Thermal-Aware Task Scheduling in 3D Chip Multiprocessor with Real-Time Constrained Workloads

JIAYIN LI and MEIKANG QIU, Huazhong University of Science and Technology and University of Kentucky
JIAN-WEI NIU, Beihang University
LAURENCE T. YANG, Huazhong University of Science and Technology and St. Francis Xavier University
YONGXIN ZHU, Shanghai Jiaotong University
ZHONG MING, Shenzhen University

Chip multiprocessor (CMP) techniques have been implemented in embedded systems due to tremendous computation requirements. Three-dimension (3D) CMP architecture has been studied recently for integrating more functionalities and providing higher performance. The high temperature on chip is a critical issue for the 3D architecture. In this article, we propose an online thermal prediction model for 3D chips. Using this model, we propose novel task scheduling algorithms based on rotation scheduling to reduce the peak temperature on chip. We consider data dependencies, especially inter-iteration dependencies that are not well considered in most of the current thermal-aware task scheduling algorithms. Our simulation results show that our algorithms can efficiently reduce the peak temperature up to 8.1°C.

Categories and Subject Descriptors: C.3 [**Special-Purpose and Application-Based Systems**]: *real-time and embedded systems*

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: Thermal, peak temperature, task scheduling, real-time constraint

ACM Reference Format:
Li, J., Qiu, M., Niu, J.-W., Yang, L. T., Zhu, Y., and Ming, Z. 2013. Thermal-aware task scheduling in 3D chip multiprocessor with real-time constrained workloads. ACM Trans. Embedd. Comput. Syst. 12, 2, Article 24 (February 2013), 22 pages.
DOI = 10.1145/2423636.2423642 http://doi.acm.org/10.1145/2423636.2423642

This work was supported in part by the NSF CNS-1249223, NSFC 61071061, the University of Kentucky Start-Up Fund; the State Key Lab of Software Development Environment Grant BUAA SKLSDE-2010ZX-13, NSFC 60873241, ASF 20091951020; the National High Tech. R&D Program of China (863) 2009AA012201 and the Shanghai International S&T Collaboration Program (09540701900); the NSFC 61170077; the SZ-HK Innovation Circle project ZYB200907060012A, NSF GD:10351806001000000, S&T project of SZ JC200903120046A, S&T Project of GD 2012B091100198.
Authors' addresses: J. Li and M. Qiu, Department of Electrical and Computer Engineering, University of Kentucky, Lexington, KY 40506; email: {jli6; mqiu}@engr.uky.edu; J.-W. Niu, State Key Laboratory of Software Development Environment, Beihang University, Beijing 100191, China; email: niujianwei@buaa.edu.cn; L. T. Yang, Department of Computer Science, St. Francis Xavier University, Antigonish, NS B2G 2W5, Canada; email: ltyang@stfix.ca; Y. Zhu, School of Microeletronics, Shainghai Jiaotong University, 200240, China; email: zhuyongxin@sjtu.edu.cn; Z. Ming (corresponding author), College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, GD 518060, China; email: mingz@szu.edu.cn.

DOI 10.1145/2423636.2423642 http://doi.acm.org/10.1145/2423636.2423642

1. INTRODUCTION

Chip multiprocessors (CMP) have been widely used in *Embedded Systems for Interactive Multimedia Services* (ES-IMS) due to tremendous computation requirements in modern embedded processing. The primary goals for microprocessor designers are to increase the integration density and achieve higher performance without correspondingly increases in frequency. However, the traditional *two dimensional* (2D) planar CMOS fabrication processes are poor at communication latency and integration density. The *three-dimensional* (3D) CMOS fabrication technology is one of the solutions for faster communication and more functionalities on chip. More functional units can be implemented while stacking two or more silicon layers in a CMP. Meanwhile, the vertical distance is shorter than the horizontal distance in a multi-layer chip [Topol et al. 2006; Black et al. 2006], which makes the systems more tight.

In CMPs, high on-chip temperature impacts circuit reliability, energy consumption, and system cost. Research shows that a 10 to 15°C increase of operation temperature reduces the lifetime of the chip by half [JEDEC 2009]. The increasing temperature causes the leakage current of a chip to increase exponentially. Also, the cooling cost increases significantly, which amounts to a considerable portion of the total cost of the computer system. The 3D CMP architecture magnifies the thermal problem. Moreover, the cross-sectional power density increases linearly with the number of stacked silicon layers, causing more serious thermal problems.

To mitigate the thermal problem, *Dynamic Thermal Management* (DTM) techniques, such as *Dynamic Voltage and Frequency Scaling* (DVFS), have been developed at the architecture level. When the temperature of the processor is higher than a threshold, the DTM can reduce the processor power and control the temperature of the processor. With DTMs, the system performance is degraded inevitably. Another way to alleviate the thermal problem of the processor is to use the operation system level task scheduling mechanisms. They either arrange the task execution order in a designated manner, or migrate "hot" threads across cores to achieve thermal balance. However, most of these thermal-aware task scheduling methods focus on independent tasks or tasks without inter-iteration dependencies. Applications in modern ES-IMS often consist of a number of tasks with data dependencies, including inter-iteration dependencies. Therefore, it is important to consider the data dependencies in the thermal-aware task scheduling.

In this article, we propose real-time constrained task scheduling algorithms to reduce peak temperature in a 3D CMP. The proposed algorithms are based on the rotation scheduling [Chao et al. 1997], which optimizes the execution order of dependent tasks in a loop. The main contributions of this article include:

(1) We present an online 3D CMP temperature prediction model.
(2) We also propose OS level task scheduling algorithms to reduce the peak temperature. The data dependencies, especially inter-iteration dependencies in the application are well considered in our proposed algorithms.

The organization of this article is as follows. In Section 2, we discuss works related to this topic. Then, models for task scheduling in 3D CMPs are presented in Section 3. A motivational example is given in Section 4. We propose our algorithms in Section 5, followed by experimental results in Section 6. Finally, Section 7 conclude the article.

2. RELATED WORK

Energy-aware task scheduling has been widely studied in the literature. Weiser et al. [1994] first discussed the problem of task scheduling to reduce the processor energy consumption. An offline scheduling algorithm for task scheduling with variable processor speed was proposed in Yao et al. [1995]. But the tasks considered in these papers

are independent tasks. Mosse et al. [2000] proposed several schemes to dynamically adjust processor speed with slack reclamation based on the DVS technique. A scheme for processor speed management at branches was presented in Shin et al. [2001] based on the ratio of the longest path to the taken paths for the branch statement to the end of the program. However, the studies above only consider the uniprocessor system.

Recently, energy reduction has become an important issue in parallel system. Research in Shivle et al. [2004, 2006] focused on heterogeneous mobile ad hoc grid environments. Authors in those works studied the static resource allocation for the application composed of communicating subtasks in an ad-hoc grid. However, the goal of the allocation in those works is to minimize the average percentage of energy consumed by the application to execute across the machines, while meeting an application execution time constraint. This goal may lead to some cases in which some machines may consume much more energy than the others, even though the average consumption is minimized. Therefore, approaches proposed in those works cannot guarantee the satisfaction of the temperature constraint.

An energy-aware task scheduling mechanism, called EcoMapS, is proposed in Tian et al. [2005]. EcoMapS incorporates channel modeling, concurrent task mapping as well as communication and computation scheduling. Qiu and Sha [2009] proposed two task scheduling algorithms for embedded system with heterogeneous functional units. One of them is optimal and the other is near-optimal heuristic. The task execution time information was stochastically modeled. Qiu et al. [2009, 2010] proposed a loop scheduling algorithm for voltage assignment problem in embedded system. The research in Qiu et al. [2007] focused on modeling task execution time as a probabilistic random variable. Two optimal algorithms, one for uniprocessor and one for multiprocessor system, were presented to solve the voltage assignment with probability problem. The goal of these algorithms is to minimize the expected total energy consumption while satisfying the timing constraint. However, none of them consider thermal issues on processors.

In chip design stage, several techniques are implemented for thermal-aware optimization. Nookala et al. [2006] and Sankaranarayanan et al. [2005] proposed different thermal-aware floorplanning algorithms. For floorplanning on 3D chips, several other approaches were proposed recently [Pathak and Lim 2008; Zhou et al. 2008; Allec et al. 2008; Han et al. 2005]. Chaparro et al. [2009] proposed the controlling TFTECs from the microarchitecture for an enhanced DTM in multi-core architectures. Research in Puttaswamy and Loh [2007] focuses in improving the efficiency of heat removal.

