Process-Variation-Aware Mapping of Best-Effort and Real-Time Streaming Applications to MPSoCs

DAVIT MIRZOYAN, Delft University of Technology
BENNY AKESSON and KEES GOOSSENS, Eindhoven University of Technology

As technology scales, the impact of process variation on the maximum supported frequency (FMAX) of individual cores in a multiprocessor system-on-chip (MPSoC) becomes more pronounced. Task allocation without variation-aware performance analysis can greatly compromise performance and lead to a significant loss in *yield*, defined as the percentage of manufactured chips satisfying the *application timing requirement*. We propose variation-aware task allocation for best-effort and real-time streaming applications modeled as task graphs. Our solutions are primarily based on the throughput requirement, which is the most important timing requirement in many real-time streaming applications.

The four main contributions of this work are (1) distinguishing best-effort firm real-time and soft real-time application classes, which require different optimization criteria, (2) using dataflow graphs, which are well suited for modeling and analysis of streaming applications, we explicitly model task execution both in terms of clock cycles (which is independent of variation) and seconds (which does depend on the variation of the resource), which we connect by an explicit binding, (3) we present two optimization approaches, which give different improvement results at different costs, (4) we present both exhaustive and heuristic algorithms that implement the optimization approaches. Our variation-aware mapping algorithms are tested on models of seven real applications and are compared to mapping methods that are unaware of hardware variation. Our results demonstrate (1) improvements in the average performance (3% on average) for best-effort applications, and (2) for firm real-time and soft real-time applications, yield improvements of up to 27% with an average of 15%, showing the effectiveness of our approaches.

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

General Terms: Algorithms, Performance, Design

Additional Key Words and Phrases: Process variation, multiprocessor system-on-chip, synchronous dataflow graphs, application mapping

ACM Reference Format:
Davit Mirzoyan, Benny Akesson, and Kees Goossens. 2014. Process-variation-aware mapping of best-effort and real-time streaming applications to MPSoCs. ACM Trans. Embedd. Comput. Syst. 13, 2s, Article 61 (January 2014), 24 pages.
DOI: http://dx.doi.org/10.1145/2490819

1. INTRODUCTION

Aggressive technology scaling has enabled the integration of multiple processors and hardware accelerators on a single silicon chip die, known as a multiprocessor system-on-chip (MPSoC). The use of such systems is increasingly popular, as they have high computational power and low power consumption, which are the main requirements for many embedded systems. However, scaling the minimum feature sizes in

This work was partially funded by projects EU FP7 288008 T-CREST and 288248 Flextiles, Catrene CA104 Cobra, and NL STW 10346 NEST.
Corresponding author's address: D. Mirzoyan, Delft University of Technology; email: d.mirzoyan@tudelft.nl.

DOI: http://dx.doi.org/10.1145/2490819

deep-submicron technologies has also brought variations in key transistor parameters, such as channel length and device and interconnect width. This phenomenon, known as process variation [Unsal et al. 2006], significantly impacts the maximum supported frequency of individual cores in an MPSoC [Bowman et al. 2002; Eisele et al. 1997]. It is shown [Miranda et al. 2009] that the variation in the longest path delay (the inverse of FMAX) of a very long instruction word (VLIW) processor, manufactured at 32nm technology, is up to 40%. Moreover, the impact of within-die variation on the system parameters is increasing as the technology scales, making it a limiting factor in efficient MPSoC design [Bowman et al. 2002].

Binding application tasks to resources in an MPSoC without considering the impact of process variation can greatly compromise the average performance and lead to a significant *yield* loss. In this work, the yield metric is not the hardware manufacturing yield, which determines the number of chips that meet the predefined frequency requirements. In our view, yield is defined at the system level and shows the percentage of manufactured chips that satisfy the application timing requirement. Existing solutions that propose variation-aware yield-driven task allocation and scheduling [Wang et al. 2007; Chon and Kim 2009; Singhal and Bozorgzadeh 2008; Huang and Xu 2010] use acyclic task graphs for application modeling and are based on latency requirements. Acyclic task graphs are not able to capture cyclic data dependencies or the streaming behavior (i.e., iterative and overlapping execution) of real-time streaming applications [Stuijk et al. 2007]. In this work, we allow arbitrary task graphs that may include cyclic data dependencies. Our solutions are primarily based on the throughput requirement, which is the most important timing requirement in many real-time streaming applications. However, our approaches can be extended to cover latency requirements [Moreira and Bekooij 2007].

The four main contributions of this work are (1) as an extension of Mirzoyan et al. [2012], we differentiate best-effort, firm real-time and soft real-time application classes which require different optimization criteria. For best-effort and firm real-time applications, we maximize the average throughput over all manufactured chips and the yield, respectively. The objective for soft real-time applications is to improve the yield and/or reduce the average throughput degradation, as the chips with low degradations can be used in this class of applications. (2) We base our solutions on synchronous dataflow graphs (SDFG), which are well suited for modeling and analysis of streaming applications and have multiple efficient techniques for throughput computation [Stuijk et al. 2007]. The novelty of our SDFG formulation lies in the *explicit modeling* of software execution in terms of clock cycles (which is independent of the variation in the hardware resource), and in terms of seconds (which does depend on the variation in the resource), which are linked by an explicit binding. (3) We present two optimization approaches, *single binding* and *multiple bindings*. With the single-binding optimization approach, the objective is to find a binding at design time that results in an optimized target function for an application class (e.g., yield for firm real-time applications). With the multiple-bindings optimization approach, a set of bindings is found and stored at design time, and based on the variation in each manufactured chip, the binding that satisfies the application throughput requirement or maximizes the throughput is selected at the runtime configuration stage. (4) We present both exhaustive and heuristic algorithms that implement the optimization approaches. The exhaustive algorithms provide optimum results and can be applied to problems of a small to medium size. The heuristic algorithms provide results close to optimum and are scalable to problems of a larger size. Our variation-aware mapping algorithms are tested on models of seven real applications and are compared to mapping methods that are unaware of the variation in the hardware resources. Our results report (1) improvements in the average throughput (3% on average) for best-effort applications, (2) for firm real-time and soft real-time applications, yield improvements of up to 27% with an average of

15%, (3) reductions of up to 2% in the average throughput degradation for soft real-time applications, showing the effectiveness of our approaches.

The rest of the article is organized as follows: Section 2 presents related work in the field. Section 3 introduces formal models of a hardware platform, an application SDFG, and a binding. In Section 4, we present the single-binding and multiple-bindings optimization approaches for best-effort, firm real-time and soft real-time streaming applications. Section 5 illustrates the variation-aware exhaustive and heuristic algorithms that implement the optimization approaches. Section 6 experimentally evaluates our methods and Section 7 concludes.

2. RELATED WORK

Several techniques have been proposed to minimize process variation at the circuit and microarchitectural levels at different costs [Unsal et al. 2006; Tschanz et al. 2002]. There has been extensive research in the area of task allocation and scheduling for MPSoC [Braun et al. 2001; Stuijk et al. 2007; Bonfietti et al. 2009, 2010; Wang et al. 2007; Chon and Kim 2009; Singhal and Bozorgzadeh 2008; Huang and Xu 2010]. Some research [Stuijk et al. 2007; Bonfietti et al. 2010, 2009] proposed methods to map throughput-constrained applications modeled as SDFGs to resources in an MPSoC. However, none of them consider the impact of process variation. With variation-unaware mapping approaches, the impact of process variation cannot be reflected by having different resources, with different frequencies, as the availability of a resource with a specific frequency is a matter of probability.

Wang et al. [2007] introduced a new design metric called *performance yield*, defined as the probability of an assigned schedule meeting a predefined performance constraint. They proposed a variation-aware scheduling algorithm that allocates and schedules tasks with latency requirements modeled as an acyclic task graph to MPSoC such that the performance yield is maximized. Resource sharing in task allocation and scheduling under process variation has been studied by Chon and Kim [2009]. They proposed an effective statistical static timing analysis technique which schedules and binds tasks in an acyclic task graph to the resources in an MPSoC in the presence of resource sharing such that the performance yield is maximized. Singhal and Bozorgzadeh [2008] introduced the problem of stochastically optimal task allocation, which tries to minimize the overall execution time of tasks in sequence and in parallel under process variation. Huang and Xu [2010] took into account the spatial correlation characteristics of systematic within-die variation and presented a scheduling algorithm that schedules tasks with latency constraints in an acyclic task graph such that the performance yield is maximized. With their solution, a set of schedules is synthesized offline and based on the variation in each chip, a runtime scheduler selects the right one such that the latency constraint is satisfied whenever possible.

All these solutions that account for process variation use acyclic task graphs for application modeling and are based on latency requirements. Acyclic task graphs are not able to capture the iterative and overlapping execution of real-time streaming applications, which are primarily constrained by throughput requirements. Several real-life streaming applications, such as our H.263 Encoder, MP3 Playback, and Modem benchmark applications presented in Section 6, include cyclic data dependencies. For these applications, none of the preceding solutions work, as they cannot capture the cyclic data dependencies in these applications. In contrast, we allow arbitrary task graphs that may include cyclic data dependencies. Our solutions are primarily based on throughput requirements but could be extended to cover latency requirements [Moreira and Bekooij 2007]. We additionally distinguish best-effort, firm real-time and soft real-time application classes, which require different optimization criteria. To the best of our knowledge, this is the first work that addresses the problem of variation-aware task allocation for cyclic task graphs.

