if the small cores cannot schedule the assigned workload, some workload needs to be moved to the big cores. In the workload reassignment, the task τ_k with the smallest product of $\frac{e_i^s}{e_i^b}$ and the average execution temperature on the big core will be moved first. If the small cores still cannot schedule the workload on them after moving the whole task, the whole τ_k will be moved to big cores. Otherwise, the algorithm only moves that part of the task such that the small cores can schedule the workload on them. Once the task assignment to cores is successfully determined according to this process, the Hetero-wrap scheduling procedure from the baseline algorithm is used to generate the task schedule.

3.3 DVFS Algorithm

Dynamic Voltage and Frequency Scaling (DVFS) has long been used to reduced the thermal impact on processors. In

Notation
u_i^b/u_i^s: utilization of τ_i on big/small core
x_i^b/x_i^s: fraction of τ_i assigned to big/small core
lo_i: minimum fraction of τ_i assigned to big core
y_i^b/y_i^s: fraction of τ_i assigned to big/small core excluding lo_i
Γ_b/Γ_s: list of tasks on big/small core
m_b/m_s: number of big/small cores
er_i: $T_{avg}^i * \frac{e_i^s}{e_i^b}$ of τ_i

Task_ alloc(T)
Stage 1:
Allocate lo_i
if $\sum lo_i * u_i^b > m_b$
  return not feasible
Stage 2:
for all τ_i in T
  $\Gamma_s \leftarrow \Gamma_s \cup \tau_i$
  $y_i^s \leftarrow 1 - lo_i$
Stage 3:
if $\sum y_i^s * u_i^s \leq m_s$
  return $\{x_i^b | x_i^b = lo_i\}$,$\{x_i^s | x_i^s = y_i^s\}$
else
  while $\sum y_i^s * u_i^s > m_s$
    find τ_k with smallest er_i in Γ_s
    if $\sum y_i^s * u_i^s - y_k^s * u_k^s > m_s$
      $y_k^b \leftarrow 1 - lo_k$
      $y_k^s \leftarrow 0$
      $\Gamma_s \leftarrow \Gamma_s - \tau_k$
    else
      $y_k^b \leftarrow \frac{\sum y_i^s * u_i^s - m_s}{u_k^s}$
      $y_k^s \leftarrow 1 - lo_k - y_k^b$
    if $\sum y_i^b * u_i^b > m_b - \sum lo_i * u_i^b$
      return not feasible
return $\{x_i^b | x_i^b = lo_i + y_k^b\}$,$\{x_i^s | x_i^s = y_i^s\}$

Fig. 2. Thermal Aware task assignment

Algorithm:IPC_DVFS
Notation:
T_{sys}: Tasks in system
P: list of partitioned tasks on core
t_{sys}: system time, initialized at 0
f_{high}/f_{low}: processor high/low frequency level
Δt: time step
t_{slice}: length of scheduling slice
F: flag array indicating if task i has finished the current iteration
IPC: Array recording the IPC of τ_i in the previous time step
SA: schedule array for core in each slice after task assignment

At every time step
For each τ_i in system
  IF τ_i finished in past Δt
    $F[i] = 1$
  IF t_{sim} mod p_i=0
    $F[i] = 0$
IF t_{sim} mod t_{slice}=0
  For all τ_i in T_{sys}
    IF $F[i]$=0
      $T \leftarrow T \cup \tau_i$
  $Task\_Alloc(T)$
  Generate SA for each core
  $Alloc\_Slack()$
  Set count=0 on each core
On each core
  execute $\tau_{SA[count].index}$ at speed SA[count].speed
  set IPC[SA[count].index] to IPC in the Δt
  IF t_{sys} mod t_{slice}=SA[count].end or task finishes
    count+=1

Fig. 3. IPC based DVFS

[21], a DVFS scheme based on Instructions-Per-Clock (IPC) monitoring is presented to improve the lifetime of processors by preferentially using slack to slowdown a high-IPC phase of workload. When executing tasks according to the slice schedule generated using the method described above, DVFS is applied using the concept proposed in [21].

As stated above, there are partitioned tasks and migrating tasks on cores. Applying DVFS on migrating tasks may result in executing the same task on different cores at the same time, which is not allowed. Thus, in this paper, only partitioned tasks are subject to slowdown by voltage scaling. The pseudo code of the DVFS algorithm is shown in Figures 3 and 4. In Figure 3, at the beginning of every time step, the tasks completed in the prior time step are indicated using a finishing flag array. If a new iteration of a task is released, this flag is reset. Also, at the beginning of each time slice, the schedule is rearranged based on current uncompleted tasks in the system; the available slack in the coming time slice is then calculated. The initial task assignment and scheduling is based on the Worst Case Execution Time (WCET) of each task. If a task finishes earlier than its WCET, there is no need to allocate time for this task in the coming time slices until its next iteration is released. Thus, at the beginning of every time slice, tasks that have not finished will be reassigned using the task assignment algorithm described in the previous section. Then, the slack will be used to slow down partitioned tasks on big cores using the algorithm shown in Figure 4.

