Temperature-Aware Scheduling and Assignment for Hard Real-Time Applications on MPSoCs

Thidapat Chantem
Department of CSE
University of Notre Dame
Notre Dame, IN 46556
tchantem@nd.edu

Robert P. Dick
Department of EECS
Northwestern University
Evanston, IL 60208
dickrp@northwestern.edu

X. Sharon Hu
Department of CSE
University of Notre Dame
Notre Dame, IN 46556
shu@nd.edu

***Abstract*—Thermal effects in MPSoCs may cause the violation of timing constraints in real-time systems. This paper presents a mixed integer linear programming based solution to this problem. Tasks are assigned and scheduled to an MPSoC to minimize peak temperature, subject to real-time constraints. The proposed approach outperforms existing methods, reducing peak temperature by up to 24.66 °C and by an average of 8.75 °C when compared to minimal-energy solutions. We also present a heuristic for use on large problem instances. Steady-state thermal analysis is used for tasks with long execution times compared to the RC thermal time constants of the cores. Transient analysis is used otherwise. The steady-state analysis based heuristic finds solutions with at most 3.40 °C deviation from optimal peak temperature (0.22 °C on average) while improving upon existing technique by as much as 25.71 °C and 10.86 °C on average. The transient analysis based heuristic further reduce peak temperature by 1 °C in the best case and 0.17 °C on average.**

I. Introduction and Contributions

In the near future, multiprocessor-system-on-chips (MPSoCs) will likely be used in many applications, as they often permit higher performance and better power efficiency than uniprocessor architectures. However, MPSoCs can have high power density and temperature, which degrade reliability and increase packaging and cooling cost. A 10–15 °C difference in operating temperature can result in a 2× difference in the lifespan of a device [1]. The cost of cooling solutions increases super-linearly in power consumption [2]. This makes designing packages and cooling solutions for the worst-case temperature prohibitively expensive. For these reasons, techniques such as processor throttling must be used to prevent the chip from reaching an unsafe temperature at run-time. However, run-time throttling can have undesirable consequences for real-time applications, which demand predictable timing behavior.

Existing power-aware techniques, such as energy minimization, peak power minimization, as well as dynamic voltage and frequency scaling (DVFS), cannot solve the temperature problem in MPSoCs because they do not consider spatial thermal variation; heat generated by an active core also affects other neighboring thermal elements, be they other cores or portions of the heatsink. The net heat flow from one thermal element to another depends on the conductance parameters and the current temperature of these thermal elements. Ignoring spatial thermal variation can lead to unnecessarily-high peak temperatures.

Researchers have recently started working on temperature-aware assignment and scheduling on MPSoCs [3], [4], [5]. Little work has been done on the use of MPSoCs for temperature-aware design of real-time systems. Xie and Hung were the first to study this problem [6]. However, their technique can deviate significantly from optimality (Section VI-B).

We present the first mixed-integer linear programming (MILP) formulation for assigning and scheduling tasks with hard real-time constraints on an MPSoC to minimize peak temperature. Our formulation considers spatial and temporal variations in power density. It relies on a *phased steady-state* thermal analysis technique that is directly integrated within the MILP formulation. This analysis approach produces a separate thermal profile for each power profile occurring during the schedule using steady-state analysis. Extensions for temperature-dependent leakage power modeling and DVFS are given. Experimental results show that our approach outperforms existing power-aware techniques in reducing the chip peak temperature and produces optimal solutions for small problem instances in which task execution times are long enough to neglect the dynamic effects resulting from heat capacity.

To solve problem instances that are large or for which the effects of heat capacity are significant, we propose a task assignment and scheduling framework in which the actual method for computing the temperature profile can be selected. Specifically, phased steady-state analysis is used when task durations are long relative to the thermal RC time constants of the cores. *Transient analysis*, in which temperatures are time-dependent, is used otherwise. Experimental results show that our heuristics produce results with small deviations from the optimal phased steady-state solutions.

II. System Model and Problem Definition

The system model and the temperature-aware real-time MPSoC assignment and scheduling problem are now described.