3. FORMAL MODELS

This section formally defines a hardware multiprocessor platform as a set of resources. We introduce a set of operating points (maximum supported frequencies) for each resource to reflect the impact of process variation. We define an SDFG model of an application, named an *unbound graph*, where the actors (i.e., tasks of an application) are characterized by execution times in clock cycles. The unbound graph is unaware of the binding of application actors to the hardware resources and is hence decoupled from hardware variation. Later, we introduce an explicit binding of actors to the resources in a platform and define an SDFG model of an application, named a *bound graph*, where application actors are characterized by execution times in seconds. The bound graph is no longer decoupled from hardware variation and enables us to analyze the impact of variation on the application performance for different actor to resource bindings.

The presented techniques are general and apply to any system that implements the models in this section. Examples of such systems are CoMPSoC [Hansson et al. 2009] and CA-MPSoC [Shabbir et al. 2010]

3.1. Model of a Hardware Platform

We refer to a hardware multiprocessor platform as a set of resources connected to each other by an interconnection network. For simplicity, we assume a zero-latency network. However, non-zero-latency networks can be dealt with by modeling the delay for sending data over the connections in the application SDFG, as shown in Stuijk et al. [2007]. We denote the set of resources as R. Each resource is a generic processing element, such as a processor, DSP, or a hardware accelerator. We assume a multiprocessor platform, where each resource is in a separate frequency domain and can be operated at any of its possible operating points. This assumption holds for globally asynchronous and locally synchronous (GALS) embedded designs.

Manufacturing process variation is classified into inter-die and intra-die variations [Eisele et al. 1997]. Inter-die variation, also referred to as global variation, acts globally on the entire chip, resulting in faster and slower chips on a wafer. Therefore, all the resources on the chip incur identical variation and can be equally faster or equally slower. Intra-die variation, also known as local variation, affects individual resources on the chip differently. Due to local variation, there can be faster and slower resources on a chip. Local variation is overlaid on the global variation. This can result in a relatively faster resource on a slow chip and vise versa. The probability density functions (PDF) of the maximum supported frequency of a resource, as a result of global and local variations, are illustrated in Figure 1. Both of these sources of variation are explicitly modeled in our framework. We characterize each resource by a *nominal operating point* (nominal maximum supported frequency), which is the target frequency specification of the resource, and what the manufacturing aims for (Definition 3.1). To reflect the impact of global variation, we introduce a set of *global operating points* (possible maximum supported frequencies due to the global variation) (Definition 3.2). These concepts are illustrated in Figure 1.

Definition 3.1 (*Nominal Operating Point*). The function $ON : R \rightarrow \mathbb{R}^+$ returns the nominal operating point of a resource $r \in R$.

Definition 3.2 (*Global Operating Points*). The function $OG : R \rightarrow \mathcal{P}(\mathbb{R}^+) \setminus \emptyset$ returns a non-empty set of all possible global operating points of a resource $r \in R$.

Each global operating point has an occurrence probability which is given in Definition 3.3. As global variation affects all the resources on the chip identically, any two resources have the same probability of being equally faster or slower. Note that the sum of the probabilities of all the global operating points of any resource is 1.

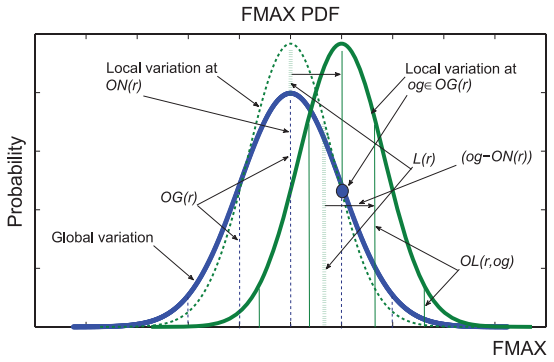


Fig. 1. Global and local operating points.

Definition 3.3 (*Probability of a Global Operating Point*). The function $PG : R \times \mathbb{R}^+ \rightarrow \mathbb{R}^+$ returns the occurrence probability of a global operating point $og \in OG(r)$ of a resource $r \in R$.

Local (intra-die) variation is modeled by introducing a set of *local operating points* for each resource. In Definition 3.4, we first define a set $L(r)$ of local operating points in respect to the nominal operating point $ON(r)$. The set $L(r)$ is graphically illustrated in Figure 1. Its elements are shown as discretized frequencies of the PDF of the local variation in respect to $ON(r)$ (green dashed distribution). Later, we obtain the local operating points $OL(r, og)$ with respect to a global operating point $og \in OG(r)$ (i.e., local variation overlaid on a global variation). We do this by adding an offset of $(og - ON(r))$ to each element of $L(r)$. Figure 1 shows the elements of $OL(r, og)$, which are discretized frequencies of the local distribution overlaid on a global operating point og (green solid distribution). The set $OL(r, og)$ is formally defined in Definition 3.5.

Definition 3.4 (*Local Operating Points with Respect to* $ON(r)$). The function $L: R \rightarrow \mathcal{P}(\mathbb{R}^+)\backslash\emptyset$ returns a non-empty set of all possible local operating points with respect to the nominal operating point $ON(r)$, for a resource $r \in R$.

Definition 3.5 (*Local Operating Points*). The function $OL : R \times \mathbb{R}^+ \rightarrow \mathcal{P}(\mathbb{R}^+)$ returns the set of all possible local operating points with respect to a global operating point $og \in OG(r)$, for a resource $r \in R$, where $OL(r, og)$ is given as

$$OL(r, og) = \begin{cases} \emptyset, & \text{if } og \notin OG(r), \\ \{ol \in \mathbb{R}^+ | \forall l \in L(r), \exists ol : ol = l + (og - ON(r))\}, & \text{otherwise.} \end{cases} \quad (1)$$

Each local operating point of a resource is associated with an occurrence probability. The probability of a local operating point with respect to the nominal operating point (i.e., $l \in L(r)$) is given by Definition 3.6. The probability of a local operating point with respect to a global operating point (i.e., $ol \in OL(r, og)$) is equal to the probability of $l \in L(r)$, where $l = ol - (og - ON(r))$ (Definition 3.7). Note that the sum of the probabilities of all local operating points of any resource is 1.

Definition 3.6 (*Probability of a Local Operating Point with Respect to* $ON(r)$). The function $P : R \times \mathbb{R}^+ \rightarrow \mathbb{R}^+$ returns the occurrence probability of a local operating point $l \in L(r)$ with respect to the nominal operating point $ON(r)$, for a resource $r \in R$.

Definition 3.7 (*Probability of a Local Operating Point*). The function $PL : R \times \mathbb{R}^+ \times \mathbb{R}^+ \rightarrow \mathbb{R}^+$ returns the occurrence probability of a local operating point $ol \in OL(r, og)$ with respect to a global operating point $og \in OG(r)$, for a resource $r \in R$, where

$PL(r, ol, og)$ is given as

$$PL(r, ol, og) = \begin{cases} 0, & \text{if } og \notin OG(r), \\ P(r, l) | l = ol - (og - \ ON\,(r)), & \text{otherwise.} \end{cases} \quad (2)$$

Given that each resource is characterized by a set of local operating points, there are multiple combinations of local operating points for the overall number of resources. We refer to an instance of local operating points for the overall number of resources as a *chip operating point*. The set of all possible chip operating points with respect to a global operating point $og \in OG(r)$ is obtained by the Cartesian product of the individual sets of local operating points of resources (Definition 3.8) and is an N-dimensional vector for N resources. The set of all possible chip operating points across all possible global operating points is derived from the union of the individual $C(og)$ sets for all $og \in OG(r)$. This set is defined in Definition 3.9. The probability that a set of resources has a certain chip operating point is given by Definition 3.10, where both probability components of local and global operating points are multiplied.

Definition 3.8 (*Chip Operating Points with Respect to a Global Operating Point*). The function $C : \mathbb{R}^+ \to \mathcal{P}(\mathbb{R}^+)$ returns the set of all possible chip operating points with respect to a global operating point $og \in OG(r)$, for a set R of resources, where $C(og)$ is given as

$$C(og) = \prod_{r \in R} OL(r, og). \quad (3)$$

Definition 3.9 (*Chip Operating Points*). The set OC of all possible chip operating points for a set R of resources is given as

$$OC = \bigcup_{og \in OG(r)} C(og). \quad (4)$$

Definition 3.10 (*Probability of a Chip Operating Point*). The function $PC : OC \to \mathbb{R}^+$ gives the occurrence probability of a chip operating point $oc \in OC$, where $PC(oc)$ is given as

$$PC(oc) = PG(r, og) \cdot \prod_{\substack{r \in R \\ ol \in oc}} PL(r, ol, og). \quad (5)$$

To illustrate the presented concepts, consider a platform comprising of two resources. Each resource r is given by a nominal operating point $ON(r)$ (in cycles/second), a set $OG(r)$ of two global operating points with associated probabilities $PG(r, og)$, and a set $L(r)$ of two local operating points with respect to $ON(r)$ with probabilities $P(r, l)$ (Table I). The local operating points $OL(r, og)$, with respect to each global operating point $og \in OG$, are derived from Definition 3.5 and are shown in Table II. The table also illustrates the occurrence probabilities $PL(r, ol, og)$ of the local operating points, which are obtained from Definition 3.7. The chip operating points and the occurrence probability of each chip operating point are derived from Definitions 3.8, 3.9, and 3.10 and are given in Table III.

