

Variation-aware Task Allocation and Scheduling for MPSoC

Feng Wang, C. Nicopoulos, Xiaoxia Wu, Yuan Xie and N. Vijaykrishnan
Pennsylvania State University, University Park, PA, 16802
Email: fenwang@cse.psu.edu

Abstract—As technology scales, the delay uncertainty caused by process variations has become increasingly pronounced in deep sub-micron designs. As a result, a paradigm shift from deterministic to statistical design methodology at all levels of the design hierarchy is inevitable [1]. In this paper, we propose a variation-aware task allocation and scheduling algorithm for Multiprocessor System-on-Chip (MPSoC) architectures to mitigate the impact of parameter variations. A new design metric, called *performance yield* and defined as the probability of the assigned schedule meeting the predefined performance constraints, is used to guide the task allocation and scheduling procedure. An efficient yield computation method for task scheduling complements and significantly improves the effectiveness of the proposed variation-aware scheduling algorithm. Experimental results show that our variation-aware scheduler achieves significant yield improvements. On average, 45% and 34% yield improvements over worst-case and nominal-case deterministic schedulers, respectively, can be obtained across the benchmarks by using the proposed variation-aware scheduler.

I. INTRODUCTION

Aggressive technology scaling enables integration capacities of billions of transistors [1]. As a result, entire systems can now be integrated on a single chip die (System-on-Chip, or SoC) [2]. In fact, many embedded systems nowadays are heterogeneous multiprocessors with several different types of Processing Elements (PEs), including customized hardware modules (such as Application-Specific Integrated Circuits, or ASICs), programmable microprocessors, and embedded Field-Programmable Gate Arrays (FPGA), all of which are integrated on a single die to form what is known as a Multiprocessor System-on-Chip (MPSoC) [2].

The Intellectual Property (IP) re-use approach has been widely advocated as an effective way to improve designer productivity and efficiently utilize the billions of available transistors in complex SoC designs [2]. In this approach, the designers are required to partition the logic functionality into hard and soft modules. By assigning the software functions to appropriate hardware PEs, the designer must then determine if the resulting embedded system can meet the real-time constraints imposed by the design specifications [2]. Design decisions taken at the early stages of the design process are critical in avoiding potentially high-cost alterations in the more advanced phases of the process. Consequently, the designer must conduct early analysis and evaluation to guarantee that the performance and cost targets are met. Early-stage evaluation tools with accurate system analysis capabilities enable the SoC designers to explore various high-level design alternatives that meet the expected targets. Traditionally, task scheduling and allocation based on a worst-case timing analysis is used to evaluate these possible designs during the early design stages. Hence, the deterministic slack can be used to assess the tightness of the timing constraints and guide the task scheduling and allocation in this early exploration process.

However, the challenges in fabricating transistors with diminutive feature sizes in the nanometer regimes have resulted in significant variations in key transistor parameters, such as transistor channel length, gate-oxide thickness, and threshold voltage. This manufacturing variability can, in turn, cause significant performance and power deviations from nominal values in identical hardware designs. For example, Intel has shown that process variability can cause up to

a 30% variation in chip frequency and up to a 20x variation in chip leakage power for a processor designed in 180 nm technology [1]. Designing for the worst case scenario may no longer be a viable solution, especially when the variability encountered in the new process technologies becomes very significant and causes substantial percentage deviations from the nominal values. Increasing cost sensitivity in the embedded system design methodology makes designing for the worst case infeasible. Further, worst-case analysis without taking the probabilistic nature of the manufactured components into account can also result in an overly pessimistic estimation in terms of performance. For example, Fig. 1 shows that simply adding the Worst Case Execution Time (WCET) of two tasks can result in a pessimistic estimation of the total WCET, and may end up necessitating the use of excess resources to guarantee real-time constraints.

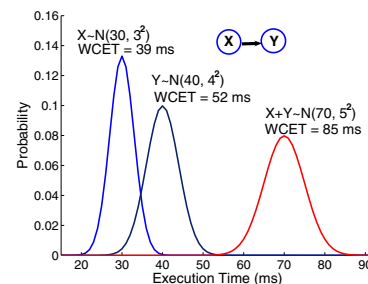


Fig. 1. The execution times of X and Y have independent Gaussian distributions $N(\mu, \sigma^2)$. The WCET is calculated as $\mu + 3\sigma$. $X+Y$ follows $(N(\mu_X + \mu_Y, \sigma_X^2 + \sigma_Y^2))$. Therefore, $WCET_{X+Y} < WCET_X + WCET_Y$.