II.A. Task Model

Let J be the set of tasks to be executed. For each task $j \in J$, its execution time when running on core m is denoted by $E(j, m)$, its deadline by $D(j)$, and its release time by $R(j)$. ($R(j) = 0$ and $D(j) = \infty$ if no release time and deadline constraints are associated with task j.) To capture dependencies among tasks, we use a directed acyclic graph in which nodes represent tasks and directed edges indicate data dependencies between pairs of tasks. Further, let

$$\Gamma_{j_1,j_2} = \begin{cases} 1 & \text{if task } j_1 \text{ is an immediate predecessor of } j_2 \\ 0 & \text{otherwise.} \end{cases}$$

This work was supported in part by NSF under grant numbers CNS-0410771, CNS-0347941, and CNS-0702761, and in part by SRC under grant number 2007-HJ-1593.

A task j may execute only after all its predecessor tasks have completed and j has been released. For periodic systems, we guarantee schedule validity by scheduling out to the hyperperiod of all graphs. The hyperperiod is the least common multiple of the periods of all periodic tasks in the system.

II.B. Thermal Model

We model an MPSoC with a set of cores, M. The width, height, and location are specified for each core $m \in M$. Based on the floorplan, the set of neighbors of m, N_m, thermal conductance to a neighbor, $G_n(m,n)$, and thermal conductance to the heatsink element above it, $G_h(m)$, can be calculated. $P(j,m)$ indicates the power consumption of core m when executing task j. We discuss an extension to this model to account for leakage power in Section IV-B.

Our thermal analysis method is based on the classical Fourier heat flow model. Each core corresponds to a discrete thermal element. (Section IV-B discusses how our approach can be modified to support finer-grained thermal element modeling.) The heatsink on top of the cores is modeled using multiple thermal elements and its partitioning corresponds to the layout of the cores. Since the heatsink is usually larger than the processor itself, we model heatsink overhang using additional thermal elements; the heatsink overhangs the chip by 25% of its length and width. The interface layer is included within the heatsink instead of being modeled explicitly. It is usually very thin, therefore lateral heatflow within it can be neglected. Lateral heatflow between cores and heatsink elements is modeled. Figure 1(a) depicts the corresponding circuit representation.

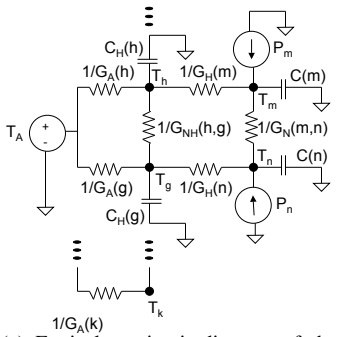
At time t, we compute the temperature at core m, $T(i,m)$:

$$0 = \sum_{n \in N_m} (T(t,m) - T(t,n)) \cdot G_n(m,n) + C(m) \cdot \frac{dT(t,m)}{dt} + (T(t,m) - T(t,h)) \cdot G_h(m) - \sum_{j \in J} \alpha(t,j,m) \cdot P(j,m) \quad (1)$$

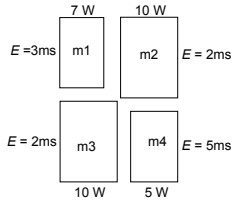
$$0 = \sum_{g \in N_h} (T(t,h) - T(t,g)) \cdot G_{nh}(h,g) + C(h) \cdot \frac{dT(t,h)}{dt} + (T(t,h) - T(t,m)) \cdot G_h(m) + (T(t,h) - T_A) \cdot G_A(h), \quad (2)$$

where $T(t,h)$ is the temperature of the heatsink element h directly above core m at time t, $C(m)$ is the capacitance at core m, and $\alpha(t,j,m) = 1$ if task j is active on core m at time t. $G_{nh}(h,g)$ is the lateral conductance between two heatsink elements h and g, $C(h)$ is the capacitance of heatsink element h, and $G_A(h)$ is the vertical conductance between heatsink element h and the ambient temperature T_A. All material thermal conductance parameters can be calculated as described in a standard physics textbook. We assume that the thickness of silicon is 0.6 mm and that of copper is 1 mm. The thermal conductivity of silicon and copper are set to 148 W/mK and 400 W/mK, respectively. Lastly, we use 45 °C for T_A and 90 °C for active layer temperature.

Problem Definition: Consider the floorplan of a chip containing a set of cores, M, and a set of real-time tasks, J as described above, determine a static assignment of tasks to cores and a static, non-preemptive schedule of tasks on the cores such that all precedence constraints and real-time deadlines are met, and the chip peak temperature, T_{max}, is minimized.



(a) Equivalent circuit diagram of the thermal model.



(b) Floorplan for the motivating example with task execution time (E) for each core. Diagram not to scale.

