Maximizing the Number of Good Dies for Streaming Applications in NoC-based MPSoCs under Process Variation

DAVIT MIRZOYAN, Delft University of Technology
BENNY AKESSON, Czech Technical University in Prague
SANDER STUIJK and KEES GOOSSENS, Eindhoven University of Technology

Scaling CMOS technology into nanometer feature size nodes has made it practically impossible to precisely control the manufacturing process. This results in variation in the speed and power consumption of a circuit. As a solution to process-induced variations, circuits are conventionally implemented with conservative design margins to guarantee the target frequency of each hardware component in manufactured multiprocessor chips. This approach, referred to as worst-case design, results in a considerable circuit upsizing, in turn reducing the number of dies on a wafer.

This work deals with the design of real-time systems for streaming applications (e.g., video decoders) constrained by a throughput requirement (e.g., frames per second) with reduced design margins, referred to as *better than worst-case design*. To this end, the first contribution of this work is *a complete modeling framework that captures a streaming application mapped to a NoC-based multiprocessor system with voltage-frequency islands under process-induced die-to-die and within-die frequency variations*. The framework is used to analyze the impact of variations in the frequency of hardware components on *application throughput at the system level*. The second contribution of this work is *a methodology to use the proposed framework and estimate the impact of reducing circuit design margins on the number of good dies that satisfy the throughput requirement of a real-time streaming application*. We show on both synthetic and real applications that the proposed better than worst-case design approach can increase the number of good dies by up to 9.6% and 18.8% for designs with and without fixed SRAM and IO blocks, respectively.

Categories and Subject Descriptors: C.3 [**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, reduced design margins

ACM Reference Format:
Mirzoyan D., Akesson B., Stuijk S., Goossens K.. Maximizing the Number of Good Dies for Streaming Applications in NoC-based MPSoCs under Process Variation. ACM Trans. Embedd. Comput. Syst. V, N, Article A (January YYYY), 25 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

Streaming applications in embedded multimedia and wireless systems are usually constrained by a throughput requirement such as frames per second for a video decoder. Considerable computational capability is required to implement these applications

This work was partially funded by projects EU FP7 288008 T-CREST and 288248 Flextiles, Catrene CA104 Cobra, CA505 BENEFIC, CA703 OpenES, ARTEMIS-2013-1 621429 EMC2 and 621353 DEWI, NL STW 10346 NEST, and the Ministry of Education of the Czech Republic under project number CZ.1.07/2.3.00/30.0034.
Author's addresses: D. Mirzoyan, Delft University of Technology, email: d.mirzoyan@tudelft.nl.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

due to an increasing amount of functionality. Additionally, many portable consumer electronics impose requirements on low power consumption for long battery life [van Berkel 2009]. To meet these requirements, streaming applications are implemented on a multiprocessor system-on-chip (MPSoC), where multiple processing cores exploit task and data-level parallelism to increase performance. The components inside a multiprocessor system were traditionally connected to each other by a bus. However, traditional buses do not provide scalable interconnection. For this reason, a paradigm shift towards network on a chip (NoC) based interconnection inside multiprocessor systems has been seen in recent years [Dally et al. 2001]. Present-day MPSoCs are implemented by means of the globally asynchronous, locally synchronous (GALS) design style [Muttersbach et al. 2000], which was introduced to alleviate the bottleneck of global clock distribution and reduce the related major source of power consumption in multiprocessor systems. The GALS architecture is composed of synchronous blocks, communicating with each other on an asynchronous basis. Within the GALS design paradigm, the concept of voltage-frequency islands (VFI) enables scaling the frequency (voltage) of each individual hardware component (clusters of components) in a multiprocessor system to further reduce power consumption.

To reduce circuit area and thus integrate more functionality on a chip die, CMOS technology has traditionally been scaled down. However, scaling in the nanoscale era has brought significant variability in the manufacturing process. This variability or inability to precisely control the manufacturing process results in significant variation in the maximum supported frequency of hardware components in a multiprocessor system [Bowman et al. 2002]. Considerable variability of up to 50% in the longest path delay of a processor is reported [Miranda et al. 2009]. As a solution, circuits are conventionally implemented with design margins or *guard-bands* to guarantee the target frequency of hardware components. Under this design paradigm, known as *worst-case design*, the hardware components in a multiprocessor platform are operated at their minimum frequencies. The tasks of an application are mapped to the hardware components such that a certain timing requirement (e.g., throughput or latency) imposed on the application is satisfied. This is illustrated in Figure 1a. However, worst-case design results in a considerable increase in circuit area and power consumption [Jeong et al. 2009], reducing the benefits of technology scaling.

This work deals with the design of real-time systems for streaming applications constrained by a throughput requirement with reduced design margins, referred to as *better than worst-case design*. With better than worst-case design, the area and the power consumption of a circuit are reduced. Smaller circuit area and thus die size results in a larger number of *gross dies* on a wafer, but the target maximum supported frequency of hardware components in a multiprocessor system is not guaranteed anymore. However, there may be hardware components with maximum frequencies higher and lower than the target frequency on the same chip die due to the impact of within-die variation. Operating each hardware component at its actual maximum supported frequency and using the available mapping freedom, the allocation of the tasks of an application to the hardware components can be tailored for each specific chip such that the throughput requirement is satisfied whenever possible. This is illustrated in Figure 1b. We introduce a design metric called *application yield* that quantifies the percentage of dies that satisfy the application throughput requirement. Figure 1c illustrates qualitative curves for die area, application yield and the number of good dies against guard-band reduction. The application yield reduces gradually, slower than die area, due to tailoring the mapping for each die. Thus, the number of *good dies* ((wafer area / die area) x application yield)) that satisfy the throughput requirement of the application is maximized. This is experimentally proven in Section 6.

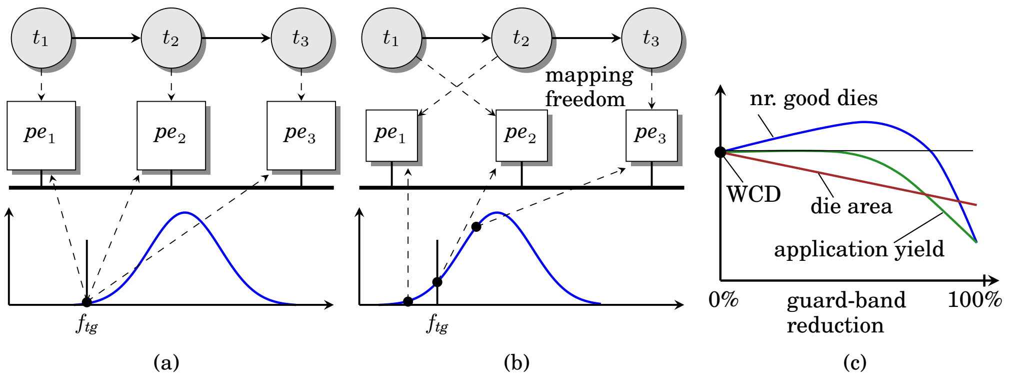


Fig. 1. An application constrained by a throughput requirement is allocated to a multiprocessor platform under (a) worst-case (WCD) and (b) better than worst-case designs. Each hardware component is operated at its (a) worst-case maximum supported (target) frequency f_{tg}, and (b) actual maximum supported frequency. Better than worst-case design (reduced guard-bands) results in more good dies due to a combination of smaller die area and high application yield (c).

This work has two contributions. The first contribution is a *complete modeling framework that captures a streaming application mapped to a NoC-based multiprocessor system with voltage-frequency islands under process-induced die-to-die and within-die frequency variations*. The framework is used to analyze the impact of variations in the frequency of hardware components on *application throughput*. Any set of clock-frequency levels can be specified per VFI domain. We use synchronous data-flow (SDF) to model a streaming application mapped to an MPSoC. The novelty of our SDF formulation lies in the *explicit modeling* of software execution in terms of *clock cycles* (which is independent of the frequency variation of hardware components), and in terms of *seconds* (which does depend on the frequency variation of hardware components), which are linked by an explicit *binding*. The second contribution of this work is *a methodology to use the proposed framework and estimate the impact of reducing circuit design margins on the number of good dies that satisfy the throughput requirement of a real-time streaming application* (see Figure 1c). This is what a system designer requires to decide by how much to reduce the design margins to maximize the number of good dies (minimize the cost per die). We show on both synthetic and real applications that the proposed design paradigm can increase the number of good dies by up to 9.6% and 18.8% for designs with and without fixed SRAM and IO blocks, respectively.

