# Temperature-aware Scheduling for Embedded Heterogeneous MPSoCs with Special Purpose IP Cores

Shervin Sharifi
Computer Science and Engineering Dept.
University of California, San Diego. La Jolla, CA
shervin@ucsd.edu

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

Tajana Simunic Rosing
Computer Science and Engineering Dept.
University of California, San Diego. La Jolla, CA
tajana@ucsd.edu

## ABSTRACT

Many embedded heterogeneous MPSoCs integrate general purpose cores along with special purpose cores. The power states of these cores are usually controlled by an internal hardware controller rather than the central operating system. In this paper, we propose a thermal management technique which reduces the performance penalty of central thermal management by considering these special purpose cores and the performance requirements of the tasks running on them. Our experimental results show that for the workloads with high priority special purpose tasks, our technique can reduce the occurence of thermal violations by at least 3X while improving the weighted execution time by up to 24%.

## Keywords

Thermal management, embedded MPSoCs, heterogeneous MPSoC, special purpose core

## 1. INTRODUCTION

As technology scales, decreasing device dimensions and increasing power densities result in higher temperatures. Operating at higher temperature degrades the reliability of the system, increases leakage power, causes performance degradation and leads to higher cooling and packaging costs [12]. These issues have made temperature one of the major factors which must be considered and addressed in design, manufacturing and test. Many embedded systems work under diverse and undesirable conditions. For example, wireless base stations may be deployed outdoors in harsh environmental conditions with ambient temperature exceeding 80°C [15]. Cell phones also must be able to operate under a wide range of ambient temperatures without the benefit of more sophisticated packaging due to cost and space considerations. Thermal and power management techniques are the key solutions for systems such as these.

Power density and thermal limitations have been among the most important reasons of prevalence of multi-processor SoCs (MPSoCs). MPSoCs provide higher performance within a specific power budget and thermal envelope. Heterogeneous MPSoCs provide even better performance and power trade-off by integrating cores of various types on the same die.



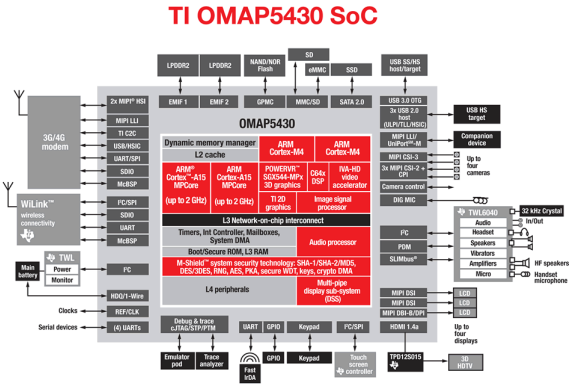


**Figure 1: TI OMAP5430 heterogeneous SoC [2]**

Integrating cores with the same instruction set architecture but operating at various power and performance points is one form of heterogeneity. The heterogeneity can also be provided by addition of diverse special purpose cores or accelerators which are specifically designed for certain applications. Existing examples of such embedded heterogeneous MPSoCs are Qualcomm's Snapdragon platform, or Texas Instruments' OMAP platform. Qualcomm's latest Snapdragon platform includes two general purpose cores also called application cores (ARM Cortex A9), a DSP, a multimedia co-processor and a graphics processing unit. Texas Instrument's OMAP5 platform 1 includes four general purpose cores (two ARM Cortex A15 and two ARM Cortex M4 cores), a DSP, 3D and 2D graphics accelerators, video accelerator, an image signal processor and an audio processor, plus some other accelerators and coprocessors.

Special purpose cores are specifically designed and optimized for the special tasks they are supposed to do. They may be third party IP (intellectual property). Many of them are hard IP cores which are optimized for certain technologies and described in low-level physical descriptions and could not be modified at the time of integration. Such components typically do not run general purpose operating systems (OS). Also they are not under full control of the central operating system. One way that the OS and general purpose cores communicate with these special purpose cores is through interrupts. They are notified by an interrupt when a job is available for them, and as soon as they finish processing their job, they notify the operating system through another interrupt. During this time interval, the operating system has little control over the special purpose cores

and their power states. In some cases, these cores might have autonomous control of their power states for thermal management purposes, but usually these controls are not coordinated with the other cores in the system. This creates challenges for thermal management of the overall MPSoCs.

