End-to-End Delay Minimization in Thermally Constrained Distributed Systems

Pratyush Kumar, Lothar Thiele
Computer Engineering and Networks Laboratory (TIK)
ETH Zürich, Switzerland
{*pratyush.kumar, lothar.thiele*}*@tik.ee.ethz.ch*

***Abstract*—With ever-increasing power densities, managing on-chip temperatures by optimizing mapping and scheduling of tasks is becoming increasingly necessary. We study the minimization of end-to-end delay for thermally constrained scheduling of an application that is specified as a task graph and is executing on parallel processors without speed scaling. We show that task graph scheduling on thermally constrained systems is monotonic, i.e., delaying the execution of a task longer than necessary cannot lead to the early completion of any other task. Using this monotonicity principle, we design the provably optimal schedule for a given mapping, called the JUST schedule. The JUST schedule can be easily implemented using temperature sensors. We then present different thermal-aware modifications to standard mapping heuristics and evaluate them on a large set of problem instances. The experimental results illustrate that with simple thermal-aware modifications, mappings with much smaller end-to-end delay can be identified.**

I. Introduction

With the inevitable, though slower, transistor scaling, the power consumption of today's electronic systems continues to rise. Higher power densities directly translate to higher on-chip temperatures, which can impact the long-term reliability and also the functional correctness of the system [1]. Furthermore, with transistors becoming increasingly leaky, higher temperatures in turn lead to higher power [2]. Improvements in hardware-based cooling using better packaging materials, more efficient active cooling techniques and novel solutions like liquid cooling, are not able to keep pace with the rise in temperatures [3]. As a result, software-based methods broadly classified as Dynamic Thermal Management (DTM) are becoming increasingly mainstream [4].

In modern processors there are two major sources of power consumption: switching activity in the capacitors leading to dynamic power consumption and power dissipation between voltage rails leading to leakage power consumption. Dynamic power consumption can be reduced by scaling the voltage and/or frequency of the system to match performance requirements: a technique referred to as Dynamic Voltage Scaling (DVS). Leakage power consumption can be reduced by putting the system into a low power state, whenever the system is not expected to perform any computation: a technique referred to as Dynamic Power Management (DPM) or execution throttling.

The objective of thermally constrained scheduling, is to use DVS and DPM techniques to optimize the system performance and/or to meet real-time deadlines while satisfying certain specified thermal constraints. Several works have formulated and studied such problems. In [5], the authors derived the maximum amount of workload that a processor can serve using on-line voltage scaling while satisfying thermal constraints. In [6], reactive speed control to schedule frame-based real-time tasks was proposed and schedulability tests under temperature constraints were derived. Proactive speed control to minimize response time of set of independent tasks under temperature constraints has been studied in [7]. In [8], completion time of tasks is minimized while each task is assigned a fixed speed.

There are two limitations with the above and similar works. Firstly, they only consider DVS for thermal-aware scheduling. While DVS is an attractive proposition in theory, there are practical requirements it assumes: availability of different system-wide voltage rails, and possibility of switching between these voltage rails at any given point of time. Further, as VLSI technology continues to scale, and as thermal issues become more important, the headroom for voltage scaling is unfortunately shrinking. Finally, peripheral devices such as I/O controllers, memory modules and network devices such as radio, intrinsically do not support speed scaling. For such devices, applying DPM techniques by putting the system in low-power modes at times of low activity is more effective.

The second limitation of these works is that they only consider independent tasks with individual performance constraints. In reality, applications comprise of several dependent tasks, modeled usually as task graphs, which need to satisfy a cumulative performance constraint such as the end-to-end delay. In such applications, one task can be slowed down or delayed to speed up or run earlier another task, with no net effect on the end-to-end delay. The scheduling freedom in such problems is thus potentially larger and this broadens the scope of thermally constrained scheduling.

Some existing works have studied the problem of thermal-aware scheduling of task graphs. In [9], authors study the problem of binding and scheduling tasks of a task graph

on multi-core systems with specified thermal-constraints. In [10], the authors propose heuristic approaches to distribute slack time across different tasks of a task graph, again for a multi-core architecture. These works consider heuristics based only on the steady-state behavior of the system. This simplifying assumption is not valid when considering distributed computing systems with non-negligible communication overhead, or for applications where execution times of tasks is much shorter than the thermal time-constants.

Interconnecting heterogeneous computing resources over communication networks is increasingly becoming a common way of computing: examples range from networked embedded systems performing co-ordinated tasks to high-performance grid computers. Representing applications as task graphs not only expresses the data dependencies of the tasks, but also exposes the inherent concurrency in the applications, that can be exploited in such parallel systems. The problem of thermally constrained scheduling of task graphs in such systems, is to minimize the end-to-end delay of the application while satisfying specified thermal constraints on each of the processors. We consider processors with no speed scaling feature and hence the only way to manage temperatures is to *force* the processors in to low-power states. In addition to such forced operation, the processor would remain in the low-power state when it does not have any tasks to run, either due to low concurrency in the application or because input data is yet to arrive. Thus, both the parallel and distributed natures of such applications and architectures have to be carefully considered in thermally constrained scheduling.

Our contributions in this work are on three fronts. Firstly, we show that task graph scheduling on thermally constraint systems is monotonic, i.e., delaying the execution of a task more than is necessary to meet the thermal constraint, cannot lead to early completion of any other task. Using this desirable property, for a *given* mapping of tasks to the processors, we derive the provably optimal `JUST` scheduling strategy, that minimizes the end-to-end delay of the application while satisfying the thermal constraints on each of the processors. We present how the optimal scheduling can be easily implemented using temperature sensors on the processors.

The second contribution is to study heuristics to optimize the mapping of the tasks. The nature of the optimal `JUST` schedule motivates us to encode the thermal constraints as part of the task graph and then apply the well-studied mapping heuristics, to decide the mapping of the tasks. We compare the end-to-end delays of `JUST` schedules for mappings generated with the original task graph and the modified thermal-aware task graph with the HEFT heuristic proposed in [11]. We present results on realistic as well as a large number of randomly generated task graphs. Our results illustrate that thermal-aware mapping and scheduling of tasks can substantially reduce the end-to-end delay of applications, in comparison with a thermal-unaware approach, even for the optimal scheduling policy. We also characterize the nature of the task graphs and system architectures where thermal-awareness is more critical.

Finally, as the third contribution, we design more sophisticated thermal-aware mapping strategies by explicitly modeling the thermal behavior of the system in the mapping phase. and compare them with the simple mapping strategy based on the modification of the task graph. From experimental results we conclude that the sophisticated methods that we propose provide only a small improvement at a substantial run-time overhead.