2. RELATED WORK

Techniques have been proposed to mitigate the impact of process variation at the circuit level. The main ideas are related to adaptive supply voltage and body biasing approaches [Meijer et al. 2012]. Both supply voltage upscaling and forward body biasing are effective for process-dependent performance compensation, and provide a large range of frequency upscaling. The disadvantage of the approaches is the increase in power consumption. The better than worst-case design approach proposed in this work is orthogonal to these methods and can be used in combination with them to further reduce design margins and increase the number of good dies.

Marculescu *et al.* analyze the probability distribution of *latency* of systems with multiple voltage-frequency islands considering within-die variation [Marculescu et al. 2008]. Their approach is only applicable to systems specified as acyclic task graphs, which are not able to capture the iterative and overlapping execution of many real-life streaming applications. In contrast, we allow arbitrary task graphs that may include cyclic data dependencies. We model a system by means of an SDF graph, which is well-suited for modeling and analysis of real-time streaming applications with throughput requirements. A methodology to perform system-level throughput analysis of multiple VFI designs, considering process variation, is presented by [Garg et al. 2008]. However, they only account for within-die variation, while we consider both within-die and die-to-die variations. Their approach is based on Homogeneous SDF (HSDF) graphs, which is a restricted case of an SDF graph that we use. An SDF graph provides much more compact application models, which is why many real-time streaming applications are modeled in an SDF formulation. To be able to use the approach in [Garg et al. 2008] for an application specified as an SDF graph, a conversion from the SDF graph to an equivalent HSDF graph is required [Sriram et al. 2000]. This can lead to an exponential increase in the graph size (in terms of the number of actors and edges), as compared to the original SDF graph. Performing throughput analysis on such a large graph results in prohibitively high computation times, making the approach in [Garg et al. 2008] unsuitable for many applications. The work in [Garg et al. 2008] assumes a one-to-one mapping of tasks to processing elements, while we allow resource sharing and assume static-order scheduling among tasks of an application allocated to the same core. They assume that the communication between tasks takes a given number of cycles, while we derive minimum bandwidth requirements based on an application and model a communication channel in a TDM-based NoC as a *latency-rate server* [Stiliadis et al. 1998]. Additionally, we define the relation between the reduction in circuit design margins and the number of good dies per wafer, which is what system designers require. This is new to the state-of-the-art. The related work on die-to-die and within-die variations is cited throughout the article whenever relevant.

In our previous work [Mirzoyan et al. 2013], we assumed a zero-latency interconnect in a multiprocessor system. As an extension to this work, we model an interconnect consisting of routers, links and network interfaces that are affected by process-induced frequency variations. We model the latency for sending data across the interconnect as a latency-rate server, and refine the previously proposed modeling framework according to the extensions. Additionally, we illustrate how the number of good dies is maximized with reduced guard-bands for a set of SDF graphs modeling real applications.

3. OVERVIEW

This section demonstrates how the different concepts introduced in this article fit together in the proposed better-than worst-case design flow. The block diagram of the design flow is illustrated in Figure 3. In Step 1, for a given guard-band value, a characterization and modeling of process-induced frequency variations is performed for each hardware component in the multiprocessor platform. The model of a multiprocessor platform is introduced in Section 4.1. The modeling of variation is presented in Section 4.2, and Section 5.2 describes how variation characterization for a given guard-band is performed. In Step 2, the multiprocessor platform as a whole is characterized by possible sets of frequencies for all hardware components (chip outcomes after manufacturing) based on the variation. This is given in Section 4.3. For each frequency set, there is a binding of application tasks to the hardware components. The real-time streaming application constrained by a throughput requirement is modeled by an application graph (Section 4.4). For each frequency set, the resource allocation in the hardware platform is modeled in the application graph based on the given binding (Step 3). The result is a resource-aware application graph for each frequency set. This is presented in Section 4.5. The application yield is then the percentage of resource-aware application graphs that provide throughput equal or greater than the requirement (Step 4). In Step 5, the die area is estimated with the specified guard-band. The number of good dies that satisfy the throughput requirement is then estimated given the application yield and the die area in Step 6. Section 5.1 presents the details for Steps 4, 5 and 6. Starting at full guard-bands, this flow is repeated with reducing guard-bands (thus die area), such that the number of good dies is maximized. The result is a guard-band value that provides an estimated maximum number of good dies.

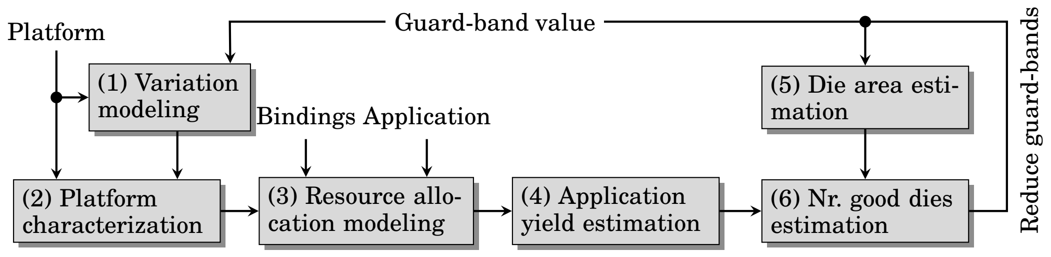

Platform
Guard-band value
Reduce guard-bands
(1) Variation modeling
Bindings
Application
(5) Die area estimation
(2) Platform characterization
(3) Resource allocation modeling
(4) Application yield estimation
(6) Nr. good dies estimation

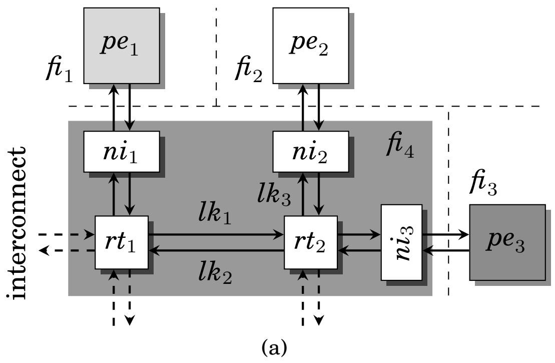
Fig. 2. Design flow for a real-time streaming application under better than worst-case designs.

4. FORMAL FRAMEWORK

This section presents the first contribution of this article, the modeling framework which is used to analyze the throughput of an application mapped to a multiprocessor system with voltage-frequency islands under the impact of process variation. To analyze the throughput of an application in a multiprocessor platform, a model of computation is required. The model needs to capture the application, the platform and the mapping of the application to the platform. The impact of process variation on the hardware resources (i.e., processing elements, routers, network interfaces and links) in the platform also needs to be captured. This section introduces the formal models of this work. We start by defining a hardware multiprocessor platform as a *platform graph*. We present how the modeling of variation in hardware resources is performed. Later, an SDF model of a streaming application, named a *resource-aware application graph*, is introduced. This model is unaware of the binding of application actors to processing elements, and is hence decoupled from hardware variation. Finally, we define another SDF model of the application, coined as a *bound application graph*. This graph captures the binding of a resource-aware application graph to a platform graph. We describe how resource allocation is modeled in a bound application graph. This model is used to perform timing analysis of the mapped application. Note that existing NoC-based multiprocessor systems such as CoMPSoC [Goossens et al. 2013], CA-MPSoC [Shabbir et al. 2010] and Daedalus^{RT} [Bamakhrama et al. 2012] use underlying models for performance modeling and analysis similar to the ones presented in this work. More specifically, these are a data-flow graph for application modeling, buffer size and latency-rate connection models for capturing resource allocation. However, these systems do not consider the impact of process variation in contrast to this work.