Fig. 1. Circuit and floorplan

III. Motivating Example

We illustrate the benefit of directly minimizing peak temperature using an example. Consider a task set containing two identical tasks, j_1 and j_2, each with a deadline of 5 ms. For this example, the MPSoC is arranged as shown in Figure 1(b). To minimize energy, tasks j_1 and j_2 are both assigned to core $m2$ with a peak temperature of 65.30 °C. To minimize peak power, task j_1 is assigned to core $m2$ and task j_2 to core $m1$, resulting in the same peak temperature. If task j_1 were executed on core $m4$ and task j_2 on core $m1$, the peak temperature would be reduced to 65.16 °C, 0.14 °C cooler. While this improvement may not seem significant, the power density of the chip in the above example is only 0.19 W/mm^2. The power density can be as high as 0.79 W/mm^2 for 90 nm processors, 2.02 W/mm^2 for 65 nm processors, and 7.24 W/mm^2 for 45 nm processors [7]. To obtain similar power densities, we increase each core power consumption by factors of 2, 5, 10, 15, and 20. Cores with high power density can are used in high-performance computing and signal processing applications. The resulting chip power densities are between 0.39 W/mm^2 and 3.89 W/mm^2. For such cores, peak temperature minimization results in a peak temperature reduction from 2–20 °C.

IV. MILP-based Approach

This section presents our solution to the problem defined in Section II-B, describes extensions for leakage power and DVFS, and discusses limitations of the MILP formulation.

IV.A. MILP Formulation

MILP formulations have long been proposed for modeling the heterogeneous multiprocessor task assignment and scheduling problem [8]. However, energy minimization has often been the main objective. Such solutions ignore both temporal and spatial thermal variation. Even peak power minimization only considers temporal thermal variation. To consider both types of thermal variations, we directly minimize the chip peak temperature, $T_{\max}$, which is the highest

temperature at any position on the chip during a schedule of duration *SL*, i.e.,

$$T_{\max} = \max_{m \in M, t \in [0, SL]} \tau_m(t), \tag{3}$$

Using Eq. 1 and 2 to compute the temperature at each node at a given time instant corresponds to dynamic or transient thermal analysis, which is computationally expensive, making its use in an MILP impractical. If transient analysis were incorporated into the MILP formulation, its use would be limited to very small problem instances. For these reasons, we set the capacitance values in Eq. 1 and 2 to zero to obtain the steady-state temperature at each node when predicting temperatures in our MILP formulation. In Section IV-B, we indicate the situations in which the MILP-based approach with steady-state analysis is appropriate and inappropriate. In addition, we present a solution to the more general dynamic temperature optimization problem in Section V.

One way to compute $T_{\max}$ in Eq. 3 is to discretize time. However, since real-time tasks can execute for hundreds of thousands or millions of time units, doing so can be costly. To overcome this difficulty, we observe that (i) core power consumptions only change at the beginning or end of task execution, and (ii) the temperature of a core only experiences a rapid change soon after the power consumption of at least one core on the chip changes. Hence, we only evaluate the temperature of each core m immediately after any task i starts executing on any core m in the MPSoC and denote this temperature by $T(i,m)$. Therefore, we write the objective of the MILP as

$$\min T_{\max}, \text{ where } T_{\max} \geq T(i,m), \forall m \in M, \forall i \in J.$$

In addition, $T(i,m)$ satisfies the constraints in Eq. 1 and Eq. 2, which are rewritten as

$$\begin{aligned} T(i,m) \equiv T_{HS}(i,h) + \frac{1}{G_H(m)} \left[\sum_{j \in J} \beta(i,j,m) \cdot P(j,m) \right] \\ + \frac{1}{G_H(m)} \sum_{n \in N_m} G_N(m,n) \cdot [T(i,n) - T(i,m)] \end{aligned} \tag{4}$$

$$\begin{aligned} 0 = (T_{HS}(i,h) - T(i,m)) \cdot G_H(m) + (T_{HS}(i,h) - T_A) \\ \cdot G_A(h) + \sum_{g \in N_h} (T_{HS}(i,h) - T_{HS}(i,g)) \cdot G_{NH}(h,g). \end{aligned} \tag{5}$$

where

$$\beta(j_1, j_2, m) = \begin{cases} 1 & \text{if task } j_2 \text{ executes on core } m, \\ & \text{precedes and overlaps with task } j_1 \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