The rest of the paper is organized thus. In Section 2, we formally present the model of the system and the problem definition. As the main contribution of this work, we present the optimal scheduling strategy, for a given mapping, in Section 3. In Section 4, we present methods to optimize the mapping of tasks. We present results of the experiments in Section 5 and conclude in Section 6.

II. Models and problem definition

A. Application and system models

An application is represented by its task graph, also called a directed acyclic graph (DAG), $G = (V, E)$, where V is the set of tasks and E is the set of directed edges between the tasks. Each edge $(u, v) \in E$ represents the precedence constraint that before task v begins to execute, required input from task u must be available.

The system that we consider consists of a set, R, of possibly heterogeneous processors. The frequency of operation of processor $p \in R$ is denoted as f_p. We consider a fully connected interconnect where communication on links can happen parallel to the computation on processors.

The task v has a worst-case execution time (WCET) on processor p given by $c_{v,p}$. The communication delay between tasks u and v, when mapped on processors p and q, respectively, is given by $q_{(u,v),p,q}$.

Thus, the application and the system are specified by the tuple $\mathbf{A} = (G, R, c, q)$.

B. Thermal model

We consider a distributed network of heterogeneous processors. Each processor can have a different thermal model based on the power consumption of the processor, the cooling solution employed and the ambient temperature around that processor. We do not consider any heat exchange between the processors, i.e. the processors are thermally insulated from each other.

In all the notation used below, we consider a specific processor $p \in R$ and denote this by using p as a subscript in all parameters. The physical process of heat transfer is a complex one. However, for architectural level thermal management, the Fourier heat transfer model is typically used [5], [6], [8]. Under this model, the evolution over time

of temperature of a processor p is given by the following differential equation:

$$C_p \frac{dT_p}{dt} = P_p - G_p(T_p - T_p^{amb}), \quad (1)$$

where T_p^{amb} is the ambient temperature around the processor, G_p represents the thermal conductance of the cooling solution, P_p represents the power being consumed in the system, C_p represents the thermal heat capacity of the processor and $T_p(t)$ is the temperature of the processor. The leakage current on a VLSI system is itself a function of temperature: at higher temperature the leakage current is higher. In [12], a linear model is shown to work well in practice. In this model, the power can be expressed as the following function of processor-specific parameters α_p, β_p

$$P_p = \alpha_p T_p + \beta_p. \quad (2)$$

Later, in Section 3.2, we consider a more general power model where tasks executing on the same processor can consume different amounts of power.

Including (2), the differential equation in (1) becomes

$$C_p \frac{dT_p}{dt} = -(G_p - \alpha_p)T_p + (\beta_p + G_p T_p^{amb}). \quad (3)$$

To simplify notation we shorten the above equation as

$$\frac{dT_p}{dt} = -a_p T_p + b_p,$$
$$\text{with } a_p = \frac{G_p - \alpha_p}{C_p}, b_p = \frac{\beta_p + G_p T_p^{amb}}{C_p}. \quad (4)$$

The closed form solution to this differential equation is

$$T_p(t) = T_p^{\infty} + (T_p(t_0) - T_p^{\infty}) \cdot e^{-a_p(t-t_0)}, \quad (5)$$

where $T_p^{\infty} = b_p/a_p$ is the steady-state temperature of the processor. From (5), the thermal behavior of the processor is characterized by the time constant a_p and the steady-state temperature T_p^{∞}.

When a processor is executing tasks it is said to be in the *active* mode, otherwise it is said to be in the *idle* mode. Intuitively, the power consumption of the processor is different in either mode. We represent the corresponding α and β terms, as expressed in (2), as α_p^{act}, β_p^{act} for the active mode and α_p^{idl} and β_p^{idl} for the idle mode. Similarly, we represent the derived time constant a_p for the active and idle modes as a_p^{act} and a_p^{idl}, respectively. While representing T_p^{∞} for the two modes, to aid readability, we drop the ∞ symbol and denote them simply as T_p^{act} and T_p^{idl}. For a valid thermal mode, we should have $T^{act} \geq T^{idl}$. Using these derived quantities, the thermal behavior of the system is specified by the tuple $\mathbf{T} = (a^{act}, T^{act}, a^{idl}, T^{idl})$.

C. Mapping and Scheduling Model

We consider a partitioned static-ordered non-preemptible scheduling of a given task graph. For partitioned scheduling there exists a binding function π where π_v denotes the processor on which task v is always executed on. The schedule is static-ordered: the tasks bound to a processor are executed according to a fixed ordering σ, where $\sigma_{p,i}$ denotes the ith task to execute on processor p. Partitioned static-ordered scheduling is a standard approach to schedule task graphs [13]. We denote by $|\sigma_p|$ the total number of tasks mapped on processor p. The scheduling is non-preemptible in nature, i.e., once a task begins to execute, it cannot be preempted. Preemption by other tasks is clearly not allowed under static-ordered scheduling. Preemption by putting the processor in the idle mode is also disallowed as such preemption would require an additional feature: the processor must be able to preserve its context in the idle mode and be able to resume its execution after the transition from idle to active mode. We do not make this assumption, and hence the processor can be in idle mode only in-between the execution of tasks.

To control the temperature of the processors we use dynamic power management (DPM), whereby, we allow insertion of idle times before execution of each task. The value $\gamma_{p,i}$ denotes the minimum amount of time processor p remains in idle mode *before* the execution of the task $\sigma_{p,i}$.

The mapping and scheduling of the task graph is thus fully specified by the tuple $\mathbf{S} = (\pi, \sigma, \gamma)$. We also refer to the binding and ordering of the tasks as given by the tuple $\mathbf{M} = (\pi, \sigma)$ as the mapping of tasks.

D. Problem Definition

For each processor $p \in R$, a thermal constraint specifies the maximum allowed temperature of that processor, denoted as T_p^{max}. A thermal constraint is valid if $T_p^{max} \leq T_p^{act}$, i.e., the maximum allowed temperature should not be above the maximum temperature that can be reached by the processor. We assume that tasks can begin execution in each of the processors at time $t = 0$. The initial temperature, at $t = 0$, of each processor $p \in R$ is given as T_p^{start}. The end-to-end delay, d, is defined as the largest finish times among all tasks of the application.

The thermally constrained task graph scheduling problem can then be defined as follows: Given an application and system specification $\mathbf{A}$ and a thermal model $\mathbf{T}$. Design mapping and scheduling $\mathbf{S}$ such that

- for all processors $p \in R$ the temperature on processor p at all times does not exceed T_p^{max}, and
- the end-to-end delay, d, is minimized.

III. Optimal end-to-end delay for a given mapping of tasks

We solve the problem of identifying the optimal mapping and scheduling in two steps. Firstly, in this section, we

assume a given mapping, $\mathbf{M}$, with binding π and static-ordering σ. For this mapping, we compute the idle times γ to optimally minimize the end-to-end delay, while satisfying the thermal constraints. We then, in the next section, consider the problem of optimizing the mapping of the tasks.

