Distributed Thermal Management for Embedded Heterogeneous MPSoCs with Dedicated Hardware Accelerators

Yen-Kuan Wu
Electrical and Computer Engineering Dept.
University of California at San Diego, La Jolla, CA
Email: yew002@ucsd.edu

Shervin Sharifi, Tajana Simunic Rosing
Computer Science and Engineering Dept.
University of California at San Diego, La Jolla, CA
Email: shervin@ucsd.edu, tajana@ucsd.edu

***Abstract*—This paper addresses thermal management in heterogeneous MPSoCs where the power states of the general purpose cores can be controlled by the operating system (OS) while OS is not able to control power states of the dedicated hardware accelerators (DHAs). We propose a scalable and cooperative distributed thermal management technique [1] which works based on the cooperation of local controllers deployed in some of the cores. Through low overhead message passing, these controllers communicate in order to exchange temperature and performance related information which is used to find the best thermally safe set of frequency settings for the cores. Experimental results show that for our technique can successfully reduce the deadline miss rate by 47.16% in average compared to localized thermal management techniques while successfully satisfying temperature constraints.**

I. Introduction

Continuously decreasing device dimensions due to technology scaling along with increasing power densities result in higher temperatures. This higher temperature can degrade reliability of the system, increase leakage power, increase performance degradation and need more expansive cooling and packaging costs [1]. To mitigate these issues, temperature should be addressed in various levels of embedded system design. Many of them operate in diverse range of environmental conditions. For example, biosensor networks implanted in animals or humans require very low temperature [2]. Cell phones must operate under a very wide range of ambient temperatures without the benefit of more advanced packaging and cooling due to cost and space considerations. Workload and power management techniques are crucial for such systems.

One of the major reasons for prevalence of *multiprocessor systems-on-chip (MPSoCs)* is their ability to provide higher performance within a specific power budget and thermal envelope compared to their single core counterparts. Heterogeneous MPSoCs provide even a better trade-off by integrating cores operating at various power and performance points and allowing a better matching of delivered performance to the performance demands of the workload. Some MPSoCs, especially in embedded applications, integrate *dedicated hardware accelerators* (DHAs) for special purpose computing such as video/audio decoding and graphics acceleration.

Existing examples of such embedded heterogeneous MPSoCs are Texas Instruments' OMAP or NVIDIA's Tegra 2 (shown in figure 1). These MPSoCs are currently used in devices such as smart phones and tablet PCs. Texas Instrument's OMAP4 platform includes two *general purpose cores* (GP cores) which are based on ARM Cortex A9, a DSP, an image signal processor and a graphics processing unit. NVIDIA Tegra 2 includes three GP cores (two Cortex A9 and one ARM7 processors), 2D/3D graphics processing units, video decode and encode processors, an image signal processor, an audio processor, etc.

These DHAs are often third party *intellectual property* (IP), and do not run the same OS as GP cores. Although some of these DHAs might have built in hardware based thermal management mechanisms, they typically operate independently from a centralized thermal controller. Due to the increasing functional demand of embedded systems, these DHAs become more complex, consume more power and contribute more to the system's thermal issues. For example, in NVIDIA's Tegra 2, the silicon area dedicated to DHAs is more than twice of the area consumed by the general purpose processors as shown in figure 1.

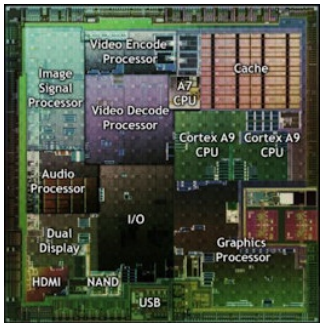

Fig. 1: NVIDIA Tegra 2 mobile processor [11]

Thermal management techniques for MPSoCs are classified into three main categories: central, localized, and distributed. Centralized is usually implemented in the OS, which as temperature increases may slow down cores or migrate threads between different cores. The complexity increases exponentially with the number of cores. They are not applicable to the cases where there is limited global control of all the cores. In a localized solution, each core controls its own temperature independently. Because temperature of the cores highly depends on the states of the other cores due to their physical proximity, this solution may result in very suboptimal

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

results. In a distributed solution, the thermal control of a core is performed locally, but in collaboration with the other cores in order to reach a good solution in a cooperative manner.

In this paper we propose a *distributed thermal management* technique for heterogeneous MPSoCs. While due to the distributed nature of this technique, it is much more scalable than the centralized techniques, it is also applicable to cases where power states of some of the cores cannot be controlled by the operating system. The algorithm relies on a cooperation of simple controllers implemented on the individual cores which collectively decide about the future frequency settings of the cores. These simple controllers can be implemented in hardware or software. They communicate through low overhead message passing in order to exchange thermal and performance information. Our experiments show the deadline miss rate can be reduced by 47.16% in average as compared to localized technique.