3.2. Model of an Unbound Graph

We model best-effort and real-time streaming applications by means of synchronous dataflow graphs (SDFG). The motivation behind this choice is that an SDFG model provides a good compromise between expressiveness, modeling ease, analysis potential, and implementation efficiency. With an SDFG model, an application is captured by a directed graph, where the nodes (called actors) represent computations (tasks) that

Table I. A Platform Comprising of Two Resources: Each with Two Global and Two Local Operating Points

Resource	$ON(r)$	$OG(r)$	$PG(r, og)$	$L(r)$	$P(r, l)$
r_1	10	8	0.5	7	0.7
		12	0.5	10	0.3
r_2	10	8	0.5	7	0.7
		12	0.5	10	0.3

Table II. Local Operating Points and Their Occurrence Probabilities

Resource	$OL(r, 8)$	$OL(r, 12)$	$PL(r, ol, 8)$	$PL(r, ol, 12)$
r_1	5	9	0.7	0.7
	8	12	0.3	0.3
r_2	5	9	0.7	0.7
	8	12	0.3	0.3

Table III. Chip Operating Points and Their Occurrence Probabilities

$C(8)$	(5 5)		(5 8)		(8 5)		(8 8)	
$C(12)$	(9 9)		(9 12)		(12 9)		(12 12)	
C	(5 5)	(5 8)	(8 5)	(8 8)	(9 9)	(9 12)	(12 9)	(12 12)
$PC(oc)$	0.245	0.105	0.105	0.045	0.245	0.105	0.105	0.045

communicate with each other by sending streams of data elements over their edges. We denote the set of all actors as A, where each actor requires a number of clock cycles to finish its execution (Definition 3.11). This is similar to the constructs used in domain-specific languages for streaming applications, such as StreamIt [Amarasinghe et al. 2005]. Note that actors are nonblocking pieces of code. Therefore, execution times in cycles can be determined on a processor in isolation, and no complete mapping of the application to the hardware platform is necessary. For real-time applications, execution cycles can be derived using worst-case execution-time (WCET) estimation tools, such those in Wilhelm et al. [2008]. For best-effort applications, typical execution times can be obtained by simulations. Note that the number of clock cycles required for an actor's execution can be different for each resource if the platform is heterogeneous.

Definition 3.11 (*Execution Time in Cycles*). The function $EC : A \times R \rightarrow \mathbb{N}$ returns the number of cycles required to execute an actor $a \in A$ on a resource $r \in R$.

Definition 3.12 defines a model of an SDFG that is unaware of the binding of actors to resources. Each actor in the graph is characterized by a number of execution times in clock cycles of the resource for the resources to which it can be bound.

Definition 3.12 (*Unbound Graph*). An unbound graph gu is a 4-tuple $\langle A, D, Init, EC\rangle$ with a set A of actors, a set $D = A \times A$ of dependency edges, a function $Init : D \rightarrow \mathbb{N}$ that gives the number of initial tokens for an edge $d \in D$, and the function $EC : A \times R$ that gives the execution times in clock cycles of actors A on a number of resources in the set R.

Figure 2 illustrates an example SDFG model of an H.263 Encoder application. It consists of five actors which are connected to each other by means of seven dependency edges. Dependency edges d_3, d_6, and d_7 contain initial tokens, illustrated by black dots in the figure. The execution of an actor is called a *firing*. When an actor fires, it removes a number of tokens from all its input ports, and at the end of the firing (after its execution), it produces a number of tokens on each output port. The set of actor firings that restores the initial configuration of the graph is termed an *iteration*. During a single iteration of the graph, each actor can fire a number of times. This is

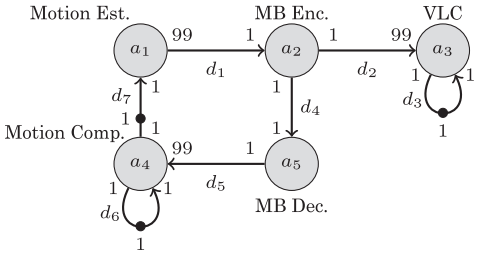


Fig. 2. Example SDFG model of an H.263 Encoder.

given by the repetition vector of the graph (Definition 3.13). The repetition vector of the SDFG shown in Figure 2 is equal to (1, 99, 1, 1, 99) for actors (a_1, a_2, a_3, a_4, a_5), respectively.

Definition 3.13 (*Repetition Vector*). The function $\gamma : A \to \mathbb{N}$ returns the number of times each actor $a \in A$ fires during a single iteration.

3.3. Model of a Bound Graph

Each actor can be bound to a number of resources from the set R. The set of resources an actor can be bound to is given by Definition 3.14.

Definition 3.14 (*Possible Bindings of an Actor*). The function $BP : A \to \mathcal{P}(R) \setminus \emptyset$ returns the set of resources to which an actor $a \in A$ can be bounded.

For a set A of actors and a set R of resources, there can be multiple bindings of actors to resources. The set of all possible actor to resource bindings can be obtained by the Cartesian product of the individual sets of possible bindings of actors (Definition 3.15).

Definition 3.15 (*Binding*). The set B of all possible actor to resource bindings is given as

$$B = \prod_{a \in A} BP(a). \tag{6}$$

For each binding, the execution time of an actor in clock cycles is known. The execution time of an actor in seconds on a resource for a specific operating point is given by Definition 3.16.

Definition 3.16 (*Execution Time in Seconds*). The function $ET : A \times R \to \mathbb{Q}$ returns the execution time in seconds of an actor $a \in A$ on a resource $r \in R$ that has a local operating point $ol \in OL(r, og)$, and is given as

$$ET(a, r) = \frac{EC(a, r)}{OL(r, og)}. \tag{7}$$

For a specific chip operating point and a specific binding of actors to resources, a model of a bound SDFG can be generated (Definition 3.17).

Definition 3.17 (*Bound Graph*). A bound graph gb is a 3-tuple $\langle gu, b, oc \rangle$ with an unbound graph gu, a binding $b \in B$ of actors A to resources R, and a chip operating point $oc \in OC$.

The throughput of an SDFG is traditionally computed by means of maximum cycle mean analysis (MCM) on the equivalent homogeneous SDFG (HSDFG) (Definition 3.18). This implies that a conversion from SDFG to HSDFG is required [Sriram and Bhattacharyya 2000].

Table IV. Optimization Criteria for Application Classes

	Best-effort	Firm real-time	Soft real-time
Average throughput	√	–	–
Yield	–	√	√
Average throughput degradation	–	–	√

Definition 3.18 (*Throughput of a Bound Graph*). The throughput of a bound graph gb is defined as $T(gu, b, oc) = 1/MCM(gb')$, where $MCM(gb')$ is the maximum cycle mean over all cycles in the equivalent HSDFG gb'. The cycle mean of each cycle c equals the sum of the execution times of actors in the cycle divided by the number of initial tokens on the cycle.

$$MCM(gb') = \max_{c \in C_{gb}} \sum_{a \in c} ET(a)/Init(c). \tag{8}$$

4. OPTIMIZATION PROBLEMS

In this section, we describe the requirements of *best-effort*, *firm real-time* and *soft real-time* applications, and define an optimization problem for each of them. For best-effort applications, we maximize the average throughput over all manufactured chips. The optimization objective for firm real-time applications is to maximize the yield, which is the percentage of manufactured chips that satisfy the application minimum throughput requirement, denoted t_{req}. For soft real-time applications, we maximize the yield and/or reduce the average throughput degradation, as the chips with a low degradation in the throughput can be used in this class of applications. The optimization criteria for the different application classes are summarized in Table IV.

For the different application classes, we present optimization approaches of *single binding* and *multiple bindings*. With the single-binding optimization approach, the objective is to find a binding at design time that results in an optimized objective function for an application class (e.g., yield for firm real-time applications). With the multiple-bindings optimization approach, a set of bindings are found and stored at design time, and based on the variation in each manufactured chip, the right binding that satisfies the application throughput requirement or maximizes the throughput is selected at the runtime configuration stage.

4.1. Single Binding

With the single-binding optimization approach, a single binding is selected at design time such that the objective function for an application class (e.g., yield for firm real-time applications) is optimized. With this approach, all manufactured chips, which have different chip operating points (variation) in the resources, have an identical binding. We proceed by presenting the optimization criteria for the different application classes and defining the single-binding optimization problem for each of them.

4.1.1. Best-Effort Applications. Best-effort applications do not have real-time performance requirements. Although there are no timing requirements set on the applications of this class, high performance is preferred by the user. Our optimization objective for best-effort applications is hence to maximize the average throughput of all manufactured chips.

