TempoMP: Integrated Prediction and Management of Temperature in Heterogeneous MPSoCs

Shervin Sharifi, Raid Ayoub, Tajana Simunic Rosing
Computer Science and Engineering Department
University of California, San Diego. La Jolla, California 92093
Email: {shervin, rayoub, tajana}@ucsd.edu

***Abstract*—Heterogeneous Multi-Processor Systems on a Chip (MPSoCs) are more complex from a thermal perspective compared to the homogeneous MPSoCs because of their inherent imbalance in power density. In this work we develop *TempoMP*, a new technique for thermal management of heterogeneous MPSoCs which leverages multi-parametric optimization along with our novel thermal predictor, *Tempo*. *TempoMP* is able to deliver locally optimal dynamic thermal management decisions to meet thermal constraints while minimizing power and maximizing performance. It leverages our *Tempo* predictor which, unlike the previous techniques, can estimate the impact of future power state changes at negligible overhead. Our experiments show that compared to the state of the art, *Tempo* can reduce the maximum prediction error by up to an order of magnitude. Our experiments with heterogeneous MPSoCs also show that *TempoMP* meets thermal constraints while reducing the average task lateness by 2.5X and energy-lateness product by 5X compared to the state of the art techniques.**

I. Introduction

Heterogeneous MPSoCs integrate cores of various performance and power characteristics to provide a better trade-off with respect to performance, power and temperature by allowing customization of performance and power of the chip to match the requirements of the workload. They are used in a broad range of applications such as cell phones (Qualcomms Snapdragon [27]) and wireless base stations (Mindspeeds Transcede 4000 [23]) which experience a wider range of environmental conditions than typically seen in data centers and offices. For example, wireless base stations are deployed outdoors in environmental conditions with ambient temperatures ranging from -20°C to over 80°C [31]. For such systems, on-chip dynamic temperature and power management techniques (DTM and DPM) are essential.

Many DTM techniques have been proposed to date, ranging from reactive ([11], [13]) to more recently proposed proactive techniques ([5], [9], [19]) which use thermal predictors. A proactive thermal management technique proposed in [9] uses ARMA model which has to be updated at run time to avoid inaccuracies due to workload changes, thus leading to overhead. A few techniques have been proposed that do not require online adaptation, such as [19] and [5], but both assume that previous thermal history is a good estimate of future thermal behavior, and thus neglect the impact of future power state changes on temperature. Proactive thermal management algorithms that leverage these predictors are heuristic in nature.

A number of thermal management techniques have been proposed that leverage control theory and optimization. In [25], convex optimization is used to control the frequency of the cores on a homogeneous MPSoC to guaranty that thermal constraints are met. In [33], a linear quadratic regulator is used to solve the frequency assignment problem for thermal balancing. To achieve a smooth control and to minimize performance loss and thermal fluctuations in an MPSoC, [32] proposes a technique based on model predictive control. In [10], the problem of scheduling a task graph on a homogeneous MPSoC to minimize the hotspots and balance the temperature distribution is formulated as Integer Linear Programming (ILP) and solved offline. A mixed-integer linear programming formulation based on steady-state thermal model is presented in [8] to minimize peak temperature under hard real time constraints and task dependencies. Because this approach is not scalable, a heuristic approach is also proposed which does not use a predictor and works based on the mobility of the tasks. Relatively little work has focused on heterogeneous embedded MPSoCs. A method is proposed in [28] for energy and temperature aware scheduling of embedded workloads on heterogeneous MPSoCs where decisions on either energy savings or thermal management operating mode of the scheduler and the frequency settings of the cores are made based on the performance requirements of the workload which are estimated at runtime.

In this work we propose a temperature predictor *Tempo* which, in contrast to previous predictors, uses the knowledge of the underlying thermal model of MPSoC, does not need any runtime adaptation and is able to predict potential temperature of any set of future power states without applying them to the system. Being completely linear, *Tempo* can be easily integrated in either model predictive control based techniques such as in [32] or multiparametric programming framework as we did in this work with no need for non-linear optimization. Our thermal management technique, *TempoMP*, considers performance, power and temperature characteristics of each core, which makes it applicable to heterogeneous (and homogeneous) MPSoCs. The performance requirements of the embedded workloads are estimated at runtime and used by *TempoMP* to find the optimal power states of the cores to meet the performance under thermal constraints. The next sections outline the design of our predictor, *Tempo*, followed by the description of how we use the predictor as a part of thermal management framework, *TempoMP*. In results section we show that *TempoMP* can meet the thermal constraints of a heterogeneous MPSoC design while reducing maximum task lateness by 2.5x and improving the energy-lateness product by up to 5x.