On big cores, the partitioned tasks are arranged to be executed first in the coming time slices. Firstly, all the uncompleted partitioned are clustered to the front part of the slice and all the uncompleted migrating tasks are clustered to the end part of the slice. The gap between these two clusters is the available slack. As is shown in Figure 4, the IPC of each task in the prior time step is recorded. The available slack is then allocated to the task with the highest IPC. Based on how much slack is available, the task with highest IPC in a given time slice is either wholly or partially slowed down. If there is still some slack left after the above process, the task with the second highest IPC will go through the same process. This process continues until the slack is used up or all partitioned tasks are slowed down.

Notation
P: list of partitioned tasks
f_{high}/f_{low}: high/low frequency level of core
$SA[i].index$: index of task that is at position i of array SA
$SA[i].start$: start time of the task at position i
$SA[i].end$: end time of the task at position i
$SA[i].speed$: frequency level to run task position i

Alloc_Slack()
On each big core
Γ_p=$\{SA[i]|SA[i].index \in$P$\}$
front=0, end=t_{slice}
FOR i in (0..SA.size-1)
  IF $SA[i] \in \Gamma_p$
    l=SA[i].end-SA[i].start
    SA[i].start=front
    SA[i].end=front+l
    front=SA[i].end
FOR i in (SA.size-1..0)
  IF $SA[i] \notin \Gamma_p$
    l=SA[i].end-SA[i].start
    SA[i].end=end
    SA[i].end=end-l
    end=SA[i].start
$slack$=end-front
FOR SA[i] in Γ_p
//access in IPC in past Δt descending order
  l=SA[i].end-SA[i].start
  sn=l*$\frac{f_{high}-f_{low}}{f_{low}}$
  //slack needed to slowdown the how part of SA[i]
  IF slack>sn
    SA[i].speed=LV
    SA[i].end+=sn
    rearrange SA[k]s following SA[i] in SA
    slack-=sn
  ELSE IF slack>0
    sp=slack*$\frac{f_{low}}{f_{high}-f_{low}}$
    //workload can be slowed by slack
    SA[i].speed=LV
    SA[i].end=SA[i].start+slack+sp
    insert (index=SA[i].index, speed=LV, start=SA[i].end, end =start+l-sp) to SA
    rearrange SA[k]s following SA[i] in SA
    slack=0
  ELSE break

Fig. 4. Slack allocation on big core

4 Experimental Results

4.1 Experiment Setup

A system with one out-of-order (big) core and one in-order (small) core is simulated to compare the thermal-aware algorithm introduced in the previous section to the baseline algorithm. The big core has two frequency levels: 2.0 GHz and 1.2 GHz. Applications from the Mibench benchmark suite [22] are used as a workload of independent tasks. Task sets are generated by selecting five tasks randomly from the benchmark suite. The WCET for each task running on big/small core of each task is found by using the Gem5 [23] out-of-order/in-order simulator. The actual execution time of each iteration is a random number; it is selected to be normally distributed with mean equal to half the WCET and standard deviation of 0.2 of the mean, conditioned to be no greater than the WCET value.

To obtain the power consumption of the workload we used Gem5 and MaPAT [24]. These power traces were then used to generate temperature traces for each of the cores; we used the fast thermal simulator TILTS [25] which is a modification of HotSpot [26]. When a task has an actual execution time that equals r_i of its WCET, its power file will be compressed in the time domain to this fraction. This ensures simulation of all the high/low power phases in the execution of a task.

The reliability of each core (thus the system) is initially set to 1. Recall that our figure-of-merit (FOM) is the operating time before the reliability of the system (i.e., the product of the individual core reliabilities) reaches a given threshold; we illustrate here the lifetimes when the FOM thresholds are $1-10^{-6}, 1-10^{-7}$, and $1-10^{-8}$, respectively. Corresponding lifetimes are evaluated for the task schedule generated by the baseline algorithm and the improvement of our algorithm over the baseline is calculated.