For a given binding $b \in B$, the different chips that have different chip operating points (variation) can have different throughputs. Figure 3 depicts the throughput $T(gb) = T(gu, b, oc)$ of the bound graph gb for the different chip operating points $oc \in OC$, given a fixed binding $b \in B$. With the single-binding optimization approach, the objective is to find a binding that maximizes the average throughput t_{avg} over all chip operating points (i.e., average throughput over all manufactured chips). This is

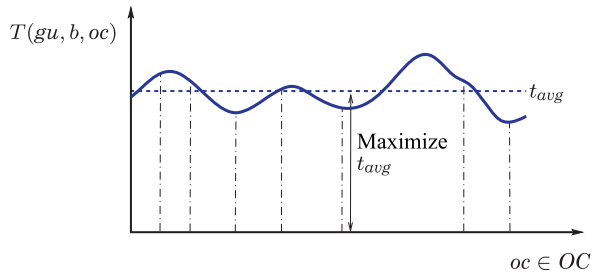


Fig. 3. Throughput against chip operating point for a fixed binding. The average throughput over all chip operating points is shown by the dotted line.

illustrated in Figure 3. Note that each chip operating point $oc \in OC$ shown on the x-axis in Figure 3 is a vector defining the local operating points of a chip. Therefore, the x-axis does not assume any increasing or decreasing ordering of $oc \in OC$. Furthermore, the figures presented in this section do not represent realistic applications but are given for illustrative purposes. For clarity, the throughput $T(gb) = T(gu, b, oc)$ is shown as a continuous curve but is discrete in reality.

The average throughput of a binding $b \in B$ over all chip operating points $oc \in OC$, where each chip operating point has a probability weight $PC(oc)$, is given by Definition 4.1.

Definition 4.1 (*Average Throughput*). Given an unbound graph gu and a binding $b \in B$, the function T_{avg} gives the probability-weighted average throughput over all chip operating points $oc \in OC$.

$$T_{avg}(b) = \sum_{oc \in OC} T(gu, b, oc) \cdot PC(oc). \tag{9}$$

The objective of the single-binding optimization approach for best-effort applications is formulated as given a set A of actors and a set R of resources, find a binding $b_{out} \in B$ of actors to resources such that the average throughput $t_{avg} = T_{avg}(b_{out})$ is maximized.

4.1.2. Firm Real-Time Applications. In firm real-time applications, violations of the timing requirements are not allowed. Examples of such applications are software defined radio, air traffic control, robotics, and military systems. The manufactured chips that have lower than the required performance cannot be used in such systems. Having higher than the required performance is not important as long as the performance requirement is satisfied. The optimization for this class of applications aims at maximizing the yield, which is the percentage of chips that satisfy the throughput requirement.

With the single-binding optimization approach for firm real-time applications, the objective is to find a binding that maximizes the yield. In our modeling framework, the yield of a binding $b \in B$ over all chip operating points $oc \in OC$, where each chip operating point has a probability weight $PC(oc)$, is given by Definition 4.2. The yield computation of a binding is graphically illustrated in Figure 4.

Definition 4.2 (*Yield of a Binding*). Given an unbound graph gu and a binding $b \in B$, the function Y gives the percentage of chips satisfying the requirement t_{req} over all chip operating points $oc \in OC$.

$$Y(b) = \sum_{oc \in OC} \begin{cases} PC(oc), & \text{if } T(gu, b, oc) \geq t_{req}, \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

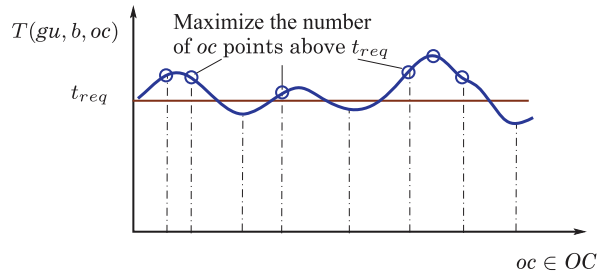


Fig. 4. Throughput against chip operating point for a fixed binding. Yield is given by the number of oc points (with associated probabilities) above t_{req}.

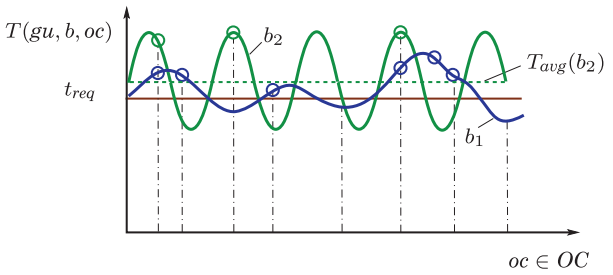


Fig. 5. Throughput against chip operating point for a fixed binding. Two bindings b_1 and b_2 are shown, where b_2 has higher average throughput but lower yield than b_1.

Figure 5 shows two bindings b_1 and b_2, where b_2 has a higher average throughput (better for best-effort applications) but a lower yield (worse for hard real-time applications) than b_1. This suggests the benefits of having different optimization criteria for best-effort and firm real-time application classes.

The objective of the single-binding optimization approach for firm real-time applications is formulated as given a set A of actors and a set R of resources, find a binding $b_{out} \in B$ of actors to resources such that the yield $y_{max} = Y(b_{out})$ is maximized.

4.1.3. Soft Real-Time Applications. Soft real-time applications are characterized by less stringent timing requirements. In such applications, missing a deadline causes only a performance degradation, often evaluated through some quality of service parameter. In addition, such applications are often dynamic in nature, where the average-case execution is more frequent and much shorter than the worst-case execution. Examples of such applications are multimedia systems, monitoring apparatuses, virtual reality, and interactive computer games. The optimization objective for this class of applications is to improve the yield but also reduce the average throughput degradation (deviation from the requirement) over all chips. Reducing the average throughput degradation is essential, as the chips with a low degradation can be used in this class of applications.

With the single-binding optimization approach, the objective is to find a binding that results in a maximized yield and/or a minimized average throughput degradation over all chip operating points (i.e., over all manufactured chips). Depending on the particular (yield, average throughput degradation) value pair, one binding may be preferred over another. Consider the following example. A binding that has a lower yield but also a lower average throughput degradation may be preferred over another binding with a higher yield. The trade-off between the yield and the average throughput degradation is captured in a cost function that guides the binding selection process (Definition 4.3).

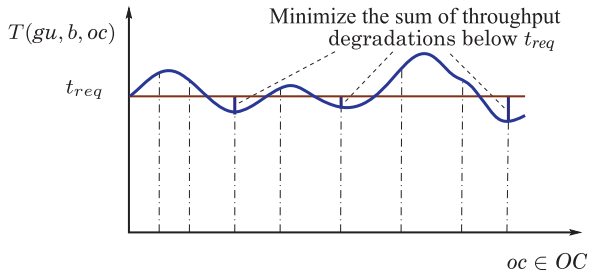


Fig. 6. Throughput against chip operating point for a fixed binding. Selecting a binding with the lowest cost function CF_{srt} is equivalent to minimizing the shaded area below t_{req}.

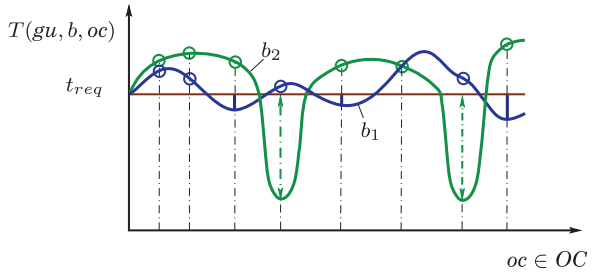


Fig. 7. Throughput against chip operating point for a fixed binding. Two bindings b_1 and b_2 are shown, where b_2 has higher yield but also a higher average throughput degradation than b_1.

The optimization problem for soft real-time applications is formulated based on the cost function and aims at minimizing it. This is graphically illustrated in Figure 6.

Definition 4.3 (*Soft Real-Time Cost Function*). Given an unbound graph gu and a binding $b \in B$, the function CF_{srt} gives the probability-weighted sum of throughput degradation over all chip operating points $oc \in OC$.

$$CF_{srt}(b) = \sum_{oc \in OC} \begin{cases} (t_{req} - T(gu, b, oc)) \cdot PC(oc), & \text{if } T(gu, b, oc) < t_{req}, \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

Figure 7 shows two bindings b_1 and b_2, where b_2 has a higher yield (better for firm real-time applications) but also a much higher average throughput degradation (worse for soft real-time applications) than b_1. This example suggests the benefits of having different optimization criteria for firm and soft real-time applications.

The objective of the single-binding optimization approach for soft real-time applications is formulated as given a set A of actors and a set R of resources, find a binding $b_{out} \in B$ of actors to resources such that the cost function $CF_{srt}(b_{out})$ is minimized.

4.2. Multiple Bindings (Runtime Configuration)

With the multiple-bindings optimization approach, a binding for each chip operating point is selected at design time. Therefore, a set of bindings are found and stored at design time. Based on the chip operating point (variation) of each manufactured chip, the right binding that satisfies the application throughput requirement or maximizes the throughput is then selected at the runtime configuration stage. The runtime binding selection for each chip is done only once at the system initial configuration stage through the operating system and is not detrimental to real-time deadlines. Per-chip binding selection always provides better or equal results compared to the case where a single

binding is present for all chips. The downside of the approach is that multiple bindings are stored for the configuration stage and diverse application instances are present for the same product, which can complicate the processes of software maintenance and upgrading.