In this paper we propose a technique specifically designed for thermal management of MPSoCs consisting of both general and special purpose cores where power states of the special purpose cores are not controlled by the general purpose OS. We call our technique HTHS (Hybrid Thermal management for Heterogeneous SoCs) as the system is composed of heterogeneous cores with hybrid combination of autonomous (in special purpose cores) and central (in general purpose cores) control of power states for thermal management. At each scheduling tick, our algorithm makes a sequence of decisions on assigning new tasks to the cores, adjusting the power states of the cores and migrating the tasks among the general purpose cores. Then the outcomes of these decisions are applied to the MPSoC. The details of how these decisions are made are described in Section 3. Our results presented in section 4 show quantitative benefits relative to the state of the art.

## 2. RELATED WORK

Task scheduling under thermal constraints on an MPSoC is in general an NP-hard problem [18]. The space of the solutions is huge due to the myriad possibilities for frequency settings, task assignment of the cores and task start times.

Various dynamic thermal management techniques have been proposed for MPSoCs. Most of these solutions (e.g. [5] and [8]) address thermal management of homogeneous MPSoCs and assume the operating system has full control on the power states of all the cores.

Techniques have been proposed for thermal management of heterogeneous MPSoCs as well. In [7], an approach is proposed for asymmetric dual core designs where the workload is off-loaded to the low power core in order to reduce the occurrence of thermal emergencies with low performance impact. In [13], a centralized approach is proposed for temperature and energy management in heterogeneous MPSoCs. This approach is called hybrid in the sense that the scheduler has two different modes which are selected based on the workload utilization. At low and moderate utilization, energy is minimized by workload scheduling and disabling the cores which are not needed. At higher utilizations where thermal issues are more likely to happen, a temperature balancing approach is taken using task assignment and DVFS. The work in [14] proposes an algorithm for scheduling embedded workloads on a heterogeneous MPSoC where at each scheduling tick, based on the performance requirements of the workload and thermal state of the cores, frequencies and tasks are assigned to the cores. All of these approaches assume centralized control of the operating system over all of the cores.

Some techniques have been also proposed for distributed thermal management of multi-core MPSoCs. In [17], it is assumed 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 mentioned before, in the case of heterogeneous MPSoCs with special purpose cores, some of the cores are not under full control of the central operating system. In some cases, they have their own temperature control which may be implemented in the hardware. On the other hand, the central operating system can control the power states of the general purpose cores to manage the temperature. This mix of central and distributed controls makes the holistic thermal management of such systems more challenging. To the best of our knowledge, there is no previous work addressing this problem.

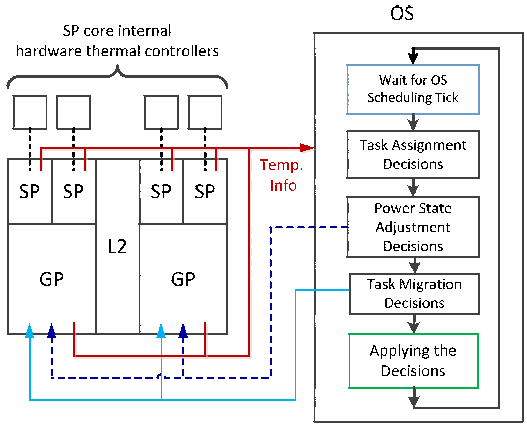


**Figure 2: Architecture of the HTHS thermal management system**

We propose a thermal management technique which coordinates central and distributed thermal controls. At every scheduling tick, our algorithm makes a sequence of decisions on assigning power states(voltage/frequency settings) and tasks to the cores. First, it makes decisions on assigning the newly arrived tasks to the available cores. Then, based on the current tasks in the system and the current temperatures, it decides if some cores should change their power states. Then the decision is made on how to migrate the tasks among the general purpose cores such that the overall performance is improved. After all these decisions are made, they are applied to the MPSoC. At the next scheduling tick, the decisions are made and applied to the MPSoC again.

