Performance Yield-Driven Task Allocation and Scheduling for MPSoCs under Process Variation

Lin Huang and Qiang Xu
CUhk REliable computing laboratory (CURE)
Department of Computer Science & Engineering
The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
Email: {lhuang,qxu}@cse.cuhk.edu.hk

ABSTRACT

With the ever-increasing transistor variability in CMOS technology, it is essential to integrate variation-aware performance analysis into the task allocation and scheduling process to improve its performance yield when building today's multiprocessor system-on-a-chip (MPSoC). Existing solutions assume that the execution times of tasks performed on different processors are statistically independent, which ignores the spatial correlation characteristics for systematic variation. In addition, a unified task schedule is constructed at design stage and applied to all products with various variation effects, which restricts the maximum performance yield that can be achieved for MPSoC products. To tackle the above problems, in this paper, we present a novel quasi-static scheduling algorithm. Based on a more accurate performance yield estimation method, a set of variation-aware schedules is synthesized off-line and, at run time, the scheduler will select the right one based on the actual variation for each chip, such that the timing constraint can be satisfied whenever possible. Experimental results demonstrate the effectiveness.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: Real-Time and Embedded Systems

General Terms

Algorithm, Design.

Keywords

Process Variation, Performance Yield, Task Scheduling

1. INTRODUCTION

Because of aggressive technology scaling, today's embedded computing systems are able to integrate multiple microprocessors and various hardware accelerators on a single silicon die, known as MPSoC. When building embedded applications, designers first partition them into hardware and software tasks. Then, task allocation and scheduling are carefully performed to effectively utilize the available processing elements (PEs) while satisfying various design constraints (e.g., timing constraints for real-time tasks).

At the same time, the relentless scaling of CMOS technology has also brought with ever-increasing variability in transistor parameters such as channel length, gate-oxide thickness and threshold voltage [4, 18]. While there have been extensive works in the literature to mitigate process variation effects at the logic level (e.g., statistical timing analysis and optimization [2]) and algorithmic level (e.g., variation-aware high-level synthesis [21]), we are not able to hide the variability at the system level any more. For embedded computing systems, designers are facing MPSoCs containing processors with variable frequencies across identically designed cores and across different identically designed chips.

In view of the above, variation-aware performance analysis needs to be integrated into the system-level task allocation and scheduling process for efficiently designing MPSoCs. Wang *et al.* [20] first addressed this problem by introducing the concept of *performance yield* for MPSoC designs, defined as the probability of the assigned schedule meeting timing constraints of the system. Statistical task graph analysis is then utilized in their task allocation and scheduling algorithm to maximize performance yield. Later, Singhal and Bozorgzadeh [17] proposed a non-probabilistic approach to reduce computational complexity in [20]. Recently, Chon and Kim [6] presented a new task allocation and scheduling method that takes the impact of resource sharing into consideration.

All the above works assume that task execution time follows Gaussian distribution and the execution times of tasks performed on different processor cores are statistically independent of each other (denoted as *s*-independent). However, a notable feature of within-die process variations is that they often exhibit themselves as *spatially-correlated systematic variations*, where devices close to each other have a higher probability of observing a similar variation level than devices that are far apart [14]. Because of this effect, identically designed processor cores that are nearby tend to have similar characteristics while variance is more severe among distant cores. Without considering the above systematic variation effects, the performance yield calculated in prior work is not accurate. Besides, without the *s*-independence assumption between the task execution times, most properties of Gaussian distribution cannot be applied directly to analyze the multivariate normal distribution. Consequently, closed-form statistical analysis becomes extremely difficult, if not impossible. To tackle the above problem, we rely on Monte Carlo simulation to estimate the performance yield for various task allocation and schedule solutions, and we show that a reasonable amount of test cases is sufficient to achieve high confidence for the accuracy.

In addition, prior work develops a unified schedule for all chips at design stage to maximize performance yield. Clearly, the effectiveness of such static solution decreases with the ever-increasing variation effects. We present a novel quasi-static scheduling strategy, wherein a set of variation-aware schedules is synthesized off-line and, at run time, the scheduler will select the right one based on the actual variation for each chip, such that the timing constraint can be satisfied whenever possible. Experimental results on various task graphs mapped onto hypothetical MPSoCs show that the proposed solution is able to dramatically improve the performance yield.

The remainder of this paper is organized as follows. Section 2 reviews related prior work and formulates the problem tackled in this paper. Section 3 presents the proposed quasi-static performance yield-driven task allocation and scheduling algorithm. Experimental results on various hypothetical platforms are presented in Section 4. Finally, Section 5 concludes this paper.

DAC'10, June 13-18, 2010, Anaheim, California, USA

2. PRIOR WORK AND MOTIVATION

2.1 Related Work

While there is a rich literature on task allocation and scheduling solutions for embedded system designs [5, 12], only a few explicitly take process variation effects into consideration. [20] introduced a new metric called performance yield to indicate the probability of MPSoC products meeting timing constraints given a certain task schedule, and proposed to enhance it by using statistical task graph analysis. The computation of two atomic functions, *sum* and *max*, is presented for the statistical timing analysis. The former one relies on the property of Gaussian distribution, while the latter one uses moment matching technique. Later, to avoid the time-consuming computations on distribution functions, [17] presented a non-probabilistic approach. It made a good attempt to handle the cases that the latency of processors follow an arbitrary distribution in the problem formulation, and reached theoretical conclusions for some simplistic scenarios. However, the proposed task allocation and scheduling technique was again based on the assumption of Gaussian delay distribution. Recently, [6] examined the impact of resource sharing on statistical static timing analysis and introduced an analytical framework to take this issue into account.