We use $ts(j)$ and $tf(j)$ to denote the start and finish times of task j, respectively, and $tf(j) \equiv ts(j) + \sum_{m \in M} \delta(j,m) \cdot E(j,m)$. Eq. 4 implies that if core m is executing task i then $\beta(i,i,m) = 1$. More generally, $\beta(i,i,m) \equiv \delta(i,m)$. Consider a situation where tasks i and j execute on cores m and n, respectively. Further, task i precedes task j and their executions overlap. At the start of task i, we only need to consider the power consumption of core m. However, at the start of task j, we must take into account the power consumptions of both cores. For this reason, we must ensure that $\beta(j_1, j_2, m) = 1$ only when $\delta(j_2, m) = 1$ and $ts(j_2) \leq ts(j_1) \leq tf(j_2) - \epsilon$, where ϵ is a small constant used to prevent imprecise floating point computations from making it appear as if contiguous tasks overlap in time. Therefore,

$$\forall m \in M, \forall j_1, j_2 \in J, j_1 \neq j_2$$

$$tf(j_2) \geq ts(j_1) + (\beta(j_1, j_2, m) - 1) \cdot \Lambda \tag{7}$$

$$ts(j_2) \leq ts(j_1) + (1 - \beta(j_1, j_2, m)) \cdot \Lambda \tag{8}$$

$$1 \geq \beta(j_1, j_2, m) + \delta(j_1, m) \tag{9}$$

$$\begin{aligned} tf(j_2) - \epsilon - \ & (1 - \eta(j_2, j_1)) \cdot \Lambda - (1 - \delta(j_2, m)) \cdot \Lambda \\ & \leq ts(j_1) + \beta(j_1, j_2, m) \cdot \Lambda \end{aligned} \tag{10}$$

where

$$\delta(j,m) = \begin{cases} 1 & \text{if task } j \text{ is assigned to core } m \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

$$\eta(j_1, j_2) = \begin{cases} 1 & \text{if task } j_1 \text{ is executed before task } j_2 \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

The constraints used to guarantee schedulability can be found in an extended technical report [9]. Note that Eq. 4 is only linear if we can treat $P(j,m)$ as a constant given task j and core m. We now assume this to ease explanation, but will relax this assumption in Section IV-B.

The above MILP formulation is guaranteed to find an assignment and schedule that minimize the peak temperature of the chip. Our formulation can readily be modified to produce an assignment and schedule that minimize peak power or total energy.

IV.B. Extensions and Limitations

The assumption that $P(j,m)$ is a constant is no longer valid if leakage power is significant. Leakage power is a superlinear function of temperature and ignoring this dependency may lead to underestimation of the chip peak temperature. It has been observed that leakage power can be approximated by a a linear function in the operating temperature ranges of integrated circuits with roughly 5% error [10]. We use the following model:

$$P(j,m) = K_1(j,m) \cdot T(j,m) + K_2(j,m),$$

where $K_1(j,m)$ and $K_2(j,m)$ are constants that depend on core m and task j. Consequently, from Eq. 4,

$$\sum_{j \in J} \beta(i,j,m) \cdot P(j,m) = \sum_{j \in J} (\lambda(i,j,m) + \beta(i,j,m) \cdot K_2(j,m))$$

where

$$\lambda(i,j,m) = \begin{cases} K_1(j,m) \cdot T(i,m) & \text{if } \beta(i,j,m) = 1 \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

(13) can be replaced by the following contraints

$$\forall i, j \in J, \forall m \in M$$

$$\lambda(i,j,m) \geq 0 \tag{14}$$

$$\lambda(i,j,m) \leq \beta(i,j,m) \cdot \Lambda \tag{15}$$

$$\lambda(i,j,m) \geq (K1_{j,m} \cdot T(i,m)) - (1 - \beta(i,j,m)) \cdot \Lambda \tag{16}$$

$$\lambda(i,j,m) \leq (K1_{j,m} \cdot T(i,m)) - (\beta(i,j,m) - 1) \cdot \Lambda \tag{17}$$

If the cores are equipped with DVFS capability, further peak temperature reductions, as well as energy savings, can be achieved by modifying the solution in Section IV-A. For each

core m, the set of discrete voltage levels, K_m, is specified. We redefine $E(j,k,m)$ to be the execution time of task j on core m at voltage level k and $P(j,k,m)$ to be the power consumption required to execute task j on core m at voltage level k. The binary variables $\delta(j,k,m)$ are also redefined to be 1 if task j is assigned to core m at voltage level k. Consequently, from Eq. 4,

$$\sum_{j\in J}\beta(i,j,m)\cdot P(j,m)=\sum_{k\in K}\sum_{j\in J}\nu(i,j,k,m)\cdot P(j,k,m)$$

where

$$\nu(j_1,j_2,k,m)=\begin{cases}1 & \text{if } \delta(j_2,k,m)=1,\ \beta(j_1,j_2,m)=1\\ 0 & \text{otherwise}\end{cases}$$

The extended formulation can be obtained by adding the constraints for the variables defined in the above equation.