4.1. Platform graph

The template of a hardware multiprocessor platform used in this work and referred to as a platform graph is illustrated in Figure 3a. It consists of generic processing elements, such as processors, DSPs, or hardware accelerators, connected to each other by a network on chip (NoC), later referred in this work as an interconnect. Processing elements are denoted by *pe*. We assume an arbitrary topology interconnect, which consists of routers, denoted by *rt*, and network interfaces, denoted by *ni*, connected by unidirectional links, denoted by *lk*. The interconnect is assumed to provide lossless and ordered data transmission. Each processing element is connected to a single network interface in the interconnect. It is assumed that the network interfaces sit close to processing elements, and that the connections between processing elements and network interfaces do not introduce any delay. The path from a network interface to another network interface in the interconnect is referred to as *a connection*. A connection provides a certain maximum bandwidth (in bytes per cycle) given the number of resources on the connection are reserved. It also has a certain hop count, given by the number of routers on the path. We require that a connection can be modeled as a *latency-rate server* [Stiliadis et al. 1998], independent of other connections. In this work, we assume a time-division multiplexing (TDM) arbitration policy (although other arbitration policies can be used). Resource reservation on a connection is performed by allocating a number of slots in the TDM slot table. This provides a certain minimum bandwidth and maximum latency on the connection, as described in more details in Section 4.5.1. Examples of network-on-chips that provide the described properties are Æthereal [Goossens et al. 2010], Nostrum [Millberg et al. 2004] and SurfNoC [Wassel et al. 2013]. We formally define an interconnect in Definition 4.1. We refer to a processing element, a router, a network interface and a link in a platform graph as *a (hardware) resource* . As such, the union of the sets of processing elements, routers, network interfaces and links represents the set of all resources in the platform graph (Definition 4.2). The multiprocessor platform is given by a globally asynchronous, locally synchronous (GALS) architecture [Meincke et al. 1999], where the processing elements and the interconnect are partitioned into voltage-frequency islands (VFI). The interconnect is placed in a single VFI, and thus the resources in the interconnect belong to that island. The set of voltage frequency islands is denoted by *FI*. Communication between islands is accomplished by means of mixed-clock first-in-first-out (FIFO) buffers, which are part of network interfaces. A clock-generation unit (CGU) that provides a set of discrete clock-frequency levels, is dedicated to each voltage-frequency island. The formal definition of a platform graph *gp* is given in Definition 4.3. The set of all platform graphs is denoted by *GP*. The multiprocessor platform depicted in Figure 3a is partitioned into four islands, namely fi_1, fi_2 and fi_3 comprising processing elements pe_1, pe_2 and pe_3 respectively, and fi_4 consisting of the interconnect. The separation between clock domains is shown by the dotted lines.



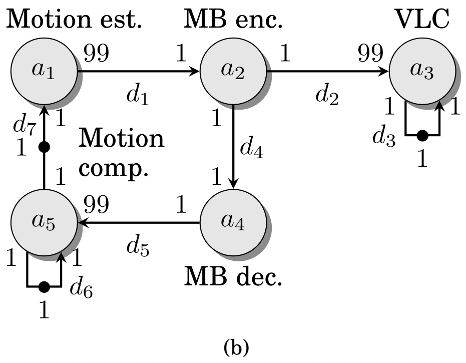


Fig. 3. (a) The template of a multiprocessor platform consisting of processing elements connected to each other by an interconnect. The processing elements and the interconnect are placed in different voltage-frequency islands. The separation between clock domains is shown by the dotted lines. (b) An example SDF model of an H.263 encoder.

Definition 4.1. (Interconnect) An interconnect *noc* is a 6-tuple $\langle RT, NI, LK, \eta, sz_{tb}, sz_{fl}\rangle$ consisting of a set RT of routers, a set NI of network interfaces, a set LK of links connecting routers and network interfaces in an arbitrary topology, a TDM slot-table size sz_{tb} (in number of slots) for all network interfaces, a flit size sz_{fl} (in bytes), a function $\eta(ni_i, ni_j)$, which for a connection from a network interface $ni_i \in NI$ to a network interface $ni_j \in NI$ ($ni_i \neq ni_j$) returns a tuple $\langle \beta, \Psi \rangle$ with β the maximum bandwidth (in bytes per cycle) assuming that all slots in the TDM table are reserved, and Ψ the number of hops.

Definition 4.2. (Set of resources) The set R of resources is the union of the sets *PE* of processing elements, *RT* of routers, *NI* of network interfaces, and *LK* of links in a platform graph, and is defined as $R = PE \cup RT \cup NI \cup LK$.

Definition 4.3. (Platform graph) A platform graph *gp* is a 5-tuple $\langle PE, noc, FI, \psi, \chi \rangle$ consisting of a set *PE* of processing elements, an interconnect *noc*, a set of voltage-frequency islands *FI*, a function $\psi(fi) : FI \rightarrow \mathcal{P}(R)$, which for each voltage-frequency island $fi \in FI$ returns the set $R_{fi} \subseteq R$ of resources belonging to the island, and a function $\chi(pe) : PE \rightarrow FI$, which for each processing element $pe \in PE$ returns the voltage-frequency island $fi \in FI$ to which the processing element belongs. Each processing element $pe \in PE$ is connected to a single network interface $ni \in NI$ in the interconnect *noc*.

4.2. Variation in hardware resources

The inability to precisely control the manufacturing process in sub-micrometer technology nodes leads to variability in key design (i.e., device and interconnect) parameters across the wafer. This variability aggregates to a higher level of logic blocks, resulting in variation in the maximum-supported frequency of hardware resources, such as processing elements. Manufacturing process variation can be classified into die-to-die and within-die variations. Die-to-die variation, also referred to as global variation, acts globally on the entire chip die, affecting parameters of all devices (i.e., transistors) and wires on the die identically. Global variation is seen between dies within a wafer and between dies of different wafers (due to wafer-to-wafer variation); therefore, overall global variation presumes multiple wafers. In contrast, within-die variation, also known as local variation, affects parameters of devices on the same die differently. It can be classified into systematic and random components. Systematic local variation exhibits spatial correlation, such that nearby devices possess similar parameter values. This correlation dies out quickly at the level of devices on a die as a function of distance. While the parameter correlation between adjacent devices on a die is high, the correlation between larger adjacent logic blocks on a die, such as a processing element, is typically much lower. There are existing works, such as [Huang et al. 2010], that capture spatial correlation in their modeling framework. They assume correlation values based on measurements performed at 90 nm technology in [Friedberg et al. 2005]. However, new measurements performed in [Pang et al. 2008; 2009] show no significant spatial correlation at 45 nm technology, in contrast to 90 nm technology. This is partially because random local variation, which is purely random from device to device, has more than doubled at 45 nm technology whereas systematic local variation has decreased. For simplicity, we assume zero correlation between maximum supported frequencies of hardware resources due to local variation. The impact of global and local process variations in the maximum supported frequency of resources in a platform graph is modeled by a normal distribution, which is shown to be a good fit for modeling the impact of global and local manufacturing process variations [Pang et al. 2008; Bowman et al. 2002]. We now proceed by presenting the models.

4.2.1. Global variation. To model the impact of global variation, we describe the maximum supported frequency of each hardware resource $r \in R$ in a platform graph *gp* by a random variable f_g^r distributed normally with μ_g^r mean and σ_g^r standard deviation. To denote that f_g^r is normally distributed, the notation $f_g^r = N(\mu_g^r, (\sigma_g^r)^2)$ is used. Global variation affects the maximum supported frequency of all hardware resources on a chip die identically. This results in equally faster or equally slower resources on each manufactured die. Therefore, we can say that the correlation between $(f_g^{r_i}, f_g^{r_j})$ for any $r_i, r_j \in R$ is equal to 1. Additionally, the standard deviation to mean ratio (σ_g^r/μ_g^r) is the same for all resources. The probability density function (PDF) and the cumulative distribution function (CDF) of a normally distributed random variable $x = N(\mu, \sigma^2)$ are denoted by $\phi(x, \mu, \sigma)$ and $\theta(x, \mu, \sigma)$, respectively. The CDF $\theta(x_0, \mu, \sigma)$ represents the probability that the random variable x takes on a value less than or equal to x_0. Note that neighboring dies on a wafer tend to have similar variation degrees. However, we do not deal with this spatial correlation as it does not influence the results presented in the work. To estimate the number of good dies, the dies with various variation profiles have to be considered, but the relative positioning of the dies does not need to be known.

4.2.2. Local variation. Let us assume that a hardware resource has a certain maximum supported frequency $f_g^r = f_0$ due to global variation. The impact of local variation on the maximum supported frequency of the resource is overlaid on f_0. We thus introduce a normally distributed random variable $f_l^r = N(f_0 - \delta^r, (\sigma_l^r)^2)$ to model the impact of local variation on the maximum supported frequency of a hardware resource with respect to a global frequency value $f_g^r = f_0$ of the resource. Here, σ_l^r is the standard deviation and δ^r models a reduction in mean frequency of the hardware resource. Processing elements often contain multiple critical paths, and the frequency of a processing element is decided by the slowest critical path [Bowman et al. 2002]. The probability that at least one of the critical paths is slowed down due to variation is higher than the probability that a single path is slowed down. This results in a mean frequency reduction, as shown in [Bowman et al. 2002]. Links contain multiple wires, and variation has a similar impact on the mean frequency as in processing elements. The reduction for links has been shown experimentally in [Hernandez et al. 2012]. In the same paper it is shown that the reduction in mean frequency is negligible for routers. We make a similar assumption of a negligible mean frequency reduction for network interfaces. As we assume no spatial correlation between the variation in maximum supported frequencies of hardware resources due to local process variation, the covariance between $(f_l^{r_i}, f_l^{r_j})$ for any $r_i, r_j \in R$ is equal to zero. Figure 4 illustrates an example PDF of f_g^r for a processing element with $\mu_g^r = 300$ MHz and $\sigma_g^r = 12$ MHz. The same figure shows the PDFs of $f_l^r = N(f_0 - \delta^r, (\sigma_l^r)^2)$ with respect to $f_0 = (\mu_g^r + k \cdot \sigma_g^r)$ for $(k = -1, 0, 1)$, where $\delta^r = 15$ MHz and $\sigma_l^r = 10$ MHz; these numbers are representative for 45 nm technology nodes, as the measurements in [Pang et al. 2008] show.