A simple partitioned algorithm is also introduced to be compared with the algorithm proposed in this paper. Using the partitioned algorithm, each task is statically assigned to core and can only be executed on that core. On each core, earliest deadline first (EDF) [20] is used to schedule tasks.

4.2 Numerical Results

Recall the main steps we used to enhance reliability in this algorithm: (A1) Assign workload to small cores as much as possible; (A2) When some tasks need to be executed on big cores, tasks that execute faster and have a lower thermal impact on big cores are assigned to big cores first; (A3) Perform workload reassignment when a task finishes before its WCET, and (A4) Use dynamic voltage and frequency scaling so long as that can be done without causing tasks to be executed on more than one core at the same time. We carry out experimental studies to evaluate the impact of these approaches on lifetime enhancement.

In Figure 5(a)-(c), the lifetime improvement without DVFS (thus using A1, A2 and A3) at different system reliabilities is shown. As already stated, the algorithm in [4] is used as baseline. "Partitioned" stands for the simple static algorithm. "A1,2" indicates that our algorithm is using A1 and A2. "A3" indicates that the algorithm is using A3. The x-axis, utilization (Worst Case), is the utilization of the task sets on the big core when the execution times of tasks equal to WCET. As shown in Figure 5, when the utilization is low, all algorithms show gains over the baseline. This is because the baseline algorithm assigns all tasks to the big core while other algorithms will assign tasks to the small core. Using A3 gains as much as 27% improvement under low utilization; under these conditions, using task reassignment, there are more chances to reassign a task to the small core and ease the thermal impact on the big core. Thus, the lifetime of the system is improved greatly. At high utilization, using A1 and A2 performs better and gains more than 10% of improvement. This is because when the system is almost fully utilized, the order in which tasks are