The thermal model (Section II-B) can further be refined by using multiple thermal elements for each core, where each thermal element may have different power consumption and/or correspond to a particular functional unit of the core. (A finer-grained thermal model in which a core is partitioned into many small thermal elements may further improve accuracy.) We omit this refinement due to lack of realistic benchmarks for which power profile variations within cores are known. When such benchmarks become available, only minor modifications to the solution in Section IV-A will be needed. Specifically, the binary variables $\delta(j,m,x)$ must be redefined as 1 if task j_1 executes on core m and x is a thermal element belonging to m, and the variables in Eq. 6 must be modified accordingly.

There are two limitations to the MILP-based approach: (i) the MILP formulation cannot be used to efficiently solve large problem instances (the problem is $\mathcal{NP}$-hard) and (ii) due to the use of steady-state analysis, the MILP formulation may overestimate the chip peak temperature when task execution times are short compared to the RC thermal time constants of the cores (i.e., the constants influencing the rate of temperature change in response to power consumption change). We will use an efficient heuristic framework to overcome these challenges.

V. Scheduling Heuristic Framework

To trade off accuracy in temperature estimation with running time, we propose a scheduling framework where either steady-state or transient analysis can be used, depending on the characteristics of the tasks under consideration.

Our framework uses a binary-search based approach to iteratively improve the solution. It takes as inputs upper and lower temperature bounds, as well as the maximum number of iterations, *maxIter*. It then uses the average of the upper and lower bound on the peak temperatures as the target temperature to find an assignment and schedule. If it succeeds, the current target temperature is used as the upper temperature bound for the next iteration of the binary search. Otherwise, it is used as the lower temperature bound.

To find an assignment and schedule for a given target temperature, we introduce the list scheduling [11] algorithm shown in Algorithm 1. ThermalSched() returns a schedule that honors the specified target temperature, if possible. Our search-based scheduling approach permits the use of an efficient list scheduler without global knowledge of temperature variation.

Algorithm 1 ThermalSched($G(V,E)$, *targetTemp*)

```
1:  compute EST(j), for all tasks // earliest start time
2:  compute LST(j), for all tasks // latest start time
3:  mobility(j) ← LST(j) - EST(j), for all tasks
4:  currentTime ← 0
5:  busy(m) ← 0, for all cores
6:  while there are unscheduled tasks do
7:    RT ← ready tasks in non-decreasing order of mobility
8:    for each j ∈ RT do
9:      invalidCount ← 0
10:     fastestCore ← −1
11:     bestExeTime ← ∞
12:     for each m ∈ M do
13:       δ(j, m) ← 0
14:       endTime ← E(j,m) + currentTime
15:       if not busy(m) and endTime ≤ D(j) then
16:         compute projected temperature profile
17:         peakTemp ← max_{m∈M} T(j, m)
18:         if peakTemp < targetTemp then
19:           if E(j,m) < bestExeTime then
20:             fastestCore ← m
21:             bestExeTime ← E(j,m)
22:       else if not busy(m) then
23:         invalidCount ← invalidCount + 1
24:     if invalidCount = |M| then
25:       return INFEASIBLE
26:     else if fastestCore ≠ −1 then
27:       δ(j, fastestCore) ← 1
28:       ts(j) ← currentTime
29:       tf(j) ← ts(j) + E(j, fastestCore)
30:       busy(fastestCore) ← 1
31:   update EST(j), for all unscheduled tasks
32:   update mobility(j), for all unscheduled tasks
33:   update busy(m), for all cores
34:   currentTime ← min{tf(j) : tf(j) > currentTime}
```

For a given task j, its earliest start time *EST(j)* and latest start time *LST(j)* are computed. The mobility of task j can then be calculated as the difference between *LST(j)* and *EST(j)*. A potential challenge in computing *EST(j)* and *LST(j)* is that the execution time of task j is unknown prior to assignment to a core. The earliest start time *EST(j)* of each task $j \in J$ is computed using the As Soon As Possible (ASAP) scheduling algorithm [11]. For each task, we use the execution time on the fastest core in order to prioritize tasks based on maximal mobility. The same is true for computing the *LST(j)*, for all $j \in J$.

