Frontiers of Information Technology & Electronic Engineering
www.zju.edu.cn/jzus; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Schedule refinement for homogeneous multi-core processors in the presence of manufacturing-caused heterogeneity*

Zhi-xiang CHEN†[1,2], Zhao-lin LI[3,4], Shan CAO[2,5], Fang WANG[3,4], Jie ZHOU[1]

([1]*Department of Automation, Tsinghua University, Beijing 100084, China*)

([2]*Institute of Microelectronics, Tsinghua University, Beijing 100084, China*)

([3]*Research Institute of Information Technology, Tsinghua University, Beijing 100084, China*)

([4]*Tsinghua National Laboratory for Information Science and Technology, Tsinghua University, Beijing 100084, China*)

([5]*The School of Information and Electronics, Beijing Institute of Technology, Beijing 100084, China*)

†E-mail: chen-zx10@mails.tsinghua.edu.cn

Received Feb. 1, 2015; Revision accepted Aug. 26, 2015; Crosschecked Nov. 4, 2015

Abstract: Multi-core homogeneous processors have been widely used to deal with computation-intensive embedded applications. However, with the continuous down scaling of CMOS technology, within-die variations in the manufacturing process lead to a significant spread in the operating speeds of cores within homogeneous multi-core processors. Task scheduling approaches, which do not consider such heterogeneity caused by within-die variations, can lead to an overly pessimistic result in terms of performance. To realize an optimal performance according to the actual maximum clock frequencies at which cores can run, we present a heterogeneity-aware schedule refining (HASR) scheme by fully exploiting the heterogeneities of homogeneous multi-core processors in embedded domains. We analyze and show how the actual maximum frequencies of cores are used to guide the scheduling. In the scheme, representative chip operating points are selected and the corresponding optimal schedules are generated as candidate schedules. During the booting of each chip, according to the actual maximum clock frequencies of cores, one of the candidate schedules is bound to the chip to maximize the performance. A set of applications are designed to evaluate the proposed scheme. Experimental results show that the proposed scheme can improve the performance by an average value of 22.2%, compared with the baseline schedule based on the worst case timing analysis. Compared with the conventional task scheduling approach based on the actual maximum clock frequencies, the proposed scheme also improves the performance by up to 12%.

Key words: Schedule refining, Multi-core processor, Heterogeneity, Representative chip operating point
doi:10.1631/FITEE.1500035 **Document code:** A **CLC number:** TP302

1 Introduction

1.1 Background

Homogeneous multi-core processors are becoming widely used for computation-intensive embedded applications. Typical homogeneous multi-core processors include Stanford's Imagine (Khailany *et al.*, 2001), MIT's RAW (Taylor *et al.*, 2002), Tilera's Tile64 (Bell *et al.*, 2008), etc., in which all cores

* Project supported by the National Natural Science Foundation of China (Nos. 61225008, 61373074, and 61373090), the National Basic Research Program (973) of China (No. 2014CB349304), the Specialized Research Fund for the Doctoral Program of Higher Education, the Ministry of Education of China (No. 20120002110033), and the Tsinghua University Initiative Scientific Research Program

ORCID: Zhi-xiang CHEN, http://orcid.org/0000-0001-7986-030X

have the same architecture and run at the same clock speed. With the development of IC technology, the performance improvement of multi-core processors benefits from continuous down-scaling of feature size in the manufacturing process. However, parameter variations within the manufacturing process, which are called process variations, are inevitable when the sizes of transistors are respectively shrunk all the way to their feasible limits with imperfect lithographic equipment and material processing systems (Dietrich and Haase, 2012). In fact, the continuous down-scaling of the feature size has resulted in significant variations, both intra-die and inter-die, of gate length, threshold voltage, and mobility, which are important to the transistor behavior (Zhao *et al.*, 2009). According to the information about process variation reported by ITRS (2014), effective channel length and gate-oxide thickness can deviate up to 12% of the nominal values. As a result, cores in homogeneous multi-core processors can run at different maximum operating frequencies, which is called manufacturing-caused heterogeneity, after fabrication due to process variations, though they are designed to have the same architecture and run at the same maximum operating frequency determined by the worst case timing analysis during the design stage. For example, experimental data reveals 40% spread in the frequencies of cores within a chip in 32 nm technology (Aguilera *et al.*, 2014).

The maximum operating frequencies (FMAXs) of all cores are determined by the slowest core in conventional homogeneously designed multi-core processors. The unified FMAX makes the task scheduling simple but has a heavy performance penalty because the manufacturing-caused heterogeneity is neglected. In this work, 'scheduling' means allocating executing core and executing order for each task, and 'schedule' means the result of scheduling. As shown in Fig. 1, the schedule sch_1 for the task graph of the application in Fig. 1a is shown in Fig. 1c with all cores operating at 200 MHz. In the schedule, v_i, v_i', and v_i'' $(1 \le i \le 4)$ are different instances of the same task. In the presence of multiple FMAXs for the cores, each different combination of the FMAXs for the cores on a chip is denoted as a chip operating point, which is the same as the term 'scenario' in Khodabandeloo *et al.* (2014) and is denoted as op. As shown in Fig. 1d, in the traditional scheduling scheme, sch_1 is applied over the three chip operating points in Fig. 1b, which exhibit the same performance with the schedule interval of 0.35 s. Therefore, the increase of FMAX on core c_2, from 200 to 250 MHz, makes no performance improvement. Twenty percent of the computing ability of core c_2 is unexploited in the traditional scheduling scheme.

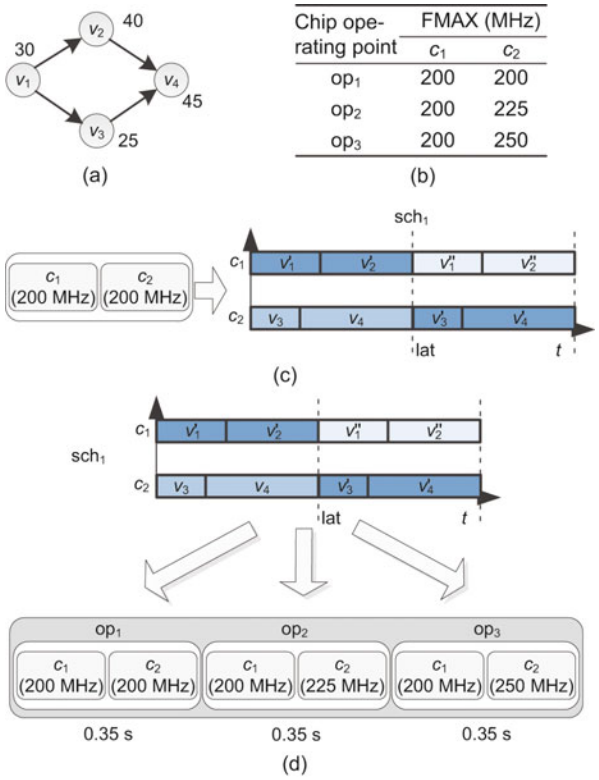

Fig. 1 A simple example: (a) an application with four tasks; (b) three chip operating points; (c) the traditional schedule; (d) traditional scheduling scheme

In the presence of manufacturing-caused heterogeneity, task scheduling has to take the heterogeneity into consideration to improve the performance for each chip. A straightforward approach is to generate an individual schedule for each chip according to the detected op. However, in the embedded domain where dynamic scheduling is almost impossible, applying this approach to masses of chips is too time-consuming, since the schedule should be generated externally for embedded systems. Another approach is to generate all optimal schedules for all chip operating points in advance and store them into the on-chip memory for each chip. However, the search space for matching the schedule with the chip is too large. Furthermore, the memory usage is too high, since embedded systems have limited on-chip

memory. Thus, both approaches are not feasible for embedded systems. Therefore, we propose an effective solution to schedule refinement, where a finite set of optimal schedules is stored into the on-chip memory and the best schedule is selected for each chip during booting.

1.2 Overview

In this work, heterogeneity-aware schedule refining (HASR) is proposed to efficiently exploit the manufacturing-caused heterogeneity to improve performance. We analyze and show how the actual maximum frequencies of cores are taken into consideration to obtain the effective schedule. The proposed HASR scheme generates multiple candidate schedules to achieve the most optimal expected performance by taking the distribution of heterogeneity into consideration. Meanwhile, a sampling method is adopted to deal with the problem of exponential growth in the quantity of operating points. HASR stores an appropriate number of schedules and the corresponding chip operating points in the on-chip memory and adopts a simple algorithm to bind the most appropriate schedule to the chip during booting. The binding is carried out during booting and thus has no negative impact on the performance. The effects of the number of tasks, the number of cores, and the number of candidate schedules on the performance improvement are analyzed. The experimental results show the effectiveness of the proposed HASR scheme.