The next section describes the details of our algorithm.

## 3. THE HTHS TECHNIQUE

Hybrid thermal management for heterogeneous SoCs(HTHS) combines task assignment and migration with core power state adjustments in order to control the temperature of heterogeneous MPSoCs which integrate both *general purpose* (GP) and *special purpose* (SP) cores. The goal is to minimize the performance penalty due to thermal constraints. This performance penalty happens when some cores are slowed down or stopped to avoid thermal emergencies. The penalty would be higher if these cores are running high priority tasks. To minimize the penalty, cores running higher priority tasks must be kept at highest possible frequencies. This process is usually controlled by the OS in a centralized manner. In the heterogeneous MPSoCs where the power states of the SP cores are controlled by autonomous hardware controllers and cannot be controlled by the OS, the problem is more complicated. For example, in such systems, if a special purpose task is critical to the overall progress, it should run as fast as possible on its corresponding SP core, which suggests that the core's hardware controller should be kept from slowing down that core. Lack of full operating system control on the power state of SP cores creates new challenges in the thermal management of these systems. Our technique addresses these challenges.

At every scheduling tick, HTHS goes through three *decision phases* in order. In the first phase, decisions are made about assigning new tasks to the available cores. In the second phase of decisions, based on the current temperatures

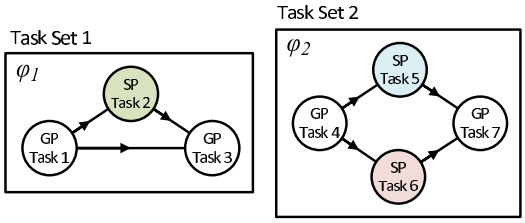


**Figure 3: Workload model**

of the cores and the current tasks in the system it is decided which cores need to change their power states. The goal is to control the temperature such that the cores running high priority tasks do not need to slow down or stop. The last phase decides about migrating the tasks among the GP cores in order to achieve higher performance and more balanced temperature. The main goal of this phase is to run high priority general purpose tasks on the GP cores running at the high frequencies and also to move the high priority tasks to the cooler areas of the die so that they are less likely to be slowed down or stopped due to thermal emergencies. When all of these decisions are made, then the decision results are actually applied to the MPSoC by setting the GP cores to the decided power states and migrating general purpose tasks as decided. The rest of the section provides more details on our system and the HTHS technique.

## 3.1 System description

Figure 2 shows the system diagram and interaction between components. Dashed lines connected to the cores show the control of their power state. The blue solid lines represent the control of the OS on the task migration of the cores. The red solid line represents temperature information sent to the OS. As the figure shows, the OS controls the power states of the GP cores while the power states of the SP cores are controlled by their own hardware thermal controller. Task migration is controlled by the OS and is possible only among GP cores. Also, the OS controls initial assignment of the tasks to all the cores. We assume a minimal communication among these cores and the operating system. HTHS relies on receiving the temperature information from the SP and GP cores.

All decisions in the operating system are made at scheduling ticks. The central operating system controls the overall temperature of the MPSoC by assigning tasks to the cores, migrating the tasks among the GP cores, and controlling the power states of the GP cores even though it is not able to control power states of the SP cores.

We assume the workload running on the heterogeneous MPSoC consists of various task sets where each task set has tasks with dependencies among them. This is shown in figure 3. Each task set has a priority associated with it. The dependencies among its tasks are described by a directed acyclic graph where each node represents a task. A task may be a special purpose task which must run on its own SP core (nodes in color in figure 3) or general tasks which could run on any of the GP cores (nodes in white).