To describe the maximum supported frequency of a hardware resource by a single distribution, global and local distributions are combined by convolution. As explained before, global and local variations in the maximum-supported frequency of a resource are modeled by means of normal distributions. It is known that the convolution of two normal distributions is also a normal distribution with added means and variances. Therefore, the maximum supported frequency of a hardware resource due to both global and local variations is described by a normally distributed random variable given by Equation (1). The combined distribution for the example described in Figure 4 is shown in the figure.

$$f^r = N(\mu_g^r - \delta^r, (\sigma_g^r)^2 + (\sigma_l^r)^2) = N(\mu^r, (\sigma^r)^2) \tag{1}$$

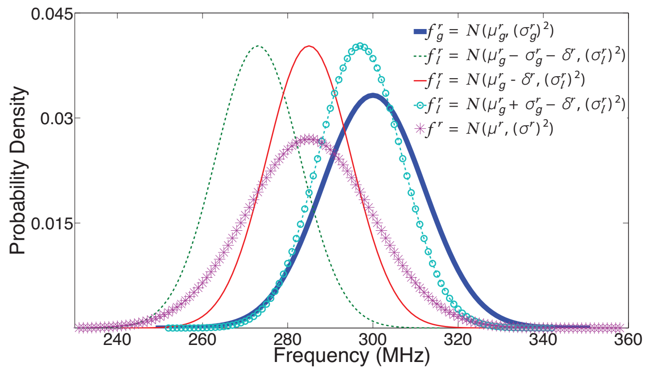


Fig. 4. f_g^r PDF (due to global variation) for a processing element with $\mu_g^r = 300$ MHz, $\sigma_g^r = 12$ MHz; f_l^r PDFs (due to local variation) with respect to $f_g^r = 273$, 285 and 297 MHz, $\delta^r = 15$ MHz, $\sigma_l^r = 10$ MHz; combined PDF of f^r is the convolution of PDFs of f_g^r and f_l^r.

4.3. Clock-frequency characterization

From an implementation perspective, all clock-generation units, associated with voltage-frequency islands in a platform graph, provide only a set of discrete clock-frequency levels. The selection of a set of clock-frequency levels for a voltage-frequency island is based on the variation in the maximum supported frequencies of hardware resources belonging to the island. It is performed in the following way. In a general case, a voltage-frequency island is comprised of multiple hardware resources (either processing elements or interconnect resources). Each resource is characterized by a combined distribution of its maximum supported frequency, reflecting both global and local process variations. For the purpose of clock-frequency selection, we consider only the frequency range within three standard deviations from mean (i.e., $\mu^r \pm 3\sigma^r$) in the distributions. The probability of the maximum supported frequency being outside the range of three standard deviations is only 0.3%. Considering the range outside the three standard deviations, and thus providing clock-frequency levels in a wider range, will result in a lower number of clock frequencies in the range of three standard deviations (for the same number of levels). This can result in a performance degradation in manufactured chips, as the gap between the actual maximum supported frequency and the clock frequency a resource is operated at will be on average larger for 99.7% of the resources. Figure 5 illustrates example combined distributions for the range of three standard deviations for two hardware resources belonging to the same island. We assume that the combined distributions can be in any arbitrary positioning with respect to each other. The clock frequency of an island is limited by the slowest resource belonging to the island. Considering all resources in a voltage-frequency island, we identify the frequency given by the lowest positive three standard deviations from mean (i.e., $\mu^r + 3\sigma^r$) in the combined distributions. In Figure 5, this frequency is shown by f_{high}. Similarly, the frequency given by the lowest negative three standard deviations is derived, as shown by f_{low} in Figure 5. Once the frequencies f_{low} and f_{high} are identified, the clock-frequency levels are selected in the range given by $(f_{high} - f_{low})$. In principle, clock-frequency levels in the range $(f_{high} - f_{low})$ can be selected in any arbitrary way. The policy of selection does not affect the rest of the methodology in this work. We choose to select the clock-frequency levels equidistantly, as formally defined in Definition 4.4. Figure 5 illustrates how five equidistant clock-frequency levels are obtained for the given example.

Definition 4.4. (Clock-frequency levels) A set of n equidistant clock-frequency levels available to a voltage-frequency island $\mathit{fi} \in \mathit{FI}$ comprised of a set $\psi(\mathit{fi})$ of hardware resources in a platform graph $\mathit{gp} \in \mathit{GP}$ is given by $c(\mathit{gp}, \mathit{fi}, n) : \mathit{GP} \times \mathit{FI} \times \mathbb{N} \to \mathcal{P}(\mathbb{R}^+)$, and for $f_{low} = \min\limits_{r \in \psi(\mathit{fi})} (\mu^r - 3\sigma^r)$, $f_{high} = \min\limits_{r \in \psi(\mathit{fi})} (\mu^r + 3\sigma^r)$, is defined as

$$c(\mathit{gp}, \mathit{fi}, n) = \{f_{low} + (k-1) \cdot \frac{(f_{high} - f_{low})}{n} | \ k = 1, 2, .., n\} \tag{2}$$

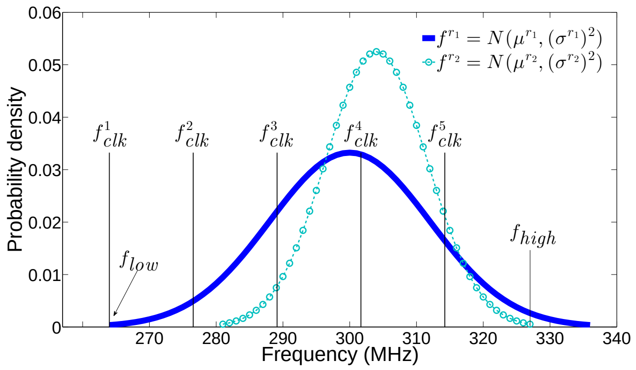


Fig. 5. An example showing how equidistant clock-frequency levels are selected for a voltage-frequency island comprising two hardware resources.

Given that each voltage-frequency island can be operated at any clock-frequency level in the set $c(\mathit{gp}, \mathit{fi}, n)$, for a set FI of islands in a platform graph, there are multiple possible combinations of clock-frequency levels. An instance of clock-frequency levels for all islands in a platform graph is captured in a *chip-frequency vector*, denoted by fc, and is an M-dimensional vector for M islands (Definition 4.5). Each element in fc represents a clock-frequency level $f_{clk} \in c(\mathit{gp}, \mathit{fi}, n)$ for a corresponding island $\mathit{fi} \in \mathit{FI}$. The set of all possible chip-frequency vectors is obtained by the Cartesian product of individual sets $c(\mathit{gp}, \mathit{fi}, n)$ (Definition 4.6).

Definition 4.5. (Chip-frequency vector) A chip-frequency vector for a set FI of voltage-frequency islands in a platform graph gp specifies a clock frequency f_{clk} from the set $c(\mathit{gp}, \mathit{fi}, n)$ for every island $\mathit{fi} \in \mathit{FI}$, and is given by $\mathit{fc}(\mathit{fi}) : \mathit{FI} \to \mathbb{R}^+$.

Definition 4.6. (All chip-frequency vectors) The set of all possible chip-frequency vectors for a set FI of voltage-frequency islands in a platform graph gp is given by

$$\mathit{FC} = \prod_{\mathit{fi} \in \mathit{FI}} c(\mathit{gp}, \mathit{fi}, n) \tag{3}$$

Each chip-frequency vector $\mathit{fc} \in \mathit{FC}$ is associated with a probability, which is the probability that voltage-frequency islands in a platform graph are operated at the particular clock-frequency levels specified by fc. From probability theory, it is known that the joint probability of independent events equals the product of their individual probabilities. However, due to the correlated global variation in hardware resources in a platform graph, frequencies described by random variables f^r are not independent. On the other hand, resource frequencies described by random variables f^r_l are independent, as we assume no spatial correlation for local variation. For this reason, the joint probability of a chip-frequency vector *fc* is represented as a sum of components. Each component is the *joint local probability*, which is the probability that the islands are operated at the particular clock-frequency levels based on local distributions with respect to a chip-level global frequency deviation. We introduce a *base hardware resource*, denoted by r_b, which is chosen arbitrarily among hardware resources in the platform graph. The base hardware resource serves as a reference for computing the global frequency values of all the resources in the platform graph. Given an absolute global frequency value $f_g^{r_b} = f_0$ of the base hardware resource, the global frequency value of any resource $r \in R$ is given by $f_0 \cdot (\mu_g^r / \mu_g^{r_b})$, where $\mu_g^{r_b}$ and μ_g^r are the global mean frequencies of hardware resources r_b and r, respectively.