2 Related work

This work deals with the schedule refining on multi-core processors in the presence of manufacturing-caused heterogeneity. The related research can be categorized into two types, traditional task scheduling on multiprocessor and process variation aware scheduling.

2.1 Traditional task scheduling

Task allocation and scheduling for multi-core processors has been extensively studied. Since precedence-constrained task allocation and scheduling has been proved to be an NP-complete problem (Ramamritham, 1995), heuristic algorithms, such as list scheduling (Topcuoglu *et al.*, 2002), genetic algorithms (Omara and Arafa, 2010), and ant colony algorithm (Ferrandi *et al.*, 2010), were widely used to quickly find a suboptimal solution. Two novel scheduling algorithms for a bounded number of heterogeneous processors were proposed in Topcuoglu *et al.* (2002) with an objective to simultaneously meet the requirements of high performance and fast scheduling time. Two genetic algorithms with some heuristic principles were introduced in Omara and Arafa (2010) to reduce the complexity of the optimization process and improve the performance. An ant colony optimization heuristic was presented in Ferrandi *et al.* (2010) to efficiently execute both scheduling and mapping to optimize the application performance. However, none of them considered the impact of manufacturing-caused heterogeneity.

2.2 Process variation aware scheduling

Task scheduling with process variation awareness is used to guide the design of a multi-core processor and aims at the optimization of performance yield, which is the percentage of manufactured chips satisfying the predefined performance requirement. Wang *et al.* (2011) scheduled tasks to enhance performance yield through statistical scheduling to mitigate the impact of process variations. Singhal and Bozorgzadeh (2008), Chon and Kim (2009), and Huang and Xu (2010) expanded the research in Wang *et al.* (2011) with other scheduling approaches. Both exhaustive and heuristic methods were proposed to achieve yield optimization for variation-aware task allocation of real-time streaming applications on a multi-core processor (Mirzoyan *et al.*, 2012). However, all these solutions are based on statistical timing analysis to optimize the performance yield rather than the performance. Momtzapour *et al.* (2010a; 2010b) used the genetic algorithm to find the best schedule that maximizes power-yield under the performance-yield constraint and extended their work for a deep investigation (Momtazpour *et al.*, 2013). Momtazpour *et al.* (2011) considered the problem of simultaneously choosing multi-processor system-on-a-chip (MPSoC) architecture and task allocation for energy optimization under a given performance constraint. However, all these solutions aim at power-yield and energy optimization under process variation. Khodabandeloo *et al.* (2014) presented a hierarchical and statistical temperature-aware quasi-static task mapping

and scheduling framework under process variation for hard real-time applications on MPSoCs. They considered the optimization of temperature and their framework is ineffective in evaluating plans by executing each plan for all scenarios. The terms 'plan' and 'scenario' have the same meaning as 'candidate schedule' and 'chip operating point' in this work. However, we aim at the optimization of expected performance and analyze and show how the effective schedule refining scheme works.

3 Model

This section lays the foundations upon which our proposed HASR scheme rests. First, the detailed architecture of the target multi-core processor is presented. Second, the task graph model of the application running on the multi-core processor is described. Third, the formulation of the problem to be solved is shown. The main notations used in this work are listed in Table 1.

Table 1 Overview of the main notations

Notation	Description
op	Chip operating point
OP	The set of all possible chip operating points
OF	The set of all possible maximum operating frequencies
N_{sch}	The number of candidate schedules
N_{c}	The number of cores in the multi-core processor
N_{OP}	The number of chip operating points
N_{OF}	The number of maximum operating frequencies
P_i^m	The performance for processor with op_i when applying the optimal schedule of processor with op_m
P_i	The best performance for processor with op_i given the candidate schedules
lat_i^m	The scheduling interval for processor with op_i when applying the optimal schedule for processor with op_m
lat_i	The optimal scheduling interval for processor with op_i
$\mathrm{freq}_{i,k}$	The actual maximum operating frequency of the kth processor for op_i
RP	The ratio between the performances of two schedules for the same op
REPP	The ratio between the expected performance of two schedules for the op
sch_i	A schedule for application execution on the multi-core processor

3.1 Model of homogeneous multi-core processors

Fig. 2 depicts a typical multi-core processor which consists of multiple cores and an on-chip interconnection network. The on-chip interconnection network is used for inter-core communication. The identically designed cores are influenced by process variation and have different maximum operating frequencies. The chip is implemented in a globally-asynchronous-locally-synchronous (GALS) style to support different operating frequencies for cores on the same chip (Yu and Baas, 2009). Each core is allocated to a frequency island (FI), where the local clock signal can be adjusted independently. This means the operating frequencies of different cores, which are in different FIs, can be controlled independently and are the corresponding actual maximum operating frequencies after speed binning. Data transfer between different FIs is realized by the mixed-clock first-in-first-out.

Due to the impact of manufacturing-caused heterogeneity, the maximum operating frequency of each core varies and can be described by a continuous distribution (Bowman *et al.*, 2002). The probability density function (PDF) of the distribution can be obtained by the model presented in Sarangi *et al.* (2008). The minimum value of the maximum operating frequency $f_{\min}$ is the maximum operating frequency under the worst case such that the cumulative probability at $f_{\min}$, $\mathrm{CDF}(f_{\min})$, equals 0.25% according to the three-sigma rule. Although the maximum operating frequency of a core is possible to be any value greater than $f_{\min}$, the physical implementation of the clock generation allows the frequency to be changed only in discrete steps. The reason is that the clock frequency of a core is tuned by multiplying the reference clock, which is usually tens of megahertz, with different multiplication

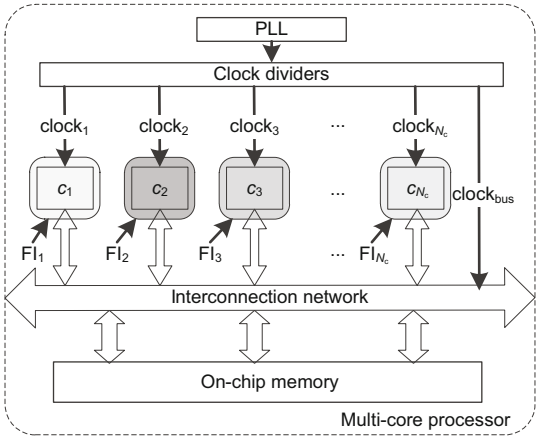

Fig. 2 The model of multi-core processors

factors. Therefore, there are only several maximum operating frequencies supported by the chip. Taking the reference clock of 50 MHz as an example, the maximum operating frequency of a core can be 450, 500, 550 MHz, etc. Thus, the maximum value of the maximum operating frequency is set to be $f_{\max}$ such that $\mathrm{CDF}(f_{\max}+\Delta f)$ equals 99.75% with Δf as the frequency step size. Denoting the set of all maximum operating frequencies as OF, each element, $\mathrm{of}_i \in \mathrm{OF}$, represents an operating frequency. The number of elements is denoted as N_{OF}. The probability of of_i is computed as

$$\mathrm{fp}(\mathrm{of}_i)=\begin{cases}\dfrac{\mathrm{CDF}(\mathrm{of}_i+\Delta f)-\mathrm{CDF}(\mathrm{of}_i)}{1-\mathrm{CDF}(f_{\min})}, & \mathrm{of}_i \neq f_{\max},\\ \dfrac{1-\mathrm{CDF}(\mathrm{of}_i)}{1-\mathrm{CDF}(f_{\min})}, & \mathrm{of}_i = f_{\max}.\end{cases}$$

Same as the term 'scenario' in Khodabandeloo *et al.* (2014), op_i stands for one combination of operating frequencies of all cores. In a given op_i, the maximum operating frequencies of all cores are determined. Denoting the maximum operating frequency of the kth core as $\mathrm{freq}_{i,k}$, op_i can be represented as $\mathrm{op}_i=\{\mathrm{freq}_{i,1},\mathrm{freq}_{i,2},\ldots,\mathrm{freq}_{i,N_{\mathrm{c}}}\}$ with the constraint of $\mathrm{freq}_{i,k}\in\mathrm{OF}$. The set of all possible chip operating points is denoted as OP, $\mathrm{OP}=\{\mathrm{op}_1,\mathrm{op}_2,\ldots,\mathrm{op}_{N_{\mathrm{OP}}}\}$. The probability of op_i is represented as $\mathrm{prob}(\mathrm{op}_i)$ and the number of chip operating points is denoted as N_{OP}. The probability $\mathrm{prob}(\mathrm{op}_i)$ is computed as $\mathrm{fp}(\mathrm{freq}_{i,1})\cdot\mathrm{fp}(\mathrm{freq}_{i,2}|A_2)\cdot\cdots\cdot\mathrm{fp}(\mathrm{freq}_{i,N_{\mathrm{c}}}|A_{N_{\mathrm{c}}})$, where $\mathrm{fp}(\mathrm{freq}_{i,j}|A_j)$ $(2\le j\le N_{\mathrm{c}})$ is the probability that the maximum operating frequency of the jth core is $\mathrm{freq}_{i,j}$ under the condition A_j. A_j stands for the condition that the maximum operating frequencies of the 1st, 2nd, 3rd, …, $(j-1)$th cores have been determined by op_i. A chip operating point represents a class of chips in which the same cores have identical maximum operating frequencies.