All the above works assume that the execution times of tasks follow Gaussian distribution. Yet it has been demonstrated that modeling the threshold voltage V_{th} and effective channel length L_{eff} with Gaussian distribution fits the empirical data well [15]. Then, according to the following processor frequency model [9], the execution time of a task induced by process variation can be approximated with Gaussian distribution in some instances at best.

$$f \propto \frac{(V_{dd} - V_{th})^{\alpha}}{L_{eff} V_{dd}} \tag{1}$$

where, V_{dd} is the supply voltage while α is a material-dependent constant.

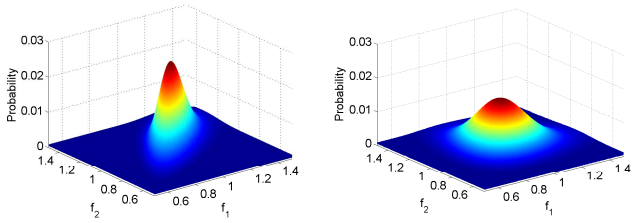
More importantly, prior works assume that the execution times of multiple tasks (on different processor cores, at least) are *s*-independent of each other. This assumption, however, ignores the spatial correlation characteristic of systematic variation, which is modeled by the following spherical function for its good agreement with empirical data [15].

$$\rho(r) = \begin{cases} 1 - \frac{3r}{2\phi} + \frac{1}{2}\left(\frac{r}{\phi}\right)^3 & r \le \phi \\ 0 & r > \phi \end{cases} \tag{2}$$

where, ϕ is the distance pass which the correlation becomes zero, and r is the distance between two elements on the chip. $\rho \in [0,1]$ is therefore a measure indicating their spatial correlation. The higher the correlation is, the closer ρ is to one. When two processors are correlated with each other in terms of threshold voltage V_{th} and effective channel length L_{eff} because of systematic process variation, their frequency values tend to have similar offsets [9, 15]. The jointly probability density function of two frequency values given $\rho = 0.8$ is illustrated in Fig. 1(a). For comparison, we also plot the jointly probability density function of two processors whose frequency values are totally uncorrelated (i.e., *s*-independent, or $\rho = 0$), as shown in Fig. 1(b). To get a clearer understanding, we use Kendall Tau rank correlation coefficient, which is defined as the probability of concordance of MPSoC pairs[1] minus that of disconcordance, to demonstrate the co-dependence of two processors fabricated on the same silicon die, as shown in Fig. 2.

With correlation, the statistical properties of *s*-independent Gaussian distributions serving as the basis of statistical task graph analysis in prior work are not applicable. For example, consider two random variables $X_1 \sim \mathcal{N}(30, 3^2)$ and $X_2 \sim \mathcal{N}(40, 4^2)$. Given strong correlation $\rho = 0.8$, the summation $X_1 + X_2$ does not follow the Gaussian distribution $\mathcal{N}(70, 5^2)$, as predicted with the mutually *s*-independent assumption. The appreciable difference in the summation distribution is depicted in Fig. 3.

[1]Consider two observations of MPSoCs whose operational frequency of processors are $< f_{\text{MPSoC}_1,P_1}, f_{\text{MPSoC}_1,P_2} >$ and $< f_{\text{MPSoC}_2,P_1}, f_{\text{MPSoC}_2,P_2} >$ respectively. If $(f_{\text{MPSoC}_1,P_1} - f_{\text{MPSoC}_2,P_1})$ and $(f_{\text{MPSoC}_1,P_2} - f_{\text{MPSoC}_2,P_2})$ have the same sign, this pair is called *concordant*; while otherwise *disconcordant*.



(a) Strong Correlation ($\rho = 0.8$) (b) No Correlation ($\rho = 0$)

Figure 1: Influence of Systematic Variation on Frequency Map.

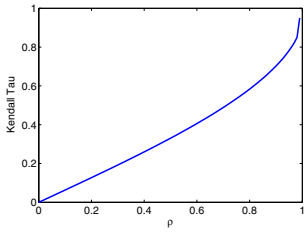


Figure 2: Kendall's Tau.

From the above, we can conclude that the performance yield calculation in prior work is not accurate enough, especially considering that the spatially-correlated systematic variation effects are increasingly significant with technology scaling [13, 16]. In addition, prior work tried to construct a unified task schedule for all chips at design stage. Such static solution ignored the unique processor frequency map for each MPSoC chip, which is available after manufacturing test. This inflexibility significantly constrains the maximum performance yield that designers can achieve.

The above limitations of prior works motivate the proposed performance yield-driven task allocation and scheduling solution in this paper.