We present some notation to characterize a schedule. Let $t^s_{p,i}$ and $T^s_{p,i}$ denote the time and temperature of processor p, respectively, when the task $\sigma_{p,i}$ starts to execute. Similarly, let $t^f_{p,i}$ and $T^f_{p,i}$ denote the time and temperature of processor p, respectively, when task $\sigma_{p,i}$ finishes. When discussing multiple schedules, we label the above variables with a superscript denoting the schedule. For each processor $p \in R$, tasks can start at time $t = 0$. To enable ease of notation later, we represent this as the finish time of an imaginary 0th task, $t^f_{p,0} := 0$, for all $p \in R$. Similarly, we set the initial temperature of each processor $T^f_{p,0} := T^{start}_p$.

The insertion of idle times, γ, before the execution of tasks gives the following relations on the start times

$$t^s_{p,i} \geq t^f_{p,i-1} + \gamma_{p,i}, \qquad \forall\ p \in R,\ 1 \leq i \leq |\sigma_p|. \tag{6}$$

The γ terms only define a lower bound on the amount of time the processor is in the idle mode between tasks; a processor may continue to remain in the idle mode for longer, if the input data required by the next task is not yet available.

For a given mapping of tasks, the end-to-end delay, d is given in terms of the finish times as

$$d = \max_{p \in R, i = |\sigma_p|} t^f_{p,i}. \tag{7}$$

Given (6), the value of d is a function of the inserted idle times γ. The *optimal end-to-end delay*, denoted as d^{opt}, is defined as the smallest end-to-end delay over all choices of idle times γ. The goal of this section is to compute this optimal assignment of idle times.

A. Worst-case Execution Pattern

The optimal end-to-end delay, d^{opt}, is a function of the execution times of tasks. In the application specification, tasks are characterized only by their WCETs, leading to an uncertainty in the execution pattern. In order to provide real-time guarantees which hold irrespective of the actual execution times of tasks, we must ascertain the combination of execution times of tasks that leads to the largest d^{opt}.

For systems with no thermal considerations, clearly, the longest end-to-end delay is obtained when the all task executions take as long as possible. But does this hold for thermally-constrained systems as well? The following theorem answers this question in the affirmative.

Theorem 1: For a given mapping of tasks, d^{opt} is maximum when all tasks run for their WCETs.

Proof: For the given mapping of tasks, let S be the optimal schedule with respect to end-to-end delay, satisfying thermal constraints, when all tasks exactly execute for their WCETs. For the same mapping of tasks, let S^* be a schedule, where tasks run for specified times, no more than their WCETs. From the monotonicity of dataflow, we know that an earlier execution of a task cannot lead to a later finish of any other task. This implies that in schedule S^*, all tasks can be started at the same time as they are started in schedule S. Let S^* indeed be such that $(t^s_{p,i})^{S^*} = (t^s_{p,i})^S$. Since all tasks run for their WCETs in S, we have $(t^f_{p,i})^{S^*} \leq (t^f_{p,i})^S$, and thus $d^{S^*} \leq d^S$.

We have to now establish that S^* satisfies thermal constraints. For each task of the task graph, in schedule S^* the corresponding processor satisfies two properties: (a) it remains in the idle mode, prior to the start of the task, for *at least* as long as in S, and (b) it runs in the active mode for *at most* as long in S. Using $T^{act} \geq T^{idl}$, and by repeatedly applying (5) with the above two properties, we can show that $(T^f_{p,i})^S \geq (T^f_{p,i})^{S^*}$, for all $p \in R$ and $1 \leq i \leq |\sigma_p|$. Hence, like in S, thermal constraints are not violated in S^*, and S^* is a valid schedule. Thus, we have

$$\begin{aligned} d^S &= d^{opt}(\text{tasks run for WCETs}) \\ &\geq d^{S^*} \\ &\geq d^{opt}(\text{tasks run for arbitrary amounts of time} \\ &\qquad \text{upper-bounded by WCETs}). \end{aligned} \tag{8}$$

■

From this theorem, we have the following conclusion: irrespective of the actual execution times of tasks, under the optimal scheduling policy, we can always guarantee a makespan constraint of d^{opt}(tasks run for WCETs). In the remainder of this paper, we will assume all tasks run for their WCETs, so as to be able to compute the makespan guarantee and use it to compare different mappings.

B. JUST Schedule

Define $T^{target}_{p,i}$ as the value of $T^s_{p,i}$ for which, given that task $\sigma_{p,i}$ runs for its WCET, the corresponding value of $T^f_{p,i}$ is T^{max}_p. In other words, $T^{target}_{p,i}$ denotes the maximum allowed temperature of processor p at the start of execution of task $\sigma_{p,i}$, without violating the temperature constraint. To simplify notation, let c^* denote the WCET of the considered task $\sigma_{p,i}$ on processor p, i.e. $c^* = c_{\sigma_{p,i},p}$. By applying (5), we can statically derive the following relation

$$T^{target}_{p,i} = T^{act}_p - \left(T^{act}_p - T^{max}_p\right) \cdot e^{-a^{act}_p c^*}. \tag{9}$$

Again applying (5), given the value of $T^f_{p,i-1}$, we can compute a lower bound on the idle time $\gamma_{p,i}$ such that $T^s_{p,i} \leq T^{target}_{p.i}$ as

$$\begin{aligned} \gamma^{min}_{p,i}\left(T^f_{p,i-1}\right) &= 0, \quad \text{if } T^f_{p,i-1} \leq T^{target}_{p,i} \\ &= \frac{1}{a^{idl}_p} \log\left(\frac{T^f_{p,i-1} - T^{idl}_p}{T^{target}_{p,i} - T^{idl}_p}\right), \text{ else.} \end{aligned} \tag{10}$$

The interpretation of γ^{min} is as follows: if the idle time before the execution of task $\sigma_{p,i}$ is less than $\gamma^{min}_{p,i}$, then the thermal constraint of processor p would be violated by the end of the execution of task $\sigma_{p,i}$. Thus, any schedule that satisfies the specified thermal constraints must keep the processor in the idle mode for at least as long as given by the above γ^{min} values. We now define the schedule which we shall, later in this section, prove to be the optimal schedule.

Definition 1 (JUST(π, σ)): For a given binding π and for a given ordering σ, the JUst Sufficient Throttling JUST(π, σ) schedule is one where all idle times, γ, are set to γ^{min}.

In a JUST schedule the processors are put in the idle mode for just sufficient amount of time to avoid violating thermal constraints, and hence the name. Note that the JUST schedule is not statically defined as the values of γ^{min} depend on the temperatures T^f, which are revealed only at run-time. However, the computation of the idle times is causal in nature: once a task completes, the length of the idle time to be inserted before the next task begins can be computed. From the definition of the JUST schedule, we have the following result.