In this paper, variation-aware performance analysis is integrated into the task scheduling and allocation process for efficiently designing SoCs in the presence of unpredictable parameters. Accurate early analysis is very critical, because system-level performance evaluation influences early design decisions that later impact the overall design complexity and cost. In addition to variation-aware analysis, we introduce the new concept of *parametric yield* to accommodate the new reality of non-negligible variability in modern MPSoC architectures. Traditionally, yield has been viewed as a metric to determine hardware implementations which meet the predefined frequency requirements. Manufacturers reject the subset of dies that fail to meet the required performance constraints. Thus, classification based on manufacturing yield is very important from a commercial point of view. In this work, we extend the notion of yield to a higher abstraction level in the design flow: we define the *parametric yield* of the SoC design as the probability of the design meeting the real-time constraints imposed by the underlying system. Subsequently, designs that cannot meet the real-time constraints have to be discarded. Hence, this definition of yield relates directly to the design cost and design effort and is a direct corollary of decisions and choices made throughout the design process. Experimental results clearly indicate that the proposed process variation-aware approach in the early design exploration phase can lead to significant yield

gains down the line. Specifically, our variation-aware scheduler can obtain 45% and 34% performance yield improvements over worst-case and nominal-case conventional (i.e. deterministic) schedulers, respectively, across a wide gamut of MPSoC benchmarks.

The rest of the paper is organized as follows: Section II reviews the related work in this area; in Section III, the process variation-aware task scheduling problem is formulated; Section IV discusses statistical timing analysis for task scheduling and introduces the criticality of the computation method; Section V presents the variation-aware task scheduling algorithm. Section VI shows experimental results and Section VII concludes the paper.

II. RELATED WORK

Related work pertaining to this paper can be divided into three different categories: traditional task scheduling for embedded systems, gate-level statistical analysis and optimization, and probabilistic analysis for real-time embedded systems.

There have been extensive studies in the literature on task allocation and scheduling for embedded systems. Precedence-constrained task allocation and scheduling has been proven to be an NP-hard problem; thus, allocation and scheduling algorithms usually use a variety of heuristics to quickly find a sub-optimal solution [3] [4]. Recently, several researchers have investigated the task scheduling problem for Dynamic-Voltage-Scaling-enabled (DVS) real-time multi-core embedded systems. Zhang et al. [5] formulated the task scheduling problem combined with voltage selection as an Integer Linear Programming (ILP) problem. Luo et al. [6] proposed a condition-aware DVS task scheduling algorithm, which can handle more complicated conditional task graphs. In a more recent work [7], Hu et al. proposed task and communication scheduling for Network-on-Chip (NoC) architectures under real-time constraints.

Early statistical timing analysis approaches were based on Monte Carlo techniques, which are expensive in terms of computation complexity and, therefore, are not suitable for large systems. Most state-of-the-art statistical timing analysis methods [8], [9] can be categorized into two major types: path-based approaches and block-based approaches. Several statistical optimization approaches have been proposed, such as statistical gate sizing to improve the probability of meeting timing constraints [10], [11], technology mapping, and statistical buffer insertion.

More related work on probabilistic real-time embedded system design will be discussed in Section III-C, after the problem formulation is presented in Section III-B.

Our contributions in this paper distinguish themselves in the following aspects: 1) we first formulate the process-variation-aware task scheduling problem for MPSoC architectures and propose an efficient variation-aware scheduling algorithm to solve it; the novelty lies in the augmentation of process-variability-induced uncertainties in the scheduling model. 2) We subsequently introduce and employ the notion of *performance yield* in the dynamic priority computation of the task scheduling process. 3) Finally, we develop a yield computation method for the partially scheduled task graphs.

III. PRELIMINARIES

This section lays the foundations upon which our proposed variation-aware task allocation and scheduling algorithm rests. We first describe the evaluation platform specification and the specifics of the assumed variation modeling. We then present the problem formulation, and finally we discuss how our proposed work complements and expands on existing probabilistic real-time embedded system research.

A. Platform Specification and Modeling

Heterogeneous multiprocessors tend to be more efficient than homogenous multiprocessor implementations at tackling inherent application heterogeneity [2], since each processing element is optimized for a particular part of the application running on the system. Motivated by this characteristic, the PEs of the MPSoC platform utilized in this work are assumed to be heterogeneous and interconnected - without loss of generality - with a bus. For instance, the PE can be a Digital Signal Processor (DSP), general-purpose CPU or an FPGA [2].