II. Temperature Prediction

The objective of our prediction method is to accurately predict future temperature of the cores based on the available temperature and power information. *Tempo* estimates the temperature of the cores at the end of the next scheduling tick $(k+1)$ based on the temperature of the cores at the beginning

978-3-9810801-8-6/DATE12/©2012 EDAA

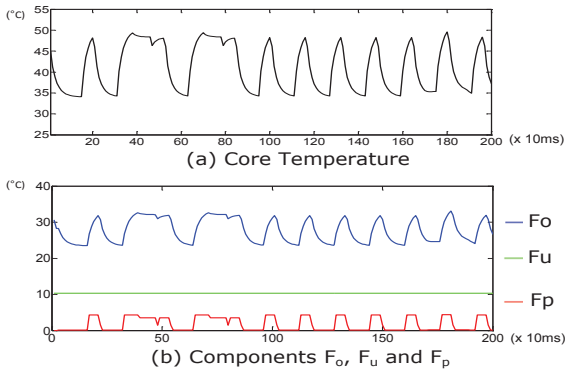


Fig. 1. (a) Temperature of the core (b) Breakdown of temperature into components of equation (4), All relative to ambient (40°C)

of scheduling ticks k and $k+1$, as well as the power states of the cores in these intervals.

Our work is based on compact thermal model of the chip in [17] which leverages the well-known duality of thermal and electrical phenomena. The dynamics of the temperature and the relation between the temperature, power consumption of the cores and thermal characteristics of the system is described as:

$$C_t \frac{d}{dt} T(t) = -G_t T(t) + P(t) \tag{1}$$

where the vectors T and P respectively represent the temperatures and power consumptions at all the nodes of thermal network. G_t and C_t are thermal conductance and capacitance matrices respectively. Assuming the number of the cores and nodes of the thermal RC network to be n and m respectively, P and T are vectors of length m both and G_t and C_t are matrices of size $m \times m$ both.

Given the temperature at all the nodes of the thermal network, estimating the future temperature based on the power is not difficult. However, usually this is not the case because thermal sensors cannot be placed at internal layers (thermal interface material, heat spreader, etc.) and available temperature information is typically limited to the temperature of the cores within silicon layer. The temperature of the internal nodes could also be obtained by simulating the thermal model at runtime, which is not computationally practical. Moreover, without feedback from the thermal sensors, the results may deviate from the actual values.

To reflect this lack of information about the internal nodes, we break the vector of temperature values (T) into two sub-vectors. Sub-vector $\tilde{T}$ represents the temperatures *observable* by thermal sensors (core temperatures), while $\hat{T}$ represents the internal nodes of the thermal network whose temperatures are *unobservable*. The size of these vectors are n and $m-n$ respectively. If each core does not have its own sensor, the technique in [29] can be used to estimate the core temperature using the available sensors.

The analytical solution to the non-homogeneous system of differential equations in equation (1) can be discretized for a scheduling tick of length t_s:

$$\begin{bmatrix} \tilde{T}[k+1] \\ \hat{T}[k+1] \end{bmatrix} = \Psi \begin{bmatrix} \tilde{T}[k] \\ \hat{T}[k] \end{bmatrix} + \Phi \begin{bmatrix} P[k+1] \\ \hat{P}[k+1] \end{bmatrix} \tag{2}$$

where $\Gamma = C_t^{-1} G_t$, $\Psi = e^{-\Gamma t_s}$, $\Phi = \Gamma^{-1}(I - e^{-\Gamma t_s}) C_t^{-1}$. The first term in equation (2) is the contribution of initial conditions and the second term is the contribution of power consumption during each scheduling tick. We break the matrices Ψ and Φ into the following sub-matrices:

$$\Psi = \left[\begin{array}{c|c} \Psi_{oo} & \Psi_{uo} \\ \hline \Psi_{ou} & \Psi_{uu} \end{array}\right], \Phi = \left[\begin{array}{c|c} \Phi_{oo} & \Phi_{uo} \\ \hline \Phi_{ou} & \Phi_{uu} \end{array}\right] \tag{3}$$

where sizes of the matrices Ψ_{oo}, Ψ_{uo}, Ψ_{ou} and Ψ_{uu} are $n \times n$, $n \times m-n$, $m-n \times n$ and $m-n \times m-n$ respectively. Each matrix Ψ_{xy} shows the effect of initial temperature of the set x of nodes on the current temperature of the set y. For example, Ψ_{uo} models the effect of initial temperature of unobservable(u) nodes on the current temperature of observable(o) nodes. Similar notation applies to matrices Φ_{xy}. Since the internal nodes do not consume any power, $\hat{P}[k+1] = 0$. Therefore:

$$\tilde{T}[k+1] = \underbrace{\Psi_{oo}\tilde{T}[k]}_{F_o} + \underbrace{\Psi_{uo}\hat{T}[k]}_{F_u} + \underbrace{\Phi_{oo}P[k+1]}_{F_p} \tag{4}$$