2.2 Problem Formulation

The problem studied in this work is formulated as: Given

- A directed acyclic task graph $\mathcal{G} = (\mathcal{T}, \mathcal{E})$, wherein each node in $\mathcal{T} = \{\tau_i : i = 1, \cdots, n\}$ represents a task in $\mathcal{G}$, and $\mathcal{E}$ is the set of directed arcs which represent precedence constraints. Each task τ_i has a deadline d_i;
- A platform-based MPSoC embedded system and its floorplan. The platform consists of a set of processors $\mathcal{P} = \{P_j : j = 1, \cdots, m\}$, belonging to $\mathcal{V}$ categories;
- Task cycle count table $\{c_{i,j} : 1 \le i \le n, 1 \le j \le m\}$, where $c_{i,j}$ represents the task cycle count of task τ_i on processor P_j;

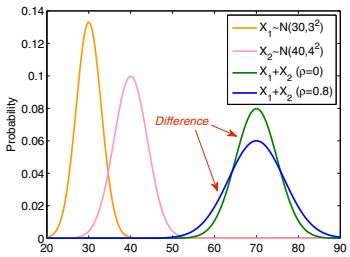


Figure 3: Influence of Correlation on Sum Operation.

- Parameters in process variation model: threshold voltage variation $\sigma_{V_{th}}$ and range ϕ;

To determine periodical task schedules such that the percentage of MPSoC products meeting performance constraints is maximized.

3. QUASI-STATIC TASK ALLOCATION AND SCHEDULING ALGORITHM

As discussed earlier, deriving a unified schedule for all MPSoC chips restricts the maximum performance yield that we can achieve and its effectiveness decreases with the ever-increasing variation effects. A straightforward thought to maximize performance yield is then to generate a specific task schedule for each individual chip. This extreme solution, however, is not quite practical because of the associated overhead of applying task allocation and scheduling algorithms on-chip and the additional design effort to prepare code and data of tasks for all kinds of processors to enable flexible task allocation. We therefore propose a quasi-static solution, wherein we prepare a set of task schedules at design stage, and at run time, the scheduler selects the right one based on the actual variation for each chip. To be specific, we first generate an initial task schedule with simulated annealing-based technique, taking stochastic properties of MPSoC frequency map into account (Section 3.1). Then, we use data clustering technique to derive additional task schedules to further improve MPSoC performance yield (Section 3.2).

3.1 Initial Task Scheduling

3.1.1 Simulated Annealing Procedure

In this section, we resort to a modified simulated annealing technique to find a task schedule such that the performance yield is maximized. When compared with the classic procedure of simulated annealing, the differences stay particularly in the function YIELD ESTIMATION and MEET ACCEPTANCE CONDITION in the flow shown in Fig. 4. The task schedule obtained with this method plays an important role in the quasi-static scheduling process, denoted by initial task schedule (or S_{init}).

The task schedule S is described with two sequences: (*scheduling order sequence*, *resource binding sequence*) [10]. For example, given the task graph shown in Fig. 5 is allocated onto an MPSoC with two processors, a feasible schedule is (τ_1, τ_3, τ_2, τ_4, τ_5; P_1, P_2, P_1, P_1, P_2), meaning that task τ_1 is scheduled first, followed by task τ_3, τ_2, τ_4, and τ_5. Task τ_1, τ_3, τ_4 are allocated onto processor P_1, while τ_2 and τ_5 on processor P_2. To generate a new schedule, we reuse the two types of random moves defined in [10] that are capable to search complete solution space: (i). swap two adjacent elements in scheduling order sequence if they do not have any precedence constraints; (ii). change an element in resource binding sequence.

	Function INITIAL TASK SCHEDULING
1	Build an initial solution S and initialize T as T_{init}
2	$\widetilde{Y}_{old} \leftarrow$ YIELD ESTIMATION (S)
3	**While** $T > T_{end}$
4	**For** the iteration i from 1 to I
5	$S' \leftarrow$ RANDOM MOVE (S)
6	$\widetilde{Y}_{new} \leftarrow$ YIELD ESTIMATION (S')
7	**If** MEET ACCEPTANCE CONDITION ($\widetilde{Y}_{old}$, $\widetilde{Y}_{new}$)
8	$S \leftarrow S'$
9	$\widetilde{Y}_{old} \leftarrow \widetilde{Y}_{new}$
10	$T \leftarrow T \times R_{cooling}$

Figure 4: Main Flow of Initial Task Scheduling.

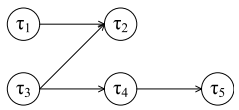


Figure 5: An Example of Task Graph.

3.1.2 Yield Estimation and Error Analysis

To find the task schedule with maximum performance yield, every time a new schedule is constructed, we need to estimate the performance yield obtained by applying this schedule on MPSoC chips. Clearly, given the process variation model and MPSoC floorplan, performance yield Y is a function of task schedule S. However, since the s-independence assumption between the execution times of tasks do not hold (due to the spatial correlations for processor cores), it is extremely difficult, if not impossible to derive a closed-form expression for Y because most properties of Gaussian distribution cannot be applied directly to analyze the multivariate normal distribution. We therefore estimate the performance yield with Monte Carlo simulation, wherein the variation effects of the sample chips are generated according to process variation model. The objective function $Y(S)$ is then approximated by the sample average

$$Y(S) \approx \frac{1}{N}\sum_{i=1}^{N} x(\omega_i, S) \tag{3}$$

where, $\omega_1, \cdots, \omega_N$ are independent identically distributed samples of MPSoC frequency map, called test chips. $x(\omega_i, S)$ is a binary value, representing whether test chip with frequency map ω_i is able to meet performance constraints, provided the task schedule S. For the ease of discussion, we hereafter use x_i to represent $x(\omega_i, S)$ and Y to represent $Y(S)$.