We proceed by explaining how the probability that an island is operated at a clock frequency with respect to a global frequency value of the base resource is computed. In a general case, an island consists of multiple hardware resources. The clock frequency of the island is decided based on the slowest resource in the island. To compute the probability that the island is operated at a clock frequency f_{clk}, the probability that the maximum supported frequency of all resources in the island is higher than f_{clk} needs to be considered. This is given by the product of probabilities $(1 - \theta(f_{clk}, \mu_l^r, \sigma_l^r))$ for all hardware resources belonging to the island. Definition 4.7 defines the CDF of the minimum of maximum supported frequencies of hardware resources in an island; the CDF $\theta_m(x_0, \mathit{fi}, f_0)$ represents the probability that the minimum of the maximum supported frequencies of hardware resources takes on a value lower than x_0 with respect to an absolute global frequency value $f_g^{r_b} = f_0$ of the base resource r_b.

Definition 4.7. (Cumulative distribution of minimum) The CDF of the minimum of maximum supported frequencies of hardware resources $r \in \psi(\mathit{fi})$ belonging to a voltage-frequency island ***fi*** in a platform graph ***gp*** $\in$ ***GP*** is given by $\theta_m(\boldsymbol{gp}, x, \boldsymbol{fi}, f_0) : \boldsymbol{GP} \times \mathbb{R}^+ \times \boldsymbol{FI} \times \mathbb{R}^+ \to \mathbb{R}^+$, and is defined as

$$\theta_m(\boldsymbol{gp}, x, \boldsymbol{fi}, f_0) = 1 - \prod_{r \in \psi(\mathit{fi})} (1 - \theta(x, \mu_l^r, \sigma_l^r)) \tag{4}$$

where μ_l^r is the mean of the normally distributed random variable f_l^r, and is computed with respect to an absolute global frequency value $f_g^{r_b} = f_0$ of the base hardware resource $r_b \in R$ by $\mu_l^r = f_0 \cdot (\mu_g^r / \mu_g^{r_b}) - \delta^r$

Depending on the maximum supported frequencies of hardware resources in a voltage-frequency island, each island is operated at the highest possible clock-frequency level. Let us consider a case where five clock-frequency levels $\{f_{clk}^1, \cdots, f_{clk}^5\}$ are provided to a voltage-frequency island ***fi***. For any actual x (minimum of the maximum supported frequencies) in the range $(f_{clk}^4,\ f_{clk}^5]$, the island is operated at f_{clk}^4. The probability of x being in the interval $(f_{clk}^4,\ f_{clk}^5]$ is computed by the difference $(\theta_m(\boldsymbol{gp}, f_{clk}^5, \boldsymbol{fi}, f_0) - \theta_m(\boldsymbol{gp}, f_{clk}^4, \boldsymbol{fi}, f_0))$. Similarly, the probability that the island is operated at f_{clk}^5 is given by $(1 - \theta_m(\boldsymbol{gp}, f_{clk}^5, \boldsymbol{fi}, f_0))$. This is formally defined in Definition 4.8.

Definition 4.8. (Probability of clock frequency) The probability that a voltage-frequency island ***fi*** $\in$ ***FI*** in a platform graph ***gp*** $\in$ ***GP*** is operated at a clock frequency f_{clk}^i for a set $\{f_{clk}^1, \cdots, f_{clk}^n\}$ of clock-frequency levels, where $f_{clk}^i < f_{clk}^{i+1}$, with respect to a global frequency value $f_g^{r_b} = f_0$ of the base hardware resource $r_b \in R$, is given by $\boldsymbol{pf}(\boldsymbol{gp}, f_{clk}^i, \boldsymbol{fi}, f_0) : \boldsymbol{GP} \times \mathbb{R}^+ \times \boldsymbol{FI} \times \mathbb{R}^+ \to \mathbb{R}^+$, and is defined as

$$\boldsymbol{pf}(\boldsymbol{gp}, f_{clk}^i, \boldsymbol{fi}, f_0) = \begin{cases} \theta_m(\boldsymbol{gp}, f_{clk}^{i+1}, \boldsymbol{fi}, f_0) - \theta_m(\boldsymbol{gp}, f_{clk}^i, \boldsymbol{fi}, f_0) & i < n \\ 1 - \theta_m(\boldsymbol{gp}, f_{clk}^i, \boldsymbol{fi}, f_0) & i = n \end{cases} \tag{5}$$

The probability of a chip-frequency vector $fc \in FC$ for a set FI of voltage-frequency islands, with respect to a global frequency value f_0 of the base hardware resource, is computed by the product of individual probabilities $pf(gp, fc(fi), fi, f_0)$ and the probability of the global frequency value. This is formally defined in Definition 4.9.

Definition 4.9. (Local probability of fc) The probability of a chip-frequency vector $fc \in FC$ for a set FI of voltage-frequency islands in a platform graph $gp \in GP$, with respect to a global frequency value $f_g^{r_b} = f_0$ of the base hardware resource $r_b \in R$, is given by $p(gp, fc, f_0) : GP \times FC \times \mathbb{R}^+ \rightarrow \mathbb{R}^+$, and is defined as

$$p(gp, fc, f_0) = \phi(f_0, \mu_g^{r_b}, \sigma_g^{r_b}) \cdot \prod_{fi \in FI} pf(gp, fc(fi), fi, f_0) \tag{6}$$

The overall probability of a chip-frequency vector fc is obtained by adding the joint local probabilities for all global frequency values in the range of three standard deviations from mean for the base hardware resource, as defined in Definition 4.10.

Definition 4.10. (Probability of fc) The probability of a chip-frequency vector $fc \in FC$ in a platform graph $gp \in GP$ is given by $pc(gp, fc) : GP \times FC \rightarrow \mathbb{R}^+$, and for $I = [\mu_g^{r_b} - 3\sigma_g^{r_b},\ \mu_g^{r_b} + 3\sigma_g^{r_b}]$, is defined as

$$pc(gp, fc) = \sum_{f_0 \in I} p(gp, fc, f_0) \tag{7}$$

4.4. Resource-aware application graph

We model a real-time streaming application by means of synchronous data-flow (SDF) graphs [Lee et al. 1987]. An SDF graph provides a good compromise between expressiveness, modeling ease, analysis potential and implementation efficiency. With an SDF model, an application is captured by a directed graph, where the nodes, called actors, represent computation, communication or storage. Actors communicate with each other by sending streams of data elements, called tokens, over their edges. We formally define an SDF graph in Definition 4.11.

Definition 4.11. (SDF graph) An SDF graph *sdfg* is a 3-tuple $\langle A, D, P\rangle$ consisting of a set A of actors, a set $D = A^2$ of dependency edges and a set P of ports. Each dependency edge $d \in D$ has a number of initial tokens $\xi(d) : D \rightarrow \mathbb{N}^0$. Each actor $a \in A$ is associated with input and output ports, where each port $pt \in P$ has a rate $rate(pt) : P \rightarrow \mathbb{N}^+$. The source of a dependency edge is an output port of an actor, and the destination of a dependency edge is an input port of an actor. Each port of each actor is connected to a single edge, and each edge is connected to ports of actors.

Figure 3b illustrates an example SDF model of an H.263 encoder application. It consists of five actors connected to each other by seven dependency edges. Edges d_3, d_6 and d_7 each contain one initial token, illustrated by black dots in the figure. An actor *fires* (executes) when it has sufficient number of tokens on each of its input ports, as specified by port rates $rate(pt)$. The port rates are shown near the channel ends in Figure 3b. When an actor fires it removes the number of tokens from all its input ports and before the end of the firing produces a number of tokens on each output port, as given by its output port rates. The sequence of actor firings that restores the initial configuration of the graph (the initial distribution of tokens on dependency edges $d \in D$) is termed an *iteration*. During a single iteration of the graph, each actor can fire multiple times. This is determined by the port rates of actors, and is captured by the repetition vector of the graph (Definition 4.12). The repetition vector of the SDF graph shown in Figure 3b is equal to $\langle 1, 99, 1, 99, 1\rangle$ for actors $\langle a_1, a_2, a_3, a_4, a_5\rangle$, respectively.

Definition 4.12. (Repetition vector) A repetition vector of an SDF graph *sdfg* specifies the number of times each actor $a \in A$ fires during a single iteration, and is given by $\kappa : A \rightarrow \mathbb{N}$, such that for each dependency edge $d \in D$ from an actor $a_i \in A$ to an actor $a_j \in A$, $i \neq j$, $\mathit{rate}(\mathit{pt}_d^{sr}) \cdot \kappa(a_i) = \mathit{rate}(\mathit{pt}_d^{ds}) \cdot \kappa(a_j)$, where $\mathit{pt}_d^{sr} \in P$ and $\mathit{pt}_d^{ds} \in P$, are the source and destination ports of the dependency edge, respectively.