where the first and second terms (F_o and F_u) are respectively the contributions of initial temperature of the observable and unobservable nodes on the temperature at the next scheduling tick. The third term (F_p) is the contribution of the power consumption of the cores. At each scheduling tick, the term F_o can be calculated based on the current temperature of the cores obtained from thermal sensors. The term F_p also can be calculated given the current power consumption of the cores. But due to lack of knowledge of the temperature of unobservable nodes ($\hat{T}[k]$), term F_u is unknown. Therefore, equation (4) is not enough for calculating the future temperature of the cores. Figure 1 shows an example of breakdown of temperature of a core on an MPSoC into three components of the equation (4) with ambient temperature of 40°C. The core temperature ($\tilde{T}[k+1]$) shown in part (a) of the figure is the sum of the components in Figure 1(b). Although the core temperature changes significantly, the term F_u changes very slowly. F_u is the aggregate contribution of initial temperature of nodes in *thermal interface material*, *heat spreader* and *heat sink*. However, it is mainly dominated by the contribution of heat sink nodes. The reason is that the time constants of the nodes in the thermal interface material and heat spreader layers are much smaller than the scheduling interval, therefore the effect of their initial temperature at the end of the scheduling tick will be insignificant. On the other hand, the time constants of the heat sink nodes is much larger than the length of a scheduling interval, therefore the temperature of heat sink nodes will not change significantly during this time. F_u is basically determined by the effect of initial temperature of the heat sink nodes, so it will remain almost constant between consecutive scheduling ticks, so we set $F_u[k] \approx F_u[k-1]$.

Equation (4) for the scheduling tick k can be written as $\tilde{T}[k] = \Psi_{oo}\tilde{T}[k-1] + F_u[k-1] + \Phi_{oo}P[k]$. Value of $F_u[k-1]$ from this equation can be substituted as $F_u[k]$ in equation (4):

$$\tilde{\mathbb{T}}[k+1] = \big((\Psi_{oo} + I)\tilde{T}[k] - \Psi_{oo}\tilde{T}[k-1]\big) + \Phi_{oo}(P[k+1] - P[k]) \tag{5}$$

where $\mathbb{T}$ is the predicted temperature. This equation is used to predict the temperature at the next scheduling tick based on the temperature of the cores and their power settings.

We define *Tempo*'s thermal state of a core, $\tilde{\mathfrak{T}}[k+1]$, in equation (6) as the predicted temperature at the end of scheduling tick $(k+1)$ if the power state of the cores do not change in this scheduling tick. It can be calculated based on

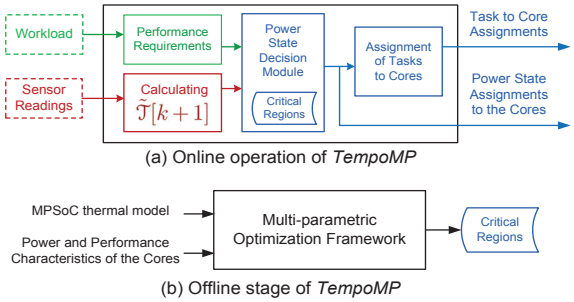


Fig. 2. (a) Online and (b) offline stages of *TempoMP*

the previous and current temperature of the cores. *TempoMP* uses $\tilde{\mathcal{T}}[k+1]$ in its optimization framework.

$$\tilde{\mathcal{T}}[k+1] = ((\Psi_{oo} + I)\tilde{T}[k] - \Psi_{oo}\tilde{T}[k-1]) \tag{6}$$

The second term of equation (5) reflects the effect of power changes on the temperature of the cores.

To account for the leakage power and its dependence on temperature, we use a linear approximation as suggested in [21] with an approximate estimation error of less than 5% [21]. Using this model, the leakage power of a core can be estimated as sum of a constant term and a term linearly dependent on the cores temperature. Therefore, the leakage power of the cores can be estimated as:

$$P_{leak}(t) = LT(t) + Q \tag{7}$$

where L is a diagonal matrix containing the coefficients for the linear terms and Q is a vector of constant terms for different cores. Adding this power to the dynamic power in equation (2) results in a similar equation with new matrices $\Gamma' = C_t^{-1}(G_t - L)$, $\Psi' = e^{-\Gamma' t_s}$, $\Phi' = \Gamma'^{-1}(I - e^{-\Gamma' t_s})C_t^{-1}$.

III. TempoMP

TempoMP systematically considers individual performance, power and thermal characteristics of the cores which makes it applicable to heterogeneous MPSoCs as well as homogeneous ones. Moreover, unlike heuristic scheduling techniques, it is able to guaranty that maximum temperature threshold will be met. This is done by evaluating the thermal impacts of alternative scheduling decisions in advance and avoiding scheduling decisions which might result in thermal emergencies. Figure 2 shows online and offline stages of *TempoMP*.

The scheduling system consists of four major modules. The embedded workloads are characterized in terms of execution time and power. Based on this information, the *performance requirement estimation* module estimates the execution time of the task at different power states, and for each core type determines at which power states the task will be able to meet the performance requirements. At each scheduling tick this module evaluates the current workload in the system and determines how many cores of each type and at what power states are required to meet the performance requirements of the workload.