Job allocation and scheduling is another approach to reduce temperature on-chip. Several temperature-aware algorithms are present in Coskun et al. [2008, 2009] Mulas et al. [2008], Lin et al. [2009], Ayoub and Rosing [2009], Zhu et al. [2008], Zhou et al. [2010], and Liu and Qiu [2010] recently. The Adapt3D approach in Coskun et al. [2009] assigns the upcoming job to the coolest core to achieve thermal balance. The method in Zhou et al. [2010] is to wrap up aligned cores into super core. Then, the hottest job is assigned to the coolest super core. Power and thermal management framework is proposed in Lin et al. [2009] for memory subsystem. In Ayoub and Rosing [2009], a thermal management scheme incorporates temperature prediction information and runtime workload characterization to perform efficient thermally aware scheduling. A scheduling scheme based on mathematic analysis is proposed on Zhu et al. [2008]. Liu and Qiu [2010] present a slack selection algorithm for thermal-aware dynamic frequency scaling. But none of these approaches considers the data dependencies in an application.

3. MODEL AND BACKGROUND

3.1. Thermal Model

Fourier heat flow analysis is the standard method of modeling heat conduction for circuit-level and architecture-level IC chip thermal analysis [Zhu et al. 2008]. It is

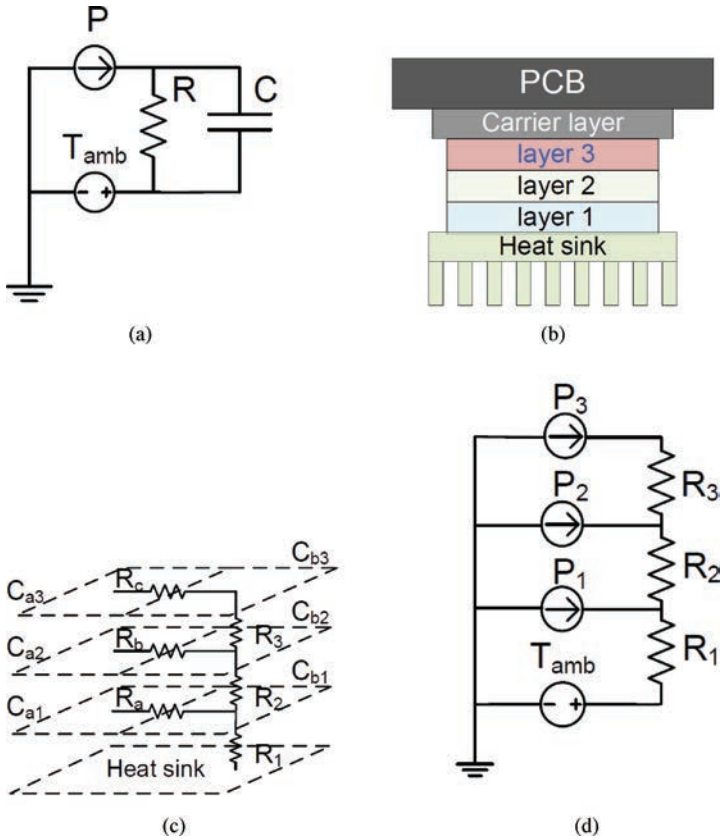


Fig. 1. (a) A Fourier thermal model of a single block. (b) The cross sectional view of a 3D chip. (c) The horizontal and vertical heat model, where the C_{a1} to C_{b3} are the IDs of the six cores in this example, the R_a to R_c are the vertical heat conductances, and R_1 to R_3 are the horizontal heat conductances. (d) The corresponding Fourier thermal model.

analogous to George Simon Ohm's method of modeling electrical current. A basic Fourier model of heat conduction in a single block on a chip is shown in Figure 1(a). In this model, the power dissipation is similar to the current source and the ambient temperature is analogous to the voltage source. The heat conductance of this block is a linear function of conductivity of its material and its cross-sectional area divided by its length. It is equivalent to electrical conductance. And the heat capacitance of this block is analogous to the electrical capacitance. Assuming there is a block on a chip with heat parameters as shown in Figure 1(a). The Fourier heat flow analysis model is

$$C\frac{d(T(t)-T_{amb})}{dt} = P - \frac{T(t)-T_{amb}}{R}. \tag{1}$$

C is the heat conductance of this block. $T(t)$ is the temperature of that block at time t. T_{amb} is the ambient temperature, P is the power dissipation, and R is the heat resistance. By solving this differential equation, we get the temperature of that block as follows:

$$T(t) = P \times R + T_{amb} - (P \times R + T_{amb} - T_{init})e^{-t/RC} \tag{2}$$

T_{init} is the initial temperature of that block.

Considering there is a task a running on this block and the corresponding power consumption is P_a, we can predict the temperature of the block by Eq. (2). Assuming that the execution time of a is t_a, we get the temperature of the block when a is finished:

$$T(t_a) = P_a \times R + T_{amb} - (P_a \times R + T_{amb} - T_{init})e^{-t_a/RC}. \quad (3)$$

When the execution of task a goes infinite, the temperature of this block reaches a stable state, T_{ss}, which is shown as follows:

$$T_{ss} = P_a \times R + T_{amb} \quad (4)$$

Substituting Eq. (4) in Eq. (3), we can get an alternative way of predicting the finish temperature of task a running on that block:

$$T(t_a) = (T_{ss} - T_{init})(1 - e^{-t_a/RC}) + T_{init}. \quad (5)$$

We can further simplify Eq. (5) as follows:

$$T(t_a) = (T_{ss} - T_{init})(1 - e^{-bt_a}) + T_{init}, \quad (6)$$

where $b = 1/RC$.

3.2. The 3D CMP and the Core Stack

A 3D CMP consists of multiple layers of active silicon. On each layer, there exist one or more processing units, which we call cores. Figure 1(b) shows a basic multi-layer 3D chip structure. A heat sink is attached to the top of the chip to remove the heat from the chip more efficiently. The horizontal lateral heat conductance is approximately 0.4 W/K (i.e. "R_a" in Figure 1(c)), much less the conductance between two vertically aligned cores (approximately 6.67 W/K, i.e., "R_2" in Figure 1(c)) [Zhu et al. 2008]. The temperature values of vertically aligned cores are highly correlated, compared with the temperatures of horizontally adjacent cores.

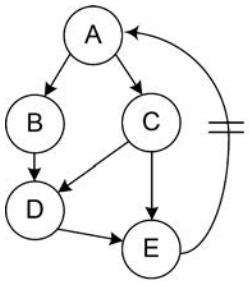
Therefore, for the online temperature prediction model used in our scheduling algorithms, we ignore the horizontal lateral heat conductance. Note that, even though we ignore this heat conductance in our model, the simulator used in our experiment is a general thermal simulator that considers both the horizontal lateral heat conductance and the vertical conductance. The efficiency of our low-computation model is tested through this general thermal simulator in our experiment. We call a set of vertically aligned cores as a *core stack*. Cores in a core stack are highly thermal correlated. The high temperature of a core caused by heavy loading will also increase the temperatures of other cores in the core stack. For cores in a core stack, the distances from them to the heat sink are different. Considering a number k of cores in a core stack, where core k is the furthest from the heat sink and core 1 is the closest to the heat sink; the stable state temperature of the core j ($j \leq k$) can be calculated as,

$$T_{ss}(j) = \sum_{i=1}^{j} \left(\sum_{l=i}^{k} P_l \times R_i \right) + T_{amb}, \quad (7)$$

where P_l is the power consumption of the core l and R_i is the inter-layer thermal conductance between cores i-1 and i (see Figure 1(d)).

In order to predict the finish temperature of task a running on core j online, we approximate this finish temperature $T_j(t_a)$ by substituting equation (7) in Eq. (5) as

$$\begin{aligned} T_j(t_a) \; = \; & \left(\sum_{i=1}^{j} \left(\sum_{l=i}^{k} P_l \times R_i \right) + T_{amb} - T_{init\_j} \right) \\ & \times (1 - e^{-t_a/R_j C_j}) + T_{init\_j} \end{aligned} \quad (8)$$



(a)

Task	P1		P2	
	t	Tss	t	Tss
A	95	84	65	92
B	70	80	60	78
C	60	95	65	82
D	80	82	80	70
E	50	75	75	82

(b)

```
for(i=2;i<N;i++) {
 A[i]=TaskA(E[i-2]);
 B[i]=TaskB(A[i]);
 C[i]=TaskC(A[i]);
 D[i]=TaskD(B[i],C[i]);
 E[i]=TaskE(C[i],D[i]); }
```

(c)

Fig. 2. (a) The DFG of an application. (b) The characteristics of the tasks. (c) The pseudo code of this application.