Our delay distribution model for the PEs is based on the maximum delay distribution model of processors, as presented in [12]. According to this model, the critical path delay distribution of a processor due to inter-die and intra-die variations is modeled as two normal distributions: $f_{inter} = N(T_{norm}, \sigma D_{inter})$ and $f_{intra} = N(T_{norm}, \sigma D_{intra})$, respectively. T_{norm} is the mean value of the critical path delay. The impact of both inter-die and intra-die variations on the chip's maximum critical path delay distribution is estimated by combining these two delay distributions. The maximum critical path delay density function resulting from inter-die and intra-die variations is then calculated as the convolution of $f_{T_{norm}}$, $f_{inter-dmax}$ and $f_{intra-dmax}$ [12]:

$$f_{chip} = f_{T_{norm}} * f_{inter-dmax} * f_{intra-dmax} \quad (1)$$

$f_{inter-dmax} = N(0, \sigma D_{D2D})$ is obtained by shifting the original distribution, f_{inter} . $f_{T_{norm}} = \delta(t - T_{norm})$ is an impulse at T_{norm} . The chip's intra-die maximum critical path delay density function is $f_{intra-dmax} = N_{cp} f_{intra} \times (F_{intra})^{N_{cp}-1}$, where F_{intra} is the chip's intra-die cumulative delay distribution and N_{cp} is the number of critical paths present in the design under evaluation.

B. Problem Formulation

In task scheduling, the application is represented as a directed acyclic precedence graph $G=(V,E)$, as shown in Fig. 2. The vertex, $v_i \in V$, in the task graph represents the computational module, i.e. task. The arc, $e(i,j) = (v_i, v_j) \in E$, represents both the precedence constraints and the communications between task v_i and task v_j . The weight $w(e(i,j))$ associated with the arc $e(i,j)$ in a task graph represents the amount of data that passes from task v_i to v_j .

As previously stated, the target MPSoC architecture in this work contains heterogenous PEs, connected by a shared bus. However, the methodology and algorithm presented in this paper can also be used with alternative interconnection fabrics, such as rings, crossbars, or on-chip routers with appropriate modifications to reflect the communication overhead of the interconnection protocol.

To account for process variation in task allocation and scheduling,

- 1) An execution time distribution table captures the effects of process variability by associating each task node, t_i , in the task graph with execution time distributions corresponding to each PE in the system; i.e. element $delay[i][j]$ in the table stores the execution time distribution of task t_i if it is executed on the j th PE in the architecture.
- 2) In addition, a new metric called *performance yield* is introduced to evaluate process scheduling. The performance yield is defined as the probability of the assigned schedule meeting the deadline constraint:

$$Yield = P(completion_time_of_the_schedule \leq deadline) \quad (2)$$

Thus the variation-aware task graph scheduling process is formulated as: *Given a directed acyclic task graph for an application*

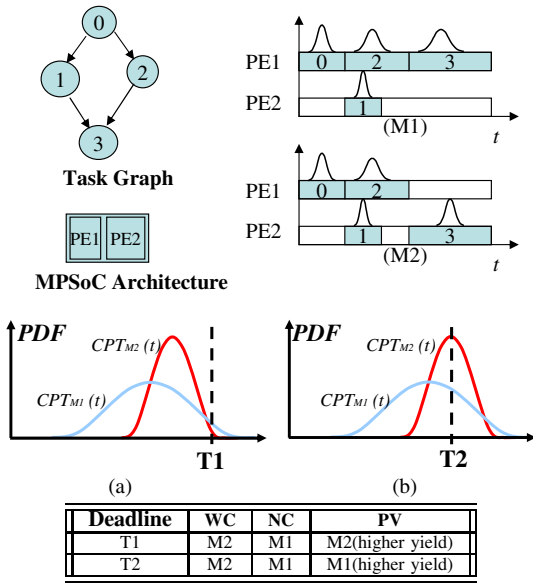


Fig. 2. An example of process-variation-aware task scheduling for an MPSoC architecture. The MPSoC architecture contains two PEs, $PE1$ and $PE2$. $PE1$ has larger delay variation than $PE2$ due to intra-die variation. $M1$ and $M2$ are two schedules for the possible placement of *Task 3*. In schedule $M1$, *task 3* is scheduled onto $PE1$, which causes a large completion time variation. In schedule $M2$, *task 3* is scheduled onto $PE2$, which results in smaller completion time variation. In (a) and (b), the distributions of the completion times of the two different task schedules $M1$ and $M2$, denoted as $CPT_{M1}(t)$ and $CPT_{M2}(t)$, are shown here as Probability Distribution Functions (PDF).

running on an MPSoC architecture that contains heterogeneous PEs, find a feasible mapping of the tasks to the PEs and determine a schedule which maximizes the performance yield of the mapping under predefined performance constraints.