3.2 Task graph model

In this work, the application is modeled by a directed acyclic graph (DAG), $G(V,E)$. In the graph, each node in V represents the computational task to be executed on the core and the edge in E between nodes represents both the precedence constraint and the data transfer. Similar to the domain-specific languages for streaming applications, each node requires a number of clock cycles to finish its execution on the core. An embedded application is usually executed many times for a stream of input data on a multi-core processor. For the periodic application considered in this work, the throughput, rather than latency, is the most concerned performance metric. Pipeline scheduling benefits from allowing tasks from different embedded application instances to be scheduled at each stage of the pipeline. The communication between tasks is assumed not to be critical to the throughput due to pipeline scheduling. In pipeline scheduling, the scheduling interval is defined as the time between the start times of two consecutive iterations of the task graph, and the throughput is computed as the reciprocal of the scheduling interval.

3.3 Problem formulation

The proposed task allocation and scheduling scheme aims at achieving high performance on chips influenced by manufacturing-caused heterogeneity. The goal can be described as maximizing the expected performance:

$$\max\ \mathrm{EPP}. \tag{1}$$

The expected performance is calculated as

$$\mathrm{EPP}=\sum_{\mathrm{op}_i\in\mathrm{OP}} P_i\mathrm{prob}(\mathrm{op}_i), \tag{2}$$

where P_i is the best performance achieved by chips with op_i, and $\mathrm{prob}(\mathrm{op}_i)$ is the probability of op_i. To achieve this goal, different schedules are adopted for performance enhancement on chips with different chip operating points. However, the number of all possible chip operating points is usually high, especially for a processor with many cores. To achieve performance improvement, we derive an appropriate number of candidate schedules and store all of them to all chips. Then, an on-chip binding of schedule to chip is conducted to find the best schedule.

Thus, the problem to be solved is formulated as: Given the frequency island based multi-core processor, the distribution of chip operating points indicating the probability for each chip operating point, and the task graph of the embedded application, find the candidate schedules and the binding of candidate schedule to chip that maximizes the performance under the constraint of the total schedule number for the processor.

4 Proposed schedule refining scheme

The optimization goal relates to the performance optimization for different chip operating points. To maintain high performance across different chip operating points, HASR adapts the schedule to each chip to exploit the processor heterogeneity. HASR takes the distribution of chip operating points into consideration to obtain multiple candidate schedules, and binds one of them to each chip during booting. In HASR, representative chip operating points are selected and the optimal schedules are generated for the representative chip operating points. The generated candidate schedules are stored in the dedicated on-chip memory space. According to the actual maximum clock frequencies of a multi-core processor, one of the candidate schedules is bound to each chip to maximize the performance. The binding is carried out on-chip during booting. Both generating candidate schedules and binding candidate schedule to chip are critical to the performance improvement.

To distinguish the schedule generated during candidate schedule generation and the schedule bound to each chip, the definitions of the optimal schedule and best schedule are presented below:

Definition 1 (Optimal schedule) The optimal schedule represents the schedule generated for the chip according to the given scheduling algorithm in Algorithm 2 (lines 11–18, see p.1024). Each element in $\mathrm{Sch_r}$ is an optimal schedule for the chip with the chip operating point in $\mathrm{OP_r}$.

Definition 2 (Best schedule) The best schedule represents the schedule bound to the chip according to the actual maximum clock frequencies of the multi-core processor. The best schedule is selected from $\mathrm{Sch_r}$ for each chip.

4.1 Example

The use of the proposed scheduling scheme on the task graph and the multi-core processor in Fig. 1 is shown in Fig. 3. In this example, the number of the generated candidate schedules is set to two. It consists of two phases: off-chip generating representative chip operating points and candidate schedules, and on-chip binding schedule to chip. In the first phase, σ_1, σ_2, and σ_3 are computed to be 400, 377, and 360 respectively according to Eq. (10). Thus, op_1 is selected as the first selected representative chip operating point. Similarly, op_2 is selected as the second representative chip operating point by computing γ_2 and γ_3 according to Eq. (14). Then, the representative chip operating point set $\{\mathrm{op}_1, \mathrm{op}_2\}$ and the corresponding optimal schedules $\{\mathrm{sch}_1, \mathrm{sch}_2\}$ are written into the on-chip memory. In the second phase, the binding of schedule to chip is done during booting. The best schedule for chips with op_1 is sch_1 according to the algorithm in Fig. 6. Similarly, the best schedules for chips with op_2 and op_3 are found to be sch_2. The performance improvements on chips with op_2 and op_3 are 4.8% and 7.1%, respectively.

4.2 Chip operating point sampling with probability consideration

In the multi-core processor with manufacturing-caused heterogeneity, the number of chip operating

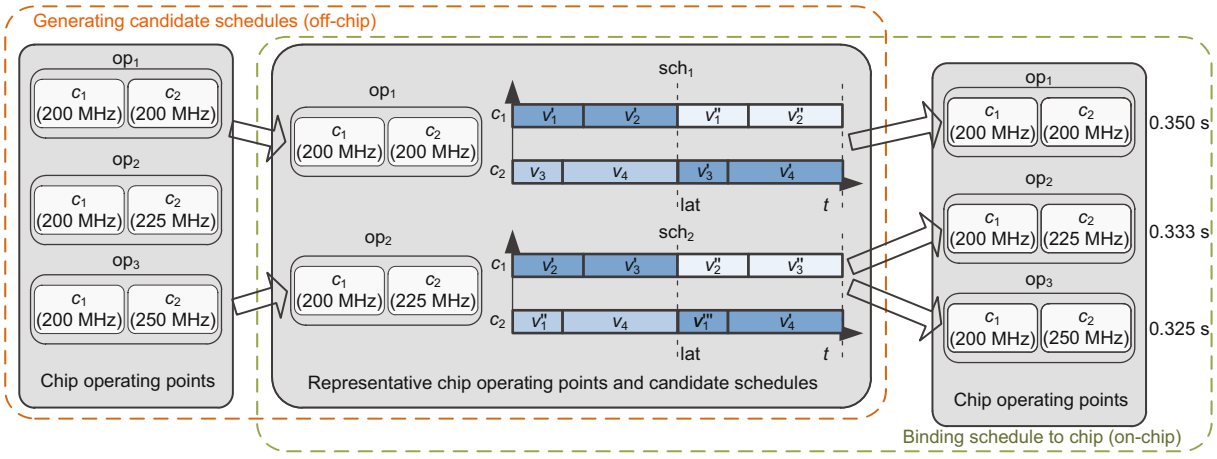

Fig. 3 Heterogeneity-aware schedule refining scheme for the task graph and processor in Fig. 1

points is $(N_{\mathrm{OF}})^{N_{\mathrm{c}}}$, because the maximum operating frequency of each core is the element of OF and the maximum operating frequencies of the N_{c} cores in the processor are independent due to intra-die variation. However, the large number of chip operating points makes it too time-consuming to select representative chip operating points, which are used for candidate schedule generation. The reason is that the selection of representative chip operating points has to traverse all chip operating points, and the selected representative chip operating point has to be evaluated over all chip operating points. Therefore, chip operating point sampling is adopted to reduce the time complexity. The influence of chip operating sampling is discussed in the experiments.

The sampling of chip operating points takes the probability distribution of chip operating points into consideration. Chip operating points with higher probabilities are more likely to be sampled. The procedure of the sampling is described in Algorithm 1. In each iteration, a chip is generated according to the PDF of the chip operating point. If its chip operating point does not belong to the sampled chip operating point set $\mathrm{OP_s}$, the chip operating point is added into $\mathrm{OP_s}$. The probability of chip operating points in $\mathrm{OP_s}$ is defined as

$$\mathrm{sprob}(\mathrm{op}_i) = \frac{\mathrm{prob}(\mathrm{op}_i)}{\sum\limits_{\mathrm{op}_j \in \mathrm{OP_s}} \mathrm{prob}(\mathrm{op}_j)}. \tag{3}$$

According to the Bernoulli law of large numbers (Von Mises, 1964), $\mathrm{OP_s}$ is close to OP for large threshold number of sampled chip operating points N_{TH}.