3.3. Application Model

A *Data-Flow Graph* (DFG) is used to model an embedded system application. A DFG typically consists of a set of vertices V, each of which represents a task in the application, and a set of edges E, showing the dependencies among the tasks. The edge set E contains edges e_{ij} for each task $v_i \in V$ that task $v_j \in V$ depends on. The weight of a vertex v_i represents the task type of task i. In our model, the number of tasks may be larger than the number of task types. And the tasks with the same task type have the same execution time. Also the weight of an edge e_{ij} means the size of data which is produced by v_i and required by v_j.

We use a cyclic DFG to represent a loop of an application in this paper. In a cyclic DFG, a delay function $d(e_{ij})$ defines the number of delays for edge e_{ij}. For example, assuming $d(e_{ab}) = 1$ is the delay function of the edge from task a to b, which means the task b in the i^{th} iteration depends on the task a in the $(i-1)$th iteration. In a cyclic DFG, edges without delay represent the intra-iteration data dependencies, while the edges with delays represent the inter-iteration dependencies. An example of a cyclic DFG is shown in Figure 2(a) where one delay is denoted as a bar. There is a real-time constraint L, which is the deadline of finishing one period of the application. To generate a schedule of tasks in a loop, we use the static *direct acyclic graph* (DAG). A static DAG is a repeated pattern of an execution of the corresponding loop. For a given cyclic DFG, a static DAG can be obtained by removing all edges with delays.

Retiming is a scheduling technique for cyclic DFGs considering inter-iteration dependencies [Chao et al. 1997]. Retiming can optimize the cycle period of a cyclic DFG by distributing the delays evenly. For a given cyclic DFG G, the retiming function $r(G)$ is a function from the vertices set V to integers. For a vertex u_i of G, $r(u_i)$ defines the number of delays drawn from each of the incoming edges of node u_i and pushed to all of the outgoing edges. Let a cyclic DFG G_r be the cyclic DFG retimed by $r(G)$, then for a edge e_{ij}, $d_r(e_{ij}) = d(e_{ij}) + r(v_i) - r(v_j)$, where $d_r(e)$ is the new delay function of edge e_{ij} after retiming and $d(e_{ij})$ is the original delay function.

3.4. Energy Model

We consider the CMP in which each core is featuring the DVFS technique. In order to reduce the energy consumption, the DVFS technique jointly decreases the processor speed and the supply voltage. Research in Tian and Ekici [2007] shows that the decrease in processor voltage causes nearly linear increase in execution time and approximately quadratic decrease in energy consumption. Without loss of generality, we assume that each core has three DVFS modes, denoted as L_1, L_2 and L_3, respectively. L_1 has the slowest frequency and the lowest supply voltage, while the L_3 has the fastest

frequency and the highest supply voltage. Note that our approach is general enough for the number of DVFS modes larger than four. Our algorithms are not limited by the assumption of the DVFS modes numbers in the system.

Assume we know the power consumption and the execution time of different tasks running on different cores. We use a two-dimensional matrix EP to represent this information. We assume the CMP system has heterogeneous cores, which is a more general assumption compared to the homogeneous CMP. When applying our approach in the homogeneous CMP system, we only need to set execution time of a given task on every core as the same. There are two values in each entry of the EP matrix, one is execution time and the other is power consumption. For example, $ep_{ij} = \{e_{ij}, p_{ij}\}$ is one entry of the EP matrix. e_{ij} is the execution time of task i running on core j, while p_{ij} is the power consumption.

4. MOTIVATIONAL EXAMPLE

4.1. An Example of Task Scheduling in CMP

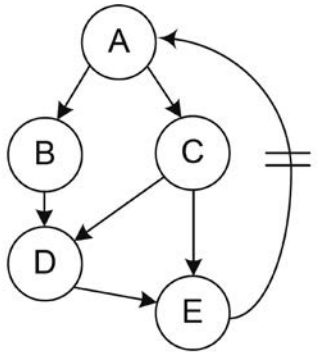
We first give an example of task scheduling in a multi-core chip. We schedule an application (see Figure 2(c)) in a two-core embedded system. A DFG representing this application is shown in Figure 2(a). There are two different cores in one layer. The execution times (t) and the stable state temperatures (T_{ss}) of each task in this application running on different cores are shown in Figure 2(b). For simplicity, we provide the stable state temperatures instead of power consumptions in this example, and we assume the value of b (see equation (6)) in each core is the same: 0.025. We also assume the initial temperatures and the ambient temperatures are 50°C.

4.2. List Scheduling Solution

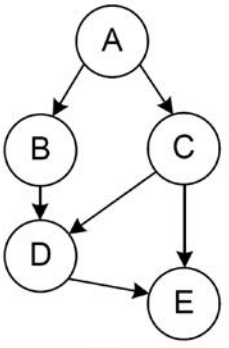
We first generate a schedule through the list-scheduling algorithm. Figure 3(b) shows a static DAG, which is transformed from the DFG (see Figure 3(a)) by removing the delay edge. For the DAG of this example, we can get the assigning order as {A, B, C, D, E}. For a task, we can calculate the peak temperatures when it is executed on different cores based on Eq. (5). Then tasks are assigned in a specific order to the core that can finish it at the coolest temperature. In the list scheduling, a task assigning order is generated based on the node information in the DAG, and the tasks are assigned to the "coolest" cores in that order. A schedule is generated as Figure 3(c). With the Eq. (5), we can get the peak temperature of each task as Figure 3(d). Task A has the highest peak temperatures in the first two iterations. In the first iteration, task A starts at the temperature of 50°C and ends at the temperature of 80.84°C. In the second iteration, task A starts immediately after the first iteration of task E finishes, which means it starts at the temperature of 67.89°C. Since it has a higher initial temperature, the peak temperature (82.50°C) in this iteration is higher.

4.3. Our Solution

Our proposed algorithm uses rotation scheduling to further reduce peak temperature. From the schedule in Figure 3(c), we can find that Task A is the first tasks executed in core P0, and Task A has inter-iteration data dependency with Task E. In this case, we can implement the rotation scheduling and Task A is the proper candidate for rotation. In Figure 4(a), we transform the original DFG into a new DFG by moving a delay from edge e_{EA} to edges e_{AB} and e_{AC}. The new corresponding static DAG is shown in Figure 4(b). In this new DAG, there are two parts: node A and the rest nodes. There is no dependency between node A and the rest nodes. The new pseudo code of this new DFG is shown in Figure 4(c), where the operation "A[i + 1] = TaskA(E[i − 1]);" can



(a)



(b)

time	P0		P1	
	task	Temperature	task	Temperature
0~95	A	50~80.84	ID	50
95~155	ID	80.84~50.18	B	50~71.75
155~220			C	71.75~79.98
220~300			D	79.98~71.35
300~350	E	50.18~67.89	ID	71.35~56.11

(c)

Task	Peak temperature	
	Iteration 1	Iteration 2
A	80.84	82.50
B	71.75	71.88
C	79.98	80.01
D	71.35	71.35
E	67.89	67.89

(d)

Fig. 3. List Scheduling. (a) The DFG. (b) The static DAG. (c) The schedule generated by list scheduling. (d) The peak temperature (°C) of each task.

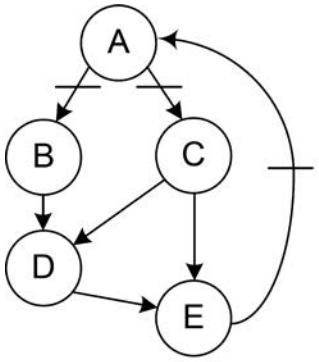
be placed anywhere in the loop, due to its independence. More details of the rotation scheduling are shown in Algorithm 7 of Section 5.

In this case, we can first assign the dependent nodes (B to E) to cores with the same policy used in the list scheduling. Tasks B, C and D are assigned to core P1 at the time slot of [0, 205]. And task E is scheduled to run on core P0 at [205, 255]. In this partial schedule, we discover that there are three time slots at which we can schedule task A. One is the idle gap of core P0 at [0, 205], another is the time slot after task E is done (time 255) on P0, and the last one is time slot after task D (time 205) on P1. Because the peak temperature of task A is the lowest when running in the idle gap of core P0 at [0, 205], this time slot is selected. Task A runs after the last iteration of task E, so the longer the idle gap between them, the cooler the initial temperature at which task A starts. Thus, we schedule task A's starting time at 110. A schedule is shown in Figure 4(d). In this schedule, the peak temperature is 81°C when task A is running in the second iteration (see Figure 4(e)). Our approach reduces the peak temperature by 1.5°C. Moreover, the total execution time of one iteration is only 255, while the total execution time generate by list scheduling is 350.