The objective of the multiple-bindings optimization for best-effort applications is to find a binding for each chip operating point $oc \in OC$ such that the throughput of the bound graph at that chip operating point is maximized. For firm real-time applications, a binding for each chip operating point is selected that satisfies the throughput requirement of the bound graph at that chip operating point. If there is no binding that satisfies the requirement, then the chips with that particular chip operating point cannot be used, reducing the yield. For soft real-time applications, the objective is to find a binding for each chip operating point such that the throughput requirement is satisfied. If there is no binding that satisfies the requirement, then a binding is selected that provides the lowest throughput degradation.

5. IMPLEMENTATION ALGORITHMS

As shown in Section 4, to implement any of the optimization problems, various bindings of application actors to resources have to be explored. In this section, we present two algorithms for the evaluation of bindings, an *exhaustive* and a *heuristic* algorithms. With the exhaustive approach, we evaluate all binding possibilities (as given by Definition 3.15) to find a binding for all chips (single-binding optimization) or a binding per chip (multiple-bindings optimization). This approach enables us to find the optimum solution. However, it is too computationally expensive for problems of a large size (i.e., large number of actors and resources). To overcome this limitation, we also implemented a heuristic algorithm that prunes the search space and obtains results close to the optimum.

The exhaustive and the heuristic algorithms presented in this section are given for the class of firm real-time application and therefore aim at maximizing the yield. The algorithms for best-effort and soft real-time applications are similar to the ones presented and are not given in this section. The differences in the algorithms are explained where necessary.

5.1. Exhaustive Algorithm

The exhaustive algorithm for the single-binding optimization problem for firm real-time applications is shown in Algorithm 1. As input, the algorithm requires an application graph gu with an associated throughput requirement t_{req} and a set R of resources. Each resource is given by a nominal operating point $ON(r)$, a set $OG(r)$ of global operating points with associated probabilities $PG(r, og)$, and a set $L(r)$ of local operating points at $ON(r)$ with associated probabilities $P(r, l)$. As can be seen, the algorithm exhaustively evaluates the yield (Definition 4.2) of all possible bindings, and returns the binding b_{out} that results in the highest yield $y_{max} = Y(b_{out})$. The algorithms for best-effort and soft real-time applications use the functions $T_{avg}(b)$ (Definition 4.1) and $CF_{srt}(b)$ (Definition 4.3) to exhaustively find a binding with the highest average throughput or the lowest cost function for soft real-time applications.

Algorithm 2 illustrates the exhaustive algorithm for the multiple-bindings optimization problem for firm real-time applications. As shown, for each chip operating point $oc \in OC$, the algorithm exhaustively evaluates all possible bindings to find a binding that satisfies the requirement t_{req}. The first binding that satisfies the requirement t_{req} is stored for each chip operating point. It is possible that no binding can satisfy t_{req} for a particular $oc \in OC$, resulting in a reduced yield. The algorithm returns a set B_{out} of bindings that includes a binding for each individual $oc \in OC$ and an estimated yield y_{max} for all chips. For best-effort applications, the algorithm exhaustively evaluates all possible bindings for each chip operating point $oc \in OC$ to find a binding with a

ALGORITHM 1: Single-Binding Exhaustive Algorithm for Firm Real-Time Applications

Require: $gu, t_{req}, R, ON(r), OG(r)\ PG(r, og), L(r), P(r, l)$
1: $y_{max} \leftarrow 0$
2: **for all** $b \in B$ **do**
3: **if** $Y(b) > y_{max}$ **then**
4: $b_{out} \leftarrow b$
5: $y_{max} \leftarrow Y(b)$
6: **end if**
7: **end for**
8: **return** b_{out}, y_{max}

ALGORITHM 2: Multiple-Binding Exhaustive Algorithm for Firm Real-Time Applications

Require: $gu, t_{req}, R, ON(r), OG(r)\ PG(r, og), L(r), P(r, l)$
1: $y_{max} \leftarrow 0, B_{out} \leftarrow \emptyset$
2: **for all** $oc \in OC$ **do**
3: **for all** $b \in B$ **do**
4: **if** $T(gu, b, oc) \geq t_{req}$ **then**
5: $B_{out} \leftarrow B_{out} \cup \{b\}$ //save b for current oc
6: $y_{max} \leftarrow y_{max} + PC(oc)$
7: BREAK
8: **end if**
9: **end for**
10: **end for**
11: **return** B_{out}, y_{max}

maximized throughput. Note that the break statement on Line 7 in Algorithm 2 does not apply for the best-effort optimization algorithm, as all bindings for each chip operating point have to be explored. For soft real-time applications, the algorithm exhaustively selects a binding for each chip operating point $oc \in OC$ such that the requirement t_{req} is satisfied, or otherwise the lowest throughput degradation is obtained.

The exhaustive algorithms enable us to find the optimum solution. The limitation of the exhaustive algorithms is that they are too computationally expensive for problems of a large size (i.e., large number of actors and resources). The total number of bindings to evaluate for throughput for each $oc \in OC$ is $|R|^{|A|}$, where $|R|$ is the number of resources and $|A|$ is the number of actors belonging to an application. To address this problem, we proceed by presenting a heuristic algorithm that prunes the search space to reduce computation time while still obtaining results close to the optimum.

5.2. Heuristic Algorithm

With the heuristic algorithm, only a small number of bindings from the total number of possibilities are explored. The bindings that are evaluated by the heuristic algorithm are generated by a two-phase procedure, *initial resource allocation* and *allocation optimization*. In the initial resource allocation, an initial binding of application actors to resources is derived. This initial binding later undergoes an optimization stage where the allocation of each actor is reconsidered to either improve the yield (the average throughput or the cost function for soft real-time applications) for the single-binding optimization problem or the throughput for each chip for the multiple-bindings optimization problem.

In the initial resource allocation, the actors whose execution times are likely to have a large impact on the throughput of an application, referred to as *critical actors*, are considered first. The criticality of an actor $a \in A$ is estimated by the product of its repetition vector $\gamma(a)$ (Definition 3.13) and average execution time (Definition 3.11) in

a number of cycles over all the resources (Definition 5.1). This is an approximate way of determining the criticality, as it intuitively estimates the average computational demand of an actor. Note that depending on the topology of an SDFG, an actor a with the highest $CA(a)$, as given by Definition 5.1, may not necessarily be on the critical cycle of the graph and therefore may not have the largest impact on the throughput of the graph. For an exact estimation of criticality, all the cycles in the equivalent HSDFG can be analyzed. This can be computationally expensive, making it unsuitable for the heuristic algorithm. The criticality estimation presented in Definition 5.1 is sufficient for the initial resource allocation, since the allocation optimization stage reconsiders the allocation of each actor, compensating for infrequent mispredictions.

Definition 5.1 (*Actor Criticality*). The function $CA : A \to \mathbb{Q}$ returns the criticality of an actor $a \in A$ and is defined as

$$\forall a \in A.\ CA(a) = \gamma(a) \cdot \frac{1}{|R|} \sum_{r \in R} EC(a, r). \tag{12}$$

When allocating the actors to resources, the initial resource allocation tries to balance the load (in terms of execution time in seconds during an iteration) on the resources. The load of a resource is computed by the sum of products of the repetition vectors and the execution times in seconds of the actors bound to the resource (Definition 5.2).

Definition 5.2 (*Resource Load*). The function $LR : R \to \mathbb{Q}$ returns the load of a resource $r \in R$ and is defined as

$$\forall r \in R.\ LR(r) = \sum_{\substack{a \in A \\ a \text{ bound to } r}} \gamma(a) \cdot ET(a, r). \tag{13}$$

Algorithm 3 shows the heuristic algorithm for the single-binding optimization problem for firm real-time applications. In the first part of the algorithm (Lines 3–10), initial

ALGORITHM 3: Single-Binding Heuristic Algorithm for Firm Real-Time Applications

Require: gu, t_{req}, R, $ON(r)$, $OG(r)$ $PG(r, og)$, $L(r)$, $P(r, l)$

```
 1: y_max ← 0
 2: //Initial resource allocation
 3: ∀r ∈ R. LR(r) ← 0
 4: Sort A in decreasing CA(a)
 5: Sort R in decreasing ON(r)
 6: for all a ∈ A do
 7:    Sort R in increasing LR(r)
 8:    Bind a to first r
 9:    Update LR(r)
10: end for //initial binding b retrieved
11:
12: //Allocation optimization
13: for all a ∈ A do
14:    for all r ∈ R do
15:       if Y(b) > y_max then
16:          b_out ← b
17:          y_max ← Y(b)
18:       end if
19:       Bind a to r //new binding b retrieved
20:    end for
21: end for
22: return b_out, y_max
```

resource allocation is performed. The actors, sorted in decreasing order of criticality, are allocated to the resources such that the load on the resources is balanced. Each time an actor is to be bound to a resource, the resource with the lowest load is selected. If the resources are not yet allocated, an actor is bound to the resource with the highest nominal operating point (Definition 3.1). This is done to ensure that the actors with a high criticality are allocated to resources with a high computational power (resources with high nominal operating points are faster on average). In the second part of the algorithm (Lines 13–21), allocation optimization is performed. The allocation of each actor in increasing order of criticality is reconsidered. Each time the allocation of an actor is changed, the new binding is evaluated for yield. The algorithm returns a binding b_{out} that has the highest yield $y_{max} = Y(b_{out})$ among the bindings that have been explored. For best-effort and soft real-time applications, functions $T_{avg}(b)$ and $CF_{srt}(b)$ are used instead of $Y(b)$ (Lines 15–18) to find a binding with an improved average throughput and a reduced cost function for soft real-time applications.