Algorithm 1 Chip operating point sampling with probability consideration

Input: Chip operating point set OP, the threshold number of sampled chip operating points N_{TH}.
Output: Sampled chip operating point set $\mathrm{OP_s}$.

```
1: i ← 0; OP_s ← ∅;
2: while i < N_TH do
3:    Sample a chip operating point, op_s, according to the probability distribution of OP;
4:    if op_s ∉ OP_s then
5:       Add op_s into OP_s;
6:       i ← i + 1;
7:    end if
8: end while
```

4.3 Generating multiple candidate schedules for sampled chip operating points

In the proposed scheme, the candidate schedule is iteratively generated and stored, as shown in Algorithm 2. It takes as input the application task graph, the multi-core processor description, and the distribution of the sampled chip operating point set. The output is the representative chip operating point set and the corresponding optimal schedules, which are candidate schedules. The quantity of the optimal candidate schedules is limited by the number of candidate schedules, N_{sch}, which is specified by the user or constrained by the memory requirement. If the number of chip operating points in $\mathrm{OP_s}$, $N_{\mathrm{OP_s}}$, is less than N_{sch}, all $N_{\mathrm{OP_s}}$ optimal schedules corresponding to the chip operating point in $\mathrm{OP_s}$ are generated and stored.

In each iteration, the representative chip operating point is selected for the generation of the candidate schedule (lines 4 to 9). The representative

Algorithm 2 Generation of multiple candidate schedules

Input: Application task graph G, distribution of sampled chip operating point $\mathrm{OP_s}$, and multi-core processor description.
Output: Representative chip operating point set $\mathrm{OP_r}$ and the corresponding optimal schedule set $\mathrm{Sch_r}$.

```
1: i ← 0; Sch_r ← ∅; OP_r ← ∅;
2: while i < max(N_sch, N_OPs) do
3:    // select the representative chip operating point
4:    if i = 0 then
5:       op_r ← Sel_first();
6:    else
7:       op_r ← Sel_remaining();
8:    end if
9:    Store op_r as the ith element of OP_r;
10:   // obtain the optimal schedule for the selected optimal operating point op
11:   Sort tasks in decreasing execution cycles;
12:   for all tasks do
13:      for all cores do
14:         Tentatively assign task to the core;
15:         Compute the scheduling interval;
16:      end for
17:      Assign task to the core that minimizes the scheduling interval;
18:   end for
19:   Store the schedule as the ith element of Sch_r;
20:   i++;
21: end while
```

chip operating point $\mathrm{op_r}$ is selected from all possible chip operating points by the functions Sel_first() and Sel_remaining(), which are stated later in Sections 4.3.1 and 4.3.2.

In each iteration, the generation of the candidate schedule is accomplished by applying the heuristic scheduling algorithm (Mirzoyan *et al.*, 2014) to chips with the selected representative chip operating point (lines 12 to 20). During task scheduling, the schedule interval is computed as the maximum time to execute tasks of an application instance among all cores. Since the optimization of candidate schedule generation is complementary to the proposed scheme, other scheduling algorithms, such as the integer linear programming method (Yi *et al.*, 2009), can also be adopted to generate the candidate schedule. The storage of candidate schedules is shown in Fig. 4. The candidate schedules and the corresponding representative chip operating points are stored in the same order. The stored representative chip operating points are used to evaluate the corresponding candidate schedules during the booting of each chip, as stated in Section 4.4. For each stored candidate schedule, the corresponding chip operating point indicates the operating frequency of each core, and the stored candidate schedule shows both the executing core and the executing order for each task.

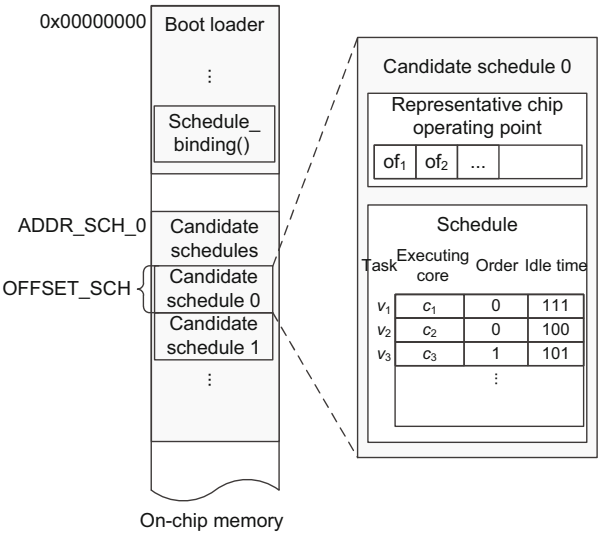

Fig. 4 The storage of candidate schedules

4.3.1 Selection of the first representative chip operating point

The procedure of selecting the first representative chip operating point is presented in Algorithm 3. The expected performance, EPP, is used to select the representative chip operating point. All chip operating points in $\mathrm{OP_s}$ are traversed to select the chip operating point with the highest priority as the representative chip operating point. The expected performance EPP_m for a selected representative chip operating point op_m is computed by applying the corresponding optimal schedule to chips of all chip operating points:

$$\mathrm{EPP}_m = \sum_{\mathrm{op}_i \in \mathrm{OP_s}} \mathrm{sprob}(\mathrm{op}_i) \cdot P_i^m. \tag{4}$$

To compare the priorities of two chip operating points op_m and op_n, the expected performance EPP_m is divided by EPP_n to obtain the ratio between the expected performances of two schedules, REPP, which is computed as

$$\mathrm{REPP} = \frac{\mathrm{EPP}_m}{\mathrm{EPP}_n}. \tag{5}$$

Since the ideal optimized performance for a homogeneous multi-core processor is achieved when the workloads, in terms of execution time in second, on all cores are identical, all cores in the chip with chip operating point op_m are assumed to consume lat_m seconds to execute the workload assigned by the corresponding optimal schedule sch_m. Thus,

$$\begin{aligned} P_i^m &= \frac{1}{\mathrm{lat}_i^m} = \frac{1}{\mathrm{lat}_m \max\limits_{0 \le k < N_\mathrm{c}} \left(\dfrac{\mathrm{freq}_{m,k}}{\mathrm{freq}_{i,k}} \right)} \\ &= P_m \min_{0 \le k < N_\mathrm{c}} \left(\frac{\mathrm{freq}_{i,k}}{\mathrm{freq}_{m,k}} \right). \end{aligned} \tag{6}$$

P_i^n can be expressed in a similar form. Here, we assume that the optimal schedules sch_m and sch_n can achieve workload balance on the target processor with chip operating points op_m and op_n, respectively. The impact of this assumption is discussed in the experiments. Since both sch_m and sch_n are optimized schedules for the same application, the total required execution cycles are assumed to be fixed for the homogeneous multi-core processor:

$$\mathrm{lat}_m \sum_{0 \le k < N_\mathrm{c}} \mathrm{freq}_{m,k} = \mathrm{lat}_n \sum_{0 \le k < N_\mathrm{c}} \mathrm{freq}_{n,k}. \tag{7}$$

Accordingly, we have

$$\frac{P_m}{P_n} = \frac{\sum\limits_{0 \le k < N_\mathrm{c}} \mathrm{freq}_{m,k}}{\sum\limits_{0 \le k < N_\mathrm{c}} \mathrm{freq}_{n,k}}. \tag{8}$$

Therefore, Eq. (5) can be rewritten as

$$\mathrm{REPP} = \frac{\mathrm{EPP}_m}{\mathrm{EPP}_n} = \frac{\sigma_m}{\sigma_n}, \tag{9}$$

where σ_m and σ_n are expressed as

$$\begin{aligned} \sigma_j &= \sum_{0 \le i < N_p} \mathrm{sprob}(\mathrm{op}_i) \cdot \rho_{j,i}, \\ \rho_{j,i} &= \sum_{0 \le k < N_\mathrm{c}} \mathrm{freq}_{j,k} \cdot \min_{0 \le k < N_\mathrm{c}} \left(\frac{\mathrm{freq}_{i,k}}{\mathrm{freq}_{j,k}} \right), \end{aligned} \tag{10}$$

with $j \in \{m, n\}$. If REPP is larger than one, the priority of op_m is higher than that of op_n. Otherwise, op_n has a higher priority. The priorities of all chip operating points are compared to select the chip operating point with the highest priority as the representative chip operating point.

Algorithm 3 Selection of the first representative chip operating point, Sel_first()

Input: Distribution of the sampled chip operating point $\mathrm{OP_s}$, task graph G.