Fig. 2 shows an example of mapping a task graph to a two-PE MPSoC platform. The example illustrates the difference between Process-Variation-aware (PV) task scheduling and conventional deterministic task scheduling based on Worst-Case (WC) or Nominal-Case (NC) delay models. The MPSoC platform contains two PEs, $PE1$ and $PE2$. $PE1$ has larger delay variation than $PE2$ due to intra-die variation. $M1$ and $M2$ are two schedules for possible placement of *Task 3*. In schedule $M1$, *task 3* is scheduled onto $PE1$, which causes large completion time variation. In schedule $M2$, *task 3* is scheduled onto $PE2$, which results in smaller completion time variation. The distributions of the Completion Time (CPT) of task schedules, $M1$ and $M2$, are denoted as $CPT_{M1}(t)$ and $CPT_{M2}(t)$, respectively. Given that the deadline of schedule M is T , with a completion time distribution of $CPT_M(t)$, the performance yield of schedule M , can be computed as

$$Yield_M(T) = \int_0^T CPT_M(t) dt \quad (3)$$

It can be observed that deterministic scheduling techniques lead to inferior scheduling decisions. When the deadline of the task is set to $T1$ as in Fig. 2(a), *task 3* should be scheduled onto $PE2$ to achieve higher yield, i.e., $Yield_{M1}(T1) < Yield_{M2}(T1)$. However, deterministic scheduling based on nominal case delay models would choose $PE1$ because the nominal case completion time of schedule $M1$ is less than that of schedule $M2$. Meanwhile, when the deadline of the task is set to $T2$, as in (b), *task 3* should be scheduled onto $PE1$ to obtain larger yield, i.e., $Yield_{M1}(T2) > Yield_{M2}(T2)$. However, deterministic scheduling based on worst case delay models would choose $PE2$ because the worst case completion time of schedule $M2$

is less than that of schedule $M1$. Thus, it is critical to adopt process-variation-aware scheduling, which takes into account the distribution of the execution time and not merely a nominal or worst case value.

C. Complementing Existing Probabilistic Real-Time Embedded System Research

The real-time community has recognized that the execution time of a task can vary, and proposed probabilistic analysis for real-time embedded systems [13]–[15], where the probability that the system meets its timing constraints is referred to as *feasibility probability* [14]. However, these statistical analyses are for execution time variations caused by software factors (such as data dependency and branch conditions), and hardware variations were not modeled. The actual meanings of *feasibility probability* and *performance yield* are quite different: for example, *feasibility probability* = 95% means that the application can meet the real-time constraints during 95% of the running time; *performance yield* = 95% means that 95% of the fabricated chips can meet the real-time constraints with 100% guarantee, while the other 5% of the chips may not meet the real-time constraints. Therefore, *feasibility probability* is a *time domain* metric and is suitable for “soft real-time” systems, while *performance yield* is a *physical domain* metric (i.e., percentage of fabricated chips) and it can guarantee that good chips meet hard deadlines.

In summary, the scheme proposed in this paper provides a complementary perspective to existing probabilistic real-time embedded system analysis by taking into account the underlying hardware variations during the task allocation and scheduling processes.

IV. STATISTICAL TASK GRAPH ANALYSIS

In statistical timing analysis for task graphs, the timing quantity is computed by using two atomic functions *sum* and *max*. Assume that there are three timing quantities, A , B , and C , which are random variables. The *sum* operation $C = sum(A, B)$ and the *max* operation $C = max(A, B)$ will be developed as follows:

- 1) The *sum* operation is easy to perform. For example, if A and B both follow Gaussian distributions, the distribution of $C = sum(A, B)$ would also follow a Gaussian distribution with a mean of $\mu_A + \mu_B$ and a variance of $\sqrt{\sigma_a^2 + \sigma_b^2 - 2\rho\sigma_a\sigma_b}$; ρ is the correlation coefficient.
- 2) The *max* operation is quite complex. Tightness probability [16] and moment matching techniques could be used to determine the corresponding sensitivities to the process parameters. Given two random variables, A and B , the tightness probability of random variable A is defined as the probability of A being larger than B . An analytical equation presented in [16] is used to compute the tightness probability, thus facilitating the calculation of the *max* operation.

With the atomic operations defined, the timing analysis for the resulting task graph can be conducted using PERT-like traversal [8].

V. PROCESS-VARIATION-AWARE TASK SCHEDULING

In this section, we first introduce the new metric of *performance yield* in the dynamic priority computation of task scheduling. We then present a yield computation method for task graphs. Finally, based on the new dynamic priority computation scheme, a new statistical scheduling algorithm is presented.

A. Process-Variation-Aware Dynamic Priority

In a traditional dynamic list scheduling approach, the ready tasks – for which the precedent tasks have already been scheduled – are first formed; the priorities of these ready tasks are then computed. Finally,

the task node with the highest priority for scheduling is scheduled. The above steps are repeatedly executed until all the tasks in the graph are scheduled. In this approach, the priority of the task is recomputed dynamically at each scheduling step, thus we call the priority of the task as *dynamic priority*. In previous work in the literature, the dynamic priority of the ready task is computed based on deterministic timing information from the PEs. We refer to this technique as *deterministic task scheduling (DTS)*.