There are two temperature thresholds $T_{mild}$ and $T_{severe}$ where $T_{mild} < T_{severe}$. If the core temperature exceeds threshold $T_{mild}$, it goes to *thermal alert mode* which means it is under a mild thermal emergency. A core in alert mode is not allowed to go to a higher power state until its temperature again reaches below a safe lower level ($T_{safe}$) and its alert mode is disabled. When the temperature of a core exceeds $T_{severe}$, the core is considered to be in severe thermal emergency. If its power state can be controlled by the operating system, it will be slowed down or stopped.

$T_{severe}$, $T_{mild}$ and $T_{safe}$ depend on the power consumption and the thermal characteristics of the chip such as the floorplan and chip material. In our experiments, maximum allowed temperature ($T_{max}$) is 105°C. $T_{severe}$ is set to around 0.5 °C less than $T_{max}$ because this is about the amount that a core temperature can change within a scheduling tick when it is close to $T_{max}$. $T_{mild}$ is set to 1°C below $T_{severe}$ and $T_{safe}$ is set about 2°C below $T_{mild}$. These values can be changed based on the desired trade-off between performance and the frequency of power state switching in the cores.

The next three subsections explain the three phases of decisions in our algorithm.

## 3.2 New task selection

This phase decides the assignment of new tasks to the available cores. It checks the existing tasks in the system in decreasing order of priorities. If the next task is a special purpose task, it is assigned to the corresponding core if it is available. For general purpose tasks, it would assign the task to one of the available GPs (if any) which has the lowest temperature. Algorithm 1 explains the process of selection of the new tasks in more detail. This algorithm runs in the OS at each scheduling tick, as shown in figure 2. $CoreType(\lambda)$ is the type of the core able to run task $\lambda$.

**Algorithm 1** New task selection

```
1: Λ ⇐ ∅
2: Φ ⇐ newly arrived tasks
3: Ω ⇐ currently available GP cores
4: while ( (Φ ≠ ∅) and (Ω ≠ ∅) ) do
5:    λ = the highest priority task in Φ
6:    κ = CoreType(λ)
7:    if (κ ∈ SP ) then
8:       assign λ to the core of type κ if available
9:    else
10:      c ⇐ The coolest core in Ω (if any)
11:      Mark λ to be assigned to c
12:   end if
13:   add λ to Λ
14:   remove λ from Φ and c from Ω
15: end while
```

## 3.3 Power state adjustments

When the temperatures approach critical thresholds, some of the cores might need to switch to lower power states. In such cases, the lack of control over SP cores might hurt the performance in two different ways. If a SP core running a high priority task might automatically slow down or stop due to a thermal emergency. Another case occurs when a GP core *gp* which runs a high priority task is adjacent to a SP core *sp* running a low priority task. If there is a thermal emergency happening on *gp*, the operating system is not able to lower the power of *sp* and is forced to adjust the power state of *gp* to keep the temperature from exceeding the threshold. Our technique prevents these cases by selective adjustment of power states. This power adjustment is triggered when the temperature of the cores approach critical levels and the cores to to thermal alert mode. When necessary, it selectively slows down/stops the cores running lower priority tasks to eliminate the thermal emergencies on the cores running high priority tasks. As figure 2 shows decisions regarding selective power adjustments are made at each scheduling tick after the new tasks are selected. Algorithm 2 describes our power state adjustment in more detail. In this algorithm, $alert(c)$ is a flag showing if core $c$ is in the alert mode, $N(c)$ is the set of the neighbors of core $c$ and $Task(c)$ is the task running on core $c$. $\phi(\tau)$ is the priority of task $\tau$ (which is between 0 and 1) and $Core(\tau)$ is the core currently running task $\tau$. $GP$ and $SP$ are the set of all gen-

eral purpose and special purpose cores respectively and $M$ is the set of cores in the alert mode. $L$ and $H$ are respectively the sets of cores marked for switching to lower and higher power states.

The cores are checked in the decreasing order of priority of the task they are running. If a core's thermal alert mode is enabled, its hottest GP neighbor is marked for lowering the power state unless one of its neighbors has been already marked. When a SP core *sp* is in alert mode, before marking its neighbors, we check to see if there is any GP core running a task with a priority lower than that of *sp*. The reason is that if all of the GP cores are running task with higher priority than *sp*, we don't want to slow them down. Instead, *sp* is left to be slowed down by its hardware controller.