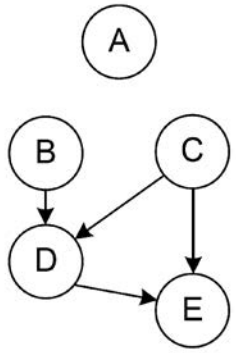
In the next section, we will discuss our thermal-aware task scheduling algorithm that deeply explores the solution space to find the good schedule meeting the real-time constraints.

5. THERMAL-AWARE TASK SCHEDULING ALGORITHM

In this section, we propose an algorithm, TARS (*Thermal-Aware Rotation Scheduling*), to solve *the minimum peak temperature without violating real-time constraints problem*. By repeatedly rotating down delays in DFG, more flexible static DAGs are generated.



(a)



(b)

```
A[2]=TaskA(E[0]);
for(i=2;i<N-1;i++) {
   B[i]=TaskB(A[i]);
   C[i]=TaskC(A[i]);
   A[i+1]=TaskA(E[i-1]);
   D[i]=TaskD(B[i],C[i]);
   E[i]=TaskE(C[i],D[i]); }
B[N]=TaskB(A[N]);
C[N]=TaskC(A[N]);
D[N]=TaskD(B[N],C[N]);
E[N]=TaskE(C[N],D[N]);
```

(c)

time	P0		time	P1	
	task	Temperature		task	Temperature
0~110	ID	50	0~60	B	50~71.75
			60~125	C	71.75~79.98
110~205	A	50~80.04	125~205	D	79.98~71.35
205~255	E	80.04~67.89	205~255	ID	71.35~56.11

(d)

Task	Peak temperature	
	Iteration 1	Iteration 2
A	80.84	81.00
B	71.75	73.12
C	79.98	80.25
D	71.35	71.39
E	76.67	76.72

(e)

Fig. 4. Rotation Scheduling. (a) The retimed DFG. (b) The new static DAG. (c) The pseudo code of the retimed DFG. (d) The schedule generated by our proposed algorithm. (e) The peak temperature (°C) of each task.

For each static DAG, a greedy heuristic approach is used to generate a schedule with minimum peak temperature. Then, the best schedule is selected among the schedules generated previously.

5.1. The TARS Algorithm

In the TARS algorithm shown in Algorithm 1, we will try to rotate the original DFG by R times. In each rotation, we get the static DAG from the rotated DFG by deleting the delay edges in DFG. A static DAG usually consists of two kinds of tasks. One kind of tasks are the tasks with dependencies, like the tasks B, C, D, and E in Figure 4(b). The other kind of tasks are the independent tasks, like the task A in Figure 4(b). The independent tasks do not have any intra-iteration relation with other tasks. Below, we first present two algorithms, the PTMM algorithm and the PTLS algorithm, to assign tasks with dependencies.

5.2. The PTMM Algorithm

The *Peak Temperature Min-Min* (PTMM) algorithm is designed to schedule the tasks with dependencies. Min-Min is a popular greedy algorithm [Ibarra and Kim 1977]. The original Min-Min algorithm does not consider the dependencies among tasks. Therefore, in the Min-Min baseline algorithm used in this paper, we need to update the assignable task set in every step to maintain the task dependencies. We define *the assignable task* as the unassigned task whose predecessors all have been assigned. Since the temperatures of the cores in a core stack are highly correlated in 3D CMP, we need to schedule tasks with consideration of vertical thermal impacts. When we

ALGORITHM 1: The TARS algorithm

Input: A DFG, the rotation times R.
Output: A schedule S, the retiming function r.

```
1: rot_cnt ← 0 /*Rotation counter.*/
2: Initial S_min, r_min, PT_min, r_cur /*The optimal schedule, the according retiming function, the
   according peak temperature and the current retiming function*/
3: while rot_cnt < R do
4:    Transform the current DFG to a static DAG
5:    Schedule tasks with dependencies. /* using the PTMM algorithm or PTLS algorithm */
6:    Schedule independent tasks, using the MPTSS algorithm
7:    Scale the frequencies, using the PPS algorithm /* A schedule S_cur for the current DFG is
      generated */
8:    Get the peak temperature PT_cur of the current schedule
9:    if PT_cur < PT_min and S_cur meets the real-time constraint then
10:      S_min ← S_cur, r_min ← r_cur, PT_min ← PT_cur
11:   end if
12:   Use RS algorithm to get a new retiming function r_cur
13:   Get the new DFG based on r_cur
14:   R ← R + 1
15: end while
16: Output the S_min, r_cur
```

consider assigning a task T_i to core C_j, we calculate the peak temperatures of cores in the core stack of C_j during the T_i running on C_j, based on Eq. (8).

Let $T_{max}(i, j)$ be the maximum value of the peak temperatures in the core stack. When we decide the assigning of T_i, we calculate all the $T_{max}(i, j)$, $for\ j = every\ core$. Due to the fact that the available times and the power characteristics of different cores in the same core stack may not be identical, the peak temperatures of the given core stack may be various when assigning the same task to different cores of this core stack respectively. Let C_{min} be the core with minimum $T_{max}(i, j)$. In each step in PTMM, we first find all the assignable tasks. Then we will form a pair $<T_i, C_{min}>$ for every assignable task. Only the $<T_i, C_{min}>$ pair which gives the minimum $T_{max}(i, j)$ will be assigned accordingly. And we also schedule the start execution time of T_i as the time when the predecessors of T_i are done and core C_{min} is ready. The PTMM is shown as Algorithm 2.

5.3. The PTLS Algorithm

The *Peak Temperature List Scheduling* (PTLS) algorithm is another algorithm that we use to schedule the tasks with dependencies. In the PTLS, we first list the tasks in a priority list considering the data dependencies (see the Algorithm 3). Some definition used in the *Task Listing* (TL) algorithm is provided as following. The *Earliest Start Time* (EST) and the *Latest Start Time* (LST) of a task are shown as in Eqs. (9) and (10). The entry-tasks have EST equals to 0. And the LST of the exit-tasks equal to their EST.

$$EST(i) = \max_{m \in pred(i)} \{EST(m) + AT(m)\} \tag{9}$$

$$LST(i) = \min_{m \in succ(i)} \{LST(m)\} - AT(i). \tag{10}$$

$AT(i)$ is the average execution time of task i. The critical node (CN) is a set of vertices in the DAG of which EST and LST are equal. After a priority list is generated, we assign the tasks, in the order of the priority list, to the core with the minimum peak temperature (see Algorithm 4).

ALGORITHM 2: The PTMM algorithm

Input: A static DAG G, m different cores, EP matrix.
Output: A schedule generated by PTMM.

```
Form a set of assignable tasks P
while P is not empty do
  for t = every task in P do
    for j = 1 to m do
      Calculate the peak temperatures of cores in the core stack of C_j, assuming t is
      running on C_j. And find the minimum peak temperature T_max(t, j)
    end for
    Find the core C_min(t) giving the minimum peak temperature T_max(t, j)
    Form a task-core pair as <t, C_min(t)>
  end for
  Choose the task-core pair <t_min, C_min(t_min)> which gives the minimum T_max(t, C_min(t))
  Assign task t_min to core C_min(t_min)
  Schedule the start time of t_min as the time when all the predecessors of t_min are finished
  and C_min(t_min) is ready
  Update the assignable task set P
  Update time slot table of core C_min(t_min) and the expected finish time of t_min
end while
```

ALGORITHM 3: The TL algorithm

Input: A static DAG, Average execution time AT of every task in the DAG.
Output: An assigning order of tasks P.

```
/*List tasks with dependencies*/
Calculate the EST and the LST of every task which has dependencies
Empty list P and stack S, and pull all tasks with dependencies in the list of task U
Push the CN task into stack S in the decreasing order of their LST, and remove them
from U
while The stack S is not empty do
  if top(S) has immediate predecessors in U then
    S ←the immediate predecessor with least LST
    Remove this immediate predecessor from U
  else
    P ← top(S)
    Pop top(S)
  end if
end while
/*List independent tasks*/
Push independent tasks in P in the decreasing order of their power consumptions.
```

5.4. The MPTSS Algorithm

Using one of the PTMM and the PTLS algorithm, we can get a partial schedule, in which the tasks with dependencies are assigned and scheduled. We need to further assign the independent tasks in the static DAG. Since the independent tasks do not have any intra-iteration relations with others, they can be scheduled to any possible time slots of the cores. In the *Minimum Peak Temperature Slot Selection* (MPTSS) algorithm, we assign the independent tasks in the decreasing order of their power consumption. Tasks with larger power consumption likely generate higher temperatures. The higher assigning orders of these tasks, the better fitting cores these tasks will be assigned to, and probably the lower resulting peak temperature of the finial schedule.