However, under large process variations, the delay variations for a PE have to be taken into account in the dynamic priority computation, leading to *statistical task scheduling (STS)*. To compute the dynamic priority in statistical task scheduling, we introduce a new metric, called conditional *performance yield* for a scheduling decision. The conditional performance yield for a task PE pair, $Yield(T_i, P_j)$, is defined as the probability of the task schedule meeting the predefined performance constraints. It is denoted as $Probability(D_{TaskGraph} < deadline | (T_i, P_j))$, where $D_{TaskGraph}$ is the completion time of the entire task graph under the condition that task T_i is scheduled onto PE P_j . The yield metric is an effective metric in guiding the task scheduling, as shown in the example in Fig. 2.

Based on the aforementioned performance yield definition, the process-variation-aware Dynamic Priority (DP) for task T_i can be computed as

$$PVaware_DP(T_i) = Yield(T_i) + \Delta Yield(T_i) \quad (4)$$

where we assume that the largest yield is obtained when task T_i is scheduled to the j th PE. P_j , $Yield(T_i)$ is computed as:

$$Yield(T_i) = Yield(T_i, P_j) \quad (5)$$

Similar to deterministic scheduling [3], we define $\Delta Yield(T_i)$ as the difference between the highest yield and the second highest yield, which stands for the yield loss if task T_i is not scheduled onto its preferred PEs. We have

$$\Delta Yield(T_i) = Yield(T_i) - 2nd_Yield(T_i) \quad (6)$$

where $Yield(T_i) = Yield(T_i, P_j)$, and $2nd_Yield(T_i) = \max\{Yield(T_i, P_k)\}, \forall k \in [1, N] \text{ and } k \neq j$.

In finding the best PE for a specific task, T_i , instead of using the relative complex yield metric of the entire task graph, we compare two delay distributions of the path directly to speed up the scheduling process. We define $path_yield(P_j)$ as $Probability(delay_P_j < deadline | (T_i, P_j))$, where $delay_P_j$ is computed as the longest path passing through task T_i .

B. Yield Computation for Partially Scheduled Task Graphs

To compute the performance yield at each scheduling step, we first estimate the delay of the path in the partially scheduled task graph, in which a part of the task graph has not been scheduled. During the scheduling, some tasks have not been mapped to PEs, and paths in both the *scheduled* and *unscheduled* sub-task graphs contribute to the delay of the entire task graph. For example, in Fig. 3, the path going through *Task 4* consists of two paths, $p1$ and $p2$, where $p1$ is the path from the start task node, *Task 0*, to *Task 4*, and $p2$ is the path from *Task 4* to the end task node, *Task 8*. The delay of that path can be computed as $delay(path_T_4) = delay(p1) + delay(p2)$. $p2$ is considered as a path in the *unscheduled* sub-task graph. Consequently, to compute the path delay, the timing quantities of the tasks in the *unscheduled* sub-task graph need to be determined. In this work, we develop two methods to approximate these timing quantities (delay distributions):

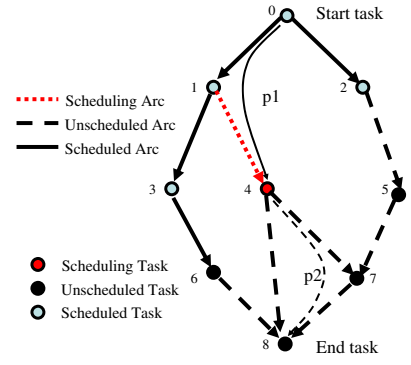


Fig. 3. During scheduling, some tasks have not been mapped to PEs, and paths in both the *scheduled* and *unscheduled* sub-task graphs contribute to the delay of the entire task graph. The path going through *Task 4* consists of two paths, $p1$ and $p2$, where $p1$ is the path from the start task node, *Task 0*, to *Task 4*, and $p2$ is the path from *Task 4* to the end task node, *Task 8*.

- 1) For each task, the *adjust mean* and *adjust variance* are used in computing the delay of the *unscheduled* task graph. We define an *adjust mean* and an *adjust variance* of the execution time, denoted $AdjMean(T_i)$ and $AdjSigma(T_i)$, respectively, for *unscheduled* task T_i as:

$$AdjMean(T_i) = \frac{\sum_{j=1}^N E(Delay(T_i, P_j))}{N} \quad (7)$$

$$AdjSigma(T_i) = \frac{\sqrt{\sum_{j=1}^N \sigma(Delay(T_i, P_j))^2}}{N} \quad (8)$$

where N is the total number of PEs in the MPSoC platform and $Delay(T_i, P_j)$ is the execution time distribution of task T_i over PE P_j .

- 2) We first perform an initial scheduling via deterministic scheduling or variation-aware scheduling based on timing information obtained through method 1). We then evaluate the timing quantities for the tasks in the *unscheduled* sub-task graph based on these initial scheduling results.