Section II discusses the related work while the details of our technique is described in Section III. Our results which are presented in section IV show quantitative benefits of our technique compared to the previous centralized thermal management techniques.

II. Related work

Thermal management techniques are able to prevent thermal emergencies by reducing the heat generation or distributing the heat generation in order to reduce the power density and temperature. By using mechanisms such as *dynamic voltage and frequency scaling* (DVFS), *dynamic power management* (DPM) or thread migration. Thread migration usually is not possible for the hardware accelerators because of the instruction set incompatibilities, but DVFS and DPM can be used for all types of cores. Scheduling tasks on MPSoCs under thermal constraints is in general an NP-hard problem due to the huge number of choices for assigning tasks to the cores and setting core frequencies. Lack of control over the frequency settings of DHAs in the embedded MPSoCs further complicates the scheduling problem in these systems.

While many dynamic thermal management techniques have been proposed for MPSoCs, most of them address thermal management in homogeneous MPSoCs and assume a full control of the operating system over the power states of all the cores. In [13], a probabilistic approach is taken for thermal management of homogeneous MPSoCs where the probability of assigning a task to a core is changed in the OS based on the temperature history of that core. Donald and Martonosi studied various combinations of thread migration, DVFS and clock gating for thermal management of a homogeneous MPSoC in [3]. For dynamic thermal management in homogeneous multi-threaded CMPs, [4] suggests temperature balancing by temperature-aware thread assignment and thread migration.

Techniques have been proposed previously for thermal management of heterogeneous MPSoCs as well. In [5], a technique is proposed for asymmetric dual core designs where the workload is migrated from high power cores to low power cores in order to reduce the occurrence of thermal emergencies with low performance impact. In [6], a temperature and energy management approach is presented for heterogeneous MPSoCs. This approach is called hybrid because the scheduler can operate in two different modes based on the workload utilization. At low or moderate utilization, energy optimization has a higher priority and is achieved through workload scheduling and disabling the cores which are not required. Since thermal issues are more likely to happen at high utilization, for these cases a temperature balancing approach is taken using task assignment and DVFS. The work in [8] proposes a technique for scheduling embedded workloads on heterogeneous MPSoCs. In this technique, at each scheduling tick, based on the thermal state of the cores and performance requirements of the workload, frequencies are chosen for the cores, and tasks are assigned to the cores based on their performance requirements.

All of these techniques assume centralized control of the operating system over all of the cores. While centralized thermal management techniques can result in more optimal solutions, they are not practical in cases such as heterogeneous MPSoCs where control over power states of some of the components is limited. Moreover, the complexity of centralized thermal management techniques increases exponentially with the number of cores which makes them impractical for MPSoCs with higher number of components.

A distributed thermal management technique for MPSoCs has been proposed in [9]. It assumes that the neighbor cores are able to migrate or exchange the tasks among them to control temperature in many-core systems in a distributed manner. This work also assumes the operating system running on each core has the full control of the power states of that core and is able to coordinate with the neighbor cores and migrate or exchange the tasks with them.

As a result, it is not applicable to heterogeneous MPSoCs with DHAs that are common in embedded systems. This combination of central and combined controls over the power states of the cores makes the holistic thermal management of such systems even more challenging. To the best of our knowledge, there is no previous work addressing this problem.

In this paper, we propose a *distributed thermal management* (*DistriTherm*) technique which addresses the above mentioned problem by using a distributed and cooperative thermal control approach using simple per-core controllers which can be implemented in hardware or software. The decisions on power states of the cores are made based on the performance and temperature related information communicated among these thermal controllers. This technique has very low overhead and is more scalable than the centralized approach. It is able to reduce the number of deadline misses in the system while keeping the temperature below the critical level. Details of the technique are described in the next section.

III. DistriTherm Technique

In this section we describe our $DistriTherm$ technique which performs distributed thermal management of heterogeneous MPSoC through communication and cooperation among the cores. *DistriTherm* is applicable to the case of heterogeneous MPSoCs where some of the cores are not under full control of the operating system and/or have their own built-in thermal management capabilities. More generally, *DistriTherm* is applicable to any MPSoC where cores act separately, but share a common communication channel. *DistriTherm*'s message passing is very low overhead and can be implemented by a simple controller and interrupt mechanism or through a shared medium such as AMBA bus which is typically present in embedded systems.

The messages passed among the individual controllers includes temperature and performance related information which is used by the controllers to estimate the thermal and performance impact of each core's power state changes on the neighboring cores. The thermal effect of a core on the others depends on various parameters such as size of the cores, their power characteristics, layout of the chip and the thermal characteristics of the system. For example, a large and high power core affects the temperature of its neighbors more than a small low power core. We use thermal correlation metric to quantify this thermal impact, and use it to quantify the trade-off between temperature benefits and performance cost of scheduling decisions.