The heuristic algorithm for the multiple-binding optimization problem for firm real-time applications is illustrated in Algorithm 4. For each chip operating point $oc \in OC$, the algorithm performs initial resource allocation and allocation optimization such that a binding is found that satisfies the requirement t_{req}. In initial resource allocation (for each $oc \in OC$), when the resources are not yet allocated, an actor is bound to the resource with the highest operating point based on the current chip operating point. This ensures that the actors with a high criticality are allocated to fast resources for each chip operating point. After initial resource allocation, allocation optimization is

ALGORITHM 4: Multiple-Binding Heuristic Algorithm for Firm Real-Time Applications

Require: gu, t_{req}, R, $ON(r)$, $OG(r)$ $PG(r, og)$, $L(r)$, $P(r, l)$
1: $y_{max} \leftarrow 0$, $B_{out} \leftarrow \emptyset$
2: Sort A in decreasing $CA(a)$
3: **for all** $oc \in OC$ **do**
4: //Initial resource allocation
5: $\forall r \in R.\ LR(r) \leftarrow 0$
6: Sort R in decreasing speed based on oc
7: **for all** $a \in A$ **do**
8: Sort R in increasing $LR(r)$
9: Bind a to first r
10: Update $LR(r)$
11: **end for** //initial binding b retrieved
12:
13: //Allocation optimization
14: **for all** $a \in A$ **do**
15: **if** $T(gu, b, oc) \geq t_{req}$ **then**
16: $B_{out} \leftarrow B_{out} \cup \{b\}$ //save b for current oc
17: $y_{max} \leftarrow y_{max} + PC(oc)$
18: BREAK
19: **end if**
20: **for all** $r \in R$ **do**
21: Bind a to r //new binding b retrieved
22: **if** $T(gu, b, oc) > t_{req}$ **then**
23: BREAK
24: **end if**
25: **end for**
26: **end for**
27: **end for**
28: **return** B_{out}, y_{max}

Table V. Application SDFG Overview

SDFG	Nr. actors	Nr. cycles
H.263 Decoder	4	0
H.263 Encoder	5	1
MP3 Playback	4	1
Sample Rate	6	0
Modem	16	5
MP3 Decoder	14	0
Satellite Receiver	22	0

performed (Lines 14–26). The allocation of each actor in increasing order of criticality is reconsidered. Each time the allocation of an actor is changed, the new binding is evaluated for throughput. The optimization for each $oc \in OC$ stops when a binding is found that satisfies the requirement t_{req}. The algorithm returns a set B_{out} of bindings that includes a binding for each individual $oc \in OC$ and an estimated yield y_{max} for all chips. For best-effort applications, the initial resource allocation and allocation optimization are performed for each chip operating point to find a binding with a maximized throughput. For soft real-time applications, a binding is found for each chip operating point such that the requirement t_{req} is satisfied, otherwise, the throughput degradation is minimized.

With the heuristic algorithm, the number of bindings to evaluate for throughput for each $oc \in OC$ is $|A| \cdot (|R| - 1)$. Given a large problem, $|A| \cdot (|R| - 1)$ is considerably lower than the total number $|R|^{|A|}$ of bindings explored by the exhaustive algorithm, that is, $|A| \cdot (|R| - 1) << |R|^{|A|}$. For example, one of our test applications that has 16 actors is allocated to three resources. With the exhaustive algorithm, it takes $3^{16} = 43{,}046{,}721$ bindings to evaluate for throughput for each chip operating point $oc \in OC$, while only $16 \cdot 2 = 32$ bindings are explored for each $oc \in OC$ by the heuristic algorithm.

6. EXPERIMENTAL RESULTS

In this section, we describe our experimental setup and present the results of our experiments. We demonstrate the improvements in the average throughput and the yield and the reductions in the average throughput degradation achieved by our variation-aware mapping algorithms over mapping approaches that are unaware of hardware variation. Additionally, we analyze how well the heuristic algorithms perform, compared to the optimum results provided by the exhaustive algorithms.

6.1. Setup

Our variation-aware mapping algorithms for best-effort, firm real-time, and soft real-time applications are evaluated on seven real DSP and multimedia applications modeled as SDFGs. From the DSP domain, the set contains a sample rate converter and modem, and from the multimedia domain, an H.263 Encoder (Figure 2), H.263 Decoder, MP3 Playback, MP3 Decoder, and satellite receiver. These application SDFGs are the unbound graphs in our formal framework. An overview of the SDFGs is shown in Table V. The table reports the number of actors and the number of cycles (feedback loops) in each application graph. The topologies of the application SDFGs, including their worst-case execution times in clock cycles, can be found online and are not presented in this article because of limited space.[1] For simplicity, we interpret the same seven applications as best effort, firm real-time, and soft real-time in our analysis when demonstrating the different application classes.

[1]SDF3 example applications. `http://www.es.ele.tue.nl/sdf3/download/examples`.

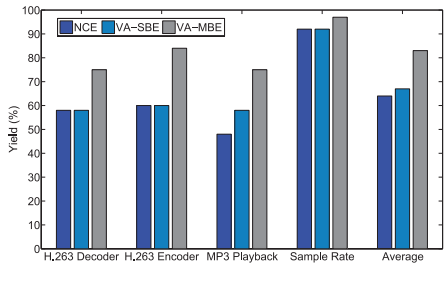
These applications are allocated to an MPSoC with 2–3 homogeneous resources with a nominal operating point of 500MHz for all resources. The sets of global and local operating points of resources are obtained by making assumptions on the impact of global and local variations on the nominal operating point (FMAX) of the resources. As shown in Eisele et al. [1997], the magnitude of the local variation depends on the logic depth of a path. As reported in Dighe et al. [2011], at high computing frequencies (in the order of GHz), the variation is large. In our work, we target embedded systems running streaming applications, where the typical frequencies are in the order of hundreds of MHz. Measurements of variation at 45nm technology in the frequency range of interest is done by Pang and Nikolic [2008]. Furthermore, Bowman et al. [2002] showed that the local variation reduces the mean of the maximum supported frequency of a core across all manufactured chips. Therefore, to reflect the impact of the local variation, we assume a 6% mean reduction of the nominal operating point $ON(r)$ and a standard deviation of 3.3% ($3\sigma = 10\%$). For the global variation, we assume a standard deviation of 3.3% ($3\sigma = 10\%$) for all resources. These assumptions are supported by Pang and Nikolic [2008]. To obtain the sets of global and local operating points, we discretize the FMAX probability density function of a resource into ten and five discrete points for global and local FMAX PDFs, respectively.

As given in Definition 3.18, the throughput of an SDFG is traditionally computed by means of MCM analysis on the equivalent HSDFG. This requires a conversion from the SDFG to an equivalent HSDFG, which can be considerably larger in size (in terms of the number of actors) than the original SDFG, making the approach inefficient for SDFG throughput analysis. In our work, we use the SDF3 tool for throughput analysis [Stuijk et al. 2006b]. To compute the throughput of an SDFG, SDF3 uses state-space exploration, which works directly on an SDFG and gives results equivalent to MCM analysis [Ghamarian et al. 2006]. Depending on what features are enabled in SDF3, the runtime of the tool can vary [Stuijk et al. 2006a]. In our experiments, SDF3 is used only for throughput analysis, and no additional features are required. This results in runtimes in the order of seconds per binding and chip operating point.

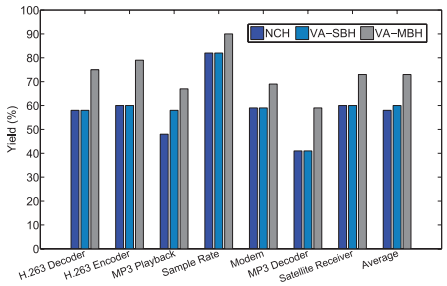
6.2. Evaluation Results

We compare the results of our optimization algorithms to those of variation-unaware nominal frequency-based mapping methods, where the binding of actors to resources is derived based on the nominal operating points of the resources. The purpose of the experiments is to show the importance of variation-aware mapping for the different application classes. Our results for the different application classes are presented for both variation-aware exhaustive and heuristic mapping algorithms. As the exhaustive mapping algorithms are too computationally expensive for problems of a large size, they are only applied to a subset of small to medium size applications from Table V. The heuristic mapping algorithms are evaluated on the complete set of applications, including the ones that are large in size. For the set of small to medium size applications, we evaluate how the heuristic mapping algorithms perform, as compared to the optimum results provided by the exhaustive algorithms.

6.2.1. Firm Real-Time Applications. Figure 8(a) illustrates the yield for the H.263 Decoder, H.263 Encoder, MP3 Playback, and Sample Rate Converter as a result of variation-aware exhaustive and nominal frequency-based mapping algorithms for the class of firm real-time applications. As shown in Table V, these applications have a small to medium number of actors (6 actors the largest), enabling the use of the exhaustive mapping algorithms. The exhaustive algorithms for the single-binding and multiple-bindings optimization approaches are denoted as VA-SBE and VA-MBE, respectively. For a nominal-frequency-based mapping, there can be multiple bindings of actors to resources that satisfy the throughput requirement for the nominal operating points



(a) For exhaustive mapping algorithms.