ALGORITHM 4: The PTLS algorithm

Input: An priority list of tasks with dependencies P, m different cores, EP matrix.
Output: A schedule generated by MPT.

```
1: while The list P is not empty do
2:    t = top(P)
3:    for j = 1 to m do
4:       Calculate the peak temperatures of cores in the core stack of C_j, assuming t is running
         on C_j. And find the minimum peak temperature T_max(t, j)
5:    end for
6:    Find the core C_min giving the minimum peak temperature T_max(t, j)
7:    Assign task t to core C_min
8:    Schedule the start time of t as the time when all the predecessors of t are finished and
      C_min is ready
9:    Remove t from P
10:   Update time slot table of core C_min and the expected finish time of t
11: end while
```

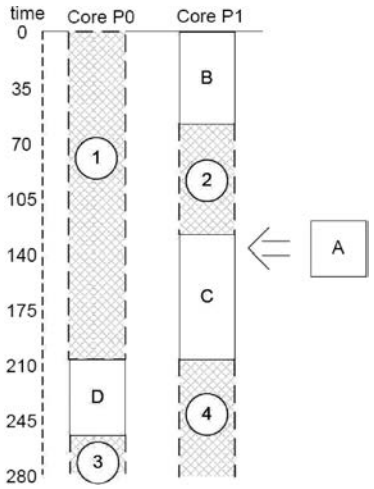


Fig. 5. An example of time slot set for an independent task.

Before we assign an independent task A, as shown in Figure 5, we first find all the idle slots among all cores, forming a time slot set TS. In the example shown in Figure 5, there are four time slots indicated with circled numbers for task A. Two of them, that is, time slot 1 and 2, are among the previously scheduled tasks. And the other two, that is, time slot 3 and 4, are at the end of cores' schedules of one iteration. The time slots that are not long enough for the execution of A will be removed from TS. Then, we calculate the peak temperature of the according core stack $T_{max}(A, core)$, which is defined in the PTMM algorithm, for every time slot. One problem arise here: since the remain time slots are long enough for the execution of A, we need to decide when to start the execution.

We use two different schemes here. The first one is the *As Early As Possible* (AEAP), which means the task T_i should be scheduled to start at the beginning of that time slot. The other one is *As Late As Possible* (ALAP), which means we should schedule the start execution time of the task T_i at a certain time so that T_i will finish at the end of the time slot. These two schemes result in different impacts on peak temperature.

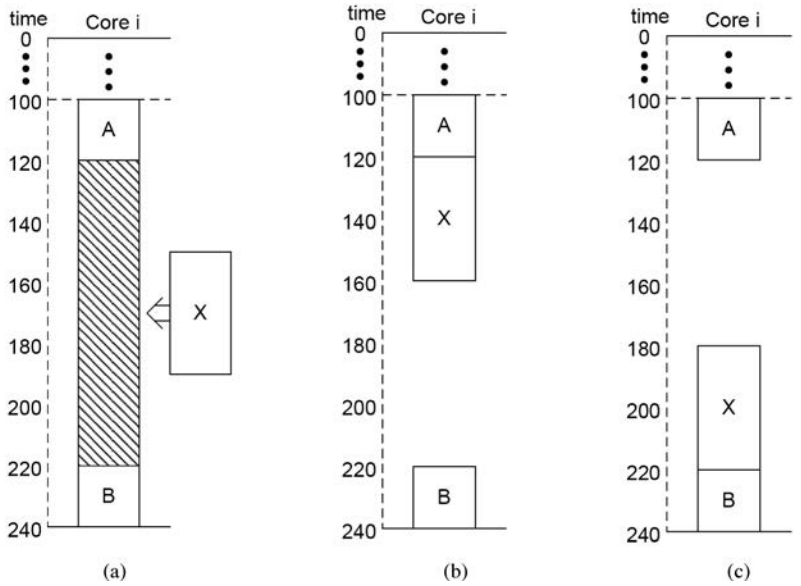


Fig. 6. (a) The task X is scheduled in a time slot in core i, (b) The task X is scheduled by the AEAP scheme, (c) The task X is scheduled by the ALAP scheme.

Let us assume we are considering scheduling task X to core i in the time slot, which is shown as a shadowed rectangle in Figure 6(a), and tasks A and B are previously scheduled on the beginning and the end of this time slot on core i. The AEAP scheme generates a time gap between X and B, as shown in Figure 6(b). The temperature of core i can be cooled down during this time gap, that is, 160 to 220. The ALAP scheme schedules X right before B without any time gap, as shown in Figure 6(c). So the initial temperature of B is lower with the AEAP scheme, that is, the schedule in Figure 6(b), than with the ALAP scheme, that is, the schedule in Figure 6(c), due to the cooling time gap (160 to 220) between the tasks X and B.

Given a certain execution time of B, lower initial temperature leads to lower peak temperature. In addition, if the power consumption of B is higher than the power consumption of X, the peak temperature of B is likely higher than the one of X, which means we should try to cool down B rather than X in this case. Implementing the AEAP in scheduling X can cool down the X at most here. On the other hand, the ALAP can create a time gap between X and the task A that is previously scheduled right before the time slot. This time gap, for example, the time gap 120 to 180, can reduce the initial temperature of X. So in the case where the power consumption of X is higher than the one of B, using ALAP can reduce the peak temperature of X. Thus, when we consider scheduling a task to a time slot, we will compare the power consumption of this task and the task previously scheduled right after this time slot. If the task being scheduled has more power consumption, we will use the ALAP scheme. Otherwise, the AEAP scheme will be implemented.

When we try to schedule tasks to the time slots which locates at the end of cores' schedules, we will determine which scheme, either AEAP or ALAP, will be used based on the power consumption comparison of this task and the task that will start first in the next iteration. For example, in Figure 5, when we try to schedule task A to time slot 4, we will compare the power consumptions of task A and B. We will schedule a large enough time slot for cooling down the task that needs more concern, that is, the

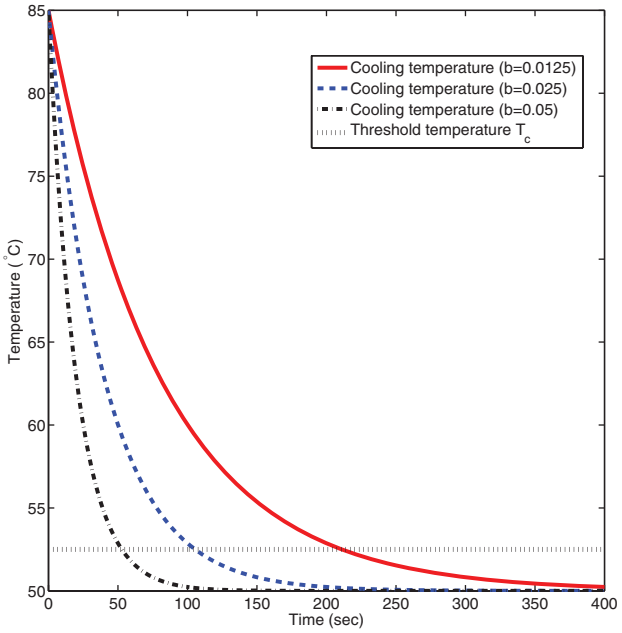


Fig. 7. Examples of cooling temperature on-chip. All three cooling temperatures start from the initial temperature of 85°C to the stable temperature of 50°C. We can observe that the cooling speeds in these three scenarios are slowing down dramatically near the threshold temperature T_C.

more power consuming one between the task to be scheduled and the task starting first in the next iteration.

Another question arises: how large the cool time slot should be scheduled? We will pre-determine a threshold cooling temperature T_c. Then we will create a cooling time slot large enough to let the more power consuming task cooling down to the threshold T_c, without violating the real-time constraint. The reason that we set the threshold temperature is that when the temperature of a core is cooling down, it drops dramatically at the beginning, as shown in Figure 7. However, it becomes stable as the core continues to cool down. Hence, if we try to cool down the core completely, it will take a significantly long time. As shown in Figure 7, if we just need to reduce the core's temperature to the threshold, that is, the horizontal dot line, it will be more time-efficient. We present our MPTSS algorithm in Algorithm 5.

5.5. The PPS Algorithm

Once we get a full schedule from the previous steps, we can further reduce the peak temperature by dynamic frequency assignment. We assume that the frequencies of different cores can be different and there are several frequencies options available for each core. From a given schedule, we can predict the task which causes the peak temperature. We can further decrease the peak temperature by changing the frequency assignment of the corresponding core when that task is running.