When in a thermal emergency, a core broadcasts its request to its *thermally correlated* cores calling for a cooperative action. The set of relevant cores exchange information regarding their thermal state and the performance overhead of engaging a temperature control mechanism. Then, based on the exchanged information and the desirable temperature-performance trade-off, the initiating core sends signals to each of the thermally correlated cores, either asking it to reduce its frequency or stating that it can keep its frequency.

Each core's thermal controller operates in a *normal* mode or *emergency shutdown* mode as shown in figure 2 using StateChart diagram. By default, the controller is in *normal* mode and switches to *emergency shutdown* mode only when the core temperature exceeds the maximum allowed temperature T_{max} also known as *critical temperature*.

In normal mode, three processes run concurrently the hardware controller as shown in Figure 2. The right process explains the $master$ mode, where the temperature of the core approaches a *threshold temperature*, and submits requests to the thermally correlated cores to cooperate as $slave$ cores in order to resolve the thermal emergency. The left process corresponds to $slave$ mode where the core receives requests from other $master$ cores to contribute as $slave$ in managing the temperature at that $master$ core.

The rest of the section describes our technique in more details. First, we describe our thermal model and define the thermal correlation between each pair of cores, which our algorithm uses to cooperatively choose a suitable core for reducing frequency to improve the temperature of the $master$. Second, we explain how our $distriTherm$ technique works in more details.

A. Thermal Correlation

We use a first order electrical network to model the temperature on chip [10], which can be formally defined as

$$C\frac{d}{dt}T(t) = GT(t) + P(t) \tag{1}$$

where $T(t)$ is the temperature vector representing the temperature of all the internal nodes at time t. $P(t)$ is the power vector that representing the power consumed by each internal node. In the thermal network model, $T(t)$ is equivalent to voltage, while $P(t)$ is equivalent to current. Therefore, we call matrix C *thermal capacitance* matrix, and matrix G *thermal conductance* matrix, while both of them are time invariant. They can be obtained by the thermal characteristic, dimensions and floorplan of the chip.

The discrete version of the temperature model in [10] is

$$T[k+1] = AT[k] + BP[k] \tag{2}$$

where $T[k]$ and $P[k]$ denote the temperature and power at kth sample respectively. Matrix A and B can be derived from the discretization of continuous model as shown in equation (3).

$$\begin{cases} A = e^{C^{-1}G\psi} \\ B = \left(\int_{\tau=0}^{\psi} e^{C^{-1}G\tau}d\tau\right)C^{-1} \end{cases} \tag{3}$$

where ψ is the sampling interval. Because both matrix C, G and constant ψ are time invariant, matrix A and B can be calculated offline. The sampling interval is determined by the *response time* of the system, that is, how fast the system can respond once it detects a thermal emergency. In our experiments, we chose a sampling interval equal to the scheduling interval which is $1ms$ as reported in table II.

As equation 3 shows, the temperature of a core on an MPSoC depends on the thermal state of the other cores. We call this relationship *thermal correlation* between each pair of cores. For example, to reduce the temperature of a core j in thermal emergency, we can reduce the power of core j itself by using DPM or DVFS, or other cores' power which are thermally correlated to core j.

The set of the cores in the system is represented by I while the set of cores in thermal emergency are represented by J. T_j is the temperature of core j, and T_{th} is the *threshold temperature*. Therefore, J would be the set of cores for which $T_j \geq T_{th}$.

To make decisions about power state settings, we need to quantify the trade-off between temperature improvement and performance overhead caused by a power state switching. According to equation (2), the temperature of a specific core i at the next time interval is

$$T_i[k+1] = \sum_{j=1}^{n}(A_{ij}T_j[k] + B_{ij}P_j[k]) \tag{4}$$

Equation (2) shows the temperature of a core i at next sample according to the discrete model, where n is the number of nodes in the temperature model. From equation (2), we can conclude that when all cores retain their power state but core i changes the power state at k, the effect on its own temperature at time $(k+1)$ is:

$$\delta T_i[k+1] = B_{ii}(P_i'[k] - P_i[k]) \tag{5}$$

We can also formally define the *temperature improvement* caused by lowering the power state of core i on core j by:

$$tempImp(i,j) \quad = B_{ji}(P_i[k] - P_i'[k]) \tag{6}$$

where $P_i'[k]$ is the new power of i if its frequency scales at time k. Please note that $tempImp(i,j)$ can be negative if the new power increases instead of decreases. This formal definition of temperature improvement makes it possible to quantify the trade-off between performance and temperature improvement.