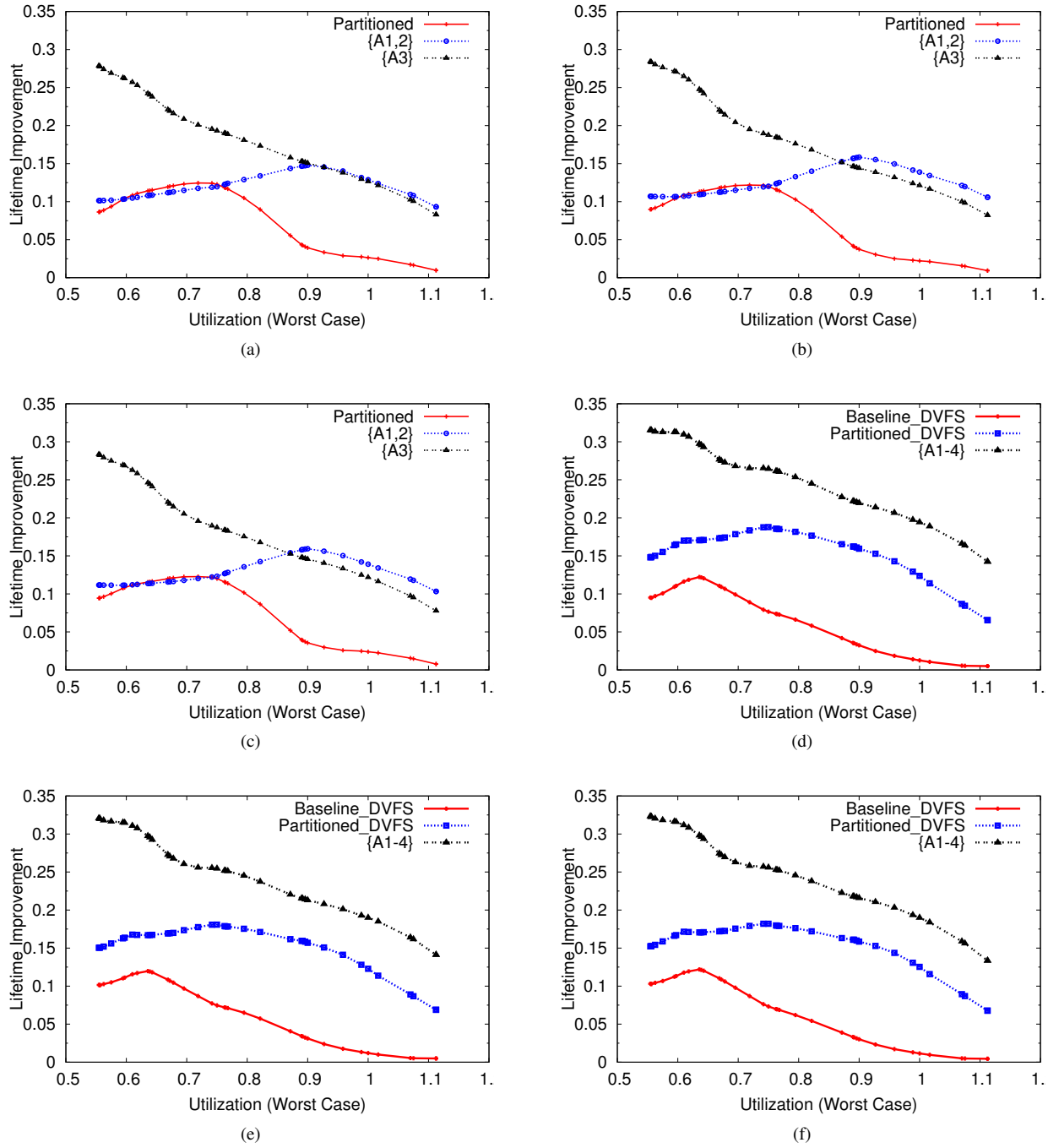

Fig. 5. Improvement for different reliability thresholds: (a) Reliability of 1-1e8 (b) Reliability of 1-1e7 (c) Reliability of 1-1e6 Improvement for different reliability thresholds with DVFS: (d)Reliabiilty of 1-1e8 (e) Reliability of 1-1e7 (f) Reliability of 1-1e6

assigned can more effectively control the thermal impact on cores; note that under these conditions, there will not be much slack for reassignment with dense workload.

Figures 5(d)-(f) show the improvement at different system reliabilities when voltage scaling is used. "Baseline_DVFS" stands for the baseline algorithm with DVFS. The baseline algorithm in [4] does not use DVFS . We add the DVFS capability by evenly assigning the slack generated by early completion to uncompleted tasks that only executed on the big core. "Partitioned_DVFS" uses the DVFS algorithm in [27]. "A1-4" uses all the 4 techniques and the algorithm we proposed in this paper. As is shown in Figures 5(d)-(f), using the techniques and algorithm proposed in this paper, the lifetime improvement is much higher than other algorithms. At high utilization, since the baseline algorithm tends to assign as much workload to the big core as possible, there is only a little slack available to apply DVFS and result in a minimal lifetime improvement. The proposed approach (using the A1-

4 techniques), on the other hand, assigns less workload to the big core, gets more slack and applies slack to tasks with higher thermal impact. These result in the the highest lifetime improvement among the three approaches. When the utilization is low, there is even more slack for DVFS on the big core using the proposed approach. Thus, the improvement over the other two is even higher.

Also note, as is shown in Figure 5, that the lifetime improvements achieved by the proposed approach at different reliabilities are almost identical.

5 Conclusion

Thermal issues have emerged as a key consideration in the management of cyber-physical systems. One increasingly popular architecture configuration uses high-end and low-end processing cores on a chip, sharing lower level cache and main memory. Such a mix allows the user a greater range of performance-to-power and performance-to-temperature trade-offs.

In this paper, we have introduced a thermal-aware task allocation and scheduling heuristic for use in periodic task workloads running on such heterogeneous platforms. Simulation experiments show that this heuristic provides substantial reliability benefits. In future work, we plan to extend this algorithm to cover sporadic tasks as well.

References

[1] (ARM 2013) "big.little technology: The future of mobile. [Online]. Available: https://www.arm.com/files/pdf/big_LITTLE_Technology_the_Futue_of_Mobile.pdf

[2] S. Baruah, "Feasibility analysis of preemptive real-time systems upon heterogeneous multiprocessor platforms," in *Real-Time Systems Symposium, 2004. Proceedings. 25th IEEE International*, Dec 2004, pp. 37–46.

[3] G. Raravi, B. Andersson, K. Bletsas, and V. Nlis, "Outstanding paper award: Task assignment algorithms for two-type heterogeneous multiprocessors," in *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, July 2012, pp. 34–43.

[4] H. S. Chwa, J. Seo, J. Lee, and I. Shin, "Optimal real-time scheduling on two-type heterogeneous multicore platforms," in *Real-Time Systems Symposium, 2015 IEEE*, Dec 2015, pp. 119–129.

[5] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas, "Single-isa heterogeneous multi-core architectures for multithreaded workload performance," in *Computer Architecture, 2004. Proceedings. 31st Annual International Symposium on*, June 2004, pp. 64–75.

[6] T. Li, D. Baumberger, D. A. Koufaty, and S. Hahn, "Efficient operating system scheduling for performance-asymmetric multi-core architectures," in *Supercomputing, 2007. SC '07. Proceedings of the 2007 ACM/IEEE Conference on*, Nov 2007, pp. 1–11.