For effective performance yield estimation, we are interested in the efficiency of Monte Carlo simulation, that is, *how many test chips do we need to consider in order to achieve a certain accuracy?* To answer this question, we perform theoretical analysis in the following.

Consider the test chip set we mentioned before. Without loss of generality, we assume M out of them meet performance constraints, where $M \leq N$. By Eq. (3), the performance yield Y is approximated by

$$\widetilde{Y} = \frac{1}{N}\sum_{i=1}^{N} x_i = \frac{M}{N} \tag{4}$$

Although we cannot tell $\widetilde{Y}$ must lie in a certain range around Y, we know *with a desired confidence level* $\widetilde{Y}$ stays in the confidence interval. To be specific, provided the variation of performance yield σ^2 is finite, we apply the central-limit theorem and have

$$\Pr\{|\widetilde{Y} - Y| < \frac{\beta\sigma}{\sqrt{N}}\} \approx \Phi(\beta) - \Phi(-\beta) \tag{5}$$

where, $\beta\sigma/\sqrt{N}$ is the half length of confidence interval, denoted by ξ in the rest of this paper. $\big(\Phi(\beta) - \Phi(-\beta)\big)$ is the confidence level of the event that the approximated performance yield $\widetilde{Y}$ is in $(Y-\xi, Y+\xi)$. $\Phi(x)$ represents the cumulative distribution function of normal distribution.

The variance σ^2 in Eq. (5) cannot be derived from multivariate normal distribution directly, but could be approximated with the sample data [11], namely

$$\widetilde{\sigma}^2 = \frac{N}{N-1}\big(\frac{1}{N}\sum_{i=1}^{N} x_i^2 - \widetilde{Y}^2\big) = \frac{N}{N-1}\big(\frac{M}{N} - \frac{M^2}{N^2}\big) \tag{6}$$

Thus, we have

$$\xi \approx \beta\sqrt{\frac{M}{N(N-1)} - \frac{M^2}{N^2(N-1)}} \tag{7}$$

which implies that the length of the confidence interval depends on both test chip quantity N and M, provided a certain confidence level. For instance, if 1,000 test chips are generated for performance yield estimation and confidence level is set to 95%, the confidence interval half length is in the range between 0 and 0.031. The confidence interval around the approximated value is depicted in Fig. 6.

3.1.3 Acceptance Condition

In the simulated annealing process, we need to compare the newfound task schedule with the original one in terms of performance yield to decide whether the newfound schedule should be accepted. Since we cannot obtain the exact value of performance yield but an approximation only, we need to use the approximated values for comparison (line 7 in Fig. 4). It is important to note that $\widetilde{Y}_1 > \widetilde{Y}_2$ does not imply $Y_1 > Y_2$ in this context. Instead, the difference of exact values $(Y_1 - Y_2)$ is in a range around $(\widetilde{Y}_1 - \widetilde{Y}_2)$. In particular, with confidence level $\big(\Phi(\beta) - \Phi(-\beta)\big)$ we have

$$\widetilde{Y}_1 - \widetilde{Y}_2 - \frac{t \cdot \sigma_{1,2}}{\sqrt{N}} < Y_1 - Y_2 < \widetilde{Y}_1 - \widetilde{Y}_2 + \frac{t \cdot \sigma_{1,2}}{\sqrt{N}} \tag{8}$$

where, t is the value of t-distribution with $(N-1)$ degrees of freedom given the confidence level. $\sigma_{1,2}$ is the variation of $(\widetilde{Y}_1 - \widetilde{Y}_2)$, which can be expressed in terms of the variations σ_1, σ_2 and the correlation $\hat{\rho}_{1,2}$ between $\widetilde{Y}_1$ and $\widetilde{Y}_2$, that is,

$$\sigma_{1,2} = \frac{1}{N}\left(\sigma_1^2 + \sigma_2^2 - 2\hat{\rho}_{1,2}\sigma_1\sigma_2\right) \tag{9}$$

We reach to the conclusion at this point that $Y_1 > Y_2$ if and only if $\widetilde{Y}_1 - \widetilde{Y}_2 - \frac{t \cdot \sigma_{1,2}}{\sqrt{N}} > 0$, given the confidence level.

With these arguments, the acceptance condition of a newfound solution is set to

$$\widetilde{Y}_{old} - \widetilde{Y}_{new} - \frac{t \cdot \sigma_{old,new}}{\sqrt{N}} > 0 \tag{10}$$

or

$$\exp\left(\frac{\widetilde{Y}_{old} - \widetilde{Y}_{new} - \frac{t \cdot \sigma_{old,new}}{\sqrt{N}}}{T}\right) > rand() \tag{11}$$

where the latter is to accept a "bad" schedule with certain probability during the annealing process to jump out of a local optimal solution.

3.2 Clustering-Based Performance Yield Enhancement

With the initial task schedule derived earlier, some test chips might cannot meet performance constraints. Consequently, we need to generate more task schedules for performance yield enhancement. This requires the collection of the test chips that cannot meet deadlines (referred to as *residual test chips* or $\mathcal{W}_{bias}$ hereafter) and the extraction of their frequency map characteristics. In this subsection, we use *k-mean algorithm* [19], a data clustering technique, to classify these test chips into a few clusters and generate additional task schedules to further improve performance yield.