At each scheduling tick, the *temperature predictor module* calculates *Tempo*'s thermal state, $\tilde{\mathcal{T}}[k+1]$, as shown in equation (6). Given the outputs of the *temperature prediction* and *performance requirement estimation* modules, *power state decision module* determines a set of thermally safe power states which are able to provide performance as close as possible to the workload and task requirements. When the power states of the cores are determined, the tasks will be assigned to the appropriate cores to achieve a good match between their individual performance requirements and performance provided by the cores. We next describes details of power state assignment and runtime operation of *TempoMP*.

A. *Power state assignment in* TempoMP

Power state decision module determines the best power state for the cores by using the results of offline optimization. As Figure 2(b) shows, the inputs to the offline optimization are power and performance characteristics of the cores and the thermal model of the MPSoC.

The optimization formulation uses *Tempo* to evaluate thermal safety of the potential power states. The decision variables in this optimization are power states in the next scheduling tick represented as $\alpha[k+1]$ (in bold in equation (8)). If core n is set to power state v, then $\alpha_{n,v}$ is 1, and 0 otherwise. Optimization chooses $\alpha[k+1]$ values such that the total power is minimized under the thermal constraints while meeting performance requirements. We denote the number of cores of type Ω which are set to power state v as $\lambda_{\Omega,v}$, where $\lambda_{\Omega,v} = \sum_{\omega \in \Omega} \alpha_{\omega,v}$. The minimum number of cores of type Ω that have to run at a given power state v to meet performance requirements of the workload is denoted as $\sigma_{\Omega,v}$. This is determined at runtime based on the performance requirements of the tasks using information such as deadlines or required throughput. For each core, if we assume core of type k has v active power states and one sleep state, the power consumption of core n which is of core type k can be written as $P[n] = \alpha_{n,1} \cdot P_{k,1} + ... + \alpha_{n,v} \cdot P_{k,v} + \alpha_{n,sleep} \cdot P_{k,sleep}$ where P_k,v is the average power consumed at power state v at a core of type k. Based on these variables, the optimization problem is formulated as:

$$\begin{aligned} \underset{\alpha}{\text{minimize}} \quad & P_{total}(\boldsymbol{\alpha[k+1]}, \tilde{\mathcal{T}}[k+1]) \\ \text{subject to} \quad & \mathbb{T}(\boldsymbol{\alpha[k+1]}, \alpha[k], \tilde{\mathcal{T}}[k+1]) \prec T_{Th} \\ & \forall \Omega, v : \lambda_{\Omega,v} \geq \sigma_{\Omega,v}. \end{aligned} \tag{8}$$

where we use $\prec$ as element-wise *less than* operator. $\tilde{\mathcal{T}}[k+1]$ is the thermal state defined by *Tempo*. The objective of this optimization is to minimize the total power (P_{total}) of the cores under the thermal limits. Leakage power is taken into account in P_{total} using equation (7). The first constraint enforces the predicted temperature at the next scheduling tick to be lower than the maximum threshold, while the second set of constraints require the power states chosen by optimization to provide at least the performance required by the workload.

Instead of solving the optimization problem at each scheduling tick, we use an approach based on multi-parametric programming [26]. Optimization parameters σ, $\alpha[k]$ and $\tilde{\mathcal{T}}[k+1]$ partition the parameter space into separate regions called *critical regions*. Each possible combination of σ, $\alpha[k]$ and $\tilde{\mathcal{T}}[k+1]$ corresponds to one and only one of these critical regions which represents the optimum power states of the cores for that specific combination. The region basically specifies the validity range of that set of power states such that temperatures of all the cores are below the threshold temperature and the total power is minimized. The actual values for the optimization parameters are found at runtime. Then the corresponding region and appropriate set of power states for the cores are found.

Multi-parametric programming allows analyzing the effect of variations and uncertainty in optimization problems where objective function is to be optimized subject to a set of constraints, and a set of parameters which are variable [26]. It

provides the critical regions defined by σ, $\alpha[k]$ and $\tilde{\mathcal{T}}[k+1]$ and the corresponding power states represented by $\alpha[k+1]$. Using this approach, the set of optimal solutions ($\alpha[k+1]$) is obtained as an explicit function of the parameters (σ, $\alpha[k]$ and $\tilde{\mathcal{T}}[k+1]$). Multi-parametric programming solves the optimization problem offline and provides the set of critical regions and parametric solutions. To solve the problem at runtime, the *power state decision module* does not need to do any optimization online. It only needs a limited number of operations to be performed at runtime to find the regions representing the $\alpha[k+1]$ corresponding to the current value of σ, $\alpha[k]$ and $\tilde{\mathcal{T}}[k+1]$.