When solving the thermal management problem using a distributed approach, the cores need to communicate to each other to set their power state appropriately. However, the communication overhead is directly proportional to the number of core pairs communicated with each other. To reduce this

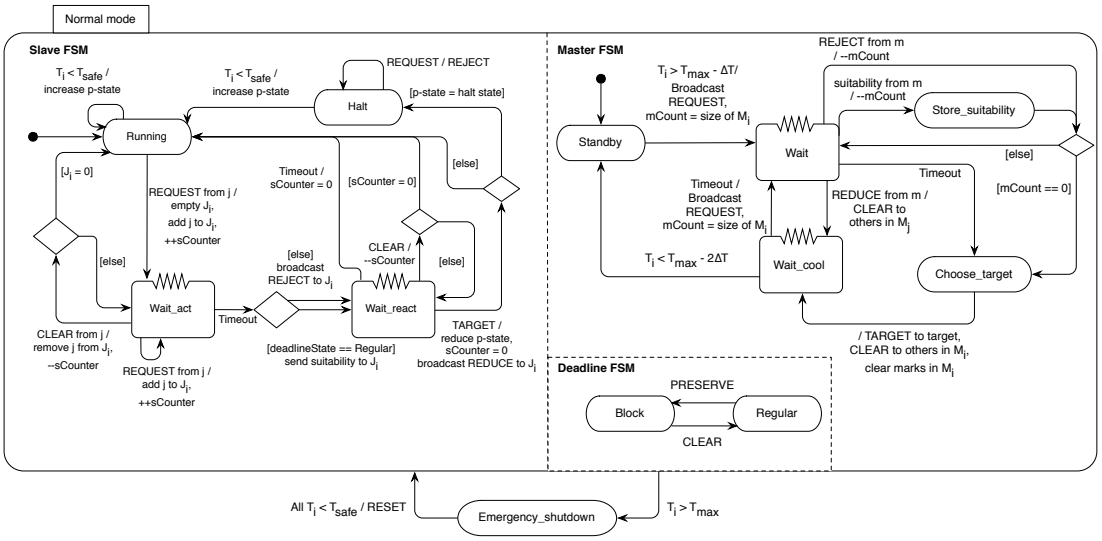

Fig. 2: Distributed thermal controller (Running on each core i)

communication overhead, we limit the information exchange to the cores which are highly thermally correlated. Based on equation (6), we define that core i is *thermally correlated* with j if

$$B_{ji} > c_{th} \tag{7}$$

where c_{th} is the *thermal correlation threshold*. If core i and j are thermally correlated, it means that changing the power and temperature of core i can affect the temperature of core j noticeably. We also define for every core i, a list M_i where M_i is the list of cores that are thermally correlated with i. In the following discussion, when core i broadcasts a signal, it means the signal is broadcast only to M_i.

To prevent significant performance loss, decisions on the power states of the cores should be carefully evaluated before being applied to the system. For example, suppose two neighboring cores run high priority tasks. Two possible solutions are to lower the frequencies of both cores, or lower the frequency of a third core which may improve the temperature of both cores with lower overall performance loss. To quantify potential effects of such decisions, we define a metric *thermal management suitability* of a core i as follows:

$$\begin{aligned} suitability(i) = \quad & (1 - priority_i)(F_i - F_i')/F_{imax} \\ & * \textstyle\sum_{j \in J_i} tempImp(i,j) \end{aligned} \tag{8}$$

where F_i and F_i' are the current frequency and target frequency of core i respectively (whose difference reflect the performance impact of slowing down the core), and J_i is list of the cores in thermal emergency which are thermally correlated with i. *Thermal management suitability* metric allows us to quantify the trade-off between the performance loss and overall temperature benefits that switching the power state of a core i can cause on the cores in set J_i. Each core in thermal emergency asks for $suitability$ information of the thermally correlated cores in order to choose one of them whose power state change could benefit more in terms of temperature with lower performance cost.

B. Distributed Thermal Controller

The distributed thermal controller of *DistriTherm* algorithm is shown in figure 2 in StateChart diagram format. Please note that in figure 2, the signals are expressed all in uppercase, while only the first character in an action's name is uppercase. As shown in the figure, the controller operates in two main modes: *normal mode* and *emergency shutdown mode*.

The controller operates in the *normal mode* until the core temperature exceeds the critical temperature T_{max}. In this case, the controller switches to *emergency shutdown* mode. In emergency shutdown mode, the *DistriTherm* controller broadcasts a signal and forces all cores to switch to sleep mode, and resets them back to default states when every core's temperature reaches a safe temperature T_{safe}.