As illustrated in Fig. 7, in the beginning of our algorithm, we only have the initial task schedule and we initialize the task schedule set $\mathcal{S}$ to be $\{S_{init}\}$. At present, the total performance yield $\widetilde{Y}_{total}$ is simply the performance yield achieved by using this only task schedule. Next, in each iteration a new task schedule S_{best} is generated according to residual test chip characteristics, and the performance yield is re-estimated (i.e., $\widetilde{Y}_{total}$ is updated). This procedure terminates if the user-defined maximum permissible number of task schedules $U_{\max}$ is reached.

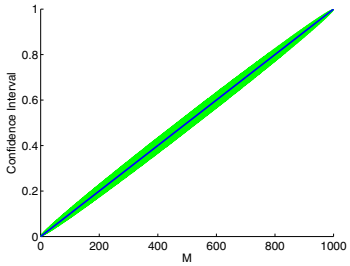


Figure 6: Confidence Interval Given 1,000 Test Chips with Confidence Level 95%.

	Function PERFORMANCE YIELD ENHANCEMENT
1	Initialize $\widetilde{Y}_{total}$ as $\widetilde{Y}(S_{init})$, $\mathcal{S} \leftarrow \{S_{init}\}$, and $k \leftarrow k_{init}$
2	**While** $\lvert\mathcal{S}\rvert < U_{\max}$ and $k < \lvert\mathcal{W}_{bias}\rvert$
3	$\{C_1, \cdots, C_k\} \leftarrow$CLUSTERING $(\mathcal{W}_{bias})$
4	**For** i from 1 to k
5	$S_i \leftarrow$TASK SCHEDULING (C_i)
6	$S_{best} \leftarrow$ BEST SCHEDULE $(S_1, \cdots, S_k)$
7	$\widetilde{Y}_{new} \leftarrow$ YIELD ESTIMATION $(\mathcal{S}, S_{best})$
8	**If** $\widetilde{Y}_{new}$ is higher than $\widetilde{Y}_{total}$ with certain confidence level
9	$\mathcal{S} \leftarrow \{\mathcal{S}, S_{best}\}$
10	$\widetilde{Y}_{total} \leftarrow \widetilde{Y}_{new}$
11	UPDATE $(\mathcal{W}_{bias})$
12	$k \leftarrow k_{init}$
13	**Else**
14	$k \leftarrow k+1$
15	SELECTION CRITERIA GENERATION $(\mathcal{S})$

Figure 7: Main Flow of Clustering-Based Performance Yield Enhancement.

To generate a new task schedule, we first classify the residual test chips into k clusters according to their frequency map by using k-mean algorithm [19]. This algorithm first makes initial guess for the centroid of every cluster. Next, it assigns every point to the cluster whose centroid is the nearest and then recomputes centroid of clusters in an iterative manner. This procedure repeats until the centroid of clusters does not change any more. For example, suppose a task graph with 31 tasks is assigned onto an MPSoC with two processor cores, 79.5% test chips out of 1,000 are able to meet performance constraints by using initial task schedule (i.e., Cluster 0 in Fig. 8). The residual test chips are categorized into three clusters and plotted in Fig. 8. Here, the quantity of clusters k is initialized as an user-defined value k_{init} before task schedule generation.

Next, a task schedule is generated for each cluster (line 4-5). To reduce runtime, we start from S_{init} and change the resource binding and/or scheduling for a few times, where the frequency map of the centroid is used for makespan calculation. To make the most of test chips in a cluster meet performance constraints, the task schedule which results in the shortest makespan during this procedure is accepted and denoted by S_i. It is necessary to highlight that we assume that the raw data of each task is prepared for one category of processors only, where the category is determined by S_{init}. That is, for heterogeneous MPSoCs, given task τ_i is allocated onto a processor belonging to category v_j, in the clustering-based performance yield enhancement it is allowed to be assigned to the processors in category v_j only. Then, the task schedule with the minimum makespan is selected and marked as S_{best} (line 6). It is possible that the resulting S_{best} cannot provide benefit in terms of $\widetilde{Y}_{total}$, because we use the centroid of each cluster for task scheduling. In this case, we increment k by one to shrink the cluster size and perform the task schedule generation again (line 14). Otherwise, S_{best} is included into schedule set $\mathcal{S}$ (line 9) and $\mathcal{W}_{bias}$ is recomputed (line 11).

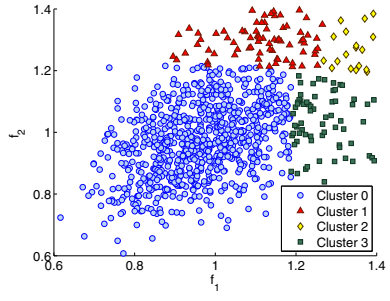


Figure 8: An Example of k-Mean Clustering.