As the first set of constraint in equation (8) suggests, temperature of the cores at the next scheduling tick depends on decision variable $\alpha[k+1]$ and optimization variables σ, $\alpha[k]$ and $\tilde{\mathcal{T}}[k+1]$. Larger number of optimization parameters results in much larger space of solutions. To reduce the number of optimization variables, we rewrite equation (5) as:

$$\tilde{\mathbb{T}}[k+1] = \underbrace{\Phi_{oo} P[k+1]}_{F_p} + \underbrace{(\tilde{\mathcal{T}}[k+1] - \Phi_{oo} P[k])}_{F_r} \quad (9)$$

At each scheduling tick, component F_r is completely known as it depends on the current and previous temperature and power values. The only unknown component is F_p. Therefore, at each scheduling tick, F_r can be calculated and used as the optimization parameter instead of two sets of parameters $\alpha[k]$ and $\tilde{\mathcal{T}}[k+1]$. This considerably reduces the dimensions and size of the parameter space. Now the set of optimization variables are only σ and $F_r[k]$ At each scheduling tick, F_r is calculated based on the observable temperatures and current power state of the cores. Then it is given to the power state decision module and is used along with performance requirements to find the corresponding $\alpha[k+1]$ at runtime. If no thermally safe power state can be found which can meet the current workload requirements, the performance requirements given to power state decision module (σ) are reduced until this module finds a set of thermally safe power states.

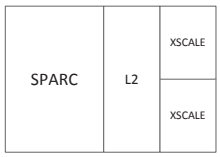
The multi-parametric programming framework is implemented in YALMIP [4] toolbox in Matlab [22] which relies on Multi-Parametric Toolbox (MPT) [3] in Matlab [22]. Optimization results are saved in the power state decision module to be accessed during the online phase of our algorithm. The memory required for the critical regions of the MPSoC of Figure 3 was less than 500KB.

The last step of *TempoMP* which is done after choosing the power states is assigning the tasks to the cores. The next subsection discusses how this step works.

B. Runtime task assignment to the cores

At each scheduling tick, at this stage the outputs of *performance estimation module* and *power state decision module* are ready. For each task, the performance estimation module can determine the power state at which this task needs to be run to be able to meet its performance requirements (deadline, throughput, etc.). Given this information from the performance estimation module and the output of power state decision module, the task assignment module tries to assign the tasks to the cores such that the tasks with higher performance requirements are assigned to the cores providing higher performance.

Algorithm 1 explains how this is done at each scheduling tick in a deadline based system. At each step of the algorithm, the available task with the earliest deadline is picked (line 7) and matched to the available core with the highest processing capability (chosen at line 8). This is repeated until no core or no task is left (checked at line 3).



(a) Floorplan of MPSoC

Thermal Parameters	
Die Thickness	0.5 mm
Convection Resistance	7 °K/W
Convection Capacitance	100 J/°K

(b) Thermal parameters

Processor		XScale-like	SPARC-like
Issue width		1	2
Area (mm^2)		1.4	5
Frequency Settings (MHz)	1.2 V	624	1200
	1.18 V	416	1140
	1.06 V	208	1000
Average Dynamic Power (W)	1.2 V	0.34	2.40
	1.18 V	0.32	2.10
	1.06 V	0.28	1.90

(c) Characteristics of the cores

Fig. 3. Characteristics of the MPSoC

Algorithm 1 Task to core assignment

```
1:  C ⇐ currently available cores
2:  J ⇐ current tasks in the system in the decreasing order of performance require-
    ments (e.g. increasing order of deadline)
3:  while (C and J are not empty) do
4:      Ω ⇐ the highest performance type of cores in C
5:      C_Ω ⇐ set of cores of type Ω in C
6:      while C_Ω is not empty do
7:          j ⇐ the first task in J
8:          i ⇐ the core at the highest power state in C_Ω
9:          assign task j to core i
10:         remove j from J and i from C_Ω
11:     end while
12:     remove C_Ω from C
13: end while
```

IV. Experimental results

A. Methodology

The cores used in our experiments are a low-power in-order architecture similar to the SPARC cores in UltraSPARC T1 [20] and a very low power core designed for embedded systems, similar to Intel's XScale [18]. Characteristics of the cores and the MPSoC are shown in Figure 3 for 65 nm technology. The areas of the cores are derived from published photos of the dies after subtracting the area occupied by I/O pads, interconnection wires, interface units, L2 cache, and control logic and scaled to 65nm. The L2 cache has 1MB size, 2 banks, 64-byte lines, and is 4-way associative. Using CACTI [16], the area and power consumption of the caches at 65nm are estimated as $14mm^2$ and 1.7W, respectively. The cache power consumption value includes leakage. We obtain the performance and power data for all the benchmarks for the different types of cores using the M5 Simulator [6] integrated with Wattch [7] power model updated with model parameters for 65nm. We compute the leakage power of CPU cores based on the areas and temperature of the units. Temperature dependence of leakage is modeled using the model introduced in [15] with the parameters for 65nm. The power traces are then utilized in the thermal simulations which are performed using HotSpot Version 4.2 [2] thermal modeling tool with a sampling interval of $100\mu s$ in block mode where each core is represented with one block. In many embedded systems such as cell phones there is no heat sink or spreader. To model