This gradual adjustments in the power state of the neighbors might not be enough to resolve the thermal emergencies and the core's temperature might exceed the $T_{severe}$ by the next scheduling tick. In that case, if the core with severe thermal emergency is under the control of the operating system, operating system slows it down (or stops it). Otherwise, if this is a SP core with its own thermal control, the operating system slows down its neighbors to prevent activation of the autonomous thermal control. The hottest neighbor is slowed down first as the hottest core is usually the most effective one in heating up the neighbor cores.

**Algorithm 2** Power state adjustments

```
1: M, L, H ⇐ ∅ (unmark all of the cores)
2: ϒ ⇐ tasks currently running on the cores
3: ϒ ⇐ ϒ ⋃ Λ ( Λ from task assignment phase)
4: while (ϒ ≠ ∅) do
5:    τ ⇐ the highest priority task in ϒ
6:    c ⇐ core(τ)
7:    ϒ ⇐ ϒ − τ (remove τ from ϒ)
8:    if (T(c) > T_mild) then
9:       alert(c) ⇐ 1, M ⇐ M + c
10:      if (T(c) ≥ T_severe) then
11:         if (c ∈ GP) then
12:            L ⇐ L + c (mark c for switching to a lower
               power state)
13:         end if
14:      end if
15:      if ( (c ∉ L) and (N(c) ⋂ L = ∅) ) then
16:         γ ⇐ the lowest priority task running on GP cores
17:         if ( φ(γ) < φ(τ)) then
18:            GPN ⇐ GP ⋂ N(c)
19:            n ⇐ the hottest core in GPN
20:            L ⇐ L + n (mark n for switching to a lower
               power state)
21:         end if
22:      end if
23:   else {T(c) <= T_mild}
24:      if (T(c) < T_safe) then
25:         alert(c) ⇐ 0
26:         if (N(c) ⋂ M = ∅) then
27:            H ⇐ H + c (mark c for switching to a higher
               power state)
28:         end if
29:      end if
30:   end if
31: end while
```

If the slowed down core has been running a high priority task, the task would be moved to a higher frequency processor in the task migration phase. If adjusting the power state of the neighbors by OS is not enough to control the temperature of a SP core, eventually the autonomous thermal control of the core slows it down or stops it.

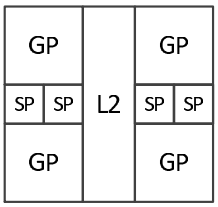


**Figure 4: Floorplan of the heterogeneous MPSoC**

## 3.4 Task migration

In this phase, decisions are made on how to migrate the general purpose tasks among the GP cores to achieve a better performance and temperature profile. The goal is to move the high priority tasks to the cores running at higher frequencies. While doing this, it also tries to move the high priority tasks to the cooler areas on the die. The current temperature at a location on the die is used as a proxy for thermal correlation of that location to the currently running cores. By assigning the high priority task to the cooler areas on the die, first we make sure that the core would be less thermally affected by the other running cores and second, it has a larger thermal slack before it reaches the thermal threshold. Algorithm 3 describes our task migration algorithm in more detail.

**Algorithm 3** Task migration

```
1: F ⇐ possible frequencies of the GP cores
2: while (F ≠ ∅) do
3:    f ⇐ highest frequency in F
4:    Θ ⇐ the set of GP cores running at frequency f
5:    while (Θ ≠ ∅) do
6:       c ⇐ the coolest core in Θ
7:       τ ⇐ the highest priority job in ϒ
8:       mark τ to be assigned to c
9:       remove c and τ from Θ and ϒ
10:   end while
11:   remove f from F
12: end while
```