Task assignment and scheduling proceed as follows. Ready tasks are ordered by non-decreasing mobility. A *ready task* is a task whose predecessors have finished executing. Given a ready task j, ThermalSched() selects the fastest available core that allows task j to meet its deadline while keeping the peak temperature below the target temperature. The fastest available core is chosen to maximize the mobility of the successors of

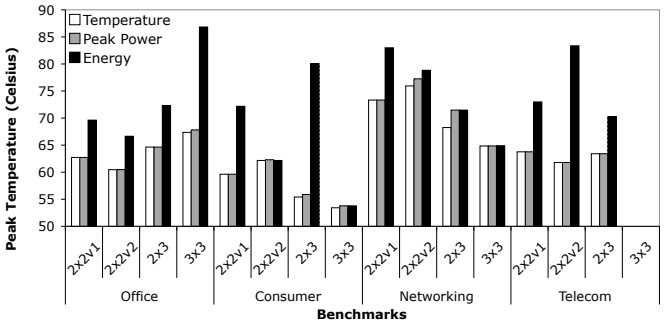


Fig. 2. Peak temp. minimization vs. energy and peak power minimization

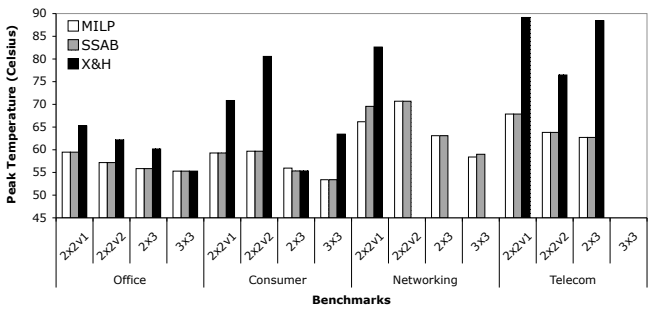


Fig. 3. Peak temp. minimization vs. energy and peak power minimization for higher power density chips

task j, thereby improving schedulability. If no core is fast enough to execute task j by its deadline, ThermalSched() terminates.

Observe that Algorithm 1 does not provide any details regarding the method for computing the thermal profile (Line 16). We propose to use two techniques based on the observations made in Section IV-B, to balance accuracy and time complexity. If task execution times are long compared to the RC thermal time constants of the cores, phased steady-state analysis will generally be accurate. The steady-state temperature profile can be computed by expressing Eq. 1 and 2 as a system of linear equations of the form $A \cdot \mathbf{T} + B = 0$, and of size $|E| \times |E|$, where $|E|$ is the total number of thermal elements in the system. Since the thermal conductivity matrix A is fixed when a floorplan is given, the inverse of the matrix can be pre-computed once. In addition, computing the temperature profile in each iteration takes $O(|M|^2)$. Therefore, the time complexity of the **steady-state thermal analysis based heuristic (SSAB)** is $O(|J|^2 \cdot |M|^3 \cdot maxIter)$.

If task execution times are short relative to core thermal RC time constants, transient analysis should be used to compute the projected temperature profile, as explained in Section IV-B. Any existing thermal analysis technique can be used in our task assignment and scheduling framework. To validate our **transient analysis based heuristic (TAB)**, we used `HotSpot` [12] in our experiments. The time complexity of the TAB algorithm is $O(|J|^2 \cdot |M| \cdot maxIter \cdot O(HotSpot))$.

As a final remark, verification using dynamic thermal analysis may be desirable even for systems with long task execution times to ensure the peak temperature bound is honored.

VI. Experimental Results

We used the Embedded System Synthesis Benchmarks Suite (E3S) [13], which follows the organization of the EEMBC benchmarks [14]. We experimented with floorplans with different number of cores, power densities and areas.

VI.A. MILP Formulation Performance

We used CPLEX 10.1 with AMPL to solve instances of the MILP formulation (Section IV-A) for optimal peak temperature, energy, and peak power. The benchmarks were run using two 2×2, one 2×3, and one 3×3 floorplans.