We propose our dynamic frequency assignment algorithm, called the *Peak Point Scaling* (PPS), in Algorithm 6. Given a schedule, we first find the task with the highest peak temperature over all the tasks. Then, the core frequency when running this task is set to one slower level. We calculate the period of this new schedule. If it meets the real-time constraint, this new schedule is acceptable. Otherwise, dynamic frequency scaling cannot reduce the peak temperature. If the new schedule is acceptable, then we

ALGORITHM 5: The MPTSS algorithm

Input: A partial schedule generated by PTMM, a set of independent tasks, m different cores, EP matrix.
Output: A schedule generated by MPTSS.

```
1: List independent tasks in a list P in the decreasing order of their power consumption
2: while The list P is not empty do
3:    t = top(P)
4:    Collect all the time slots which is long enough for t across all cores, form a time slot set
      TS.
5:    for Every time slot ts_i in TS do
6:       j ← the according core of ts_i
7:       Find the task t_next which is schedule to start right after ts_i on the core C_j.
8:       if Power(t) < Power(t_next) then
9:          Find the start time with the AEAP scheme
10:      else
11:         Find the start time with the ALAP scheme
12:      end if
13:      Get the T_max(t, j) /*similar to the one in PTMM*/
14:   end for
15:   Find the time slot ts_min giving the minimum peak temperature T_max(t, j)
16:   Assign task t to core C_min /*C_min is the core of time slot ts_min*/
17:   Schedule the start time of t in time slot ts_min based on the scheme selected in the if
      statement (line 8)
18:   Remove t from P
19:   Update time slot table of core C_min
20: end while
```

ALGORITHM 6: The PPS algorithm

Input: An initial schedule S_{init}, EP matrix, a real-time constraint TC
Output: A schedule generated by PPS.

```
1: S_temp ← S_init
2: while Period(S_temp) ≤ TC do
3:    S ← S_temp
4:    Find the task t_max generating the highest peak temperature in S_temp, and the core C_max
      which runs t_max
5:    if frequency of C_max when running t_max is the slowest level then
6:       Break
7:    end if
8:    Set the frequency of C_max when running t_max to one slower level
9:    Update S_temp
10: end while
11: Output S
```

find the task with the highest peak temperature in the new schedule, and repeat the frequency scaling again. This frequency scaling repeats until a schedule which violates the real-time constraint is generated. We output the last version of the acceptable schedules.

5.6. The RS Algorithm

At the end of each iteration of the TARS algorithm, we create a new DFG by rotating the current DFG. First, we need to form a set of rotation tasks. If a task is the first task scheduled on a core and there is at least one delay in each of its incoming edge, this task is a rotation task. The *Rotation Scheduling* (RS) algorithm is shown in Algorithm 7.

ALGORITHM 7: The RS algorithm

Input: An input DFG D_{in} and a schedule S based on D_{in}, a retiming function r.
Output: An output DFG D_{out} generated by rotation scheduling, a new retiming function r_{new}.
1: Form the set of rotation tasks RT based on D_{in} and S
2: **for** Every task t_i in RT **do**
3: Reduce one delay from every incoming edges of task t_i in D_{in}
4: Increase one delay from every outgoing edges of task t_i in D_{in}
5: $r(t_i) \leftarrow r(t_i) + 1$
6: **end for**
7: $D_{out} \leftarrow D_{in}$ and $r_{new} \leftarrow r$

Figure 8 shows an example of our RS algorithm. Assuming an initial DFG shown in Figure 8(a), we can transform the DFG into DAG by removing the edges with delays. Then a schedule is generated by the algorithms presented in the previous subsections.

In the first rotation, we can find the task A and C are the first tasks executed in two cores. So the rotation task set includes these two tasks. Since there is none delay on the incoming edge and the outgoing edge of task C, we keep the edges of task C unchanged. For task A, there are three delays on its incoming edge, that is, edge e_{EA}. Thus, in this rotation, we reduce one delay on edge e_{EA}, and increase the delays of all three outgoing edges of task A by one, respectively, as shown in Figure 8(b). We can find that task A now becomes independent in the corresponding DAG. A new schedule is generated based on this new DAG. In this schedule, task B and C are the first tasks in two cores. These two tasks form the set of rotation tasks in the next rotation.

In the second rotation, the delays of the incoming edges of task B and C, that is, e_{AB}, e_{AC}, are all reduced by one. The outgoing edges of task B and C, that is, e_{BD}, e_{BE}, and e_{CE}, increase their delays by one, as shown in Figure 8(c). According to this new DFG, task D and E become independent. The third schedule is created in this rotation.

As shown in this example, the RS algorithm can redistribute the delays in the DFG. Therefore, various DAGs can be reached. In these various DAGs, different tasks become independent, which leads to diverse scheduling orders of tasks and different schedules. As we implement the RS algorithm at the end of each iteration of our TARS algorithm, and we repeat the TARS algorithm for a predetermined number of iterations, we can select the rotations with the best schedule among a number of schedules in the sense of reducing peak temperature.

6. EXPERIMENTAL RESULTS

In this section, we present the experimental results of our algorithms. We develop our experiments as follows: we first use a precise microprocessor simulator, Wattch 1.0.2 [Brooks et al. 2000], to get the execution and power characteristics of a set of benchmarks. Then we generate a number of random DFGs consisting of this set of benchmarks. Task schedules and power traces are created by our algorithm. We input these schedules and power traces into a thermal analysis simulator, called Hotspot 4.1 [Skadron et al. 2004]. Finally, we evaluate our algorithms with the comprehensive thermal analysis generated by Hotspot 4.1. All experiments are conducted on Linux machine equipped with an Intel Core 2 Duo E8400 CPU and 3GB of RAM.

6.1. Experiment Setup

The 3D CMP architecture simulated in our experiments is a two-layer front-to-back architecture. There are eight Alpha 21264 (EV6) microprocessor cores in each layer with configuration as Table I. We use per core DFVS in our simulation with three

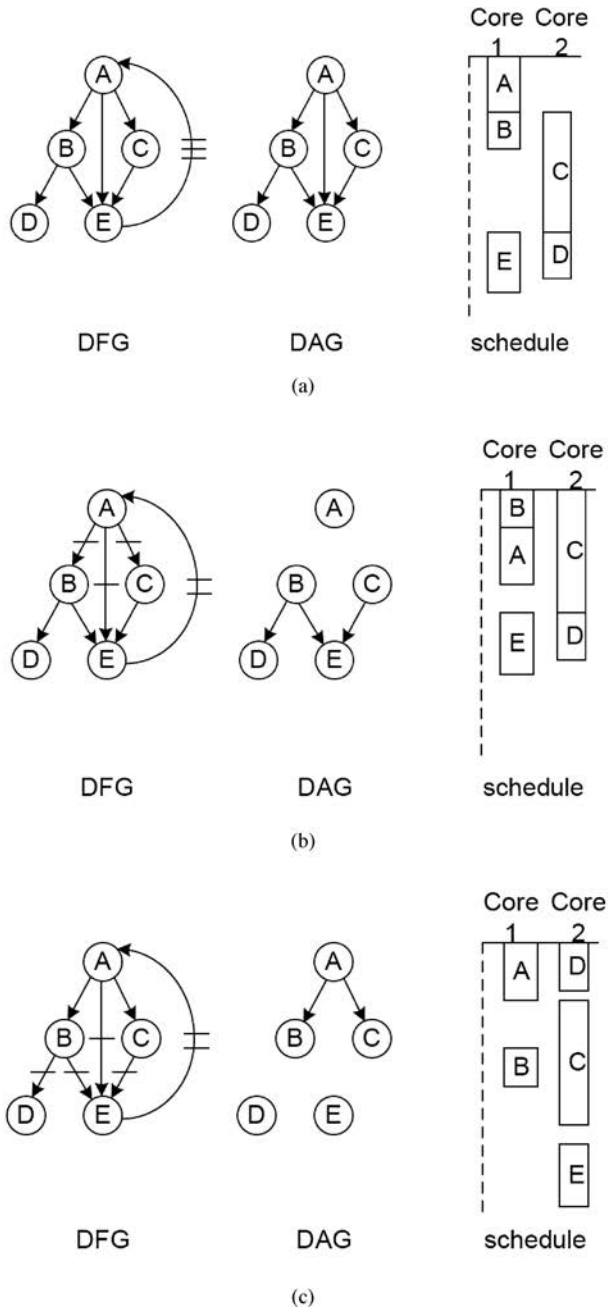


Fig. 8. (a) The initial DFG, the corresponding DAG and schedule. (b) The rotated DFG in the first rotation, the corresponding DAG and schedule. (c) The rotated DFG in the second rotation, the corresponding DAG and schedule.