Thus, with the timing quantities of the tasks nodes and edges in the *unscheduled* task graph determined, we compute the Critical Path Delay, CPD , from a particular task node to the end task node in the task graph. We perform statistical timing analysis to compute the CPD for each task node in a task graph through a single PERT-like graph traversal. After the path delay has been determined, the

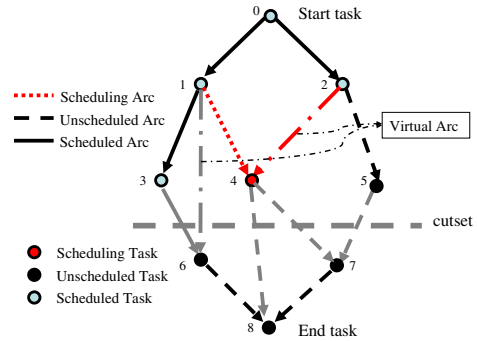


Fig. 4. Yield computation for a partially scheduled task graph to address *structural dependency*. The *virtual arc* from task 1 to task 6 indicates that task 1 and task 6 are scheduled onto the same PE and there is no data transferring on the arc (1,6). Virtual arcs have to be included in the cut-set, since they may lie on the critical path.

completion time of the partially scheduled task graph needs to be computed. Based on the same concept of cut-set as in [11], we develop a yield computation method for a partially scheduled task graph. From graph theory, all the paths from the source node to the sink node have to pass through any cut-set which separates the source and sink nodes in a directed acyclic graph. Consequently, the delay distribution of the entire task graph can be obtained by the *Max* operation over the delays of all the paths going through the edges in that cut-set.

However, the task graphs have to be modified to include *structural dependency*. We borrow the term *structural dependency* from pipeline design and define it in our context as two tasks being scheduled onto the same PE. As shown in Fig. 4, let us assume that task 6 and task 1 are scheduled onto the same PE. Although there is no data transfer from task 1 to task 6, *virtual arc* (1,6) is added to the task graph. Despite the fact that structural dependencies have been taken into account in the preceding timing computation, *one still has to include these virtual arcs in the cut-set extraction, as illustrated in Fig. 4, because these arcs may, in fact, lie on the critical path*. Given the longest path delay through each task node in the cut-set, the performance of the entire task graph can be computed as

$$D_{TaskGraph} = \text{Max}\{\text{path_delay}(T_i)\}, \forall T_i \in \text{cutset} \quad (9)$$

where $\text{path_delay}(T_i)$ is the delay of the longest path from the start task node to the end task node passing through task T_i in the cut-set. At each step of task scheduling, the only changes to the timing quantities in a partially scheduled task graph are those of the task node being scheduled and its incoming arcs, as shown in Fig. 4. The approximate timing quantities of the task being scheduled and its incoming arcs are replaced with the values computed using the delay information of the PE onto which the particular task is being scheduled. Assume task T_i is being scheduled onto PE P_j . The delay of the longest path from the start task node to task node h , $AVT(T_i, P_j)$, can be computed as $\text{max}[\text{data_available}(T_i, P_j), PE_available(P_j)] + \text{execution_time}(T_i, P_j)$. $\text{data_available}(T_i, P_j)$ represents the time the last of the required data becomes available from one of its parent nodes. $PE_available(P_j)$ represents the time that the last node assigned to the j th PE finishes execution. Thus the longest path delay going through task T_i , $\text{path_delay}(T_i)$, can be computed as

$$\text{path_delay}(T_i) = AVT(T_i, P_j) + CPD(T_i) \quad (10)$$

C. Scheduling Algorithm

```

PVschedule (Task Graph, Platform){
1. Perform initial scheduling for Task Graph
2. Perform statistical timing analysis and initialize the RTS;
3. While (RTS is not empty){
4.   Select N most critical tasks and form CTS;
5.   For each task  $T_i$  in the CTS{
6.     Tentatively schedule  $T_i$  to PEs to obtain the best and
       2nd best  $(T_i, P_j)$  pair;
7.   }
8.   Select the  $(T_i, P_j)$  pair with maximum DP and Schedule  $T_i$  onto  $P_j$ ;
9.   Add new ready tasks to RTS;
10.  }
11. Compute the yield;}

```

Fig. 5. The Pseudo Code of the proposed Variation-Aware Task Scheduling Process

With the notion of variation-aware dynamic priority defined, the corresponding scheduling algorithm designed to mitigate the effects of process variation and improve the yield is shown in Fig. 5. Our scheduling algorithm takes the *Task Graph* and the *specification* of the MPSoC platform as inputs, and outputs the mapping of tasks

to PEs. It also computes the performance yield of that mapping. The scheduling algorithm is described in detail below:

- 1) In the initial scheduling of the task graph, the tasks are scheduled with either a deterministic scheduling method or a variation-aware method with the assumption that the distribution of the delay of each task is determined by the *adjust mean* and the *adjust variance* over the PEs in the MPSoC architecture (Line 1).
- 2) Perform statistical timing analysis for the initial scheduled task graph and initialize the Ready Tasks Set (RTS) (Line 2).
- 3) Generate the RTS; that is, the tasks for which the precedent tasks have already been scheduled (Line 3). After the task is scheduled, the new ready tasks are added to the RTS (Line 9).
- 4) We choose N top-most critical tasks in the RTS as the Critical Task Set (CTS) and give higher priority to those tasks that have larger impact on the performance yield of the schedule (Line 4). The critical task is the task that has higher probability of being on the critical path. The CTS scheme enables us to focus on the most critical tasks. Thus, the computation cost can be greatly reduced, especially for large task graphs.
- 5) During tentative scheduling, task T_i is scheduled onto all the possible PEs in the MPSoC platform (Line 6). Two feasible mappings with the highest and second highest yield values are then selected. Finally, after all the tasks in the CTS finish tentative schedule, the dynamic priority (DP) for each task is computed and the task-PE pair (T_i, P_j) with the maximum DP value is selected. Task T_i is then scheduled onto PE P_j (Line 8).

VI. EVALUATION RESULTS

In this section, we present our evaluation platform and analyze the simulation results. It will be shown that the proposed method can effectively reduce the impact of manufacturing process variability and maximize performance yield, significantly outperforming traditional deterministic task schedulers that use worst-case or nominal-case delay models.

Our variation-aware scheduling algorithm was implemented in C++ and experiments were conducted using various benchmarks, including an MPEG2 benchmark [17], three embedded system synthesis benchmarks from the E3S suites [18] (these are based on data from the Embedded Microprocessor Benchmark Consortium (EEMBC)), and three benchmarks (BM1-BM3) from [19]. These benchmarks are allocated and scheduled onto an MPSoC platform with 2-4 heterogeneous PEs, each of which has a Gaussian distribution of clock frequency to reflect the effects of process variation. Two sets of experiments are performed to demonstrate the effectiveness of our Statistical Task Scheduling (STS) algorithm. The first set of results is related to the yield improvement of our variation-aware task scheduling process. The second set of experiments evaluates our algorithm within the context of cost saving.

To demonstrate the yield improvement of the proposed method, the simulation results are compared against those of traditional Deterministic Task Scheduling (DTS) procedures using Nominal-Case (NC) and Worst-Case (WC) delay models. In phase one of the evaluation process, deterministic and statistical scheduling for the task graphs are performed. Phase two computes the yield after the timing analysis of the scheduled task graph, by using equation (2). Table I shows the results of the proposed statistical method against those of deterministic scheduling techniques. The first column shows the benchmarks employed in this analysis. From the second column to the fourth column, we show the absolute yield results of

the Process-Variation-aware (PV) scheduler, the deterministic Worst-Case (WC) scheduler, and the deterministic Nominal-Case (NC) scheduler, respectively. In the fifth and sixth columns, we show the yield improvement of our statistical method over worst-case (PV-WC) and nominal-case (PV-NC) methods. As illustrated in Table I, significant yield improvement can be obtained through variation-aware task scheduling. Although DTS can result in high yield values for a few benchmarks, it is not able to guarantee the yield across all benchmarks. The yield results show that WC-based techniques are grossly pessimistic with an average 45% yield loss. Similarly, NC-based techniques result in a 34% yield loss on average.

TABLE I
YIELD IMPROVEMENT OVER A DETERMINISTIC SCHEDULER

Benchmark	PV	WC	NC	PV-WC	PV-NC
MPEG2	95%	94 %	87 %	1 %	8 %
Consumer-asic	100%	34 %	51 %	66 %	49 %
Telecom	100%	100 %	60 %	0 %	40 %
Auto-indust	100%	90 %	61 %	10 %	39 %
BM1	99%	50 %	0 %	49 %	99 %
BM2	93%	0 %	89 %	93 %	4 %
BM3	99%	0 %	99 %	99 %	0 %
Average	98%	53 %	64%	45 %	34%

We define *cost* as the amount of additional performance required by the MPSoC architecture to meet a target yield of 99% when using a deterministic scheduler (DTS), as compared to using the variation-aware scheduler (STS): $cost = \frac{required_frequency(DTS)}{required_frequency(STS)} - 1$. This cost metric is an effective indicator of actual manufacturing cost, as higher performance requirements translate to an increase in design and manufacturing effort. Furthermore, a higher frequency would increase power density and result in even larger variations, larger voltage drops and elevated thermal issues. Table II shows the cost saving results based on this cost definition. In the second and third columns, we show the percentage of additional speed required by the MPSoC architecture when using a deterministic WC scheduler and a deterministic NC scheduler, respectively. As can be seen from the table, an average of 10% and 8% cost can be saved over these two deterministic approaches by using the proposed process-variation-aware scheduler.