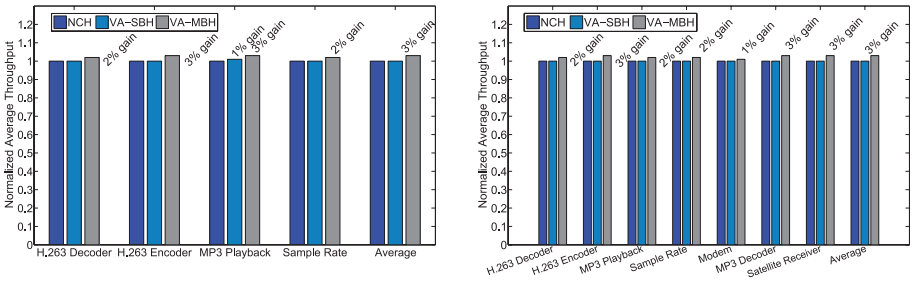


(b) For heuristic mapping algorithms.

Fig. 8. Yield of applications using the exhaustive (NCE, VA-SBE and VA-MBE) and heuristic (NCE, VA-SBE and VA-MBE) mapping algorithms for the class of firm real-time applications.

of the resources. A binding that just satisfies the requirement could potentially result in a very low yield, as any negative deviation of the nominal operating points of the resources could lead to a violation. For a fair comparison to the nominal-frequency-based mapping, we choose a binding that satisfies the throughput requirement and gives the highest throughput among all other bindings. The exhaustive nominal-frequency-based mapping is denoted as NCE in Figure 8(a). The figure shows that VA-SBE improves the result of the variation-unaware NCE mapping for only the MP3 Playback application, providing a 10% higher yield. For the rest of the applications, NCE results in an identical yield as VA-SBE. Therefore, we observe that the bindings providing a high throughput for the nominal operating points of the resources typically result in a high yield under variation. However, as shown with the MP3 Playback application, variation-aware mapping is necessary for finding a better solution. As expected, VA-MBE performs better than VA-SBE, resulting in improvements for all applications. Yield improvements of up to 27% (MP3 Playback) over the NCE mapping approach are reported. Figure 8(a) additionally illustrates the average yield for the set of applications. The variation-unaware NCE mapping results in a 64% average yield, which is improved to 67% and 83% by VA-SBE and VA-MBE, respectively. These results show the importance of variation-awareness in the resource allocation process. The runtimes of the exhaustive VA-SBE and VA-MBE algorithms for the applications in Figure 8(a) are in the order of hours (the highest) on a dual-core 2.8GHz machine. The nominal-frequency-based exhaustive NCE algorithm takes around ten minutes at most to finish.

The yield achieved by the variation-aware heuristic and nominal frequency-based mapping algorithms for the complete set of applications is shown in Figure 8(b). The nominal-frequency-based mapping, denoted as NCH, is also implemented by the same heuristic algorithm presented in Section 5. With NCH mapping, application actors are initially allocated to the resources for the nominal operating points, followed by an allocation optimization, where the initial allocation is optimized for a higher throughput. The heuristic NCH mapping for the H.263 Decoder, H.263 Encoder, and MP3 Playback results in the same yield as the exhaustive NCE mapping illustrated in Figure 8(a). In contrast, a 10% lower yield is achieved for the Sample Rate Converter. The heuristic mapping algorithm for the single-binding optimization problem (VA-SBH) for the H.263 Decoder, H.263 Encoder, and MP3 Playback provides the optimum results found by the computationally expensive VA-SBE exhaustive algorithm, previously shown in Figure 8(a). For the Sample Rate Converter, VA-SBH results in a 10% lower yield than the optimum result found by VA-SBE and performs equally to NCH. The heuristic algorithm for the multiple-binding mapping approach, denoted as VA-MBH, gives the optimum result for the H.263 Decoder. For the H.263 Encoder, MP3 Playback, and Sample Rate Converter, VA-MBH results in a 5%, 8%, and 7% lower yield, respectively.



(a) For exhaustive mapping algorithms. (b) For heuristic mapping algorithms.

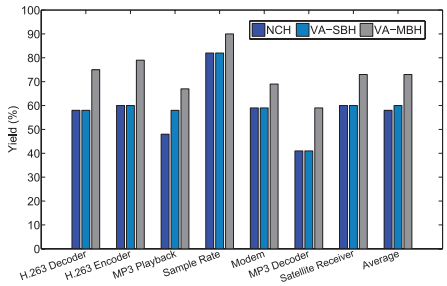
Fig. 9. Normalized average throughput of applications using the exhaustive (NCE, VA-SBE and VA-MBE) and heuristic (NCH, VA-SBH and VA-MBH) mapping algorithms for the class of best-effort applications.

These results show that the heuristic algorithms provide results close to the optimum with an average difference of 5%.

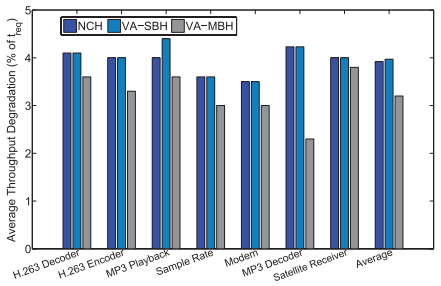
Figure 8(b) additionally illustrates the yield improvements achieved by the variation-aware heuristic mapping algorithms for the Modem, MP3 Decoder, and Satellite Receiver applications of a large size (22 actors the largest). As shown, for these applications, the variation-unaware NCH mapping provides the same results as the variation-aware VA-SBH, strengthening the preceding observation that the bindings providing a high throughput for the nominal operating points of the resources result in a high yield under variation. VA-MBH provides improvements in a yield of up to 18% for the large applications. The runtimes of the heuristic VA-SBH and VA-MBH algorithms for the applications of a large size are in the order of several hours on a dual-core 2.8GHz machine. The nominal-frequency-based heuristic NCH mapping algorithm takes around 10–15 minutes for these applications. The exhaustive algorithms for these applications are infeasible and do not finish in reasonable time. These results shows that the heuristic algorithms can be efficiently applied to problems of a large size.

6.2.2. Best-Effort Applications. Figure 9 illustrates the normalized average throughput for the set of applications using the exhaustive NCE, VA-SBE, and VA-MBE (for the subset of applications), and the heuristic NCH, VA-SBH, and VA-MBH mapping algorithms for the class of best-effort applications. The heuristic mapping algorithms provide optimum results for all small to medium size applications, except for MP3 Playback, where only a 1% lower average throughput is reported for both VA-SBH and VA-MBH. Figure 9 shows that VA-SBH improves the average throughput by only 1% for only the MP3 Playback application compared to the nominal-frequency-based allocation. On average, improvements in the average throughput of 3% are achieved by the variation-aware VA-MBH mapping algorithm. This shows that for the class of best-effort applications, variation awareness in the task allocation procedure is not as effective as for the class of firm real-time applications.

6.2.3. Soft Real-Time Applications. As explained in Section 4, the optimization objective for soft real-time applications is to improve the yield and/or reduce the average throughput degradation, as the chips with a low degradation can be used in this class of applications. The selection of a particular binding with (yield, average throughput degradation) value pair is accomplished by minimizing the cost function presented in Section 4.1.3. For the class of soft real-time applications, we illustrate the yield and the average throughput degradation for the set of applications as a result of nominal-frequency-based and variation-aware mapping algorithms.

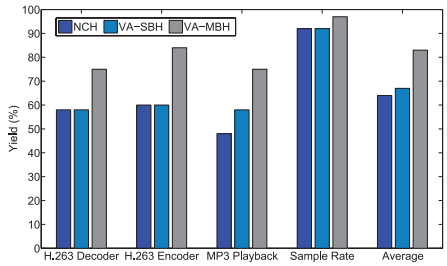


(a) Yield of applications.

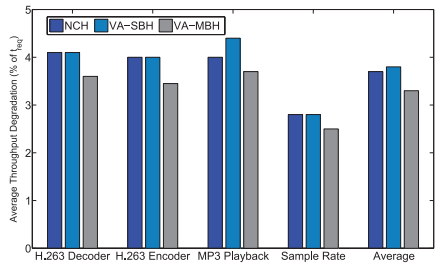


(b) Average throughput degradation of applications.

Fig. 10. Average throughput degradation and yield of applications using the NCH, VA-SBH, and VA-MBH heuristic mapping algorithms for the class of soft real-time applications.



(a) Yield of applications.



(b) Average throughput degradation of applications.

Fig. 11. Average throughput degradation and yield of applications using the NCH, VA-SBH, and VA-MBH exhaustive mapping algorithms for the class of soft real-time applications.

Figure 10 presents the results for the complete set of applications obtained by the heuristic NCH, VA-SBH, and VA-MBH mapping algorithms. It can be seen that the heuristic VA-SBH and VA-MBH mapping algorithms improve the yield of the variation-unaware NCH mapping by, on average, 2% and 15%, respectively. At the same time, Figure 10(b) illustrates reductions in the average throughput degradation of up to 2% (measured in percents of the throughput requirement) achieved by VA-MBH over NCH (MP3 Decoder). For the MP3 Playback application, VA-SBH results in a slightly higher average throughput degradation compared to NCH. However, the yield is improved by 10% over NCH mapping.