Table I. Configuration of Alpha Cores

Processor core	Alpha 21264
Core technology	65nm
Nominal frequency	5GHz
L1 data cache	64K, 2-way
L1 instruction cache	64K, 2-way
L2 cache	2M

Table II. Thermal Parameter for Hotspot

Layer	Conductivity	Capacitance per unit volumn
Silicon	100 $W/(m \cdot K)$	1.75×10^6 $J/(m^3 \cdot K)$
TIM	4 $W/(m \cdot K)$	4.0×10^6 $J/(m^3 \cdot K)$
Copper	400 $W/(m \cdot K)$	3.55×10^6 $J/(m^3 \cdot K)$

Table III. Temperature Parameter Setting

Temperature parameter	Value
Ambient temperature	35°C
Initial temperature	55°C
Critical temperature	85°C

DVFS levels (3.88GHz, 4.5GHz, and 5GHz) configured based on the parameters of Alpha 21264 [Kessler 1999].

We choose the SPEC CPU 2000 benchmark suite and the MiBench benchmark suite in our experiment. The execution time and the power consumption of each benchmark on Alpha core are tested through the Wattch 1.0.2 simulator with the above configuration. For each benchmark, we run it under those three DFVS levels via out-of-order mode to get the task characteristic of this benchmark. We generate 10 random DFG-based applications. The tasks in these applications are randomly selected from the SPEC2000 and the MiBench benchmarks. For each application, we set the real-time constraint TC (i.e., deadline) as follows:

$$TC = \frac{\sum_{i=1}^{N} t_i}{P} \times c, \tag{11}$$

where N is the number of tasks in this application, t_i is the execution of time of task i under the highest frequency, P is the total number of cores, that is, 16 in our simulation, and c is a constant which is set to 5, generating neither too tight nor too loose constraints.

The thermal simulation is conducted in the Hotspot 4.1 simulator by using the power consumption traces created by our program. In the Hotspot 4.1 simulator, the lateral and vertical thermal interactions among adjacent core are all carefully considered and modeled. As we mentioned previously, the architecture model used in the Hotspot simulator is a two-layer architecture, in which the thickness of the top layer (the one far from the heat sink) is 50μm, and the thickness of the bottom (the one close to the heat sink) is 300μm. There is a *Thermal Interface Material* (TIM) layer between these two layers. The core size is 4mm $\times$ 8mm. Some other parameters is listed in Table II. We also set the temperature parameters as shown in Table III [Liu et al. 2010].

6.2. Peak Temperature

As our algorithms are to reduce the peak temperature in 3D CMP architectures, we show the average peak temperature of all 16 cores over 10 applications in Figure 9. By

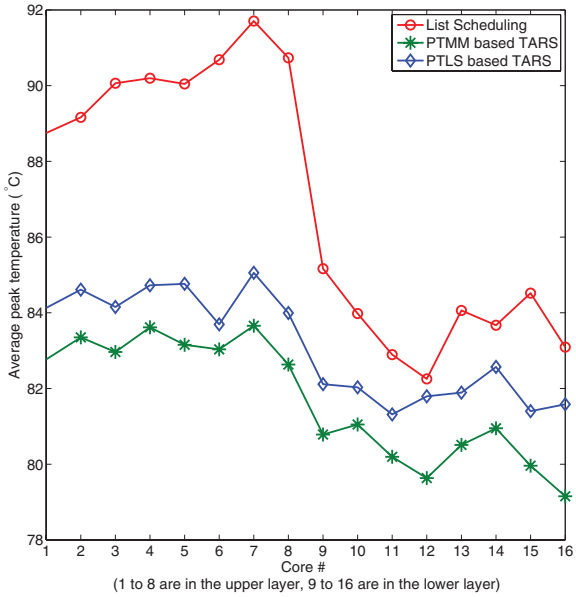


Fig. 9. Core peak temperatures comparison. The "Core #" in the x-axle represents the IDs of the sixteen cores, where cores 1 to 8 are in the upper layer and the cores 9 to 16 are in the lower layer.

comparing the result of list scheduling, we find that both of our algorithms can reduce the peak temperatures. The PTLS based TARS reduces up to 7°C. And the PTMM based TARS is even better, reducing up to 8.1°C. Both the peak reductions happen on the cores in the upper layer. For the cores in the top layer (core 1 to 8), the peak temperatures are consistently higher than the ones in bottom layer (core 9 to 16). This result is aligned to our online thermal prediction model. The peak temperatures of top layer cores is around 83°C with our PTMM based TARS algorithm, about 84.5°C with our PTLS based TARS algorithm, and about 90°C with the list scheduling. With the two phases consideration in the PTMM, that is, the Min-Min initial scheduling algorithm, more global information is used in making the assigning decisions. Thus it generates better initial schedules leading to better performance than our PTLS based TARS algorithm does.

Larger improvements are made in the top layer cores. The reason is that in our proposed algorithm, more effort is made in reducing the temperature of the hottest core, which is usually located in the top layer. Even though the improvements for cores in the bottom layer are not as significant as the ones in top layer, lower peak temperatures are achieved, due to the more flexible execution order explored in our algorithm and less impact from the aligned cores on the top layer. The reduction of peak temperature in the bottom layer is about 4.5°C with our PTMM based TARS algorithm, about 3.1°C with our PTLS based TARS algorithm.

6.3. Temperature Violations

In this section, we compare the schedules in the sense of avoiding or minimizing the number of temperature violations, which is shown in Figure 10. We define the temperature violation as the situation where the core's temperature is higher than the critical temperature. The differences of temperature violations of cores depend on a few factors, such as the workloads of cores, the location relationship with other cores. The cores 5, 6, and 7 have more temperature violations than that of cores 10–13. The

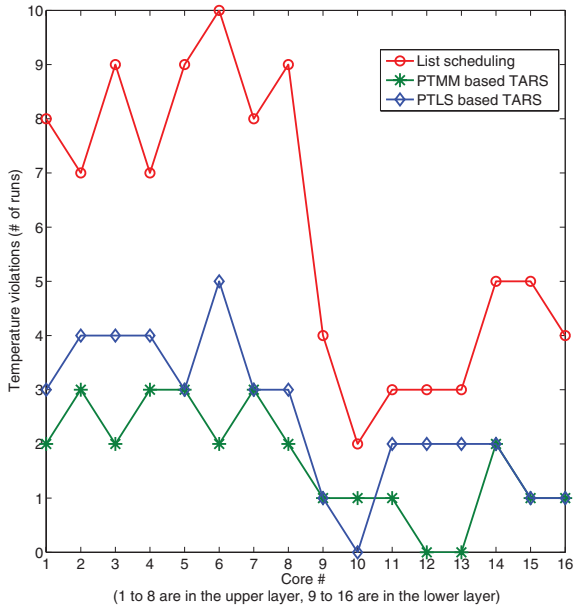


Fig. 10. Core temperature violations comparison. The "Core #"s in the x-axle represent the IDs of the sixteen cores, where cores 1 to 8 are in the upper layer and the cores 9 - 16 are in the lower layer. Out of the 10 runs in the experiment, the temperature violations are number of runs in which the corresponding core has the peak temperature higher than the temperature constraint.

reason is that the cores 5, 6, and 7 is on the upper layer of the 3D CMP. The cores in the top layer are more likely to have higher temperature than the critical temperature. Since more efforts are made to reduce the temperature of the hottest core in our TARS algorithms, our TARS algorithms can dramatically reduce the number of times of temperature violations in the top layer cores. Up to 80% temperature violations in the list scheduling are avoided in the top layer. Aligned to the result of the above subsection, the PTMM based TARS algorithm outperforms the PTLS based TARS algorithm.

For the cores in the bottom layer, only a small number of of violations occur. In both TARS algorithms, there is one core that never has temperature higher than the critical cores. No more than two violations happen in any core in the bottom layer. In summary, both our TARS algorithms can reduce the temperature violations in both the top layer and the bottom layer.

7. CONCLUSION

In this article, we presented an online 3D CMP temperature prediction model for multimedia embedded systems. We also proposed our real-time constrained task scheduling algorithms, the TARS algorithms, to reduce peak temperature in a 3D CMP. By considering the the inter-iteration data dependencies and frequencies assignment collaboratively, our proposed TARS algorithms can significantly reduce the peak temperature on chip and avoid most of the temperature violations. Our simulation results showed that our TARS algorithms can reduce peak temperature by 8.1°C, and avoid up to 80% violations in the top layer and up to 100% violations in the bottom layer.

Our future works are twofold: (1) we will investigate the implementation of stochastic approaches in our CMP temperature prediction models; and (2) we will also further consider the priorities of tasks in our task scheduling algorithms.

REFERENCES

ALLEC, N., HASSAN, Z., SHANG, L., DICK, R. P., AND YANG, R. 2008. Thermalscope: Multi-scale thermal analysis for nanometer-scale integrated circuits. In *Proceedings of the ACM/IEEE International Conference on Computer Aided Design (ACM/IEEE ICCAD)*. 75–82.