In normal mode, there are three *finite state machines* (FSM) operating concurrently: Deadline FSM, Master FSM and Slave FSM. The deadline FSM makes sure that when a lower power state cannot meet the deadline, the core stays at its current power state as long as possible. The master FSM engages when the core is in thermal emergency. It requests the other cores to contribute in lowering its temperature. The slave FSM engages when requests are received from other master cores. Here these FSMs are explained in more detail.

Deadline FSM: This FSM keeps track of the performance requirement of the core. If it is in *Block* state, it means lowering power state of that core will cause its deadline to be missed. Please note that in this work, each core has a finite number of frequencies that it can switch to, and the core power state is an integer that indicates which level of the frequency it is using. Higher power state corresponds to higher frequency. Power state 0 corresponds to core's sleep mode with smallest power consumption. To minimize the number of deadline misses, we distinguish the cores whose power state change might cause deadline miss and protect these cores from being switched to a lower power state due to thermal emergencies. This is done by our *deadline preserving algorithm* (DPA) which is shown in Algorithm 1.

At each scheduling tick, the remained slack for each deadline constrained task is estimated. Based on the current task

Algorithm 1 Deadline preserving algorithm

$slack = deadline - time\_elapsed$
$remain = (1 - progress) * base\_length * F_{max}/F_{cur-1} * \alpha$
if $(slack < remain + switching\_overhead)$ **then**
  if $STATE(deadline\_FSM) = Regular$ **then**
    send PRESERVE to deadline_FSM
  end if
else
  if $STATE(deadline\_FSM) = Block$ **then**
    send CLEAR to deadline_FSM
  end if
end if

progress, the remaining length of the task for the lower power state is estimated as well. A value between 0 and 1 is used to represent the progress of a task, where 0 means no instruction has been committed yet, and 1 means all the instructions of the task are completely committed. The execution time of a task on the core at its highest frequency is called *base length*. A linear model is used here to predict the task progress when the frequency is scaled. The scale factor α in Algorithm 1 can be obtained from pre-characterization of the task. If estimated remaining length of the task plus the power state switching overhead) is larger than the slack, it means that switching to a lower power state will lead to a deadline miss. Therefore, in this case a signal PRESERVE switches the deadline FSM to *Block* state to prevent the core from going to a lower power state. If the estimated remaining length is shorter than slack, this means it is safe to switch to a lower power state without causing a deadline miss. In this case, a CLEAR signal resets the state of the deadline FSM to *Regular* allowing slave FSM to lower power state of the core.

Sometimes the core might need to lower its power state despite the state of deadline FSM being set to $Block$. This could happen when no other core is able to reduce its frequency to prevent thermal emergency, as in the case where all requests from a $master$ core are rejected. Therefore, while this deadline preserving algorithm tries to prevent such cases, it might not able to guaranty meeting all the deadlines due to resource constraints and thermal requirements, which will be discussed in the next section.

Master FSM: In the case of thermal emergency (temperature of the core exceeding $(T_{max} - \Delta T)$), this FSM broadcasts REQUEST to every thermally correlated core and waits for every thermally correlated core to send its *suitability* or REJECT signal. ΔT is set to be 2°C to avoid too frequent power state switches. As it receives the incoming *suitability* information, the master FSM marks the *suitability* in list M_i, or 0 if the incoming signal is REJECT rather than $suitability$.

Among all the thermally correlated cores, the master FSM needs to choose a core with the highest $suitability$ as a *target core*, and send a signal TARGET to request the target core to reduce its power state. In the case that all the thermally correlated cores reject the request, the master controller has no choice but to reduce its power state. After the target core is chosen, the master controller switches into *Wait_cool* state and waits until the temperature drops below $(T_{max} - 2\Delta T)$ to avoid *Master FSM* switching frequently between *Standby* and *Wait* state if the workload changes. . If the temperature does not drop below $(T_{max} - 2\Delta T)$ before timeout, the master controller broadcasts new REQUEST to further reduce its temperature.

Slave FSM: This FSM handles the requests from master FSMs. When REQUEST signal arrives, the slave FSM puts the requesting core into list J_i and switches to *Wait_act* state. Before the timeout happens in *Wait_act* state, slave FSM keeps accepting requests from other master FSMs. This *Wait_act* timer is a very important parameter in this distributed algorithm. It allows the controller to wait for *suitability* information from all of the thermally related cores. This enables the controller to choose the best candidate among these cores. If some of the cores cannot respond in time, the controller goes ahead and makes its decision assuming that the core cannot contribute. The wait length of *Wait_act* timer should be chosen according to the thermal time constant of the core. If the timer length is too short, the slave FSM cannot capture all the requests in one single iteration which results in less optimal solutions. If the timer length is longer than the time constant, the slave FSM cannot respond on time and the temperature might significantly increase before an action is taken.