Output: Representative chip operating point $\mathrm{op_r}$.

```
Select the first element of OP_s as the initial value of op_r;
Compute σ_r for op_r according to Eq. (10);
i ← 0;
for all op ∈ OP_s do
    Select the ith chip operating point of OP_s, op_i;
    Compute σ_i for op_i according to Eq. (10);
    Compute REPP of op_i to op_r according to Eq. (5);
    if REPP > 1 then
        op_r ← op_i;
        σ_r ← σ_i;
    end if
    i++;
end for
```

4.3.2 Selection of the remaining representative chip operating point

The selection of the remaining representative chip operating point is more complicated due to the presence of already selected candidate schedules (Algorithm 4). All chip operating points are traversed to construct the tentative representative chip operating point set to compute the priority, which is represented as EPP. Then, the set with the highest priority is selected as the representative chip operating point set. The priority of each tentative representative chip operating point set is computed based on

Algorithm 4 Selection of the remaining representative chip operating point, Sel_remaining()

Input: Distribution of sampled chip operating point $\mathrm{OP_s}$, application task graph G, previous representative chip operating point set $\mathrm{OP_{pre}}$.

Output: Representative chip operating point $\mathrm{op_{best}}$.

```
Select the first element of OP_s as the initial value of op_best;
Obtain the best schedule for each chip operating point with the tentative representative chip operating point set OP_nxt = {OP_pre, op_best};
Compute γ_nxt for OP_nxt according to Eq. (14);
i ← 0;
for all op ∈ OP_s do
    Select the ith chip operating point of OP_s, op_i;
    Obtain the best schedule for each chip operating point with the tentative representative chip operating point set OP_i = {OP_pre, op_i};
    Compute γ_i for OP_i according to Eq. (14);
    Compute REPP of OP_i to OP_nxt according to Eq. (13);
    if REPP > 1 then
        op_best ← op_i;
        OP_nxt ← OP_i;
        γ_nxt ← γ_i;
    end if
    i++;
end for
```

the best performance, and the best performance is obtained by applying the best schedule for each chip operating point in $\mathrm{OP_s}$. The best schedule for each chip operating point is selected according to the procedure given in Fig. 6. The performance for chips with chip operating point op_i is expressed as

$$P_i = \max_{l \in \mathrm{OP}_a} P_i^l, \tag{11}$$

where OP_a is the tentative representative chip operating point set. The expected performance EPP_a of the tentative representative chip operating point set OP_a is computed as

$$\mathrm{EPP}_a = \sum_{\mathrm{op}_i \in \mathrm{OP_s}} \mathrm{sprob}(\mathrm{op}_i) \cdot \max_{l \in \mathrm{OP}_a} P_i^l. \tag{12}$$

The ratio of the expected performance for tentative representative chip operating point set OP_a to that of another tentative representative chip operating point set OP_b is

$$\mathrm{REPP} = \frac{\mathrm{EPP}_a}{\mathrm{EPP}_b} = \frac{\gamma_a}{\gamma_b}, \tag{13}$$

where γ_a and γ_b are expressed as

$$\gamma_j = \sum_{0\le i<N_p} \text{sprob}(\text{op}_i)\cdot\beta_{j,i},$$
$$\beta_{j,i} = \max_{l\in \text{OP}_j}\Bigg(\sum_{0\le k<N_\text{c}} \text{freq}_{l,k}\cdot \min_{0\le k<N_\text{c}}\Big(\frac{\text{freq}_{i,k}}{\text{freq}_{l,k}}\Big)\Bigg), \tag{14}$$

with $j\in\{a,b\}$. The expected performances for all tentative representative chip operating point sets are compared to select the one with the highest priority as the new representative chip operating point set.

4.4 Binding schedule to chip during booting

For each chip, only one schedule is adopted to execute the application. Given multiple candidate schedules, schedule binding is done during booting to achieve the best performance according to the detected chip operating point (Fig. 5). The chip operating point of the chip is detected by using the testing techniques and speed binning (Lin *et al.*, 2005; Raychowdhury *et al.*, 2005). The booting process performs the essential initialization including PLL configuration, memory control register configuration, exception vector table construction, etc. The schedule binding is inserted at the end of the booting process. The schedule binding is done only once at the initial booting by modifying the beginning address of the best schedule. Since schedule binding is done before the execution of applications, the time cost for schedule binding has no impact on the performance of application execution.

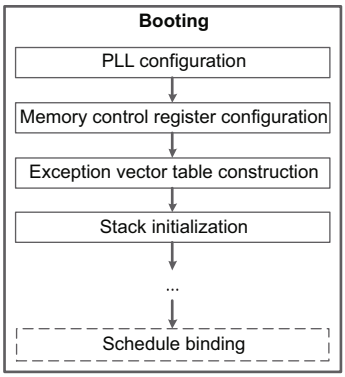

Fig. 5 The booting process

The pseudocode of schedule binding is shown in Fig. 6. For each chip operating point, all candidate schedules are traversed to find the one that results in the highest performance according to Eq. (16). The

```
Schedule_binding()
{
    idx_sch <- 0;
    P_old <- 0;
    while(idx_sch < N_sch)
    {
        P_new <- Compute_performance(idx_sch,of);
        if(P_new > P_old)   /*New best schedule*/
        {
            P_old <- P_new;
            idx_best <- idx_sch;
         /*The beginning address of the best schedule*/
            ADDR_SCH<-ADDR_SCH_0+idx_best * OFFSET_SCH;
        }
        idx_sch <- idx_sch + 1;
    }
}

Compute_performance(idx_sch, of)
{
    total_freq <- freq_sch[0];
    min_ratio <- of[0] / freq_sch[0];
    i <- 1;
    while(i < N_c)
    {
        total_freq <- total_freq + freq_sch[i];
        if (op[i] / freq_sch[i] < min_ratio)
            min_ratio <- op[i] / freq_sch[i];
    }
    P_temp <- total_freq * min_ratio;
    return(P_temp);
}
```

Fig. 6 The pseudocode of schedule binding

schedule binding is done by comparing the performance for every two candidate schedules. Let candidate schedules sch_m and sch_n be the optimal schedules for chip operating points op_m and op_n, respectively. To select the best candidate from these two candidates for chip operating point op_i, the throughput of the target chip is predicted when the chip adopts either sch_m or sch_n by using Eq. (6). To compare these two values, the ratio of P_i^m to P_i^n is computed as

$$\text{RP} = \frac{P_i^m}{P_i^n} = \frac{P_m \min\limits_{0\le k<N_\text{c}}\Big(\frac{\text{freq}_{i,k}}{\text{freq}_{m,k}}\Big)}{P_n \min\limits_{0\le k<N_\text{c}}\Big(\frac{\text{freq}_{i,k}}{\text{freq}_{n,k}}\Big)}. \tag{15}$$

Based on Eq. (7), Eq. (15) can be rewritten as

$$\text{RP} = \frac{\sum\limits_{0\le k<N_\text{c}} \text{freq}_{m,k} \min\limits_{0\le k<N_\text{c}}\Big(\frac{\text{freq}_{i,k}}{\text{freq}_{m,k}}\Big)}{\sum\limits_{0\le k<N_\text{c}} \text{freq}_{n,k} \min\limits_{0\le k<N_\text{c}}\Big(\frac{\text{freq}_{i,k}}{\text{freq}_{n,k}}\Big)}. \tag{16}$$

If $\text{RP}>1$, schedule sch_m is considered to be better than schedule sch_n for chips of chip operating point op_i. Otherwise, sch_n is better than sch_m. Since the schedule binding is based on the assumption that schedules sch_m and sch_n can achieve workload balance over all cores, the bound schedule may lead to

performance degradation under a certain case. For soft real-time systems, this is acceptable since little degradation is observed on a small portion of the chips as shown by the experimental results. For hard real-time systems, such performance degradation can be eliminated by comparing the performance of the bound schedule and the performance of the schedule derived under the worst case, and the one that exhibits better performance is selected.

4.5 Discussion

4.5.1 Complexity analysis

The generation of each candidate schedule requires the selection of a representative chip operating point and the generation of each optimal schedule. In the proposed scheme, the selection of the representative chip operating point is the most time-consuming part. The complexity of this part is $O(N_{\mathrm{OP}}^2)$ for each candidate schedule, since the obtaining of the best schedule for each chip operating point has the complexity of $O(N_{\mathrm{OP}})$. Therefore, the complexity of candidate schedule generation is $O(N_{\mathrm{sch}} \cdot N_{\mathrm{OP}}^2)$. According to the pseudocode of schedule binding (Fig. 6), the complexity of schedule binding is $O(N_{\mathrm{c}} \cdot N_{\mathrm{sch}})$.