When mapping an application described by an SDF graph to a multiprocessor platform, the minimum throughput requirement (in iterations per second) of the application (for real-time applications) and information on resource requirements of the application must be known. Such information includes the number of clock cycles each actor requires to finish its execution on a processing element, for all processing elements to which the actor can be bound (Definition 4.13), the buffer space (in number of tokens) assigned to each dependency edge for storing the data tokens produced by actors (in a real implementation these buffers can be allocated in local memories of processing elements), and the size (in bytes) of data tokens sent across each dependency edge. Execution cycles can be derived using worst-case execution-time estimation tools [Wilhelm et al. 2008]. After the actors in an application are bound to the processing elements in a multiprocessor platform, dependency edges may be allocated to connections in the interconnect (i.e., if two actors connected by a dependency edge are bound to different tiles). To reserve resources on the interconnect, additional information on the minimum bandwidth (in bytes per cycle) required by dependency edges has to be known. We formally define an SDF model of an application, called a *resource-aware application graph* (*ga*) that includes the described information in Definition 4.14. The set of all resource-aware application graphs is denoted by *GA*. A resource-aware application graph states the resource requirements, but does not include the binding of actors to processing elements and dependency edges to connections in the interconnect.

Definition 4.13. (Execution cycles) The number of cycles required to execute an actor $a \in A$ on a processing element $\mathit{pe} \in \mathit{PE}$ to which it can be bound is given by $\mathit{ec}(a, \mathit{pe}) : A \times \mathit{PE} \rightarrow \mathbb{N}$.

Definition 4.14. (Resource-aware application graph) A resource-aware application graph *ga* is a 4-tuple $\langle \mathit{sdfg}, t_{\mathit{req}}, \mathit{ec}, \omega\rangle$ consisting of an SDF graph *sdfg*, a minimum throughput requirement t_{req} for real-time applications, the function $\mathit{ec}(a, \mathit{pe})$ that assigns each actor $a \in A$ with execution cycles for the subset of processing elements in the set *PE* that a can be bound to, and a function $\omega(d)$, which for each dependency edge $d \in D$ returns a 3-tuple $\langle \mathit{sz}_d, \beta_d^{rq}, \alpha_d\rangle$ with sz_d the size of the data token (in bytes) sent across the dependency edge, β_d^{rq} the required bandwidth (in bytes per cycle), and α_d the buffer size (in number of tokens) assigned to the dependency edge.

The impact of the buffer size α_d assigned for storing data tokens on each dependence edge $d \in D$ on the throughput of an SDF graph has been previously explored in [Stuijk et al. 2006a]. Here it is fixed to a sufficiently large number $2 \cdot \mathit{rate}(\mathit{pt}_d^{sr}) \cdot \kappa(a^{sr})$, where $\mathit{rate}(\mathit{pt}_d^{sr})$ is the rate of the source port of the edge and $\kappa(a^{sr})$ is the number of times the source actor fires in an iteration. The term $\mathit{rate}(\mathit{pt}_d^{sr}) \cdot \kappa(a^{sr})$ represents the number of tokens produced on the edge in a single iteration. We use a buffer size twice larger due to possible overlapping of multiple iterations. Any buffer dimensioning can be used by a system designer based on the requirements on the throughput.

The required bandwidth β_d^{rq} (in bytes per cycle) for a dependency edge $d \in D$ in a resource-aware application graph *ga* is estimated by $\mathit{rate}(\mathit{pt}_d^{sr}) \cdot \kappa(a^{sr}) \cdot \mathit{sz}_d \cdot t_{\alpha_d}$, where t_{α_d} is the throughput (in iterations per cycle) of the resource-aware application graph with the specified α_d buffer size for each dependency edge. The product of the first three terms (i.e., the number of bytes that are produced on the edge during a single iteration) and the throughput t_{α_d} represents the number of bytes produced on the edge in a single cycle. Note that the throughput t_{α_d} is evaluated on the resource-aware application graph assuming that only a single execution of each actor is allowed at the same point in time (an execution of an actor can only start after the previous execution has finished). This is done as we do not allow multiple concurrent executions of an application actor on a processing element, as described in Section 4.5.1. As such, the throughput t_{α_d} is simply used to derive an estimated required bandwidth (independent of mapping) that is not overly pessimistic.

4.5. Bound application graph

When implementing the application on a multiprocessor platform, designers need to make choices on mapping of application actors to processing elements and thus dependency edges to connections in the interconnect, reserving resources on the connections, and scheduling of actors on the same processing element. Once these decisions have been made, a new SDF model, the *bound application graph*, that captures these decisions is constructed such that it can be used to perform timing analysis of the system. While a resource-aware application graph describes performance in terms of execution time in cycles, the essence of a bound application graph is that it considers performance in terms of execution time in seconds. This allows us to take process-induced variation in the frequency of processing elements into account. This section details how a bound application graph is constructed.

4.5.1. Modeling resource allocation. We start by capturing the binding of the resource-aware application graph to the platform graph. We assume that each actor can be bound to a number of processing elements from the set *PE*, as given by Definition 4.15. Therefore, there are multiple bindings of a set A of actors to a set *PE* of processing elements. The set of all possible bindings is obtained by the Cartesian product of the individual sets of possible bindings (Definition 4.16).

Definition 4.15. (Actor bindings) The function $ba(a) : A \to \mathcal{P}(PE) \setminus \emptyset$ returns the set of processing elements to which an actor $a \in A$ can be bound.

Definition 4.16. (Binding) The set B of all possible actor to processing element bindings is given as

$$B = \prod_{a \in A} ba(a) \tag{8}$$

A given binding of actors to processing elements is captured in a *binding vector*, denoted by b, and is an N-dimensional vector for N actors (Definition 4.17). Each element in b specifies the processing element to which each actor is bound. For example, let us assume that the resource-aware application graph shown in Figure 3b is mapped to a multiprocessor platform comprising two processing elements; a_2, a_3 are bound to pe_1 and a_1, a_4, a_5 are bound to pe_2. This is given in a binding vector $\langle a_1, a_2, a_3, a_4, a_5 \rangle \to \langle pe_2, pe_1, pe_1, pe_2, pe_2 \rangle$.

Definition 4.17. (Binding vector) A binding vector for a set A of actors specifies the processing element $pe \in PE$ to which each actor $a \in A$ is bound, and is given by $b(a) : A \to PE$.

For a given binding vector b, the execution cycles $ec(a,pe)$ of each actor $a \in A$ is known from Definition 4.13. The execution time $et(gp, a, pe)$ in seconds is obtained by the division of $ec(a, pe)$ by the clock frequency $fc(\chi(b(a)))$ of the voltage-frequency island to which the processing element belongs (Definition 4.18).

Definition 4.18. (Execution time) The execution time (in seconds) of an actor $a \in A$ on a processing element $pe \in PE$, which belongs to a voltage-frequency island $\chi(b(a))$ operated at a clock frequency $fc(\chi(b(a)))$ in a platform graph $gp \in GP$, is given by $et(gp, a, pe) : GP \times A \times PE \to \mathbb{R}^+$, and is defined as $et(gp, a, pe) = ec(a, pe)/fc(\chi(b(a)))$

In this work, we assume that only a single execution of an actor is allowed on a processing element at any point in time. In an SDF graph, this is modeled by adding a self-edge with a single initial token and with rates equal to one on source and destination ports on all actors (Figure 6a). This prohibits an execution from starting until the previous execution has finished. To model buffers with a finite capacity in an SDF graph, the approach in [Poplavko et al. 2003] is used. Figure 6b shows how the available buffer space on the dependency edge d with $\xi(d)$ initial tokens (Definition 4.11) and a buffer size α_d (Definition 4.14) is modeled in the graph. For this, a backward edge d_b from a_j to a_i is added with $(\alpha_d - \xi(d))$ initial tokens on the edge, for which $(rate(pt^{sr}_{d_b}) = rate(pt^{ds}_{d}))$ and $(rate(pt^{ds}_{d_b}) = rate(pt^{sr}_{d}))$ hold, where $(pt^{sr}_{d}, pt^{ds}_{d})$ and $(pt^{sr}_{d_b}, pt^{ds}_{d_b})$ are the source and destination ports of the edges d and d_b, respectively. Token consumption from the edge d_b can be seen as claiming space in the buffer for writing the data tokens. Similarly, token production on d_b can be seen as freeing space in the buffer.