Once timeout happens in *Wait_act* state, the slave FSM first reads the state of deadline FSM in the same core. If the state of deadline FSM is $Regular$, the slave FSM computes and broadcasts *suitability* to cores in list J_i, or broadcasts REJECT signal to cores in list J_i if the state is $Block$. Deadline FSM changes its state according to the performance needs of workloads.

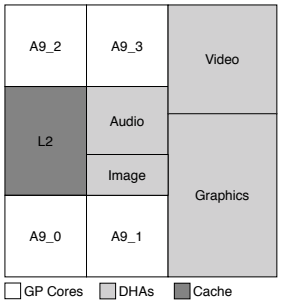

Fig. 3: Floorplan of the MPSoC used in our experiments

IV. Experimental Results

We have built a scheduling system on top of HotSpot temperature simulation tool [21] in order to evaluate our distributed temperature aware scheduling of tasks on MPSoCs, and compare it to state of the art algorithms. To integrate power and performance data for different types of cores, this system is a modular and allows easy integration of data from different sources. It decouples the overall system simulation from the core-level performance and power simulations. The performance and power data can be collected offline from a simulator or from real measurements. This enables extension of the set of cores simulated in the heterogeneous MPSoC.

We use M5 Simulator [7] which is integrated with McPAT power model [19] to get power/performance data for GP cores. These cores are based on a simple out of order architecture similar to the ARM Cortex A9 [20] which is used in embedded platforms such as TI's OMAP5 and NVIDIA's TEGRA 2. Various SPEC2000 benchmarks are simulated on M5 with the power model from [19].

Core	Area (mm^2)	Peak power (W)	Frequencies@Voltages (MHz@V)
A9	9	0.5	600@0.85, 900@0.9, 1200@1.0
audio	7.5	0.4	600@0.85, 900@1.0
video	16	0.8	400@0.85, 600@0.9 , 900@1.0
graphics	24	1.1	400@0.85, 600@0.9, 900@1.0
image	4.5	0.25	600@0.85, 900@1.0

TABLE I: System configuration

For DHAs, we use various coder, decoder and DSP architectures. To represent workloads running on high end smart phones, we consider various lengths of video decoding and encoding on the video codec DHA with the power values reported in [16]. We create traces by scaling the execution times of the MediaBench II benchmarks [18] for DSPs.

Table III summarizes the key characteristics of the benchmarks used in this paper. A DHA task is always assigned to its corresponding DHA as shown in figure 3. In our experiments, new instances of a DHA task type are issued periodically. We assume that the deadline of a task is equal to its period. Once an instance of a task misses its deadline, a new instance is created, and the previous instance is dropped. The number in the execution time column in table III, is the execution time of each DHA task at the highest frequency of its corresponding DHA. In our experiments, all GP cores are always running GP tasks unless they are stopped due to thermal issues.

As our metric for performance of DHAs, we use *deadline miss rate* which is calculated by dividing the number of deadline misses by the number of issued tasks. Because general purpose tasks do not have deadlines, average *instruction per second (IPS)* is used as their performance measure.

Power states of the cores and their corresponding voltage and frequencies are reported in I. For switching between various voltage and frequency settings, we assume an overhead of 100μs [14]. For leakage power and its dependence on temperature, we use the leakage model introduced in [12] with the same constants used in the paper for 65 nm. Power state adjustments for temperature management are done using DVFS and DPM mechanisms. In DVFS, the cores have several voltage/frequency settings which provide various operating points with various power/performance choices. In DPM, there are only two power states. The core is either running at its highest frequency or is turned off with a switching overhead 100 μs in our setting.

Ambient	45°C
Convection resistance	0.311 K/W
Chip thickness	$0.15mm$
Chip footprint	$9 \times 9mm^2$
Freq. switching overhead	$100\mu s$
Sleep/wake overhead	$100\mu s$
Wait_act timer	$1ms$
Scheduling interval	$1ms$
Sampling interval	$1ms$
c_{th}	0.2 K/W
T_{max}	107°C
ΔT	2°C
T_{safe}	100°C

TABLE II: Evaluation setup

The MPSoC used in our experiments is assumed to be implemented in 65 nm technology. The floorplan of the heterogeneous MPSoC used in our experiments is shown in the figure 3. The areas of these cores are derived from the published photos of the dies after subtracting the area occupied by I/O pads, interconnection wires, interface units, L2 cache, etc. The leakage model in [12] is used to account for the temperature dependence of the leakage. We use the same constants mentioned in [12] for 65 nm. HotSpot Version 5 [21] is used with a sampling interval of 100us which provides sufficient accuracy with reasonable overhead.