TABLE II
COST SAVING OVER A DETERMINISTIC SCHEDULER

Benchmark	WC	NC
MPEG2	3%	1%
Consumer-asic	15%	15%
Telecom	0%	7%
Auto-indust	3%	5%
BM1	11%	30 %
BM2	20%	1%
BM3	21%	0 %
Average	10 %	8 %

VII. CONCLUSIONS AND FUTURE WORK

Increasing impact of process variations due to technology scaling has led to a marked shift from deterministic to statistical design methodologies across all levels of the design hierarchy. In this paper, we formulate a variation-aware scheduling process for heterogeneous multi-core MPSoC architectures and propose a statistical scheduling algorithm to mitigate the effects of parameter variations. A new metric, known as *performance yield*, is used at each step of the iterative dynamic priority computation of the task scheduler. Further, an efficient yield computation method for task scheduling and a fast criticality-based routing algorithm have been proposed to improve

the performance of the scheduling algorithm. Simulation results show that significant yield gains can be obtained with our variation-aware scheduling methodology. More specifically, performance yield improvements of 45% and 34% over worst-case and nominal-case deterministic schedulers, respectively, are reported.

VIII. ACKNOWLEDGEMENT

This work is partially supported by NSF CAREER 0643902, NSF CCF 0702617, NSF CNS 0720659, and a grant from DARPA/MARCO GSRC.

REFERENCES

- [1] Shekhar Borkar. Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *IEEE Micro*, 25(6):10–16, 2005.
- [2] A. Jerraya and W. Wolf. *Multiprocessor systems-on-chips*. Morgan Kaufmann Publishers, 2005.
- [3] G.C. Sih and E.A. Lee. A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. *IEEE Transactions on Parallel and Distributed Systems*, 04(2):175–187, 1993.
- [4] W.H. Wolf. An architectural co-synthesis algorithm for distributed, embedded computing systems. *IEEE Trans. Very Large Scale Integr. Syst.*, 5(2):218–229, 1997.
- [5] Y. Zhang, X. Hu, and D. Z. Chen. Task scheduling and voltage selection for energy minimization. In *Design Automation Conference*, pages 183–188, 2002.
- [6] L. Jiong and N. Jha. Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems. In *7th Asia and South Pacific Design Automation Conference*, pages 719–726, 2002.
- [7] J. Hu and R. Marculescu. Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints. *Design, Automation, and Test in Europe Conference*, pages 234–239, 2004.
- [8] C. Hongliang and S. S. Sapatnekar. Statistical timing analysis considering spatial correlations using a single pert-like traversal. In *International Conference on Computer Aided Design*, pages 621–625, 2003.
- [9] A. Agarwal, D. Blaauw, and V. Zolotov. Statistical timing analysis for intra-die process variations with spatial correlations. In *International Conference on Computer Aided Design*, pages 900–907, 2003.
- [10] D. Sinha, N. V. Shenoy, and H. Zhou. Statistical gate sizing for timing yield optimization. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 1037–1041, 2005.
- [11] K. Chopra, S. Shah, A. Srivastava, D. Blaauw, and D. Sylvester. Parametric yield maximization using gate sizing based on efficient statistical power and delay gradient computation. *International Conference on Computer-Aided Design*, pages 1023–1028, November 2005.
- [12] K. A. Bowman, S. G. Duvall, and J. D. Meindl. Impact of Die-to-Die and Within Die Parameter Fluctuations on the Maximum Clock Frequency Distribution for Gigascale Integration. *Journal of Solid-State Circuits*, pages 183–190, February 2002.
- [13] T. S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L. C. Wu, and J. W. S. Liu. Probabilistic performance guarantee for real-time tasks with varying computation times. In *Real-Time Technology and Applications Symposium*, pages 164–173, 1995.
- [14] X. S. Hu, Z. Tao, and E. H. M. Sha. Estimating probabilistic timing performance for real-time embedded systems. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 9(6):833–844, 2001. 1063-8210.
- [15] G. D. Veciana, M. F. Jacome, and J.-H. Guo. Hierarchical algorithms for assessing probabilistic constraints on system performance. In *IEEE/ACM Design Automation Conference (DAC)*, pages 251–256, 1998.
- [16] C. Clark. The greatest of a finite set of random variables. *Operations Research*, pages 145–162, 1961.
- [17] Y. B. Li and J. Henkel. A framework for estimation and minimizing energy dissipation of embedded hw/sw systems. *Design Automation Conference (DAC)*, pages 188–193, 1998.
- [18] R. Dick. Embedded systems synthesis benchmarks suite (e3s). <http://www.ece.northwestern.edu/dickrp/e3s/>.
- [19] Y. Xie and W. Hung. Temperature-aware task allocation and scheduling for embedded systems. *Journal of VLSI Signal Processing*, 45(3): pages 177-189, 2006.