We compared the temperature differences between using peak temperature minimization and using energy or peak power minimization. The solutions from the MILP are shown in Figure 2. The x-axis shows the different benchmarks and floorplans. The y-axis shows the resultant peak temperatures. Some results are unavailable due to the MILP solver running out of memory before finding a solution. Our approach achieves a peak temperature reduction of up to 24.66 °C, and 8.75 °C on average, when compared to the method that minimizes energy. Such a significant difference in peak temperatures occurs because energy minimization does not consider the effects of temporal thermal variations.

Our approach did not yield significant improvements over peak power minimization because the cores used in the above experiment have low power densities. For example, the chip power density for the Consumer benchmarks ranges from 0.27 W/mm^2 to 0.36 W/mm^2 with the average chip power density of 0.32 W/mm^2. These results confirm the observations of Coskun et al. for low power density MPSoCs. However, higher power densities change the problem. To obtain chip power densities similar to those described in [7] for 65 nm processors, we multiplied each core's power consumption by 10. As shown in Figure 3, for these cores our method achieves a peak temperature reduction of up to 23.25 °C, and 9.24 °C on average when compared to peak power minimization and further proves that spatial thermal variations need to be considered.

VI.B. Heuristic Performance

We assess the performance of our SSAB algorithm (Section V) by comparing its solutions to the ones from MILP (Section IV-A) as well as Xie's and Hung's *Heuristic 1* [6], which we refer to as the X&H heuristic. The X&H heuristic calls HotSpot to compute the temperatures. Figure 4 compares the the results from the SSAB and X&H heuristics to the optimal solution from the MILP formulation. We used HotSpot to compare the peak temperatures for a fair comparison. Results for benchmarks that were not successfully solved by the X&H algorithm are omitted.

The X&H heuristic deviates from the optimal solution by 11.10 °C on average and 25.71 °C in the worst case. The SSAB heuristic finds an optimal solution in many cases while giving results that deviate by at most 3.40 °C from optimality (and 0.22 °C on average) requiring at most 50 binary search iterations for each run. Both heuristics require similar running

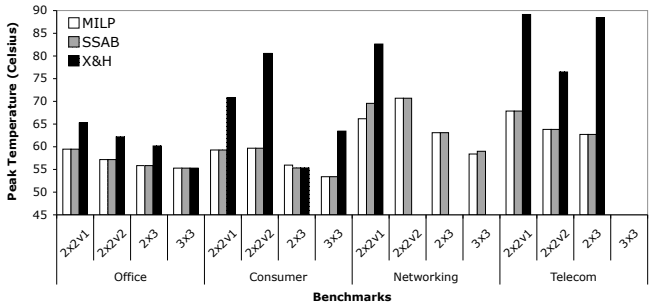


Fig. 4. Perf. of steady-state analysis based heuristics (based on HotSpot)

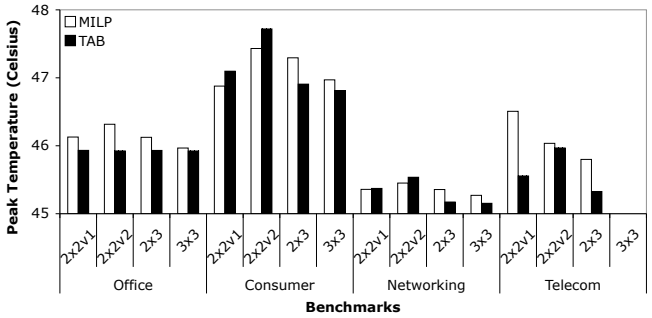


Fig. 5. Perf. of transient analysis based heuristic (based on HotSpot)

times, but the SSAB heuristic never performs worse than the X&H heuristic.

To show the scalability of the SSAB heuristic, we ran a benchmark with 30 tasks on a system with 4×4 homogeneous core arrangement. The MILP solver could not solve the problem because the 3.58 GB RAM workstation ran out of memory. However, the SSAB heuristic found a solution using no more than 50 binary search iterations within 9 s.

We now assess the performance of the TAB heuristic (Section V) using the same set of benchmarks. The TAB heuristic calls HotSpot to determine transient temperatures. Since the original task execution times for the E3S benchmarks tend to be short, the significance of dynamic thermal effects can be significant. We compare the peak temperatures obtained by the MILP and the TAB heuristic, as shown in Figure 5. (Note the difference in the y-axis scale between Figure 4 and Figure 5.) The reduction in peak temperatures for the TAB heuristic relative to the MILP solver is just under 1 °C in the best case, and 0.17 °C on average. This is because transient analysis can, at times, more accurately predict temperatures when task durations are short.