# 4. EXPERIMENTAL RESULTS

In order to evaluate various scheduling techniques, we have built a simulation platform for scheduling which integrates HotSpot [1] for temperature modeling. In this platform, the performance and power simulations of the cores are decoupled from the overall system simulation. The modularity of this framework allows for easy extension of the set of cores simulated in the heterogeneous MPSoC and simplifies integrating data from various simulators or real-life experiments. The performance and power data are collected offline and could be from a simulator or from real measurements. In our setup, we used the M5 Simulator [4] integrated with McPAT power model [11]. To take into account the temperature dependence of the leakage, we use the model introduced in [9] with the same constants mentioned in the paper for 65 nm. We assume the MPSoC we use here is implemented in 65 nm technology. The architecture of the GP cores used in our experiments is an architecture similar to the ARM Cortex A9 [3] which is a simple out of order execution architecture used in embedded platforms such as TI's OMAP4. The area of these cores is derived from the published photos of the dies after subtracting the area occupied by I/O pads, interconnection wires, interface units, L2 cache, and control logic. For SP cores, we use video codecs and DSPs. The floorplan of the heterogeneous MPSoC is shown in the figure 4. As mentioned earlier, HotSpot Version 5 [1] in integrated in our scheduling platform. Hotspot is used with a sampling interval of 1 ms for sufficient accuracy. We use two different mechanism to adjust the power states of the cores. In DVFS, the power state is incremented or decremented

one step at a time. If there is no lower operating voltage and frequency setting, the core would be stopped. In DPM, there are only two power states. The core is either running at its highest frequency or turned off. We assume that the GP cores run at frequency steps 1.5, 1.2 and 0.9 GHz which correspond to voltages 1.0, 0.9 and 0.85. For switching between various voltage and frequency settings, we assume an overhead of 50$\mu$s [16].

The workloads for our evaluations include general purpose tasks and special purpose tasks. For the GP cores we use PARSEC benchmarks on A9 which are simulated on M5 simulator [4] for this architecture with the power model in [11]. To represent a class of workloads run on high end smart phones, we consider various lengths of video decoding and encoding on the video codec cores with the power values reported in [10]. Also for the DSPs, we create traces by scaling the execution times of the MediaBench II benchmarks [6]. The SP tasks cannot migrate, but the general purpose tasks can be migrated among the GP cores with an overhead of 10$\mu$s [8].

We assume that the power states of the SP cores are controlled by their own hardware thermal controllers. When the core's temperature exceeds $T_{severe}$, the controller switches it to a lower power state. The controller switches to a higher power state when the temperature reaches below $T_{safe}$.

We also consider the following thermal management approaches for the GP cores.

- *Base:* There is no thermal management technique applied. All the cores are operating at their highest power state regardless of their temperature.
- *UniPM:* In this technique, if the temperature of any of GP cores reaches $T_{severe}$, then power states of all GP cores is switched to a lower power state. All GP cores will switch to a higher power state when the temperature of all of them is below $T_{safe}$.
- *SoloPM:* In this technique, the power state of each GP core is adjusted independently. Whenever the temperature of a GP core exceeds $T_{severe}$, the operating system switches it to a lower power state, and when its temperature is lower than $T_{safe}$, its power state switches to a higher one.
- *HTHS:* Our proposed thermal management algorithm.

Based on these thermal management approaches for SP and GP cores, we compare 7 different strategies as shown in Table 1 and 2. In both tables, column *AvgGPVio* shows the average number of temperature violations within one second among all GP cores, and *AvgSPVio* is the average number of violation for all SP cores. The last column shows the relative weighted execution times compared to the *Base*.

To evaluate the efficiency of our technique under various conditions, we created two different sets of workloads. $Workload1$ is dominated by relatively high priority special purpose tasks while $Workload2$ is mainly dominated by high priority general purpose tasks. Next subsections describe the results of running these workloads under various strategies.

## 4.1 Temperature Analysis

Figure 5(a) represents the temperature of the SP core with the highest priority when running $Workload1$ under HTHS technique with DVFS. As the figure shows, HTHS successfully avoids the situations in which a high priority SP core exceeds the temperature threshold. It also shows a more stable and smooth temperature profile with less frequent changes. In this figure, $UniPM$ also has a smooth and even lower temperature, but it should be noted that this comes at a higher performance cost, which is shown and explained in the next subsection.