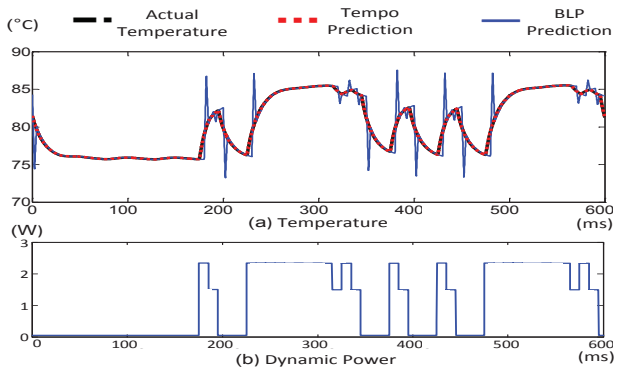


Fig. 4. Comparison of *Tempo* and *BLP* predictor [5]

this within HotSpot, we keep the spreader very thin ($0.1mm$). Heat sink is replaced by a package with thermal parameters shown in Figure 3(b) which are within the ranges suggested by [1] and [24]. Although we use an embedded package for these experiments, we have verified our technique for various configurations of embedded and general purpose packages.

The workloads in our experiments consist of integer benchmarks provided in MiBench benchmark suite [14] which include automotive/industrial, network and telecommunications applications. In addition to the datasets provided in MiBench suite, we use datasets provided by [12]. We create workloads consisting of varying number of tasks from MiBench suite. Instances of each task are generated regularly at every arrival period. We set deadline (d) and (τ) of the tasks to twice the execution time of that task at the slowest frequency on the slowest core (XScale). This way the tasks can meet their deadlines irrespective of the core type they are assigned to.

The overhead of switching to a new power state is assumed to be $50\mu s$. We use the power consumption of the higher power state during DVFS transition. The overhead of migration of the tasks between the cores is $10\mu s$ [13].

B. Results

First we compare *Tempo* with the state of the art (**Band-Limited Predictor (*BLP*)** proposed in [5]). The coefficients used in calculations of *BLP* are also calculated at design time and no training phase is required. We use the same parameters as in [5], namely $\alpha = 0.135$, $m = 3$ and $N = 3$ which lead to best BLP predictions. Figure 4(a) shows the actual temperature of a SPARC-like core along with the prediction results of *Tempo* and *BLP*. Figure 4(b) shows the trace of dynamic power applied to the core. As shown in the figure, as long as the temperature changes are smooth and there are no quick power state changes, both predictors do well. However, *BLP* fails when the power state of the core changes. For example, in Figure 4(a), right after the first power state change at around $190ms$, *BLP* underestimates the temperature. This is because *BLP* relies exclusively on the temperature history and recent trend. Therefore, even if the new power state is known, *BLP* cannot predict the temperature before the new power state is applied and has impacted the temperature trend. Moreover, as the figure shows, even after the first sample of temperature signal is observed after the change in temperature trend, *BLP* considerably overestimates the temperature. However, *Tempo* accurately predicts the future temperature given the future power state of the cores. As Figure 4 shows, maximum

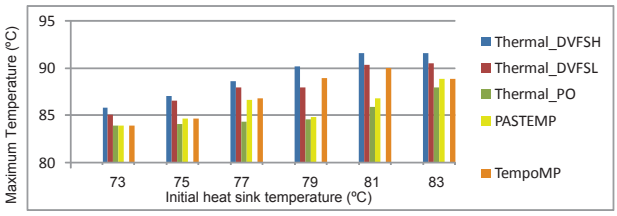


Fig. 5. Comparison of Maximum Temperature

prediction error of *BLP* can be up to 5°C while it is less than 0.5°C for *Tempo*.

The results of *TempoMP* further prove the efficiency of *Tempo*. We next compare *TempoMP* with three other state of the art temperature-aware scheduling techniques. For all of them, maximum safe temperature (T_{th}) is 90°C. ***PASTEMP*** and ***Thermal_PO*** [30] are both proactive techniques which use optimization to assign thermally safe power states to the cores based on the performance requirements of the workload and also the thermal state of the system. The optimization in *PASTEMP* uses a modified dynamic thermal model called *instantaneous thermal model* [30], while optimization in *Thermal_PO* is based on a steady state thermal model. The third technique, ***Thermal_DVFS*** relies on the direct temperature readings from the thermal sensors. It switches the core to a lower power state when the core temperature reaches a critical threshold of T_{top}. When the temperature gets below a lower threshold, T_{bottom}, the power state of the core can be switched to a higher level again. This lower threshold prevents oscillations between power states. We test two variations of *Thermal_DVFS* with different thresholds: *Thermal_DVFSL* with T_{top}=85°C and T_{bottom}=83 °C and *Thermal_DVFSH* with T_{top}=87°C and T_{bottom}=85°C. *Thermal_DVFS* does default load balancing to create a balanced distribution of the workload across the cores.