4.5.2 Memory usage analysis

The amount of memory space to store the candidate schedules is determined by both the number of candidate schedules and the memory consumed by each candidate. Denote the number of tasks as N_{t}. As shown in Fig. 4, each candidate consumes $N_{\mathrm{t}}(\log_2 N_{\mathrm{c}} + \log_2 N_{\mathrm{t}} + 32)$ bits to specify the executing core, the executing order, and the idle time for each task (Khodabandeloo *et al.*, 2014). In the proposed scheme, the memory usage can be reduced to $N_{\mathrm{sch}} N_{\mathrm{t}}(\log_2 N_{\mathrm{c}} + \log_2 N_{\mathrm{t}} + 32)$ from $N_{\mathrm{op}} N_{\mathrm{t}}(\log_2 N_{\mathrm{c}} + \log_2 N_{\mathrm{t}} + 32)$. For a system with 8 cores, 16 tasks, 5 FMAXs, the memory cost is reduced from 29.0 MB to 1.2 KB by setting N_{sch} as 16.

5 Experiments

5.1 Experimental setup

To show the improvements in throughput, the results of the proposed scheme are compared with those of the heterogeneity-unaware approaches. We set the throughput of the worst-case frequency-based schedule on the processor with the lowest operating frequencies as the base performance. The results of the proposed scheme are also compared with those of the conventional task scheduling approach (Mirzoyan *et al.*, 2014), which is denoted as DTS for the deterministic timing model, on the variation-aware processor with multiple chip operating points.

Twelve synthetic applications generated by using TGFF (Dick *et al.*, 1998) and three real-world benchmarks (Stuijk *et al.*, 2006) were adopted to evaluate the proposed HASR scheme. The 12 synthetic applications, TG1 to TG12, are generated with four different patterns and each pattern with three different task number specifications (Table 2). The three real-world benchmarks, TG13 to TG15, are H.263 decoder, MP3 decoder, and MODEM, respectively, which cover both the DSP domain and the multimedia domain (refer to Stuijk *et al.* (2006) for details).

Table 2 Overview of the benchmark applications

Benchmark	Number of tasks	Benchmark	Number of tasks
TG1	51	TG9	200
TG2	101	TG10	70
TG3	202	TG11	105
TG4	51	TG12	209
TG5	109	TG13	4
TG6	202	TG14	14
TG7	50	TG15	16
TG8	101		

The applications are scheduled onto a multi-core processor with two to eight homogeneous cores. TGFF is adopted to generate the first 12 applications and the corresponding multi-core processors. The execution time of each task is uniformly distributed between 40 and 70 ms, and the floorplans of the processors are regular grids. For the last three applications, each core of the multi-core processor is an ARM7 core with the lowest maximum operating frequency of 500 MHz. To obtain the probability distribution of the relative maximum operating frequency, we follow the model in Momtazpour *et al.* (2013) and Khodabandeloo *et al.* (2014) by applying simplifying assumptions of uniformly distributed critical paths and equally sized inverters for each critical path (Bowman *et al.*, 2009). The size of the ARM7 core is scaled down to 22 nm technology

based on the scaling factor in Huang *et al.* (2011). We adopt parameter variations suggested by ITRS (2014) and implement the model in R language.

To generate multiple candidate schedules, the set of chip operating points is sampled to be fed into the proposed scheme. Since the number of chip operating points is exponential to the number of cores in the processor, the runtime of the scheme quickly climbs up to hours from less than one minute when increasing the number of cores from four to eight. The runtime of scheduling would be up to days, which is unbearable, when the number of integrated cores further increases to a large one, e.g., 12. To reduce the complexity, the chip operating point set is sampled to obtain an approximation of the distribution to guide candidate schedule generation.

In the sampling method, a set of training samples are adopted to obtain the candidate schedules. To validate the sampling method, the generated candidate schedules are applied onto another set of validation samples. In the experiments, the performance improvement is obtained by averaging the performance improvement on 10 different sets of validation samples of the same size. In Fig. 7, the CDFs of the chips are presented with the variation of throughput improvement under different sizes of training samples. The target processor consists of eight cores and the number of candidate schedules is set to 12. In the figure, the throughput improvements over base performance are shown for different sizes of training samples (N_{TH} varies from 1000 to 100 000). The generated candidate schedules are applied onto a validation sample with 100 000 chips to obtain the throughput. It is shown that the differences between

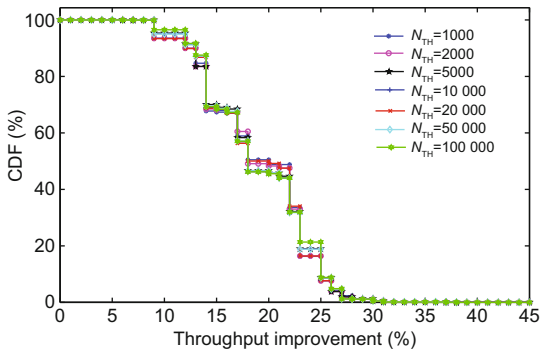

Fig. 7 The cumulative distribution function (CDF) of chips over the performance improvement achieved by HASR against base performance with different sizes of training samples for TG2

different training sample sizes are little. Similar results are observed for other applications. Thus, the sampling method is able to collect enough information to guide candidate schedule generation. For the following experiments, the size of the training sample (N_{TH}) is set to 10 000. The runtime is in the order of second for all tested applications.

5.2 Impact of the number of tasks

Fig. 8 presents the throughput improvements of HASR and DTS over base performances for the given validation sample. In this set of experiments, the number of cores on the multi-core processor is set to 8 and the number of candidate schedules is 12. It is shown that the results of HASR and DTS are better than the base performances for all applications. This is because both approaches exploit hardware heterogeneity induced by process variation. Furthermore, HASR behaviors are better than DTS behaviors for all applications. That is to say, for any given throughput requirement, the CDF of chips meeting the requirement by using HASR would be no less than that of DTS. The reason is that HASR further exploits the heterogeneity by adaptively changing the schedule. Specifically, the throughput improvements of DTS and HASR are 16% and 22.3%, respectively, when CDF is set to 50%. The average performance improvements achieved by DTS and HASR are 16.5% and 22.2%, respectively. The results show that the adaptivity of the schedule to hardware heterogeneity is important for enhancing the performance.

The results above present throughput improvements by considering the validation sample as a whole. In addition, the change of throughput on each chip is shown in Figs. 9 and 10. Since the base performance keeps unchanged across chips, only the differences of throughputs between results of HASR and DTS are presented. Although the CDFs in Fig. 8 show that HASR statistically outperforms DTS, throughput degradation occurs on some chips from the distribution of throughput improvement in Fig. 9. These two observations are not conflicted, because the statistical description does not guarantee that the set of chips meeting a given performance requirement with DTS is the subset of that with HASR. The reason of performance degradation is that the actual workloads on each core are not perfectly balanced as it is supposed to be when

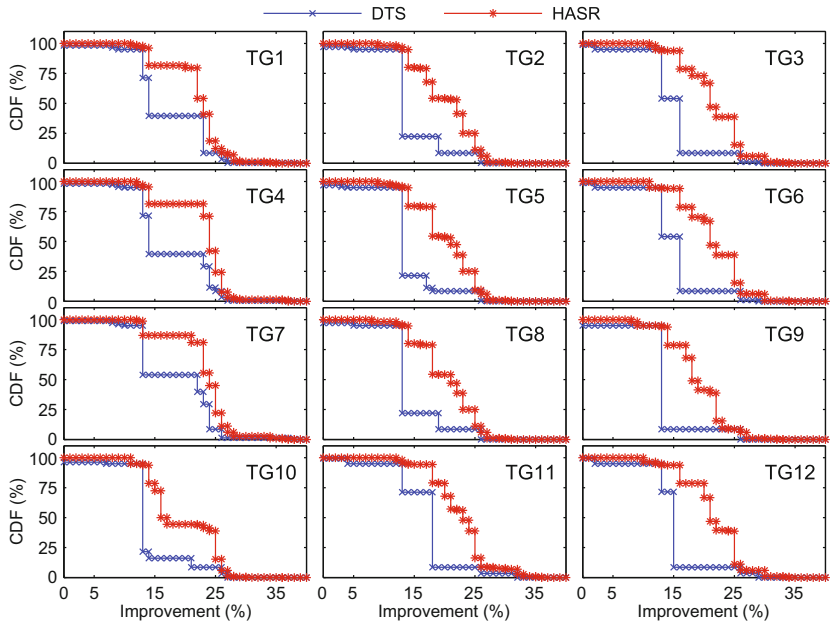

Fig. 8 The CDF of chips over the improvement of results achieved by HASR and DTS against base performance for different applications