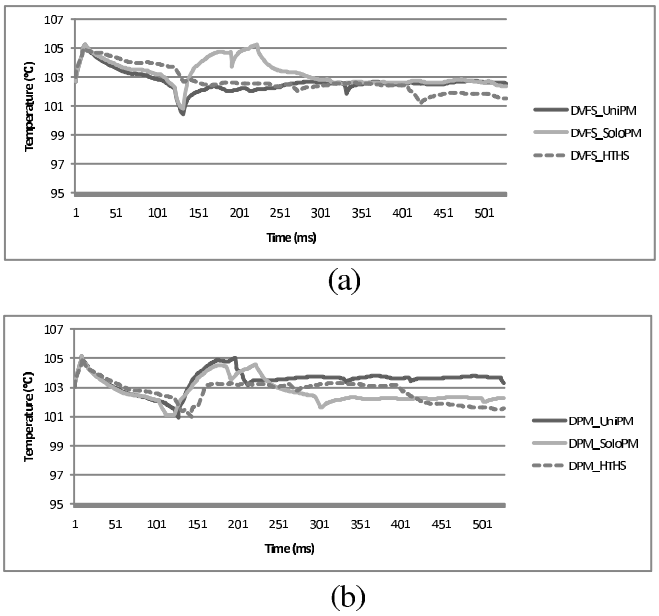


**Figure 5: Temperature of a special purpose cores when (a) SPs doing DVFS and (b) SPs doing DPM**

Figure 5(b) shows the same workload running under HTHS with DPM. Again HTHS technique provides a more smooth and stable, and lower temperature. However, compared to DVFS_ HTHS, it has more abrupt and frequent changes. The reason is that the granularity of power changes in DPM is not as fine as DVFS. These quick and large power changes cause quick temperature changes which sometimes result in oscillation between on and off power state. Therefore, to make the best use of HTHS technique, we need to set the difference between $T_{mild}$ and $T_{severe}$ properly. Also, both Tables 1 and 2 show that our HTHS technique provides the lowest temperature violation frequency for either GP cores or SP cores which also shows our HTHS can provide more stable temperature over time.

## 4.2 Performance Analysis

We use weighted sum of execution times as our performance measure to highlight the importance of meeting requirements of high priority tasks. When the performance of a specific setting is compared against a baseline, the relative performance is defined as

$$\rho = \frac{\sum_{\tau \in \Psi} T_{bl}(\tau).\phi(\tau)}{\sum_{\tau \in \Psi} T_{exec}(\tau).\phi(\tau)} \quad (1)$$

where $\Psi$ is the set of the tasks, $\phi(\tau)$ is priority of task $\tau$, while $T_{bl}(\tau)$ and $T_{exec}(\tau)$ are the execution time of task $\tau$ under baseline and the specific setting respectively.

As both Tables 1 and 2 show, our HTHS technique provides at least 9.1% better performance over all other techniques, and have at most 24.22% performance improvement. Statistics also show that our HTHS technique provides better performance while SP cores are using DPM.

# 5. CONCLUSION

In this work, we proposed a scheduling technique (HTHS) for heterogeneous MPSoCs integrating both general and special purpose cores. Our technique performs task assignment,

**Table 1: Experimental Results on Workload 1**

| SP | GP | AvgGPVio (# of GP violations/sec.) | AvgSPVio (# of SP violations/sec.) | Relative Performance compared to *Base* (%) |
|---|---|---|---|---|
| DVFS | UniPM | 2.5 | 0.875 | 79.34 |
| DVFS | SoloPM | 4.675 | 0.75 | 78.33 |
| DVFS | HTHS | 0.75 | 0.125 | 97.67 |
| DPM | UniPM | 1.875 | 0.375 | 76.87 |
| DPM | SoloPM | 2.875 | 0.375 | 69.23 |
| DPM | HTHS | 0.5 | 0.125 | 85.97 |