We run a same mix of MiBench benchmarks for 100 seconds. Before each run, the heat sink is pre-heated to the initial temperature T_i ranging from 73 °C to 83 °C. Figure 5 reports the maximum core temperature observed under various conditions. At low T_i, all techniques are able to meet the 90°C temperature threshold. Only *Thermal_DVFS* violates the threshold at high T_is, because in this region the core temperature quickly reaches above the threshold before *Thermal_DVFS* can respond. *PASTEMP* and *Thermal_PO* have consistently lower maximum temperature, because their thermal models overestimate temperature. As a result of these pessimistic estimates, they tend to make more conservative scheduling decisions and use lower power states. Although this keeps the temperature lower, it results in more performance loss compared to the other techniques as shown next.

To compare how techniques address individual performance requirements of the tasks, we measure *lateness* of a task which is defined as the time it takes to finish the task after its deadline is missed. Figure 6 shows that as T_i increases, so does the average lateness. *TempoMP* misses deadlines only at the highest T_i where turning on the SPARC without violating T_{th} is not possible. In this case, no temperature aware scheduling technique can meet all the deadlines because while the performance of SPARC core is necessary to meet the deadlines, turning this core on will violate T_{th}. At the highest T_i, *TempoMP* and *PASTEMP* perform similarly because in this

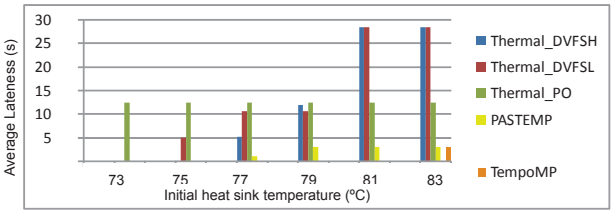


Fig. 6. Average lateness (in seconds)

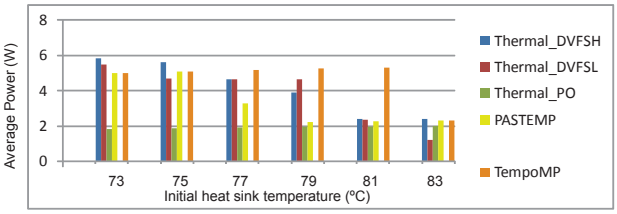


Fig. 7. Average power consumption

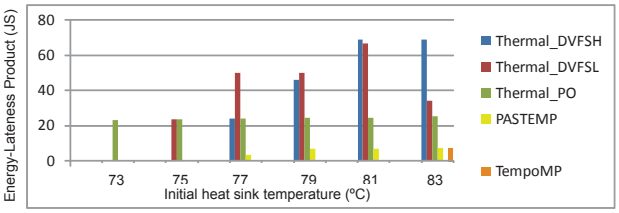


Fig. 8. Average energy lateness product

case the larger core cannot be turned on and even *TempoMP* cannot use this core. Therefore its performance is similar to *PASTEMP*. Due to the pessimistic temperature estimates of steady state thermal model, *Thermal_PO* consistently performs worse than the other techniques in terms of performance.

The average power consumption of MPSoC using DTM techniques is presented in Figure 7. In some cases *TempoMP* consumes more power than the other techniques. Due to the accurate estimation of thermal slack, *TempoMP* is able to turn on the higher power large cores to meet the performance requirements. Other techniques do not use these cores due to their pessimistic temperature estimates and as a result have much lower performance.

In Figure 8 we compare the techniques using a combined measure of energy efficiency and performance, energy-lateness product (ELP). It is similar to energy-delay product (EDP) metric, but applied to the systems with deadlines. In terms of ELP, *TempoMP* is 5X better compared to the other techniques. *Thermal_PO* is the least energy efficient because while it uses lower power states of the lower power cores, due to longer execution time and leakage power, it is not energy efficient.

V. Conclusion

This paper introduces *TempoMP*, a multi-parametric thermal optimization method for heterogeneous MPSoCs that leverages our new low overhead thermal predictor, *Tempo*, to obtain locally optimal thermal management decisions for embedded tasks with deadlines while effectively leveraging task migration and DVFS. Compared to the other proactive techniques, *TempoMP* consistently performs better in terms of performance while always meeting thermal requirements. On average, it reduced the lateness of the tasks by 2.5X and energy-lateness product by 5X. Our temperature predictor, *Tempo* also shows up to an order of magnitude reduction in the maximum prediction error compared to the previous techniques.