[7] J. C. Mogul, J. Mudigonda, N. Binkert, P. Ranganathan, and V. Talwar, "Using asymmetric single-isa cmps to save energy on operating systems," *IEEE Micro*, vol. 28, no. 3, pp. 26–41, May 2008.

[8] D. Shelepov, J. C. Saez Alcaide, S. Jeffery, A. Fedorova, N. Perez, Z. F. Huang, S. Blagodurov, and V. Kumar, "Hass: A scheduler for heterogeneous multicore systems," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 2, pp. 66–75, Apr. 2009.

[9] K. V. Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer, "Scheduling heterogeneous multi-cores through performance impact estimation (pie)," in *Computer Architecture (ISCA), 2012 39th Annual International Symposium on*, June 2012, pp. 213–224.

[10] Y. Zhang, L. Duan, B. Li, L. Peng, and S. Sadagopan, "Energy efficient job scheduling in single-isa heterogeneous chip-multiprocessors," in *Quality Electronic Design (ISQED), 2014 15th International Symposium on*, March 2014, pp. 660–666.

[11] S. I. Kim, J.-K. Kim, H. U. Ha, T. H. Kim, and K. H. Choi, "Efficient task scheduling for hard real-time tasks in asymmetric multicore processors," in *Proceedings of the 12th International Conference on Algorithms and Architectures for Parallel Processing - Volume Part II*, ser. ICA3PP'12. Springer-Verlag, 2012, pp. 187–196.

[12] G. Levin, S. Funk, C. Sadowski, I. Pye, and S. Brandt, "Dp-fair: A simple model for understanding optimal multiprocessor scheduling," in *Real-Time Systems (ECRTS), 2010 22nd Euromicro Conference on*, July 2010, pp. 3–13.

[13] J. Srinivasan, S. Adve, P. Bose, and J. Rivers, "The impact of technology scaling on lifetime reliability," in *Dependable Systems and Networks, 2004 International Conference on*, June 2004, pp. 177–186.

[14] J. Srinivasan, S. V. Adve, P. Bose, S. V. A. P. Bose, and J. A. Rivers, "The case for lifetime reliability-aware microprocessors," in *In Proc. of the 31st International Symposium on Computer Architecture*, 2004, pp. 276–287.

[15] J. R. Black, "Mass transport of aluminum by momentum exchange with conducting electrons," in *Reliability Physics Symposium, 2005. Proceedings. 43rd Annual. 2005 IEEE International*, April 2005, pp. 1–6.

[16] Z. Lu, W. Huang, M. Stan, K. Skadron, and J. Lach, "Interconnect lifetime prediction with temporal and spatial temperature gradients for reliability-aware design and run 134 time management: Modeling and applications. very large scale integration (vlsi) systems," *IEEE Transactions on*, 2006.

[17] J. Srinivasan, S. Adve, P. Bose, and J. Rivers, "Lifetime reliability: toward an architectural solution," *Micro, IEEE*, vol. 25, no. 3, pp. 70–80, 2005.

[18] C. Zhuo, D. Sylvester, and D. Blaauw, "Process variation and temperature-aware reliability management," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, 2010, pp. 580–585.

[19] E. Wu, J. Su, W. Lai, E. Nowak, J. McKenna, A. Vayshenker, and D. Harmon, "Interplay of voltage and temperature acceleration of oxide breakdown for ultra-thin gate oxides," *Solid-State Electronics*, vol. 46, no. 11, pp. 1787 – 1798, 2002.

[20] J. Liu, *Real-Time Systems*. Prentice Hall, 2000.

[21] S. Xu, I. Koren, and C. Krishna, "Improving processor lifespan and energy consumption using dvfs based on ilp monitoring," in *Green Computing Conference and Sustainable Computing Conference (IGSC), 2015 Sixth International*, Dec 2015, pp. 1–6.

[22] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, Dec 2001, pp. 3–14.

[23] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[24] 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 *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, Dec 2009, pp. 469–480.

[25] Y. Han, I. Koren, and C. M. Krishna, "Tilts: A fast architectural-level transient thermal simulation method." *J. Low Power Electronics*, vol. 3, no. 1, pp. 13–21, 2007.

[26] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. Stan, "Hotspot: a compact thermal modeling methodology for early-stage vlsi design," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 14, no. 5, pp. 501–513, May 2006.

[27] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," *SIGOPS Oper. Syst. Rev.*, vol. 35, no. 5, pp. 89–102, Oct. 2001.