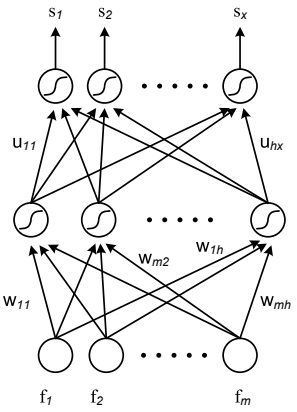


Figure 9: Selection Criteria Network.

The last step (line 15) is to generate the task schedule selection criteria by using multilayer perceptron [8], a machine learning technique. At run time, the scheduler will select task schedule according to these criteria. For this purpose, we build a multilayer sigmoid network with one input, one hidden, and one output layer, as shown in Fig. 9. This network is off-line trained by using the well-studied backpropagation algorithm [8], taking test chips and their corresponding feasible task schedules as training samples. That is, we train the weight parameters of the network ($\vec{u}$ and $\vec{w}$) such that given the input vector $\vec{f}$ (i.e., the frequency map of an MPSoC) the network provides an output vector $\vec{s}$, indicating which task schedules are able to meet performance constraints. For instance, suppose the second task schedule is able to meet performance constraints for a particular MPSoC chip, the inputs are the frequency values of processors on this MPSoC and the outputs should be $\vec{s} = (0,1,0,\cdots,0)$. Note that, both hidden layer and output layer employ sigmoid function[2], because the value of an output node i naturally implies the probability that the i^{th} task schedule meets performance constraints. It is also worth to note that the storage overhead for the network is acceptable since the number of task schedules is bounded by $U_{\max}$.

At run time, given an MPSoC product, its exact frequency map becomes available with the mature speed binning techniques. The scheduler takes this information as the input of selection criteria network, and compute the output vector with forward propagation. Suppose s_i has the highest values among all the elements in $\vec{s}$, the i^{th} task schedule is loaded and executed by scheduler. Since this selection is conducted only when the system is being initialized before usage or the frequency map is changed due to aging effects, the performance overhead for schedule selection is negligible.

4. EXPERIMENTAL RESULTS

To evaluate the effectiveness and efficiency of the proposed algorithm, we conduct a set of experiments on hypothetical MPSoCs. The task graphs are generated by TGFF [7], whose attributes are described in Table 1. These task graphs are allocated onto some hypothetical MPSoCs whose floorplan are illustrated in Table 2. The process variation model is assumed to follow multivariate normal distribution with spatial correlation [15]. The variation $\sigma_{V_{th}}$ is set to 3.2%, and the distance pass which the correlation becomes zero is set to $\phi = \{0.1, 0.5\}$ [15] unless noted otherwise.

The simulated annealing parameters are set to $I = 10^3$, $T_{init} = 10^2$, $T_{end} = 10^{-3}$, $R_{cooling} = 0.9$. For performance yield estimation, 1,000 test chips are generated with process variation model (i.e., $N = 10^3$) while the confidence level is set as 95% ($\beta = 1.96$). In the performance yield enhancement, at most 10 task schedules are generated, that is, $U_{\max} = 10$. In addition, the quantity of clusters is initialized as $k_{init} = 5$.

[2]Sigmoid function has the form $g(o) = \frac{1}{1+e^{-o}}$, ranging between 0 and 1.

label	task #	edge #	in-degree mean/std	out-degree mean/std	task # on longest path
$\mathcal{G}_a$	31	40	1.29 / 0.81	1.29 / 1.17	10
$\mathcal{G}_b$	69	100	1.45 / 0.99	1.45 / 1.17	13
$\mathcal{G}_c$	152	233	1.53 / 0.79	1.53 / 0.74	28

Table 1: Description of Task Graphs.

label	MPSoC floorplan	processor types
$\mathcal{P}_{\text{small}}$	2×2 mesh	$v_1\ v_1$ $v_1\ v_1$
$\mathcal{P}_{\text{homo}}$	2×4 mesh	$v_1\ v_1\ v_1\ v_1$ $v_1\ v_1\ v_1\ v_1$
$\mathcal{P}_{\text{hete}}$	2×4 mesh	$v_1\ v_2\ v_2\ v_1$ $v_1\ v_2\ v_2\ v_1$

Table 2: Description of MPSoCs.

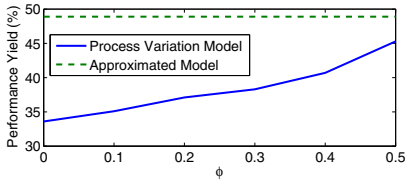


Figure 10: Difference in Performance Yield Estimation.

Existing variation-aware task allocation and scheduling techniques (e.g., [20, 17]) calculate MPSoC performance yield under the assumption that task execution times follow Gaussian distribution and tasks conducted on different processor cores are s-independent. We use one example schedule to show the difference between such calculated performance yield and our estimated one based on Monte Carlo simulation. The task schedule is generated by using the algorithm proposed in [3], an extension of classic Highest Levels First with Estimated Times (HLFET) [1] for heterogeneous systems, for the case with $\mathcal{G}_b$, $\mathcal{P}_{\text{homo}}$, $d = 350$. On one hand, we generate a set of test chips with the process variation model and estimate the performance yield with 95% confidence level, for ϕ from 0 to 0.5. On the other hand, we use curve fitting technique to approximate the execution time of each task with Gaussian distribution and calculate the performance yield according to [20] under the s-independent assumption. The significant difference in the results obtained by using these two methods is illustrated in Fig. 10[3]. Clearly, the calculated performance yield in prior work is rather inaccurate.