Lemma 2: For a given mapping of tasks, in any schedule that does not violate thermal constraints, if executing a single task earlier does not violate thermal constraints at the end of its execution, then it will not violate thermal constraints during the entire schedule.

Proof: For given mapping, let S be a given valid schedule. For the same mapping, let S^* be another given schedule that varies from S only with an earlier start time for some task σ_{p^*,i^*}, i.e. $(t^s_{p^*,i^*})^{S^*} < (t^s_{p^*,i^*})^S$ and for all other tasks $\sigma_{p,i}$, $(p,i) \neq (p^*,i^*)$, $(t^s_{p,i})^{S^*} = (t^s_{p,i})^S$. The given value of $(t^s_{p^*,i^*})^{S^*}$ is such that task σ_{p^*,i^*} satisfies all precedence constraints and the thermal constraint on processor p^* at the end of its execution.

Again from the monotonicity principle, we know that S^* would satisfy precedence constraints on all tasks. We need to show that S^* also satisfies thermal constraints. The start and finish of all tasks on all processors $p \neq p^*$, is the same as in S, and hence the temperature constraints are satisfied on these processors. Similarly, on processor p^*, the thermal constraints are satisfied until the start of task σ_{p^*,i^*}. It is also given that the thermal constraints are satisfied at the end of the execution of task σ_{p^*,i^*}. We thus need to show that S^* satisfies thermal constraints on processor p^* from the beginning of the execution of task σ_{p^*,i^*+1}. During this time, the tasks start and run for exactly the same time in both S and S^*. Notice that (5) is monotonic with respect to the starting temperature of the processor. Therefore, the composition of (5) for the different tasks and idle times from the beginning of execution of task σ_{p^*,i^*+1} is also monotonic in the starting temperature, $T^s_{p^*,i^*+1}$. Thus, it is sufficient to show that $(T^s_{p^*,i^*+1})^{S^*} \leq (T^s_{p^*,i^*+1})^S$.

To show the above, we first derive an analytical result. Consider a processor that is at temperature T_1. It is put in the idle mode for x units of time, then in the active mode for y units of time and finally in the idle mode again for z units of time. Let the temperature of the processor at the end of the three phases be T_2. Then using (5) repeatedly we derive the following relation:

$$T_2 = T^{idl} + (T_1 - T^{idl})e^{-a^{idl}(x+z)}e^{-a^{act}y} + (T^{act} - T^{idl})(1 - e^{-a^{act}y})e^{-a^{idl}z}. \quad (11)$$

We have dropped the subscript denoting the processor. If we changed x to $x - \delta$ and z to $z + \delta$, for some $\delta > 0$, the value of T_2 decreases, for a given value of T_1. This can be interpreted as follows: For a given total idle time, $(x + z)$, the final temperature, T_2, can be reduced by *shifting* some of the earlier idle time (x) to a the later idle time (z).

We can apply (11) separately to the two schedules S and S^*, on the processor p^*, where $T_1 = T^f_{p^*,i^*-1}$, $T_2 = T^s_{p^*,i^*+1}$, $x = t^s_{p^*,i^*} - t^f_{p^*,i^*-1}$, $y = c_{\sigma_{p^*,i^*},p^*}$, and $z = t^s_{p^*,i^*+1} - t^f_{p^*,i^*}$. In S^*, the value of x is reduced than in S as σ_{p^*,i^*} is started earlier than in S, whereas the value of z is increased by the same amount. Hence, using the above result, we have $(T^s_{p^*,i^*+1})^{S^*} \leq (T^s_{p^*,i^*+1})^S$. ■

The above lemma implies that modifying any valid schedule by executing a task earlier entails only local checks: whether the input data is available at the earlier starting time and whether the thermal constraints are satisfied after the execution of the task. If these local checks hold true, the entire modified schedule is a valid one. We use this result to derive the central theorem of this work.

Theorem 3: For a given binding π and ordering σ of tasks on processors, the JUST(π, σ) schedule optimally minimizes the end-to-end delay while satisfying the thermal constraints.

Proof: For binding π and ordering σ, let J denote the JUST(π, σ) schedule. For the same mapping, let S denote any other schedule, that satisfies the thermal constraints. We are to show that $d^J \leq d^S$.

Let μ be a sorting of the tasks of the task graph in non-decreasing order of their start times in the JUST schedule, $(t^s)^J$, with ties broken randomly. The ith task, for $1 \leq i \leq |V|$, in the order μ is denoted as μ_i.

Let i^* be the smallest number such that task μ_{i^*} has different start times in schedules J and S. The start time of task μ_{i^*} in J cannot be later than in S as it would contradict the definition of the JUST schedule. Hence, the start time of task μ_{i^*} is greater in S than in J. We modify schedule S such that task μ_{i^*} starts at the same time as in J, while the starting times of other tasks are kept the same. Lemma 2 shows that this modification leads to a valid schedule.

We perform the above process iteratively, modifying the start time of one task in S in each iteration. The number of tasks for which the start times differ in S and J reduces by 1 in each iteration. Hence, after a finite number of iterations, S would be transformed to J. Since in each iteration we are executing tasks in S earlier, the end-to-end delay cannot increase. Thus, any given schedule can be transformed to the JUST schedule iteratively without increasing the end-to-end delay in any iteration. ■

As discussed earlier, dataflow graphs are monotonic: executing any task later than the time of arrival of all its inputs, cannot lead to an earlier finish time of any task. With the above theorem, this monotonicity can be extended for thermally constrained dataflow scheduling as follows.

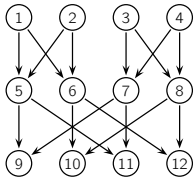
Definition 2: (MONOTONICITY IN THERMALLY CONSTRAINED DATAFLOW SCHEDULING) Executing any task later than the time of arrival of all its inputs *and* the time of the processor cooling to corresponding T^{target} as given in (9), cannot lead to an earlier finish time of any task.

This motivates modeling the idling of the processor by dummy tasks in the task graph: dummy task $\rho_{p,i}$ is inserted between $\sigma_{p,i-1}$ and $\sigma_{p,i}$. The task $\rho_{p,i}$ is said to complete execution (or fire in dataflow vocabulary) when the temperature of the processor falls below $T_{p,i}^{target}$. These dummy tasks can be easily implemented in a real system by using temperature sensors on each of the processors. Once the temperature of the system falls below the T^{target}, as given in (9), the temperature sensor can provide an enable signal to start the execution of the next task; in some sense providing a pre-programmed *temperature interrupt* to enable the next task. The JUST schedule then is simply the as-soon-as-possible (asap) scheduling of this modified task graph. Indeed, the desirable monotonicity principle allows for this simple implementation.