Acknowledgements. This work has been funded by NSF CCF grant 0916127, NSF grant 1029783, UCSD Center for Networked Systems, Qualcomm, Texas Instruments, SRC, MuSyC.

References

[1] JEDEC Standard 51-2A, Integrated Circuit Thermal Test Method Environmental Conditions-Natural Convection (Still Air), 2008. http://www.jedec.org/standards-documents/docs/jesd-51-2a.
[2] Hotspot temperature modeling tool, 2011. `http://lava.cs.virginia.edu/HotSpot/`.
[3] Multi-parametric toolbox (mpt), 2011. `http://control.ee.ethz.ch/mpt/`.
[4] Yalmip, 2011. `http://users.isy.liu.se/johanl/yalmip/`.
[5] R. Ayoub and T. S. Rosing. Predict and act: dynamic thermal management for multi-core processors. In *ISLPED '09*, pages 99–104, 2009.
[6] N. L. Binkert et al. The m5 simulator: Modeling networked systems. *IEEE Micro*, 26, July 2006.
[7] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *ISCA 2000*, pages 83 –94, 2000.
[8] T. Chantem, R. Dick, and X. Hu. Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs. In *DATE*, 2008.
[9] A. Coskun, T. Rosing, and K. Gross. Utilizing predictors for efficient thermal management in multiprocessor socs. *Trans. CAD*, October 2009.
[10] A. Coskun, T. Rosing, K. Whisnant, and K. Gross. Temperature-aware mpsoc scheduling for reducing hot spots and gradients. *ASPDAC*, 2008.
[11] J. Donald and M. Martonosi. Techniques for multicore thermal management: Classification and new exploration. In *ISCA 2006*, pages 78–88.
[12] G. Fursin et al. Midatasets: creating the conditions for a more realistic evaluation of iterative optimization. HiPEAC'07, pages 245–260, 2007.
[13] M. Gomaa et al. Heat-and-run: leveraging smt and cmp to manage power density through the operating system. In *ASPLOS-XI*, 2004.
[14] M. R. Guthaus et al. Mibench: A free, commercially representative embedded benchmark suite. In *IEEE International Workshop of the Workload Characterization*, pages 3–14, 2001.
[15] S. Heo, K. Barr, and K. Asanović. Reducing power density through activity migration. In *ISLPED*, 2003.
[16] HP. Cacti, 2011. `http://www.hpl.hp.com/research/cacti`.
[17] W. Huang et al. Hotspot: a compact thermal modeling methodology for early-stage vlsi design. *IEEE Trans. on VLSI*, 14(5):501 – 513, 2006.
[18] Intel. Intel pxa270 processor, electrical, mechanical and thermal specification data sheet., 2011. `http://www.intel.com`.
[19] O. Khan and S. Kundu. Hardware/software co-design architecture for thermal management of chip multiprocessors. In *DATE*, 2009.
[20] A. S. Leon et al. A power-efficient high-throughput 32-thread sparc processor. *IEEE Journal of Solid-State Circuits*, 42(1):7 –16, 2007.
[21] Y. Liu, R. Dick, L. Shang, and H. Yang. Accurate temperature-dependent integrated circuit leakage power estimation is easy. In *DATE*, 2007.
[22] MATLAB. *version 7.10.0 (R2010a)*. The MathWorks Inc., 2010.
[23] MindSpeed. Transcede 4000 baseband processor, 2011. `http://www.mindspeed.com`.
[24] F. Mohammadi and M. Marami. Dynamic compact thermal model of a package. In *ISCAS*, 2008.
[25] S. Murali et al. Temperature control of high-performance multi-core platforms using convex optimization. DATE '08, pages 110–115, 2008.
[26] D. Narciso, N. Faisca, and E. Pistikopoulos. A framework for multi-parametric programming and control; an overview. In *IEEE International Engineering Management Conference*, 2008.
[27] Qualcomm. Snapdragon Processor, 2011. `http://www.qualcomm.com/snapdragon`.
[28] S. Sharifi et al. Hybrid dynamic energy and thermal management in heterogeneous embedded multiprocessor socs. *ASPDAC*, 2010.
[29] S. Sharifi and T. S. Rosing. Accurate direct and indirect on-chip temperature sensing for efficient dynamic thermal management. *IEEE Trans. on CAD*, 2010.
[30] S. Sharifi and T. S. Rosing. Package-aware scheduling of embedded workloads for temperature and energy management on heterogeneous mpsocs. In *International conference on computer design*, 2010.
[31] A. Telikepalli. Designing for power budgets and effective thermal management. *Xcell Journal*, (56), 2006.
[32] F. Zanini, D. Atienza, L. Benini, and G. De Micheli. Multicore thermal management with model predictive control. In *European Conference on Circuit Theory and Design, 2009.*, pages 711 –714, 2009.
[33] F. Zanini, D. Atienza, and G. De Micheli. A control theory approach for thermal balancing of mpsoc. In *ASP-DAC '09*, pages 37–42, 2009.