The TAB heuristic also improve the benchmark finishing times. Let the speedup be the ratio between the end of the MILP schedule and that of the TAB schedule. The maximum, minimum, and average speedups are $63.29\times$, $0.87\times$, and $9.02\times$, respectively. Such speedup is due to the fact that the latter is much less pessimistic in estimating temperatures and hence schedules more tasks in parallel. However, the SSAB heuristic is much more efficient than the TAB heuristic. Specifically, the SSAB heuristic is about $175\times$ faster than the TAB heuristic on average for benchmarks with short task execution times; this difference further increases for benchmarks with longer task execution times.

VII. Conclusions

We have presented an assignment and scheduling technique that uses an MILP solver to optimize chip peak temperature under hard real-time constraints based on phased steady-state thermal analysis. Our technique outperforms existing power-aware techniques with a peak temperature reduction of up to 24.66 °C, and 8.75 °C on average, for embedded processors compared to the method of energy minimization.

To efficiently solve this $\mathcal{NP}$-hard assignment and scheduling problem, we propose a scheduling heuristic framework in which the actual method for temperature prediction depends on task durations. Phased steady-state analysis is appropriate when task execution times are long and transient analysis should be used otherwise. Our phased steady-state analysis based heuristic finds an optimal solution in many cases, with a maximum deviation from optimality of 3.40 °C. When compared to previous work, the heuristic achieves a temperature reduction of 10.86 °C on average. The transient analysis based heuristic accurately models and exploits the transient thermal effects of short tasks to further improve on the existing solutions by just under 1 °C in the best case.

References

[1] R. Viswanath, et al., "Thermal performance challenges from silicon to systems," *Intel Technology Journal*, vol. 4, no. 3, pp. 1–16, Aug. 2000.

[2] S. H. Gunther, et al., "Managing the impact of increasing microprocessor power consumption," *Intel Technology Journal*, vol. 5, no. 1, pp. 1–9, Feb. 2001.

[3] A. K. Coskun, T. S. Rosing, and K. Whisnant, "Temperature aware task scheduling in MPSoCs," in *Proc. Design, Automation & Test in Europe Conf.*, Apr. 2007, pp. 1659–1664.

[4] R. Rao, S. Vrudhula, and C. Chakrabarti, "Throughput of multicore processors under thermal constraints," in *Proc. Int. Symp. Low Power Electronics & Design*, Aug. 2007, pp. 201–206.

[5] S. Murali, et al., "Temperature-aware processor frequency assignment for MPSoCs using convex optimization," in *Proc. Int. Conf. Hardware/Software Codesign and System Synthesis*, Oct. 2007, pp. 111–116.

[6] Y. Xie and W.-L. Hung, "Temperature-aware task allocation and scheduling for embedded multiprocessor systems-on-chip (MPSoC) design," *J. VLSI Signal Processing*, vol. 45, no. 3, pp. 177–189, Dec. 2006.

[7] G. Link and N. Vijaykrishnan, "Thermal trends in emerging technologies," in *Proc. Int. Symp. Quality of Electronic Design*, Mar. 2006, pp. 625–632.

[8] L.-F. Leung, C.-Y. Tsui, and W.-H. Ki, "Simultaneous task allocation, scheduling and voltage assignment for multiple-processors-core systems using mixed integer nonlinear programming," in *Prof. Int. Symp. Circuits and Systems*, May 2003, pp. 309–312.

[9] T. Chantem, R. P. Dick, and X. S. Hu, "Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs," Tech. Rep., 2007, University of Notre Dame.

[10] Y. Liu, et al., "Accurate Temperature-Dependent Integrated Circuit Leakage Power Estimation is Easy," in *Proc. Design, Automation & Test in Europe Conf.*, Mar. 2007, pp. 204–209.

[11] G. De Micheli, *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Book Company, NY, 1994.

[12] K. Skadron, et al., "Temperature-aware microarchitecture," in *Proc. Int. Symp. Computer Architecture*, June 2003, pp. 2–13.

[13] R. P. Dick, "E3S: The embedded system synthesis benchmarks suite," E3S link at http://robertdick.org/tools.html.

[14] "Embedded microprocessor benchmark consortium," http://www.eembc.org.