We have modeled our system such that the power consumption of a processor, as given in (2), is independent of the running task. In practice, different tasks use different amounts of power. The proposed JUST scheduling can model such task-dependent power consumption. To this end, we can modify the computation of T^{target} in (9) by obtaining the parameters T^{act} and a^{act} specific to the next task that is to be executed. The thermal sensor, as before, can wait for the temperature to fall below this T^{target} and then enable the next task. This also means that the monotonicity property of Definition 2 holds for systems with task-dependent power, if T^{target} is appropriately computed.

IV. OPTIMIZING MAPPING OF TASKS

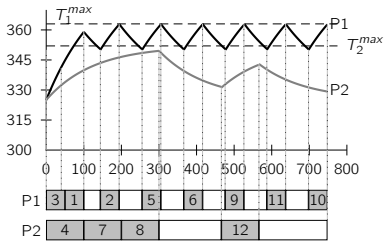
In the previous section we showed how to construct the optimal schedule with respect to end-to-end delay given the mapping of tasks. We now turn our attention to optimizing the mapping of tasks, with respect to the end-to-end delay of the corresponding JUST schedules.



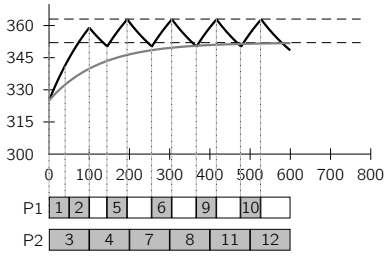
(a) Task graph

Parameter	Value
f_1	100 MHz
f_2	50 MHz
$c_{p,*}$	$5000/f_p ms$
$d_{(*,*),*,*}$	$50ms$
T_*^{start}	327K
α_*^{idl}	0.1 W/K
β_*^{idl}	-11 W
α_p^{act}	$0.1\times(f_p/100)^2$ W/K
β_p^{act}	$-25\times(f_p/100)^2$ W
T_*^{amb}	300 K
T_1^{max}	363 K
T_2^{max}	352 K
G_*	0.3 W/K
C_*	0.03 JK

(b) Problem specification



(c) Temperature (in K) vs Time (in ms) for mapping $\mathbf{M_1}$



(d) Temperature (in K) vs Time (in ms) for mapping $\mathbf{M_2}$

Figure 1. FFT example

A. Motivating examples

We start by illustrating, with two practical examples of task graphs, the key considerations for mapping of tasks in thermally constrained systems. In either example, we consider two mappings: a) mapping $\mathbf{M_1}$ generated by the HEFT algorithm [11] on the original task graph G, and b) mapping $\mathbf{M_2}$ that is generated by the same HEFT algorithm on a modified version of the task graph. Details of how these mappings are generated are presented later in the paper.

FFT: The first example is an FFT application characterized by high concurrency as shown in Figure 1(a). It is executing on a system of two *different* processors, with frequency in the ratio 2:1, and with a small communication delay between them. The system parameters are shown in Figure 1(b).

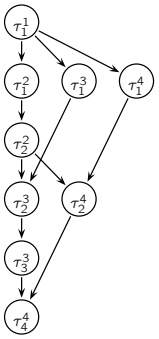
The mapping $\mathbf{M_1}$ maps two-thirds of the tasks on to the faster processor 1. For a schedule with no thermal constraint, $\mathbf{M_1}$ leads to an optimal schedule with full utilization of the both processors for the duration of the schedule. On the other hand, $\mathbf{M_2}$, in spite of the difference in processor frequency, maps equal number of tasks to either processor.

The JUST schedules corresponding to both mappings are shown in Figures 1(c) and 1(d). The JUST schedule corresponding to $\mathbf{M_2}$ has a much smaller end-to-end delay than that of $\mathbf{M_1}$. As is clear from the figure, this happens because the tasks mapped on to processor 1 require large idle times before they can begin, due to the higher power consumption of the fast processor. On the other hand, even for a smaller temperature bound, tasks running on the slower processor 2 do not need any idling time.

This example illustrates how a faster processor is not necessarily a good choice in a thermally constrained schedule. The enforced idle time changes the *effective* execution time of task on different processors to different extents.

LU Decomposition: The second example is an LU decomposition application characterized by low concurrency as shown in Figure 2(a). Its executing on a system of two *identical* processors with a large communication delay between them with system parameters as shown in Figure 2(b). The individual processors considered in this example have the same parameters as the faster processor in the last example.

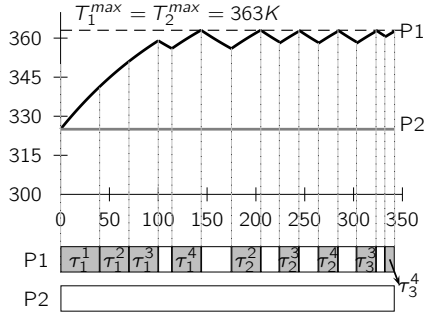
In $\mathbf{M_1}$ all tasks are mapped on to processor 1, as the communication costs are found to be too large for mapping tasks on processor 2. In $\mathbf{M_2}$, despite the large communication costs, one of the tasks, τ_1^4, is mapped to processor 2. As shown in Figures 2(c) and 2(d), JUST schedule corresponding to $\mathbf{M_2}$ has a smaller end-to-end delay than that of $\mathbf{M_1}$. This difference arises because communication along channels is allowed to happen parallel to a processor being in the idle mode. Thereby, the *effective* cost of communication can decrease. In the example, the enforced idling of processor 1, was sufficient to amortize the communication delay incurred in processing task τ_1^4 on the far-away processor 2. More than an analysis challenge, this example illustrates the potential to exploit this parallelism: allowing processors to idle while input data is being transferred.