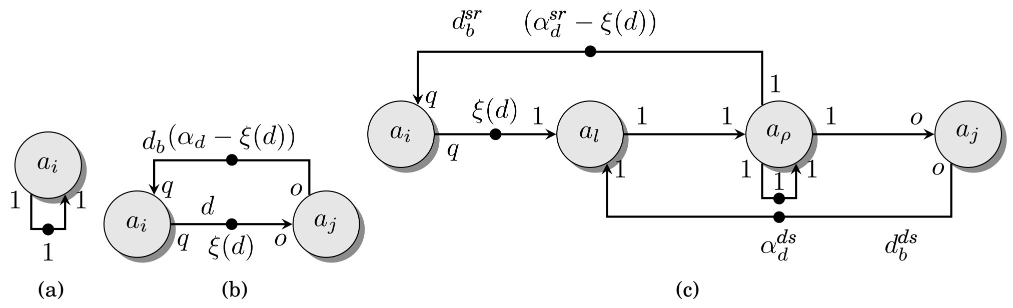


Fig. 6. Modeling resource allocation in an SDF graph. (a) Actor bound to a processing element. (b) Model of available buffer space between actors. (c) SDF model of a connection in the interconnect.

Now we consider how to model latency of data communication across a connection in the interconnect in an SDF graph. In Figure 6b, the latency for sending data tokens across the dependency edge d equals zero. We assume that this is the case when two actors are bound to the same processing element, as they communicate through a shared local memory. However, when two actors are bound to different processing elements, the dependency edge that connects the actors is bound to a connection in the interconnect. In this scenario, the interconnect introduces latency for sending a data token across the connection. We now describe a connection model that considers the interconnect architecture is constructed in an SDF graph. As detailed in Section 4.1, we model a connection as a latency-rate server [Stiliadis et al. 1998]. We use a TDM-based NoC modelled similarly to [Hansson et al. 2009], see in Figure 6c. It assumes the SDF graph of Figure 6b, where actors a_i and a_j are bound to different processing elements, and thus the dependency edge d is bound to a connection in an interconnect. The execution time of actor a_l captures the maximum latency a data token can ever experience in a connection. To model pipelining between data tokens the actor does not have a self-edge with an initial token on it. The edge d^{ds}_b, with α^{ds}_d initial tokens on it, models the buffer space at the destination tile. Actor a_l fires only when sufficient tokens are available on the edge, corresponding to a data transaction being initiated only when buffer space is available at the destination processing element. The execution time of the actor is the sum of the slot-table injection latency and path latency. Data injection into the router network is regulated by the reserved slots in the TDM table of a connection. The slot-table injection latency depends on the number of reserved slots. Based on the required bandwidth β_d^{rq} of a dependency edge $d \in D$, a number $\Lambda = \lceil (\beta_d^{rq}/\beta) \cdot sz_{tb} \rceil$ of slots are reserved in the TDM table (with a size sz_{tb}) of a connection, where β is the maximum bandwidth (in bytes per cycle) of a connection from a network interface ni_i to a network interface ni_j, $i \neq j$, (Definition 4.1). The bandwidth β assumes that all slots in the TDM table are reserved. The allocated bandwidth (in bytes per cycle) is computed by $\beta_{al} = (\Lambda/sz_{tb}) \cdot \beta$.

The slot-table injection latency not only depends on the number of reserved slots, but also on the distance between them. In this work, we assume a continuous slot reservation strategy due to the simplicity of its analysis and implementation. With this strategy, in the worst-case situation, a data token arrives right after the reserved Λ slots, resulting in a waiting time of a number $(sz_{tb} - \Lambda)$ of slots. The slot-table injection latency (in cycles) is given by $\Theta_{tb} = (sz_{tb} - \Lambda) \cdot (sz_{fl}/\beta)$, where sz_{fl}/β represents the latency (in cycles) of a single slot. Note that other reservation strategies, such as distributed slot reservation, are possible. These strategies may give a lower service latency, but are more difficult to analyze and implement.

The path latency depends on the number Ψ of hops, given by the function η (Definition 4.3), and the pipelining depth α_{rt} of the routers. In the case of the Æthereal network [Goossens et al. 2010], the pipelining depth of the routers is three, which is also chosen in this work. The path latency (in cycles) is given by $\Theta_{hp} = \Psi \cdot \alpha_{rt}$. Combining the path latency and the slot-table injection latency, the execution time (in seconds) of actor a_l is given by Equation (9).

$$et_l(a_l) = \frac{(\Theta_{tb} + \Theta_{hp})}{fc(fi_{noc})} \tag{9}$$

Actor a_ρ bounds the rate at which data tokens are sent. With a self edge and a single initial token on it, the execution time of the actor captures the time it takes to serve a data token after an initial latency et_l. The execution time (in seconds), given an allocated bandwidth β_{al} and a token size sz_d, is given by Equation (10). The edge d_b^{sr} with $(\alpha_d^{sr} - \xi(d))$ initial tokens models the available buffer space at the source processing element. In this work, both α_d^{sr} and α_d^{ds} are chosen to be equal to $\alpha_d/2$. Having presented the modeling of resource allocation in the resource-aware application graph, a bound application graph gb is formally defined in (Definition 4.19). The set of all bound application graphs is denoted by GB.

$$et_\rho(a_\rho) = \frac{sz_d/\beta_{al}}{fc(fi_{noc})} \tag{10}$$

Definition 4.19. (Bound application graph) A bound application graph gb is a 5-tuple $\langle sdfg, ga, gp, b, fc, \rangle$ consisting of an SDF graph $sdfg$ that models resource allocation when the resource-aware application graph ga is mapped to the platform graph gp, a binding vector b that specifies the processing element $pe \in PE$ to which each actor $a \in A$ is bound, and a chip-frequency vector $fc \in FC$ that specifies the clock-frequency of each voltage-frequency island $fi \in FI$.

4.6. Throughput computation and scheduling

Throughput is an important design metric in streaming applications. In the context of an SDF graph, it quantifies the rate at which output data tokens are produced. An example is frames per second for an H.263 decoder. In this work, the throughput of a bound application graph, denoted as $\tau(gb)$, is computed based on the timed execution of the graph, as proposed in [Ghamarian et al. 2006]. We use static-order scheduling to schedule the execution of actors on the same processing element. The schedules are constructed during the timed execution of the SDF graph, as demonstrated in [Stuijk et al. 2007]. Libraries offered by the publicly available SDF^3 tool are used to perform throughput computation based on timed execution of an SDF graph [Stuijk et al. 2006b].

5. BETTER THAN WORST-CASE DESIGN

Reducing the design margins or guard-bands when implementing a circuit provides the benefit of reduced circuit area, resulting in a larger number of gross dies on a wafer. This in turn may provide a larger number of good dies that satisfy the throughput requirement of the application. In this section, we present the methodology to use the variation-aware modeling framework introduced in Section 4 and estimate the number of good dies with reduced design margins (i.e., better than worst-case design).

5.1. Number of good dies

Circuit guard-banding is typically done by using corner-files during the design and verification stages; these files describe the worst-case and best-case delay values of standard-cells, corresponding to slow and fast process corners, respectively. The change in circuit area when reducing these guard-bands (i.e., implementing the circuit with reduced worst-case and increased best-case delay values) is assessed by [Jeong et al. 2009]. They use open-source cores and an industrial embedded processor core with target clock frequencies ranging from 300 to 600 MHz; the cores are synthesized using 90, 65 and 45 nm technology model libraries. Based on measured data, the authors provide a linear regression model for circuit-area reduction versus guard-band reduction. The provided model is $v = 1 - 0.0033 \cdot u$, where v is the area reduction factor and u is the guard-band reduction in percent.