**Table 2: Experimental Results on Workload 2**

| SP | GP | AvgGPVio (# of GP violations/sec.) | AvgSPVio (# of SP violations/sec.) | Relative Performance compared to *Base* (%) |
|---|---|---|---|---|
| DVFS | UniPM | 3.09 | 1.08 | 63.54 |
| DVFS | SoloPM | 5.77 | 0.94 | 60.58 |
| DVFS | HTHS | 0.93 | 0.16 | 82.20 |
| DPM | UniPM | 2.37 | 0.47 | 59.75 |
| DPM | SoloPM | 3.58 | 0.47 | 47.83 |
| DPM | HTHS | 0.62 | 0.15 | 72.05 |

power state adjustments and task migration in order to maximize the performance under the thermal constraints. The task assignment and migration phase keeps the high priority tasks on the high frequency cores and also moves the high priority tasks to the cooler areas of the core. This phase results in better performance and lower temperature, but at high utilizations when the temperatures are higher, these phases might not be able to resolve all emergencies by itself. Such cases are addressed by selectively switching the cores to lower power states to avoid exceeding thermal threshold while keeping the cores running high priority tasks cooler. Our proposed algorithm was able to reduce the average number of thermal violations by at least 3X and achieve up to 24.22% reduction in weighted execution time compared to the other techniques.

## 6. ACKNOWLEDGMENTS

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

## 7. REFERENCES

[1] Hotspot temperature modeling tool, 2010. `http://lava.cs.virginia.edu/HotSpot/`.

[2] Ti omap5 platform, 2011. `http://www.ti.com`.

[3] ARM. Cortex a9 processor, 2010. `http://www.arm.com/products/processors/cortex-a/cortex-a9.php`.

[4] 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, July 2006.

[5] J. Donald and M. Martonosi. Techniques for multicore thermal management: Classification and new exploration. In *Proceedings of the 33rd Annual International Symposium on Computer Architecture*.

[6] 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.

[7] S. Ghiasi and D. Grunwald. Design choices for thermal control in dual-core processors. In *Workshop on Complexity-Effective Design*, 2004.

[8] M. Gomaa, M. D. Powell, and T. N. Vijaykumar. Heat-and-run: leveraging smt and cmp to manage power density through the operating system. *SIGARCH Comput. Archit. News*, 32, October 2004.

[9] S. Heo, K. Barr, and K. Asanović. Reducing power density through activity migration. In *Proceedings of the 2003 international symposium on Low power electronics and design*, ISLPED, 2003.

[10] K. Iwata, S. Mochizuki, T. Shibayama, F. Izuhara, H. Ueda, K. Hosogi, H. Nakata, M. Ehama, T. Kengaku, T. Nakazawa, and H. Watanabe. A 256mw full-hd h.264 high-profile codec featuring dual macroblock-pipeline architecture in 65nm cmos. In *IEEE Symposium on VLSI Circuits*, 2008.

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

[12] M. Pedram and S. Nazarian. Thermal modeling, analysis, and management in vlsi circuits: principles and methods. In *Proceedings of the IEEE*, 2006.

[13] S. Sharifi, A. K. Coskun, and T. S. Rosing. Hybrid dynamic energy and thermal management in heterogeneous embedded multiprocessor socs. In *Proceedings of the 2010 Asia and South Pacific Design Automation Conference*, ASPDAC, 2010.

[14] 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.

[15] A. Telikepalli. Designing for power budgets and effective thermal management. *Xcell Journal*, (56), 2006.

[16] M. Ware, K. Rajamani, M. Floyd, B. Brock, J. Rubio, F. Rawson, and J. Carter. Architecting for power management: The ibm power7 approach. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, 2010.

[17] P. M. Yang Ge and Q. Qiu. Distributed task migration for thermal management in many-core systems. In *Design Automation Conference*, 2010.

[18] S. Zhang and K. S. Chatha. Approximation algorithm for the temperature-aware scheduling problem. In *Proceedings of the IEEE/ACM international conference on Computer-aided design*, ICCAD, 2007.