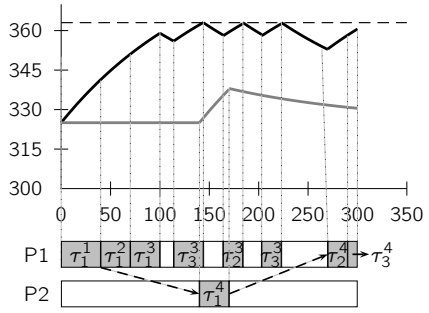
(a) Task graph

Parameter	Value
c_{*,τ_y^x}	$(5-y)\cdot 10\ if\ x=y$
	$(4-y)\cdot 10\ if\ x\neq y$
$d_{(*,*),*,*}$	$100ms$
T_*^{start}	327 K
α_*^*	0.1 W/K
β_*^{idl}	-11 W
β_*^{act}	-25 W
T_*^{amb}	300 K
T_*^{max}	363 K
G_*	0.3 W/K
C_*	0.03 JK

(b) Problem specification



(c) Temperature (in K) vs Time (in ms) for mapping $\mathbf{M_1}$



(d) Temperature (in K) vs Time (in ms) for mapping $\mathbf{M_2}$

Figure 2. LU decomposition example

B. Modification of execution and communication times

The discussed examples show that thermal constraints translate to changes in both the effective execution times and communication latencies. Motivated by the monotonicity property of thermally constrained scheduling (Theorem 3), we would like to model the effective execution times and communication latencies as part of the application task graph. To this end, we make two sets of modifications. Firstly, we change the WCET of tasks to model the idle time before the execution of that task. As shown in (10), the idle time inserted in the JUST schedule before a task, depends on the temperature of the processor at the end of the execution of the previous task. This temperature can be conservatively upper-bounded by assuming it to be the maximum allowed temperature on that processor, T_p^{max}. Crucially, this conservative bound is a valid bound independent of the mapping or the state of other processors. For a processor $p \in R$ and for a task $v \in V$, using (9) and (10), we can compute this conservative bound as

$$\gamma_{p,v}^{max} = \frac{1}{a_p^{idl}} \log\left(\frac{T_p^{max} - T_p^{idl}}{(T_p^{act} - T_p^{idl}) - (T_p^{act} - T_p^{max})e^{-a_p^{act} c_{p,v}}}\right). \tag{12}$$

We can then modify the execution times of the task v on each processor p to include this conservative bound as

$$c^T_{p,v} := c_{p,v} + \gamma^{max}_{p,v}. \tag{13}$$

The second change relates to the communication delays. As indicated in the LU decomposition example, communication can happen in parallel to the idling of the processors. In light of the modification of the WCETs of tasks according to (13), we reduce the communication latencies as follows:

$$d^T_{(u,v),p,q} := \max(0, d_{(u,v),p,q} - \gamma^{max}_{q,v}). \tag{14}$$

In the above equation, we reduce the communication latency along each channel by up to the inserted idle time of the destination task, but subject to no delay becoming negative.

While the above modifications to the timing properties of the task graph conservatively model thermally constrained scheduling, they are only approximate. On the other hand, the modifications allow us to incorporate the thermal constraints in the convenient task graph representation of the application, whereby we can apply well-studied task-graph-based mapping techniques.

C. Mapping using HEFT algorithm

Minimizing the end-to-end delay of a task graph on a set of parallel processors has been an active area of research. Apart from two very simple cases, all such mapping problems are known to be NP-hard [13]. Heuristics are, thus, the common approach in such problems. For the problem of task graph scheduling on a given set of heterogeneous processors, the mapping generated by the Heterogeneous-Earliest-Finish-Time (HEFT) algorithm presented in [11] has been shown to perform well and with small computation requirements.

For completeness, we present a brief overview of the HEFT algorithm. Like other similar algorithms, there are two phases in the HEFT algorithm: a) a ranking phase wherein the tasks of a task graph are ordered based on a ranking function, and b) a mapping phase, wherein tasks, in the derived ranking order, are assigned to be executed at specific times on chosen processors. The upward rank, as used in the HEFT algorithm, is defined recursively for every task $v \in V$ as follows

$$\begin{aligned} &rank_u(v) = \overline{c_v} + \max_{w \in succ(v)} (\overline{q_{v,w}} + rank_u(w)), \text{where} \\ &\overline{c_v} = \frac{1}{|R|} \sum_{p \in R} c_{v,p}, \text{ and } \overline{q_{v,w}} = \frac{1}{|R|^2} \sum_{p,q \in R} q_{(v,w),p,q}.. \end{aligned} \tag{15}$$

Here $succ(v)$ denotes the set of tasks which are direct successors of the task v in the task graph G.

In the mapping phase, tasks ordered by non-increasing rank are mapped sequentially. At all points of time, a partial schedule is maintained of already mapped tasks. A task is mapped to that processor which leads to the smallest *earliest finish time* (EFT) based on the current partial schedule. Once the processor with the smallest EFT is chosen, the task is mapped on to it, a specific time window for its execution is assigned, and the partial schedule is modified to include this window. This process is continued till all tasks are mapped. A more detailed description is available in [11].

We generate two sets of mappings with the HEFT algorithm. Firstly, we apply the HEFT algorithm directly to the task graph, G, with its original properties. While the output of the HEFT algorithm is an entire schedule, we only retain the mapping of the tasks, i.e. the binding and the ordering of tasks. The mapping thus obtained is denoted as $\mathbf{M_1}$. Then, we modify the timing properties (execution times of tasks and communication delays along channels) of the task graph as discussed in (13) and (14). We then again apply the HEFT algorithm to this modified thermal-aware task graph. The mapping so obtained is denoted as $\mathbf{M_2}$. We then construct the optimal `JUST` schedules for the obtained mappings considering the thermal constraints on each processor. We compare the mappings, $\mathbf{M_1}$ and $\mathbf{M_2}$, by comparing the makespan of the respective `JUST` schedules.

D. Thermal-Aware modifications to HEFT algorithm

$\mathbf{M_1}$ provides a good baseline to compare $\mathbf{M_2}$ to ascertain the need for thermal-aware mapping optimization. However, it does not suffice to evaluate $\mathbf{M_2}$ with respect to the potential of thermal-aware mapping optimization. It is necessary to compare $\mathbf{M_2}$ against more sophisticated thermal-aware mapping strategies. To the best of our knowledge, there is no existing work in this direction. Thus, we propose techniques to generate two other mappings denoted as $\mathbf{M_3}$ and $\mathbf{M_4}$. To generate these mappings we explicitly and more accurately model the thermal behavior of the processors during the mapping optimization at the cost of added complexity.

In both $\mathbf{M_3}$ and $\mathbf{M_4}$, the ranking phase is the same as in the HEFT algorithm. However, the mapping phases differ. While modifying the WCETs of the tasks in (13), we assumed that the temperature at the end of the execution of each task is the highest allowed temperature on that processor. This can be inaccurate for systems with large communication latencies or for task graphs with low concurrency. We change this in $\mathbf{M_3}$ and $\mathbf{M_4}$ by explicitly computing the temperatures of the processors for the partial schedules during the mapping phase of HEFT algorithm using (5). We then derive the earliest finish time (EFT) of tasks on different processors by computing the idle times accurately given the temperature of the processors using (10). Thus, at each step of the mapping, for the current partial schedules temperature behavior of each processor is computed, and then the exact EFTs of the tasks are computed.