AYOUB, R. AND ROSING, T. S. 2009. Predict and act: dynamic thermal management for multi-core processors. In *Proceeding of the 2009 International Symposium on Low Power Electronics and Design (ISLPED'09)*. ACM, 99–104.

BLACK, B., ANNAVARAM, M., BREKELBAUM, N., DEVALE, J., JIANG, L., LOH, G. H., MCCAULE, D., MORROW, P., NELSON, D. W., AND PANTUSO, D. 2006. Die stacking (3D) microarchitecture. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*. 469–479.

BROOKS, D., TIWARI, V., AND MARTONOSI, M. 2000. WATTCH: A framework for architectural-level power analysis and optimizations. In *Proceedings of the IEEE Annual International Symposium on Computer Architecture (IEEE ISCA)*. 83–94.

CHAO, L.-F., LAPAUGH, A., AND SHA, E. H.-M. 1997. Rotation scheduling: A loop pipelining algorithm. *IEEE Trans. Comput. Aided Design Integ. Circ. Syst. 16,* 3 (Mar.), 229–239.

CHAPARRO, P., GONZÁLEZ, J., CAI, Q., AND CHRYSLER, G. 2009. Dynamic thermal management using thin-film thermoelectric cooling. In *Proceedings of the 2009 International Symposium on Low Power Electronics and Design (ISLPED'09)*. ACM, 111–116.

COSKUN, A., ROSING, T., AND GROSS, K. 2008. Proactive temperature balancing for low cost thermal management in MPSoCs. In *Proceedings of the ACM/IEEE International Conference on Computer-Aided Design (ACM/IEEE ICCAD)*.

COSKUN, A. K., AYALA, J. L., ATIENZA, D., ROSING, T. S., AND LEBLEBICI, Y. 2009. Dynamic thermal management in 3D multicore architectures. In *Proceedings of the ACM/IEEE Conference and Exhibition on Design, Automation, and Test in Europe (DATE)*. 1410–1415.

HAN, Y., KOREN, I., AND MORITZ, C. A. 2005. Temperature aware floorplanning. In *Proceedings of the Workshop on Temperature-Aware Computer Systems*.

IBARRA, O. H. AND KIM, C. E. 1977. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *J. ACM 24,* 2, 280–289.

JEDEC. 2009. Failure mechanisms and models for semiconductor devices. *http://www.jedec.org*.

KESSLER, R. E. 1999. The Alpha 21264 microprocessor. *IEEE Micro 19,* 2, 24–36.

LIN, C., YANG, C., AND KING, K. 2009. PPT: joint performance/power/thermal management of DRAM memory for multi-core systems. In *Proceedings of the 2009 International Symposium on Low Power Electronics and Design (ISLPED'09)*. ACM, 93–98.

LIU, S. AND QIU, M. 2010. Thermal-aware scheduling for peak temperature reduction with stochastic workloads. In *Proceedings of the IEEE Real-Time and Embedded Technology and Appreciation Symposium (IEEE/ACM RTAS)*.

LIU, S., ZHANG, J., WU, Q., AND QIU, Q. 2010. Thermal-aware job allocation and scheduling for three dimensional chip multiprocessor. In *Proceedings of the IEEE International Symposium on Quality Electronic Design*. 390–398.

MOSSE, D., AYDIN, H., CHILDERS, B., AND MELHEM, R. 2000. Compiler-assisted dynamic power-aware scheduling for real-time applications. In *Proceedings of the Workshop on Compilers and Operating Systems for Low-Power*.

MULAS, F., PITTAU, M., BUTTU, M., CARTA, S., ACQUAVIVA, A., BENINI, L., AND ATIENZA, D. 2008. Thermal balancing policy for streaming computing on multiprocessor architectures. In *Proceedings of the ACM/IEEE Conference and Exhibition on Design, Automation, and Test in Europe (DATE)*. 734–739.

NOOKALA, V., LILJA, D. J., AND SAPATNEKAR, S. S. 2006. Temperature-aware floorplanning of microarchitecture blocks with IPC-power dependence modeling and transient analysis. In *ACM/IEEE ISLPED*. 298–303.

PATHAK, M. AND LIM, S. 2008. Thermal-aware steiner routing for 3D stacked ICs. In *Proceedings of the ACM/IEEE International Conference on Computer-Aided Design (ICCAD)*. 205–211.

PUTTASWAMY, K. AND LOH, G. 2007. Thermal herding: microarchitecture techniques for controlling hotspots in high-performance 3d-integrated processors. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 193–204.

QIU, M., YANG, L., SHAO, Z., AND SHA, E. H.-M. 2010. Dynamic and leakage energy minimization with soft real-time loop scheduling and voltage assignment. *IEEE Trans. TVLSI Syst. 18*, 3, 501–504.

QIU, M., JIA, Z., XUE, C., SHAO, Z., AND SHA, E. H. M. 2007. Voltage assignment with guaranteed probability satisfying timing constraint for real-time multiproceesor DSP. *J. VLSI Sig. Proc. 46,* 1, 55–73.

QIU, M. AND SHA, E. H.-M. 2009. Cost minimization while satisfying hard/soft timing constraints for heterogeneous embedded systems. *ACM Trans. Design Automat. Electron. Syst. 14,* 2, 1–30.

QIU, M., YANG, L., SHAO, Z., AND SHA, E. H.-M. 2009. Rotation scheduling and voltage assignment to minimize energy for SoC. In *Proceedings of the International Conference on Computational Science and Engineering*. 48–55.

SANKARANARAYANAN, K., VELUSAMY, S., STAN, M., AND SKADRON, K. 2005. A case for thermal-aware floorplanning at the microarchitectural level. *IEEE Trans. Comput. Aided Des. Integ. Circ. Syst. 7*, 1–16.

SHIN, D., KIM, J., AND LEE, S. 2001. Intra-task voltage scheduling for low-energy, hard real-time applications. *IEEE Des. Test Comput. 18,* 2, 20–30.

SHIVLE, S., CASTAIN, R., SIEGEL, H. J., ET AL. 2004. Static mapping of subtasks in a heterogeneous ad hoc grid environment. In *Proceedings of the 13th IEEE Heterogeneous Computing Workshop (HCW 2004)*.

SHIVLE, S., SIEGEL, H. J., MACIEJEWSKI, A. A., ET AL. 2006. Static allocation of resources to communicating subtasks in a heterogeneous ad hoc grid environment. *J. Parall. Distrib. Comput. 66,* 4, 600–611.

SKADRON, K., STAN, M., SANKARANARAYANAN, K., HUANG, W., VELUSAMY, S., AND TARJAN, D. Mar. 2004. Temperature-aware microarchitecture: Modeling and implementation. *ACM Trans. Architect. Code Optim. 1,* 1, 94–125.

TIAN, Y. AND EKICI, E. 2007. Cross-layer collaborative in-network processing in multihop wireless sensor networks. *IEEE Trans. Mob. Comput. 6,* 3, 297–310.

TIAN, Y., EKICI, E., AND OZGUNER, F. 2005. Energy-constrained task mapping and scheduling in wireless sensor networks. In *Proceedings of the IEEE International Conference on Mobile Adhoc and Sensor Systems Conference*. 211–218.

TOPOL, A. W., LA TULIPE JR., D. C., AND SHI, L. 2006. Three-dimensional integrated circuits. *IBM J. Res. Development 50,* 4/5, 491–506.

WEISER, M., WELCH, B., DEMERS, A., AND SHENKER, S. 1994. Scheduling for reduced CPU energy. In *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation*.

YAO, F., DEMERS, A., AND SHENKER, S. 1995. A scheduling model for reduced CPU energy. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS'95)*. 374–382.

ZHOU, P., MA, Y., LI, Z., DICK, R. P., SHANG, L., ZHOU, H., HONG, X., AND ZHOU, Q. 2008. 3D-STAF: scalable temperature and leakage aware floorplanning for three-dimensional integrated circuits. In *Proceedings of the ACM/IEEE International Conference on Computer-Aided Design (ICCAD)*. 590–597.

ZHOU, X., YANG, J., XU, Y., ZHANG, Y., AND ZHAO, J. 2010. Thermal-aware task scheduling for 3D multicore processors. *IEEE Trans. Panal. Distr. Syst. 21,* 1 (Jan.), 60–70.

ZHU, C., GU, Z., SHANG, L., DICK, R. P., AND JOSEPH, R. 2008. Three-dimensional chip-multiprocessor run-time thermal management. *IEEE Trans. Comput. Aided Des. Integ. Circ. Syst. 27,* 8 (Aug.), 1479–1492.

Received November 2010; revised March 2011; accepted March 2011