Because the performance yield were obtained differently, the proposed method and prior work actually have different optimization objectives and it would be not fair to compare the effectiveness of the task schedules obtained from them. For this reason, we take [3] instead of the existing process variation-aware scheduling results as the baseline solution in our experiments. Since this baseline solution does not consider process variation effects, we should treat it as a reference only.

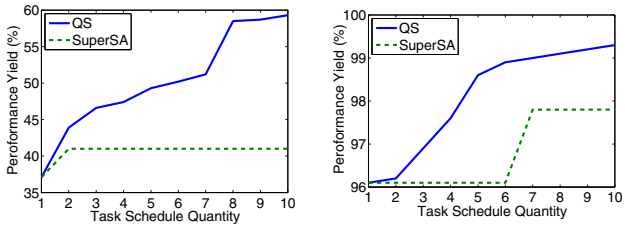
Table 3 compares the performance yield achieved by various approaches, where QS is used to label results obtained with the proposed quasi-static scheduling technique. The difference between QS and baseline (namely, Δ_2 in the table) is in the range of 42.3-99.9%. When spatial correlation is strong ($\phi = 0.5$), the improvement is mainly credited to initial task schedule S_{init} while the effectiveness of clustering-based enhancement is limited (e.g., the case with $\mathcal{G}_b$, $\mathcal{P}_{\text{homo}}$, $\phi = 0.5$, $d = 300$). This is because the processors on the same MPSoC tend to have similar variations and a task schedule suitable for this characteristic is able to bring the performance yield to a high level. Given weak spatial correlation ($\phi = 0.1$), on the other hand, the performance yield enhancement tends to rely on

[3]Note that, the inaccurate model in prior work can result in arbitrary biases of performance yield with changing correlation effects. The example shown in Fig. 10 is a particular case and it does not imply that prior works estimate performance yield more accurately with higher correlation effects.

ϕ	d	Performance yield (%)					
		baseline	S_{init}	Δ_1	QS	Δ_2	Δ_3
$\mathcal{G}_a$, $\mathcal{P}_{\text{small}}$							
0.5	175	0	37.8	37.8	59.5	59.5	21.7
	200	0.2	82.6	82.4	89.8	89.6	7.2
	225	2.5	95.1	92.6	97.8	95.3	2.7
0.1	175	0	24.1	24.1	50.2	50.2	26.1
	200	0	74.3	74.3	92.2	92.2	17.9
	225	0.1	96.3	96.2	100	99.9	3.7
$\mathcal{G}_b$, $\mathcal{P}_{\text{homo}}$							
0.5	275	0.8	63.7	62.9	72.6	71.8	8.9
	300	**5.7**	**79.8**	**74.1**	**89.6**	**83.9**	**9.8**
	325	19.7	92.7	73.0	95.3	75.6	2.6
0.1	275	0	24.1	24.1	53.6	53.6	29.5
	300	0.3	81.0	80.7	94.3	94.0	13.3
	325	8.0	92.9	84.9	100	92.0	7.1
$\mathcal{G}_c$, $\mathcal{P}_{\text{homo}}$							
0.5	700	12.4	63.2	50.8	70.6	58.2	7.4
	750	30.9	81.6	50.7	87.1	56.2	5.5
	800	52.5	92.5	40.0	94.8	42.3	2.3
0.1	700	1.2	12.1	10.9	71.3	70.1	59.2
	750	13.5	85.6	72.1	91.8	78.3	6.2
	800	42.7	94.1	51.4	98.5	55.8	4.4
$\mathcal{G}_b$, $\mathcal{P}_{\text{hete}}$							
0.5	275	1.4	84.0	82.6	91.1	89.7	7.1
	300	9.2	92.8	83.6	95.7	86.5	2.9
	325	28.4	95.6	67.2	98.6	70.2	3.0
	275	**0.1**	**40.7**	**40.6**	**79.7**	**79.6**	**39.0**
0.1	300	1.4	96.1	94.7	99.3	97.9	3.2
	325	13.3	99.3	86.0	100	86.7	0.7

d: deadline; Δ_1: the difference between baseline and S_{init};
Δ_2: the difference between baseline and QS;
Δ_3: the difference between S_{init} and QS

Table 3: Effectiveness of the Proposed Approach.



(a) $\mathcal{G}_b$, $\mathcal{P}_{\text{homo}}$, $\phi = 0.3$, $d = 275$ (b) $\mathcal{G}_b$, $\mathcal{P}_{\text{hete}}$, $\phi = 0.1$, $d = 300$

Figure 11: Performance Yield Increment with Quasi-Static Task Schedule Increase.

clustering-based task schedule generation (e.g., the case with $\mathcal{G}_b$, $\mathcal{P}_{\text{hete}}$, $\phi = 0.1$, $d = 275$). In this case, the deviation of frequency maps is significant. As a result, the simulated annealing-based technique that considers the overall characteristics of all chips is not as effective as that in strong correlation case. Under such circumstances, the characteristic of test chips exhibits separate clustering effects and hence generating individual task schedule for each cluster is more effective.