The second change is with regards to what is called *insertion* scheduling. In the HEFT algorithm, mapping of tasks is performed in the order of their ranks, and thus later tasks must not delay already scheduled, perhaps, more

critical tasks. For systems with no thermal constraints, this amounts to checking that the scheduling windows of different tasks are non-overlapping. However, for thermally constrained systems, even for non-overlapping scheduling windows, earlier scheduled tasks can be delayed. This would happen if scheduling a new task v_1 before an already scheduled task v_2 raises the temperature sufficiently to require more idle time before execution of v_2. Thus, checking that already scheduled tasks are not delayed is computationally more demanding for thermally constrained systems. The two techniques $\mathbf{M_3}$ and $\mathbf{M_4}$ differ from each other in their approach to deal with this check. In $\mathbf{M_3}$ we allow tasks to be scheduled only at the end of existing partial schedules, thereby trivially not delaying any of the earlier scheduled tasks, but perhaps at the loss of optimality. In $\mathbf{M_4}$ on the other hand, we allow tasks to be scheduled within an existing partial schedule, but subject to the fact that no earlier scheduled task is delayed. To accomplish this, we attempt every possible ordering of a new task into a partial order and re-compute the thermal behavior.

Clearly, the modeling accuracy and the computation requirement of the mapping phase increases from $\mathbf{M_2}$ to $\mathbf{M_3}$ to $\mathbf{M_4}$. We experimentally evaluate how this additional complexity translates into improvement in the end-to-end delays of the JUST schedules.

V. Experimental results

In Section 4.1, we presented two examples for FFT and LU decomposition applications, on different architectures. Indeed, the mappings $\mathbf{M_1}$ and $\mathbf{M_2}$ as used in those examples correspond to the respective mappings described in the previous section. As evidenced in the figures, the thermal-aware mapping $\mathbf{M_2}$ clearly leads to JUST schedules of smaller end-to-end delay.

Heuristics such as HEFT algorithms, are often compared to each other using several randomly generated task graphs. We employ a similar approach to compare the four mapping techniques. There are three aspects of a problem which have to be randomly generated: the topology of the task graph, the timing properties of the task graph and the system architecture. We briefly discuss each of them.

We use three parameters to model the topology of the task graph: the number of nodes $|V|$, the density of the graph in terms of average degree of vertices, and the width of the graph in terms of the average width of a topological sort of the graph.

We use two parameters to model the timing properties of the task graph: the average execution time of the tasks and the Computation to Communication Ratio (CCR), which is the ratio of average execution time to average communication time.

We set the number of processors in the system to $\lceil |V|/5 \rceil$, where $|V|$ is the number of tasks in the generated task graph. We use two parameters to model the system architecture: the variability in the frequency of the processors f_{var} about a nominal frequency and the variability in the bandwidth of the links B_{var} about a nominal bandwidth. The values of f_{var} and B_{var} denote the standard deviation of the Gaussian distribution about the nominal values.

Parameter	Value	End-to-end delay ratio	$\mathbf{M_1}$ compared to $\mathbf{M_2}$		
			Better	Equal	Worse
#Nodes	10	1.6	402	375	5298
	20	2.4	134	40	5901
	30	2.9	111	5	5959
	40	3.2	129	0	5946
	50	3.4	127	0	5948
Density	1	2.6	405	155	9565
	3	3.0	237	142	9746
	5	2.8	261	123	9741
Width	3	2.8	371	143	9611
	5	2.8	314	148	9663
	7	2.8	218	129	9778
Nominal execution time	10	2.2	360	154	9611
	20	2.7	259	134	9732
	30	3.1	284	132	9709
CCR	0.2	2.4	369	92	5614
	0.33	2.9	228	95	5751
	1	3.0	110	89	5876
	3	2.9	112	70	5893
	5	3.0	84	74	5917
Processor frequency variability	0.1	1.8	431	10	9684
	0.2	3.0	252	126	9747
	0.3	3.6	220	284	9621
Link bandwidth variability	0.33	2.9	277	140	9700
	0.66	2.8	293	141	9691
	1	2.7	333	139	9653
Total	-	2.8	420	903	29052

Figure 3. Comparison of end-to-end delays of JUST schedule for mappings $\mathbf{M_1}$ and $\mathbf{M_2}$

The thermal model used in the experiments is for an ARM-like processor sourced from [14] as shown in Figure 1(b), for a nominal frequency say f_0. For other frequency settings, we quadratically scale the power parameters in the active mode, α^{act} and β^{act}, with frequency. The values in the idle mode, α^{idl} and β^{idl} are assumed to be constant: independent of the frequency.

The WCET of a task v on a processor p running at frequency f_p is given by $(f_0/f_p) \cdot c_v$, where c_v is the randomly generated execution time of task v with mean value equal to the average execution time. Similarly, we derive the delay along links as being inversely proportional to the bandwidth of the links.

Each of the above parameters are varied through a set of values. The different values of the parameters are listed in Figure 3. We perform a fully factorial experiment: for each combination of parameter values we generate 5 random problem instances totaling to 30375 problem instances to evaluate the proposed mappings.

A. Comparison of $\mathbf{M_1}$ and $\mathbf{M_2}$ - The need for thermal-aware scheduling

Figure 3 compares, the end-to-end delay obtained for JUST schedules for mappings $\mathbf{M_1}$ and $\mathbf{M_2}$ for the different values of the parameters. Since, we perform a fully factorial experiment, we present the results averaged for each value of each parameter. For each such set of values, we present the end-to-end delay ratio, which is the ratio of the end-to-end delay of the JUST schedule with mapping generated by $\mathbf{M_1}$ to that generated by $\mathbf{M_2}$. The larger this ratio, the greater is the benefit of the thermal-aware mapping step. We also present the number of cases where $\mathbf{M_1}$ performs better, equal and worse than $\mathbf{M_2}$.

Firstly, note that on average, the JUST schedule generated using mapping $\mathbf{M_2}$ has an end-to-end delay about 2.8 times smaller than that generated by mapping $\mathbf{M_1}$. This clearly indicates that for thermally constrained systems, the performance penalty of being thermal-unaware even for the optimal scheduling strategy is severe. Since, the underlying HEFT is only a heuristic, we observe that for about 1% of the problem instances $\mathbf{M_1}$ performs better than $\mathbf{M_2}$.

We now discuss the results for each parameter. The results indicate that the performance gap between $\mathbf{M_1}$ and $\mathbf{M_2}$ is super-linear in the size of the task graph. This implies that the thermal-awareness is more critical in large and distributed task graphs. Results for variation in density and width parameters remain consistent across different values. With increasing nominal execution time, the relative performance of $\mathbf{M_2}$ improves. This confirms the intuitive result that thermal-awareness is more critical when for longer periods of time, the processors are run without preemption. Relative performance of $\mathbf{M_2}$ is poor for very small values of CCR. This perhaps is because with lower CCR values higher communication delays cause greater inaccuracies in the conservative estimate of (13). Relative performance of $\mathbf{M_2}$ greatly increases with higher processor frequency variation. To moderate the results, very small values of f_{var}, up to 0.3, were chosen. For higher variations of frequency, which are not atypical in today's heterogeneous networks, the benefit of thermal-aware scheduling would be even more significant. A similar dependence on B_{var} is not as evident.