Task	Core type	Priority	Period (ms)	Exec time @ F_{max} (ms)
gcc	GP core	1.0	N/A	N/A
mcf	GP core	1.0	N/A	N/A
crafty	GP core	0.5	N/A	N/A
parser	GP core	0.5	N/A	N/A
audio	DHA	1.0	10	8.95
video	DHA	1.0	33	19.52
graphics	DHA	1.0	50	28.80
image	DHA	1.0	12	7.45

TABLE III: Core configuration and workload

A. Results

To show the benefits of our proposed thermal management technique, we compared our distributed thermal management technique against following thermal management techniques which are widely used in embedded MPSoCs.

Local TM (Local thermal management): Each core uses a simple thermal management mechanism implemented in a hardware controller. Whenever temperature reaches ($T_{max} - \Delta T$), it reduces its power state by one step, and increases it also by one step when the temperature drops below T_{safe}.

Deadline TM (Deadline driven thermal management): The OS scheduler gathers temperature information of all GP cores in the system, and execute a proactive thermal management policy at every scheduling tick. However, OS has no control over DHA's power states, thus, all DHAs run at the highest power state such that the deadlines can always be met. In this technique we set the scheduling interval to be $1ms$ asn shown in table II.

DistriTherm: Our distributed thermal management uses the configuration shown in table II.

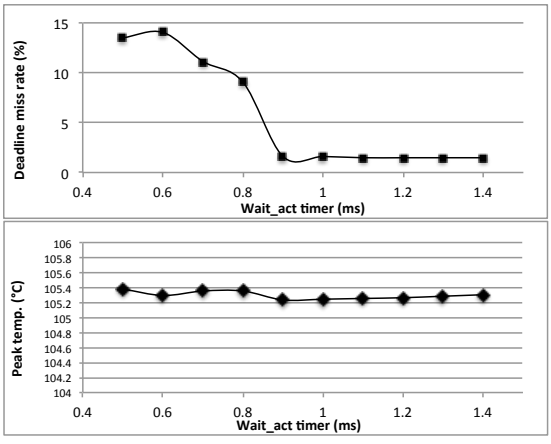

Fig. 4: Wait_act timer trade-off

One of the most important parameters in our technique is the *Wait_act* timer. Longer *Wait_act* time allows the master to make a better decision while it also increases the delay

Policy	Miss rate (%) for DHA	Avg. perf. loss (%) for GP	Peak temp. (°C)	Energy (J)
Local TM	48.77	baseline	105.00	2.95
Deadline TM	0.00	39.23	113.67	3.17
DistriTherm	1.61	27.67	105.25	2.90

TABLE IV: Performance comparison

of responding to thermal emergencies which might lead to higher maximum temperature. Figure 4 compares the effect of setting *Wait_act* timer to a range of different values when the controller operates in the *normal mode*. As this figure shows, miss rate of the system constantly decreases as *Wait_act* timer increases, but the reduction is negligible after $1ms$, while the peak temperature starts increasing after $1ms$. For the rest of the results, we use $1ms$ as the wait time for *Wait_act* timer.

The performance loss of GP cores is measured by comparing the IPS observed in the experiment (IPS_{EXP}), and IPS when the GP core runs at its highest frequency (IPS_{MAX}). Performance loss is calculated by the following equation

$$\text{Perf. loss} = \sum_i \text{IPS}_{\text{EXP}_i}/\text{IPS}_{\text{MAX}_i} \times priority_i \qquad (9)$$

As our baseline, we use *Local TM* as a technique which can always keep the temperature below thermal threshold. In table IV, other techniques are compared in term of performance to *Local TM* across the same benchmarks.

Table IV compares different techniques in terms of the average performance loss across various combinations of GP core workloads, along with average deadline miss rate of different DHA tasks. Our experiments show that *Deadline TM* results in the highest peak temperature $113.67°\text{C}$ which is significantly higher than the critical temperature. *Deadline TM* which does not have control over DHAs, consumes the highest energy among the techniques we compare. This shows that in modern MPSoC designs where DHAs consume a great proportion of total power, controlling the power state of DHAs is can significantly affect power, energy and thermal profile .

Average deadline miss rate of DHAs is 1.61% using *DistriTherm*, which is an order of magnitude lower than *Local TM*. Also, our technique successfully satisfies the temperature constraint while *Deadline TM* fails. On the other hand, it increases the performance loss of general purpose tasks by 27.67% compared to *Local TM*. This is because of higher priority of tasks running on DHAs compared to general purpose tasks as described before. Due to this higher priority, in the case of thermal emergencies, *DistriTherm* sacrifices performance of lower priority general purpose tasks selectively in order to reduce the deadline misses of higher priority DHA tasks.