In Fig. 11, we show the tradeoff between the number of task schedules and the corresponding performance yield. For comparison, we perform the proposed simulated annealing-based task scheduling for the same cases, while more schedules (at most $U_{\max}$) with the highest performance yield are maintained in the searching process, referred to as SuperSA. In most cases, the initial task schedule is the most effective one. An example is shown in Fig. 11(a), where S_{init} provides 36.9% performance yield improvement when compared with the baseline solution (0.2% in this case). The remaining schedules in set $\mathcal{S}$ further enhance the result to 59.3%. SuperSA, however, only results in slight improvement with more task schedules, around 3.9%. The observations in Fig. 11(b) are similar, where the proposed QS results in higher performance yield than SuperSA. The above results demonstrate the effectiveness of our clustering-based performance yield enhancement technique.

5. CONCLUSION

In this paper, we present a novel quasi-static variation-aware task allocation and scheduling technique for MPSoC designs. Based on a more accurate performance yield estimation method, the proposed simulation annealing based scheduling algorithm together with the novel clustering-based performance yield enhancement technique can significantly improve the performance yield of MPSoCs.

6. ACKNOWLEDGEMENT

The authors would like to thank Dr. Yuan Xie and Mr. Yibo Chen from Penn State University and Dr. Zonghua Gu from Zhejiang University for their insightful comments to this work.

This work was supported in part by the General Research Fund CUHK417807 and CUHK418708 from Hong Kong SAR Research Grants Council (RGC), in part by National Science Foundation of China (NSFC) under grant No. 60876029, and in part by a grant N_CUHK417/08 from the NSFC/RGC Joint Research Scheme.

7. REFERENCES

[1] T. Adam, K. Chandy, and J. Dickson. A Comparison of List Scheduling for Parallel Processing Systems. *Communications of the ACM*, 17(12):685–690, December 1974.

[2] A. Agarwal, D. Blaauw, and V. Zolotov. Statistical Timing Analysis for Intra-Die Process Variations with Spatial Correlations. In *Proc. ICCAD*, pages 900–907, 2003.

[3] P. Bjorn-Jorgensen and J. Madsen. Critical Path Driven Cosynthesis for Heterogeneous Target Architectures. In *Proc. International Conference on Hardware Software Codesign*, pages 15–19, 1997.

[4] S. Borkar et al. Parameter Variations and Impact on Circuits and Microarchitecture. In *Proc. DAC*, pages 338–342, 2003.

[5] T. D. Braun, H. J. Siegel, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61(6):810–837, June 2001.

[6] H. Chon and T. Kim. Timing Variation-Aware Task Scheduling and Binding for MPSoC. In *Proc. ASP-DAC*, pages 137–142, 2009.

[7] R. P. Dick, D. L. Rhodes, and W. Wolf. TGFF: Task Graphs for Free. pages 97–101, 1998.

[8] S. Haykin. In *Neural Networks and Learning Machines*. Prentice-Hall, 3rd edition, 2008.

[9] S. Herbert and D. Marculescu. Characterizing Chip-Multiprocessor Variability-Tolerance. In *Proc. DAC*, pages 313–318, 2008.

[10] L. Huang, F. Yuan, and Q. Xu. Lifetime Reliability-Aware Task Allocation and Scheduling for MPSoC Platforms. In *Proc. DATE*, 2009.

[11] P. Jäckel. In *Monte Carlo Methods in Finance*. John Wiley & Sons, 2002.

[12] Y.-K. Kwok and I. Ahmad. Static task scheduling and allocation algorithms for scalable parallel and distributed systems: Classification and performance comparison. In Y. C. Kwong, editor, *Annual Review of Scalable Computing*, pages 107–227. 2000.

[13] X. Liang, G.-Y. Wei, and D. Brooks. ReVIVaL: A Variation-Tolerant Architecture Using Voltage Interpolation and Variable Latency. In *Proc. ISCA*, pages 191–202, 2008.

[14] S. Nassif. Delay Variability: Sources, Impacts and Trends. In *Proc. ISSCC*, pages 368–369, 2000.

[15] S. R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas. Varius: A model of process variation and resulting timing errors for microarchitects. *IEEE Transactions on Semiconductor Manufacturing*, 21(1):3–13, February 2008.

[16] SIA. International Technology Roadmap for Semiconductors. http://public.itrs.net.

[17] L. Singhal and E. Bozorgzadeh. Process Variation Aware System-Level Task Allocation Using Stochastic Ordering of Delay Distributions. In *Proc. ICCAD*, pages 570–574, 2008.

[18] B. E. Stine, D. S. Boning, and J. E. Chung. Analysis and Decomposition of Spatial Variation in Integrated Circuit Processes and Devices. *IEEE Transactions on Semiconductor Manufacturing*, 10(1):24–41, Feb. 1997.

[19] P.-N. Tan, M. Steinbach, and V. Kumar. In *Introduction to Data Mining*. Addison-Wesley, 2006.

[20] F. Wang, C. Nicopoulos, X. Wu, Y. Xie, and N. Vijaykrishnan. Variation-Aware Task Allocation and Scheduling for MPSoC. In *Proc. ICCAD*, pages 598–603, 2007.

[21] F. Wang, X. Wu, and Y. Xie. Variability-Driven Module Selection with Joint Design Time Optimization and Post-Silicon Tuning. In *Proc. ASP-DAC*, pages 2–9, 2008.