Thus, overall $\mathbf{M_2}$ substantially out-performs $\mathbf{M_1}$. The advantage is expected to further increase for larger and more heterogeneous networks.

B. Comparison of $\mathbf{M_2}$, $\mathbf{M_3}$ and $\mathbf{M_4}$

For the same set of random test cases, we compare the end-to-end delay for JUST schedules generated for the three presented thermal-aware mappings $\mathbf{M_2}$, $\mathbf{M_3}$ and $\mathbf{M_4}$. The results averaged across the parameter variations are shown in Figure 4. As before, a higher end-to-end delay ratio indicates that the more sophisticated methods, $\mathbf{M_3}$ and $\mathbf{M_4}$, perform better than $\mathbf{M_2}$.

Comparison	End-to-end delay ratio	Better	Equal	Worse
$\mathbf{M_2}$ vs $\mathbf{M_3}$	1.03	7619	6993	15763
$\mathbf{M_2}$ vs $\mathbf{M_4}$	1.09	4919	4651	23505

Figure 4. Comparison of end-to-end delay of JUST schedule of mappings $\mathbf{M_2}$, $\mathbf{M_3}$ and $\mathbf{M_4}$

As expected, the average optimal end-to-end delay of the different sets of mappings is highest for $\mathbf{M_2}$, which is followed by $\mathbf{M_3}$ and $\mathbf{M_4}$, in order. In percentage terms the reduction in optimal end-to-end delay of mapping $\mathbf{M_3}$ and $\mathbf{M_4}$ in comparison with that of $\mathbf{M_2}$ is 3% and 9%, respectively: a modest gain in comparison to the 2.8x improvement from $\mathbf{M_1}$ to $\mathbf{M_2}$. The average computation overhead of generating mappings $\mathbf{M_3}$ and $\mathbf{M_4}$ with respect to generating mapping $\mathbf{M_2}$ was found to be 1.1x and 2.8x, respectively. Furthermore, the number of solutions where $\mathbf{M_2}$ is better or equal to $\mathbf{M_3}$ and $\mathbf{M_4}$ are substantial (45% and 25%, respectively). Thus, the more sophisticated mapping strategies used for generating $\mathbf{M_3}$ and $\mathbf{M_4}$ provide modest improvements in end-to-end delay over the mapping $\mathbf{M_2}$ and at significant computation overhead. The heuristic nature of the mapping strategies presented here do not enable us to comment on the generality of these results. Perhaps, other mapping heuristics can be derived, which explicitly model the thermal behavior and generate mappings that greatly outperform $\mathbf{M_2}$.

VI. Conclusions

With increasing power densities in today's systems, thermal-aware scheduling must be considered to alleviate the problem of high on-chip temperatures. In this paper, we studied the problem of minimizing end-to-end delay of a task graph when scheduled on a parallel heterogeneous system, while working under specified thermal constraints, and with no speed scaling. When the mapping (binding and ordering of tasks) of the task graph on to the processors is fixed, we showed that the proposed JUST schedule optimally reduces the end-to-end delay. Under such a schedule, enforcing thermal constraints was reduced to checking, before executing a task, if the temperature of the processor is under a pre-computed target temperature: a check that can be easily implemented using on-chip temperature sensors. We then studied the mapping problem. We presented a modification of the timing properties of the original task graph to model specified thermal constraints. On an extensive set of problem instances, we showed that using a standard mapping heuristic, HEFT, on the modified task graph gave mappings with much smaller optimal end-to-end delays than the mappings generated using the original task graph. We also presented more sophisticated thermally-aware mapping strategies which explicitly modeled the thermal behavior of the processors. With results, we concluded

that these methods generate mappings which provide only small improvements over the mappings generated by the simple approach of using the modified task graph.

A possible further work is to study implementation of thermally constrained scheduling without the help of thermal sensors, i.e., without the use of temperature interrupts to enable the next task. This requires off-line computed idle times to be inserted between the execution of consecutive tasks, such that the thermal constraints are satisfied for all possible execution patterns. Considering that tasks always run for their WCETs may not provide the correct value of the idle times.

Another interesting direction forward is to consider systems where processors exchange heat, such as multicore systems. For such systems, the lower bound on γ^{min} derived in (10) would depend on the state of the other processors. Conservative approximations without much deviation from optimality have to be explored.

ACKNOWLEDGEMENTS

This work was supported by EU FP7 projects EURETILE and PRO3D, under grant numbers 247846 and 249776.

REFERENCES

[1] Y.-W. Wu and *et al*, "Joint exploration of architectural and physical design spaces with thermal consideration," in *ISLPED*, 2005.

[2] H. Su, F. Liu, A. Devgan, E. Acar, and S. R. Nassif, "Full chip leakage estimation considering power supply and temperature variations," in *ISLPED*, 2003.

[3] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture," in *ISCA*, 2003.

[4] D. Brooks and M. Martonosi, "Dynamic thermal management for high-performance microprocessors," in *HPCA*, 2001.

[5] N. Bansal, T. Kimbrel, and K. Pruhs, "Dynamic speed scaling to manage energy and temperature," in *FOCS*, 2004.

[6] S. Wang and R. Bettati, "Reactive speed control in temperature-constrained real-time systems," *Real-Time Systems*, vol. 39, no. 1-3, 2008.

[7] J.-J. Chen, S. Wang, and L. Thiele, "Proactive speed scheduling for real-time tasks under thermal constraints," in *RTAS*, 2009.

[8] S. Zhang and K. S. Chatha, "Approximation algorithm for the temperature-aware scheduling problem," in *ICCAD*, 2007.

[9] Y. Xie and W.-L. Hung, "Temperature-aware task allocation and scheduling for embedded multiprocessor systems-on-chip (mpsoc) design," *VLSI Signal Processing*, vol. 45, no. 3, 2006.

[10] Z. Wang and S. Ranka, "A simple thermal model for multicore processors and its application to slack allocation," 2010.

[11] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, 2002.

[12] Y. Liu, R. P. Dick, L. Shang, and H. Yang, "Accurate temperature-dependent integrated circuit leakage power estimation is easy," in *DATE*, 2007.

[13] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Comput. Surv.*, vol. 31, no. 4, 1999.

[14] C.-Y. Yang, J.-J. Chen, L. Thiele, and T.-W. Kuo, "Energy-efficient real-time task scheduling with temperature-dependent leakage," in *DATE*, 2010.