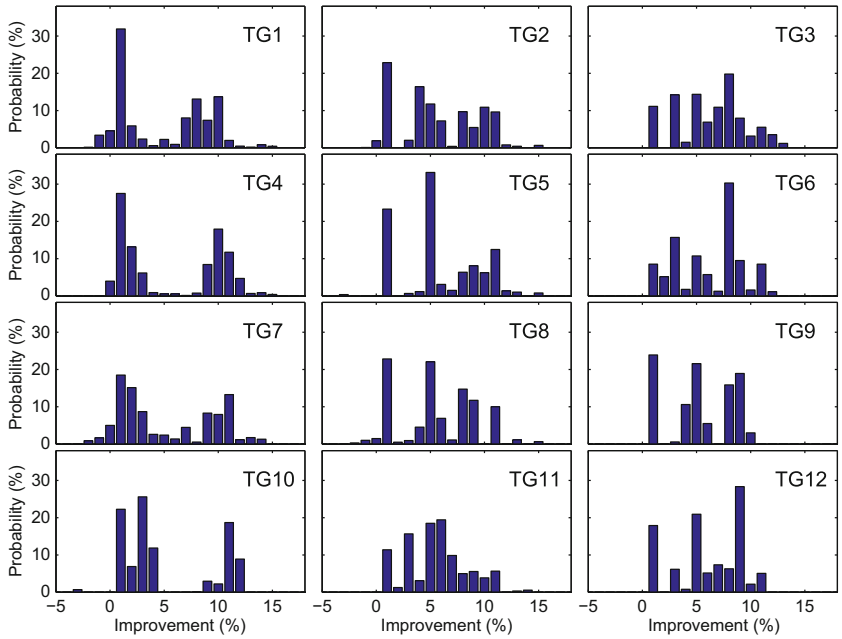

Fig. 9 The probability density function of throughput improvement achieved by HASR against DTS for different applications

applying the optimized schedules. This would lead to the misjudgment during binding schedule to the chip operating point. It can also be observed that the chips with performance degradation account only for a small portion, around 2%, of the total chips. The performance improvement on chips can be up to 12% across all tested applications. Taking a further look at the results of applications generated by the same pattern but with different task quantities, it is concluded that the portion of chips with performance degradation decreases with the increase of the task number. The boxplots in Fig. 10 obviously exhibit this observation. When the number of task is around 200, HASR is able to achieve performance

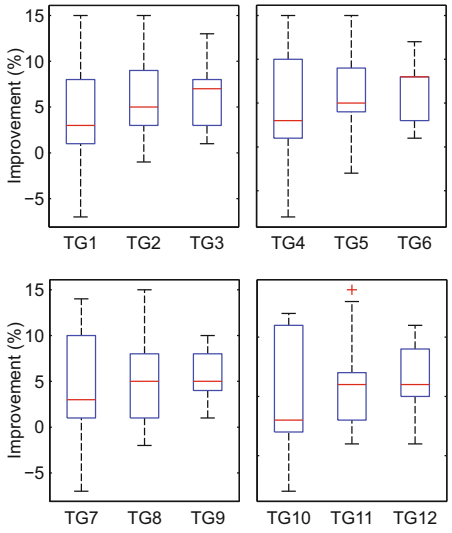

Fig. 10 The throughput improvement achieved by HASR against DTS for different applications

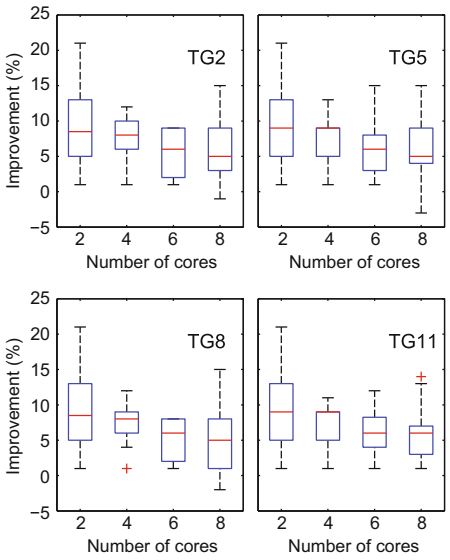

Fig. 11 The throughput improvement achieved by HASR against DTS on different multi-core processors for synthetic applications

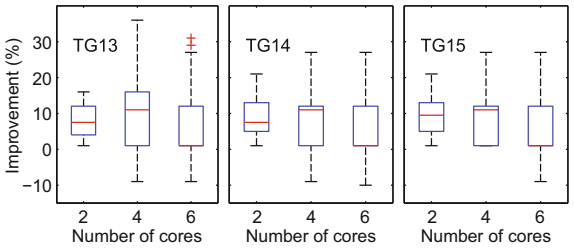

Fig. 12 The throughput improvement achieved by HASR against DTS on different multi-core processors for real applications

improvement over DTS across all chips except TG12. Furthermore, the average throughput improvement increases with the task quantity. The reason for both is that the workload imbalance across cores is narrowed with the increase of the task quantity. The binding of the schedule to the chip operating point would achieve a better result with the increase of the task quantity.

5.3 Impact of the number of cores

Figs. 11 and 12 show the variation of throughput improvement on processors with different numbers of cores. Only the differences between results of HASR and DTS on each chip are shown to study the impact of different numbers of cores. In the first set of experiments, applications with a task quantity around 100 are applied onto the multi-core processor with two to eight cores. In the second set of experiments, applications are applied onto the multi-core processor with two to six cores, because the task quantities are lower than those in the first set. It is shown that the improvement achieved by HASR over DTS increases when the core quantity decreases to two. Also, the portion of chips with performance degradation decreases to zero when the processor has two cores. The reason is that allocating an application to fewer cores leads to slighter workload imbalance. Even in the presence of workload imbalance, HASR still outperforms DTS on most chips.

5.4 Impact of the number of candidate schedules

The throughput improvements achieved by HASR over DTS under different candidate schedule quantities are shown in Figs. 13 and 14. The test applications are the same as those in Section 5.3. The applications shown in Figs. 13 and 14 are applied onto a multi-core processor with eight or two cores, respectively. As expected, the throughput improvement increases by adopting more candidate schedules, since a chip is more likely bound with a better schedule with more candidate schedules. Simultaneously, the throughput improvement by adding candidate schedules is observed to decrease when the original candidate schedule quantity becomes higher, e.g., from four to eight. This is because the heterogeneity induced by process variation is not uniformly distributed and the influences of representative chip operating points are not equal to each other. It

can be seen that in both figures the curves under 12 and 16 candidate schedules almost overlap with each other. So, increasing the candidate schedule quantity when it is larger than 16 would lead to less or no benefit. In other words, the proposed multiple candidate schedules scheme is practical since the required number of candidate schedules is low.

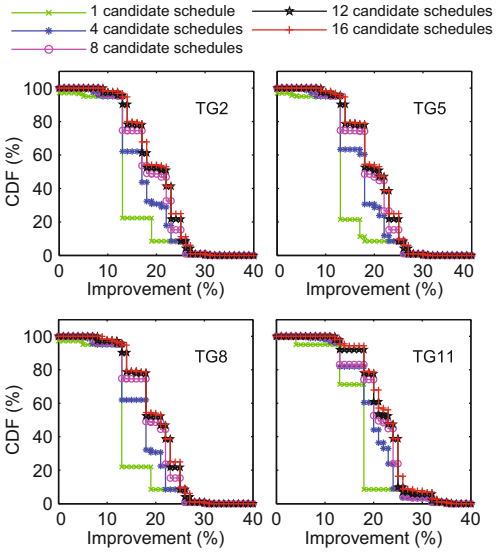

Fig. 13 The CDF of chips over the improvement of results achieved by HASR and DTS against base performance for synthetic applications with different quantities of candidate schedules

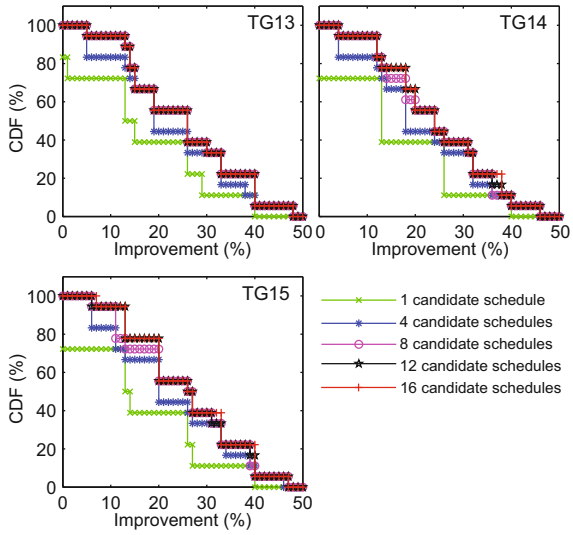

Fig. 14 The CDF of chips over the improvement of results achieved by HASR and DTS against base performance for real applications with different candidate schedule quantities

6 Conclusions and future work

The technology scaling leads to process variation induced heterogeneity for homogeneous designed multi-core processors. We show that the exploitation of such heterogeneity during task scheduling is necessary for performance improvement on multi-core processors. This work presents HASR, a heterogeneity-aware schedule refining scheme that can adaptively refine schedules according to the heterogeneity of each chip. This scheme constructs a representative chip operating point set to generate candidate schedules. Each chip is bound with one of the candidate schedules to achieve optimal performance during booting. Experimental results show that HASR outperforms the traditional single schedule based approach. The proposed HASR scheme provides the opportunity for leveraging process variation induced heterogeneity for task scheduling. With the technology scaling further, the process variation induced heterogeneity would be more notable and the exploitation of such heterogeneity would be more remarkable. While this work is focused on performance improvement, an extended work of the proposed scheme is to take power consumption into consideration during task scheduling, since power consumption also shows large variation under current technology. Another meaningful future work is to adapt the scheduling to dynamic variation and permanent faults caused by aging.