To evaluate the effectiveness of the heuristic mapping algorithms for the soft real-time optimization problem, we present the results of the exhaustive mapping algorithms for the H.263 Decoder, H.263 Encoder, MP3 Playback, and Sample Rate Converter applications. As in the case of firm real-time optimizations, the heuristic algorithms targeting soft real-time optimizations provide yield results close to the optimum with an average difference of 5%. The largest difference in the average throughput degradation from the optimum is obtained for the Sample Rate Converter application, where only around a 1% higher average throughput degradation is reported by the VA-SBH heuristic mapping algorithm.

Our experiments for best-effort, firm and soft real-time applications showed that many of the bindings selected for an application by VA-MBH are identical and that not more than 20 different bindings are selected for any of the applications. This

observation shows the applicability of the multiple-bindings optimization approach, as it provides significant improvements with a low impact on storage requirements.

With the proposed variation-aware mapping algorithms, we showed noticeable gains for the different classes of applications. These improvements were obtained for the particular local and global variation numbers assumed in Section 6.1. These numbers were selected based on the data available for the 45nm technology node in our frequency range of interest, as presented in Pang and Nikolic [2008]. The same work presents measurements of variation at 90nm technology and shows that going from 90nm to 45nm technology results in increased local variation, while global variation reduces in magnitude. Therefore, we can expect that local variation will further increase at technology nodes below 45nm. With increased local variation, resources have larger differences in the operating frequencies on the same chip. Therefore, the per-chip binding approach (i.e., VA-MB) is expected to give better results for future technology nodes.

7. CONCLUSIONS AND FUTURE WORK

This article introduces two approaches, single binding and multiple-bindings, for mapping real-time streaming applications to MPSoCs for a maximized yield under process variation. For application modeling, we use dataflow graphs that capture the iterative and overlapping execution of real-time streaming applications and have multiple efficient techniques for throughput computation. The novelty of our SDF formulation lies in the explicit modeling of software execution (1) in terms of clock cycles (which is independent of hardware variation), and (2) in terms of seconds (which does depend on the hardware variation), which are linked by an explicit binding. We present exhaustive and heuristic algorithms that implement the single-binding and multiple-binding optimization approaches. Our results show that (1) variation awareness is important in the resource allocation process, resulting in yield improvements of up to 27% with an average of 15% over the mapping methods that are unaware of hardware variation. (2) The heuristic mapping algorithm effectively reduces the exponential complexity of the exhaustive algorithm while only giving a slight reduction in the yield (5% on average). (3) The runtime storage requirements for the multiple-bindings optimization approach are very low, as only a few bindings are selected and stored for an application.

From the perspective of the number of bindings that must be evaluated, the scalability problem of the exhaustive mapping algorithms is addressed by the proposed heuristic algorithm. However, given a large number of resources and local, as well as global, operating points per resource, the number of chip operating points becomes very large, resulting in a very large number of throughput evaluations. This limits the scalability of the proposed mapping algorithms. As future work, we intend to address this scalability issue. The approaches presented in this work target real-time streaming applications which are mostly constrained by requirements on throughput. This is why our optimization approaches are throughput oriented. However, latency requirements may still be present in such applications. Latency analysis for synchronous dataflow graphs could be introduced in our framework such that optimizations based on latency requirements can be implemented. Future work furthermore involves extending our formal models to include power consumption, thus enabling variation-aware mapping in the presence of power constraints.

REFERENCES

Saman Amarasinghe, Michael I. Gordon, Michal Karczmarek, Jasper Lin, David Maze, Rodric M. Rabbah, and William Thies. 2005. Language and compiler design for streaming applications. *Int. J. Parallel Program.* 33, 2/3.

Alessio Bonfietti, Luca Benini, Michele Lombardi, and Michela Milano. 2010. An efficient and complete approach for throughput-maximal SDF allocation and scheduling on multi-core platforms. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE)*. European Design and Automation Association, 897–902.

Alessio Bonfietti, Michele Lombardi, Michela Milano, and Luca Benini. 2009. Throughput constraint for synchronous data flow graphs. In *Proceedings of the International Conference on Integration of AI and OR Techniques in Constraint Programming (CPAIOR)*. Springer-Verlag, Berlin, 26–40.

K. A. Bowman, S. G. Duvall, and J. D. Meindl. 2002. Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration. *IEEE J. Solid-State Circuits* 37, 2, 183–190.

Tracy D. Braun, Howard Jay Siegel, Noah Beck, Lasislau L. Bölöni, Muthucumara Maheswaran, Albert I. Reuther, James P. Robertson, Mitchell D. Theys, Bin Yao, Debra Hensgen, and Richard F. Freund. 2001. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *J. Parallel Distrib. Comput.* 61, 810–837.

HaNeul Chon and Taewhan Kim. 2009. Timing variation-aware task scheduling and binding for MPSoC. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASPDAC)*. 137–142.

S. Dighe, S. R. Vangal, P. Aseron, S. Kumar, T. Jacob, K. A. Bowman, J. Howard, J. Tschanz, V. Erraguntla, N. Borkar, V. K. De, and S. Borkar. 2011. Within-die variation-aware dynamic-voltage-frequency-scaling with optimal core allocation and thread hopping for the 80-core TeraFLOPS processor. *IEEE J. Solid-State Circuits* 46, 1 (Jan. 2011), 184–193.

M. Eisele, J. Berthold, D. Schmitt-Landsiedel, and R. Mahnkopf. 1997. The impact of intra-die device parameter variations on path delays and on the design for yield of low voltage digital circuits. *IEEE Tran. (VLSI) Systems* 5, 4, 360–368.

A. H. Ghamarian, M. C. W. Geilen, S. Stuijk, T. Basten, A. J. M. Moonen, M. J. G. Bekooij, B. D. Theelen, and M. R. Mousavi. 2006. Throughput analysis of synchronous data flow graphs. In *Proceedings of the International Conference on Application of Concurrency to System Design (ACSD)*. 25–36.

Andreas Hansson, Kees Goossens, Marco Bekooij, and Jos Huisken. 2009. CoMPSoC: A template for composable and predictable multi-processor system on chips. *ACM Trans. Des. Autom. Electron. Syst.* 14, 2:1–2:24.

Lin Huang and Qiang Xu. 2010. Performance yield-driven task allocation and scheduling for MPSoCs under process variation. In *Proceedings of the Design Automation Conference (DAC)*. 326–331.

M. Miranda, B. Dierickx, P. Zuber, P. Dobrovoln, F. Kutscherauer, P. Roussel, and P. Poliakov. 2009. Variability aware modeling of SoCs: From device variations to manufactured system yield. In *Proceedings of the Quality of Electronic Design (ISQED)*. 547–553.

D. Mirzoyan, B. Akesson, and K. Goossens. 2012. Process-variation aware mapping of real-time streaming applications to MPSoCs for improved yield. In *Proceedings of the International Symposium on Quality of Electronic Design (ISQED)*. 41–48.

Orlando Moreira and Marco Bekooij. 2007. Self-timed scheduling analysis for real-time applications. *EURASIP J. Adv. Signal Process.*

Liang Teck Pang and B. Nikolic. 2008. Measurement and analysis of variability in 45nm strained-Si CMOS technology. In *Proceedings of the Custom Integrated Circuits Conference (CICC)*. 129–132.

A. Shabbir, A. Kumar, S. Stuijk, B. Mesman, and H. Corporaal. 2010. CA-MPSoC: An automated design flow for predictable multi-processor architectures for multiple applications. *J. Syst. Archit.* 56, 265–277.

L. Singhal and E. Bozorgzadeh. 2008. Process variation aware system-level task allocation using stochastic ordering of delay distributions. In *Proceedings of the International Conference on Computer Aided Design (ICCAD)*. 570–574.

S. Sriram and S. S. Bhattacharyya. 2000. *Embedded Multiprocessors: Scheduling and Synchronization*. CRC Press.

S. Stuijk, T. Basten, M. C. W. Geilen, and H. Corporaal. 2007. Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs. In *Proceedings of the Design Automation Conference (DAC)*. 777–782.

S. Stuijk, M. Geilen, and T. Basten. 2006a. Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs. In *Proceedings of the Design Automation Conference (DAC)*. 899–904.

Sander Stuijk, Marc Geilen, and Twan Basten. 2006b. SDF3: SDF for free. In *Proceedings of the 6th International Conference on Application of Concurrency to System Design*. 276–278.

J. W. Tschanz, J. T. Kao, S. G. Narendra, R. Nair, D. A. Antoniadis, A. P. Chandrakasan, and V. De. 2002. Adaptive body bias for reducing impacts of die-to-die and within-die parameter variations on microprocessor frequency and leakage. *IEEE J. Solid-State Circuits*. 37, 11, 1396–1402.

O. S. Unsal, J. W. Tschanz, K. Bowman, V. De, X. Vera, A. Gonzalez, and O. Ergin. 2006. Impact of parameter variations on circuits and microarchitecture. *Proc. Microarchitec. (MICRO)* 26, 6, 30–39.

Feng Wang, C. Nicopoulos, Xiaoxia Wu, Yuan Xie, and N. Vijaykrishnan. 2007. Variation-aware task allocation and scheduling for MPSoC. In *Proceedings of the International Conference on Computer Aided Design (ICCAD)*. 598–603.

Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. 2008. The worst-case execution-time problem overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.* 7, 3, 36:1–36:53.

Received January 2012; revised July 2012, February 2013; accepted May 2013