V. Conclusion

In this work we present a scalable distributed thermal management technique for a mixture of workloads consisting of deadline driven and general purpose tasks. We first quantify the thermal correlation between the cores. Then, using the correlation, when temperature reaches a threshold, *DistriTherm* controllers of the thermally correlated cores communicate to determine the best core to slow down. This is the core whose frequency reduction can benefit the other cores more in terms of temperature, while minimizing deadline misses and throughput loss. The experiments show that our *DistriTherm* technique can successfully reduce the deadline miss rate by 47.16% on average while limiting the peak temperature as compared to completely localized thermal management.

References

[1] M. Pedram and S. Nazarian, "Thermal Modeling, Analysis, and Management in VLSI Circuits: Principles and Methods", *Proc. of the IEEE* 94(8) (2006), pp. 1487–1501.

[2] Y. Oasais, F. R. Yu and M. St-Hilaire, "Thermal Management of Biosensor Networks", *IEEE Consumer Communications and Networking Conference*, 2010, pp. 1–5.

[3] J. Donald and M. Martonosi, "Techniques for Multicore Thermal Management: Classification and New Exploration", *Proc. Intl. Symp. on Computer Architecture*, 2006, pp. 78–88.

[4] M. Gomaa, M. D. Powell and T. N. Vijaykumar, "Heat-and-Run: Leveraging STM and CMP to Manage Power Density Through the Operating System", *SIGARCH Computer Architecture News* 32(5) (2004), pp. 260–270.

[5] S. Ghiasi and D. Grunwald, "Design Choices for Thermal Control in Dual-Core Processors", *Proc. Workshop on Complexity-Effective Design*, 2004, pp. 260–270.

[6] S. Sharifi, A. Coskun and T. S. Rosing, "Hybrid dynamic energy and thermal management in heterogeneous embedded multiprocessor SoCs", *Proc. Asia and South Pacific Design Automation Conference*, 2010.

[7] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt. "The m5 simulator: Modeling networked systems", *IEEE Micro*, 26(4) (2006), pp. 52–60.

[8] S. Sharifi and T. S. Rosing, "Package-Aware Scheduling of Embedded Workloads for Temperature and Energy Management on Heterogeneous MPSoCs", *Proc. Intl. Conf. on Computer Design*, 2010, pp. 521–527.

[9] Y. Ge, P. Malani and Q. Qiu, "Distributed Task Migration for Thermal Management in Many-Core Systems", *Proc. Design Automation Conference*, 2010, pp. 579–584.

[10] K. Skadron, K. Sankaranarayanan, S. Velusamy, D. Tarjan, M. R. Stan and W. Huang, "Temperature-Aware Microarchitecture: Modeling and Implementation", *ACM Trans. on Architecture and Code Optimization* 1 (2004), pp. 94–125.

[11] NVIDIA Tegra2, http://forum.beyond3d.com/showthread.php?p=1392979.

[12] S. Heo, K. Barr, and K. Asanovic. "Reducing power density through activity migration", *Int'l Symp. on Low Power Electronic Design*, pp. 217-222, 2003.

[13] A. K. Coskun and T. S. Rosing and Keith Whisnant. "Temperature Aware Task Scheduling in MPSoCs," *Proc. Design Autom. and Test in Europe (DATE)*, pp. 1659–1664, 2007.

[14] M. Ware, K. Rajamani, M. Floyd, B. Brock, et al, "Architecting for Power Management: The IBM POWER7 Approach", *Proc. Int'l Symp. on High Performance Computer Architecture*, pp. 1–11, 2010.

[15] R. Ayoub and T. Rosing, "Predict and Act: Dynamic Thermal Management for Multi-Core Processors", *Int'l Symp. on Low Power Electronic Design*, pp. 99–104, 2009.

[16] K. Iwata, S. Mochizuki, T. Shibayama, F. Izuhara, et al, "A 256mw full-hd h.264 high-profile codec featuring dual macroblock-pipeline architecture in 65nm cmos", *In IEEE Symposium on VLSI Circuits, 2008.*

[17] Q. Wu, P. Juangm, M. Martonosi, and D. W. Clark, "Voltage and Frequency Control with Adaptive Reaction Time in Multiple-Clock-Domain Processors", *Proc. Int'l Symp. on High Performance Computer Architecture*, pp. 178–189, 2005.

[18] J. E. Fritts, F. W. Steiling, J. A. Tucek, and W. Wolf. "Mediabench ii video: Expediting the next generation of video systems research. Microprocess", *Microsyst.*, 33, June 2009.

[19] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, et al, "Mcpat: an integrated power, area, and timing modeling framework for multicore and manycore architectures", *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, 2009.

[20] ARM. Cortex A9 processor, 2011. http://www.arm.com/products/processors/cortex-a/cortex-a9.php.

[21] Hotspot temperature modeling tool, 2011. http://lava.cs.virginia.edu/HotSpot/.