References

Aguilera, P., Lee, J., Farmahini-Farahani, A., *et al.*, 2014. Process variation-aware workload partitioning algorithms for GPUs supporting spatial-multitasking. Design, Automation and Test in Europe Conf. and Exhibition, p.176.1-176.6. [doi:10.7873/date.2014.189]

Bell, S., Edwards, B., Amann, J., *et al.*, 2008. TILE64 processor: a 64-core SoC with mesh interconnect. IEEE Int. Solid-State Circuits Conf., p.588-598. [doi:10.1109/isscc.2008.4523070]

Bowman, K.A., Duvall, S.G., Meindl, J.D., 2002. Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration. *IEEE J. Solid-State Circ.*, **37**(2):183-190. [doi:10.1109/4.982424]

Bowman, K.A., Alameldeen, A.R., Srinivasan, S.T., *et al.*, 2009. Impact of die-to-die and within-die parameter variations on the clock frequency and throughput of multi-core processors. *IEEE Trans. VLSI Syst.*, **17**(12):1679-1690. [doi:10.1109/TVLSI.2008.2006057]

Chon, H., Kim, T., 2009. Timing variation-aware task scheduling and binding for MPSoC. Proc. Asia and South Pacific Design Automation Conf., p.137-142. [doi:10.1109/aspdac.2009.4796470]

Dick, R.P., Rhodes, D.L., Wolf, W., 1998. TGFF: task graphs for free. Proc. 6th Int. Workshop on Hardware/Software Codesign, p.97-101. [doi:10.1109/hsc.1998.666245]

Dietrich, M., Haase, J., 2012. Process Variations and Probabilistic Integrated Circuit Design. Springer, New York, p.69-89. [doi:10.1007/978-1-4419-6621-6]

Ferrandi, F., Lanzi, P.L., Pilato, C., *et al.*, 2010. Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst.*, **29**(6):911-924. [doi:10.1109/tcad.2010.2048354]

Huang, L., Xu, Q., 2010. Performance yield-driven task allocation and scheduling for MPSoCs under process variation. Proc. 47th Design Automation Conf., p.326-331. [doi:10.1145/1837274.1837358]

Huang, W., Rajamani, K., Stan, M.R., *et al.*, 2011. Scaling with design constraints: predicting the future of big chips. *IEEE Micro*, **31**(4):16-29. [doi:10.1109/MM.2011.42]

ITRS, 2013. International Technology Roadmap for Semiconductors. Available from http://www.itrs.net/reports.html [Accessed on Feb. 1, 2015]

Khailany, B., Dally, W.J., Kapasi, U.J., *et al.*, 2001. Imagine: media processing with streams. *IEEE Micro*, **21**(2):35-46. [doi:10.1109/40.918001]

Khodabandeloo, B., Khonsari, A., Gholamian, F., *et al.*, 2014. Scenario-based quasi-static task mapping and scheduling for temperature-efficient MPSoC design under process variation. *Microprocess. Microsyst.*, **38**(5):399-414. [doi:10.1016/j.micpro.2014.05.006]

Lin, Y.C., Lu, F., Cheng, K.T., 2005. Pseudo-functional scan-based BIST for delay fault. Proc. 23rd IEEE VLSI Test Symp., p.229-234. [doi:10.1109/vts.2005.69]

Mirzoyan, D., Akesson, B., Goossens, K., 2012. Process-variation aware mapping of real-time streaming applications to MPSoCs for improved yield. Proc. 13th Int. Symp. on Quality Electronic Design, p.41-48. [doi:10.1109/isqed.2012.6187472]

Mirzoyan, D., Akesson, B., Goossens, K., 2014. Process-variation-aware mapping of best-effort and real-time streaming applications to MPSoCs. *ACM Trans. Embed. Comput. Syst.*, **13**(2s):61.1-61.24. [doi:10.1145/2490819]

Momtazpour, M., Goudarzi, M., Sanaei, E., 2010a. Variation-aware task and communication scheduling in MPSoCs for power-yield maximization. *IEICE Trans. Fundament. Electron. Commun. Comput. Sci.*, **93**(12):2542-2550. [doi:10.1587/transfun.e93.a.2542]

Momtazpour, M., Sanaei, E., Goudarzi, M., 2010b. Power-yield optimization in MPSoC task scheduling under process variation. Proc. 11th Int. Symp. on Quality Electronic Design, p.747-754. [doi:10.1109/isqed.2010.5450497]

Momtazpour, M., Ghorbani, M., Goudarzi, M., *et al.*, 2011. Simultaneous variation-aware architecture exploration and task scheduling for MPSoC energy minimization. Proc. 21st Symp. on GLSVLSI, p.271-276. [doi:10.1145/1973009.1973063]

Momtazpour, M., Goudarzi, M., Sanaei, E., 2013. Static statistical MPSoC power optimization by variation-aware task and communication scheduling. *Microprocess. Microsyst.*, **37**(8B):953-963. [doi:10.1016/j.micpro.2012.02.008]

Omara, F.A., Arafa, M.M., 2010. Genetic algorithms for task scheduling problem. *J. Parall. Distrib. Comput.*, **70**(1):13-22. [doi:10.1016/j.jpdc.2009.09.009]

Ramamritham, K., 1995. Allocation and scheduling of precedence-related periodic tasks. *IEEE Trans. Parall. Distrib. Syst.*, **6**(4):412-420. [doi:10.1109/71.372795]

Raychowdhury, A., Ghosh, S., Roy, K., 2005. A novel on-chip delay measurement hardware for efficient speed-binning. Proc. 11th IEEE Int. On-Line Testing Symp., p.287-292. [doi:10.1109/iolts.2005.10]

Sarangi, S.R., Greskamp, B., Teodorescu, R., *et al.*, 2008. VARIUS: a model of process variation and resulting timing errors for microarchitects. *IEEE Trans. Semicond. Manufact.*, **21**(1):3-13. [doi:10.1109/tsm.2007.913186]

Singhal, L., Bozorgzadeh, E., 2008. Process variation aware system-level task allocation using stochastic ordering of delay distributions. Proc. IEEE/ACM Int. Conf. on Computer-Aided Design, p.570-574. [doi:10.1109/iccad.2008.4681633]

Stuijk, S., Geilen, M., Basten, T., 2006. SDF^3: SDF for free. Proc. 6th Int. Conf. on Application of Concurrency to System Design, p.276-278. [doi:10.1109/acsd.2006.23]

Taylor, M.B., Kim, J., Miller, J., *et al.*, 2002. The raw microprocessor: a computational fabric for software circuits and general-purpose programs. *IEEE Micro*, **22**(2):25-35. [doi:10.1109/mm.2002.997877]

Topcuoglu, H., Hariri, S., Wu, M.Y., 2002. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parall. Distrib. Syst.*, **13**(3):260-274. [doi:10.1109/71.993206]

Von Mises, R., 1964. Mathematical Theory of Probability and Statistics. Academic Press, New York, p.329-367. [doi:10.1016/b978-1-4832-3213-3.50010-5]

Wang, F., Chen, Y., Nicopoulos, C., *et al.*, 2011. Variation-aware task and communication mapping for MPSoC architecture. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst.*, **30**(2):295-307. [doi:10.1109/tcad.2010.2077830]

Yi, Y., Han, W., Zhao, X., *et al.*, 2009. An ILP formulation for task mapping and scheduling on multi-core architectures. Design, Automation and Test in Europe Conf. and Exhibition, p.33-38. [doi:10.1109/date.2009.5090629]

Yu, Z., Baas, B.M., 2009. High performance, energy efficiency, and scalability with GALS chip multiprocessors. *IEEE Trans. VLSI Syst.*, **17**(1):66-79. [doi:10.1109/tvlsi.2008.2001947]

Zhao, W., Liu, F., Agarwal, K., *et al.*, 2009. Rigorous extraction of process variations for 65-nm CMOS design. *IEEE Trans. Semicond. Manufact.*, **22**(1):196-203. [doi:10.1109/tsm.2008.2011182]