The number of gross dies on a wafer corresponding to a $u\%$ guard-band reduction is given by Equation (11), where l is the radius of the wafer and $V_{u\%}$ is the die area corresponding to a $u\%$ guard-band reduction [Jeong et al. 2009]; the second term in the equation accounts for wasted area around the edges of the circular wafer. The die area $V_{u\%}$ is given by Equation (12), where n_{pe} and n_{cgu} are the number of processing elements and clock-generation units, respectively; V_{pe}, V_{noc} and V_{cgu} are the area of a processing element, the interconnect and the clock-generation unit, respectively; and $v_{sc} \in [0, 1]$ is the fraction of the scaled logic. When reducing guard-bands, either the whole die area is scaled (i.e., $v_{sc} = 1$ and thus $V = v \cdot (n_{pe} \cdot V_{pe} + V_{noc}) + n_{cgu} \cdot V_{cgu}$) or only a fraction of it scales (i.e., $v_{sc} = [0, 1)$) in case the area consisting of embedded SRAM and IO cells does not scale. Note that an assumption that the area of clock-generation units does not change due to guard-band reduction is made.

$$N_{u\%}^{\text{gross}} = \pi \cdot \left(\frac{l^2}{V_{u\%}} - \frac{2l}{\sqrt{2V_{u\%}}} \right) \tag{11}$$

$$V_{u\%} = ((v-1) \cdot v_{sc} + 1) \cdot (n_{pe} \cdot V_{pe} + V_{noc}) + n_{cgu} \cdot V_{cgu} \tag{12}$$

We are interested in the number of good dies that provide throughput at least equal to the minimum throughput requirement of an application. The number of good dies is given by the product of the number of gross dies and *application yield*, which is a system-level metric quantifying the percentage of manufactured chips that satisfy the throughput requirement imposed on an application. The application yield is computed using the variation-aware framework proposed in Section 4. For this, the throughput of the bound application graph $gb = \langle sdfg, ga, gp, b, fc, \rangle$ for each chip-frequency vector is computed. Whenever the throughput requirement is satisfied, the application yield is incremented by the probability of the chip-frequency vector. Note that for this computation, a binding for each chip-frequency vector is required. Thus, a set $B' \in B$ of different bindings for all chip-frequency vectors is required (this set is derived by a heuristic algorithm as later explained in Section 6.1). For a given set B' of bindings, the application yield over all chip-frequency vectors $fc \in FC$, where each chip-frequency vector has a probability weight $pc(gp, fc)$, is given by Definition 5.1. As expressed in the definition, the application yield is incremented by the probability $pc(gp, fc)$ if the throughput requirement is satisfied with at least one binding in the set B'.

Definition 5.1. (Application yield for a set of bindings) The percentage of chips, characterized by a bound application graph $gb = \langle sdfg, ga, gp, b, fc, \rangle$, that satisfy the throughput requirement t_{req} over all chip-frequency vectors $fc \in FC$, for a specified resource-aware application graph $ga \in GA$, platform graph $gp \in GP$ and a set B' of bindings from the set B of all possible bindings, is given by $\gamma(ga, gp, B') : GA \times GP \times \mathcal{P}(B) \setminus \emptyset \to \mathbb{R}^+$, and is defined as

$$\gamma(ga, gp, B') = \sum_{fc \in FC} \begin{cases} pc(gp, fc) & \text{if } \exists b \in B' \mid \tau(gb) \geq t_{req} \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

5.2. Variation characterization

Each guard-band value results in a particular circuit implementation with a certain area, after the design and verification stages. By performing statistical characterization (Monte Carlo simulations) on each circuit implementation, the PDFs of the maximum supported frequency of the hardware resources (processing elements, routers, network interfaces and links) in a platform graph is obtained. A statistical characterization flow is proposed in [Miranda et al. 2009]. The following is required for performing statistical characterization. 1) Standard-cell library models for guard-band reduction, 2) a hardware platform in RTL, 3) synthesis (place and route) and verification flows, and 4) a statistical characterization flow. Implementing this requires considerable time and specific expertise. Thus, we opt out of statistical characterization. Instead, we make the following intuitive assumptions. With the original guard-band (i.e., 0% guard-band reduction), we assume that about 99.85% of manufactured hardware resource instances satisfy the target frequency f^r_{tg}. This corresponds to a combined normal distribution of maximum supported frequency such that $(\mu^r_{0\%} - 3\sigma^r_{0\%} \cdot (\mu^r_{0\%}/100)) = f^r_{tg}$, where $\sigma^r_{0\%}$ is the standard deviation in percents of mean frequency. Therefore, the mean frequency is computed by $\mu^r_{0\%} = f^r_{tg}/(1 - 3\sigma^r_{0\%}/100)$. For a resource with $f^r_{tg} = 300$ MHz target frequency, $\sigma^r_g = 4\%$, $\sigma^r_l = 3.3\%$, and thus $\sigma^r_{0\%} = \sqrt{(\sigma^r_g)^2 + (\sigma^r_l)^2} \approx 5.186\%$, the mean frequency $\mu^r_{0\%}$ is equal to ≈ 355 MHz. The combined distribution $f^r = N(\mu^r_{0\%}, \sigma^r_{0\%})$ is shown in Figure 7. These numbers are based on the available data at 45nm technology [Pang et al. 2008].

With no guard-band (i.e., 100% guard-band reduction), we assume that half of manufactured hardware resource instances satisfy the target frequency f_{tg} considering only global variation. This corresponds to a global normal distribution of maximum supported frequency with f^r_{tg} mean. Therefore, the mean of the combined distribution is

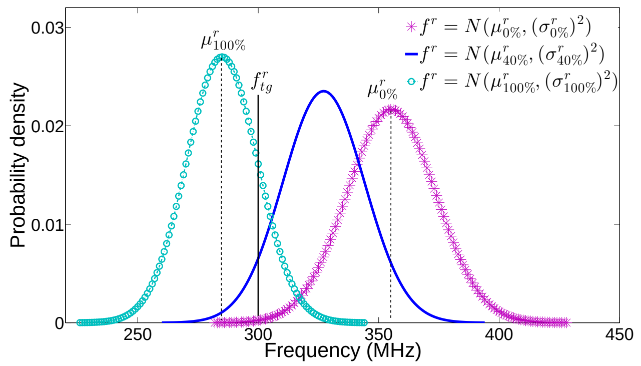


Fig. 7. Combined f^r PDF of a hardware resource due to a 0%, 40%, 100% guard-band reduction. The target frequency f^r_{tg} is 300 MHz, $\delta^r = 5\%$, $\sigma^r_g = 4\%$, $\sigma^r_l = 3.3\%$, and thus $\sigma^r_{u\%} \approx 5.186\%$ of mean frequency.

given by $\mu^r_{100\%} = (f^r_{tg} - \delta^r \cdot (f^r_{tg}/100))$, and thus $\mu^r_{100\%} = f^r_{tg} \cdot (1 - \delta^r/100)$, where δ^r is the reduction in mean frequency (in percents) due to local variation. We assume that the standard deviation to mean ratio $\sigma^r_{u\%}/\mu^r_{u\%}$ of the combined normal distribution for any $u\%$ guard-band reduction is constant. In reality, it may change due to local variation that depends on circuit implementation, which is different for each $u\%$ guard-band reduction (global variation does not depend on circuit implementation). However, only small differences in a close range of mean frequency are expected (e.g., the ratio difference for 300 MHz and 355 MHz mean frequency circuit implementations is much lower than for 300 MHz and 2 GHz circuit implementations). For the example described above, where $f_{tg} = 300$, $\sigma^r_g = 4\%$, $\sigma^r_l = 3.3\%$, and thus $\sigma^r_{100\%} \approx 5.186\%$, and assuming that $\delta^r = 5\%$, the distribution $f^r = N(\mu^r_{100\%}, \sigma^r_{100\%})$ is shown in Figure 7.

Given the values $\mu^r_{0\%}$ and $\mu^r_{100\%}$, a $u\%$ guard-band reduction results in a new combined normal distribution with a mean frequency $\mu^r_{u\%} = \mu^r_{0\%} - u \cdot (\mu^r_{0\%} - \mu^r_{100\%})/100$. Figure 7 shows the combined distribution $f^r = N(\mu^r_{40\%}, \sigma^r_{40\%})$ for a 40% guard-band reduction. For a given mean frequency $\mu^r_{u\%}$, the frequency μ^r_g corresponding to a guard-band reduction value can be computed by Equation (14), where δ^r is the mean reduction in percents of μ^r_g. It is assumed that the ratio δ^r/μ_g for any $u\%$ guard-band reduction is constant (only small differences in the local variation in a close range of mean frequency $\mu^r_{u\%}$ are expected as explained above).

$$\mu^r_g = \frac{\mu^r_{u\%}}{(1 - \frac{\delta^r}{100})} \tag{14}$$

6. EXPERIMENTAL RESULTS

On case studies, this section presents the improvements in the number of good dies achieved by the proposed better than worst-case design methodology. Note that the provided improvements are estimates. An experiment to confirm the numbers would be very expensive and impractical to conduct. Related to the confidence factor in the estimates, there are two aspects to be considered. The correctness (suitability) of the model of computation (MoC), i.e. application, hardware platform and resource allocation modeling, and the model of variation in hardware resources. As a MoC, we use a data-flow graph (more specifically an SDF graph). Data-flow graphs are shown to well capture the behavior of streaming applications and are used in modern multiprocessor systems, such as CoMPSoC [Goossens et al. 2013], CA-MPSoC [Shabbir et al. 2010], for modeling and analysis. Both global and local variations in hardware resources are modeled using a normal distribution, which is shown to be a good fit for capturing the impact of manufacturing process variations [Bowman et al. 2002]. Although we do not acquire the variation numbers, we use the available measured data for current technology nodes at specified frequencies. We assume no spatial correlation between hardware resources due to systematic local variation, but the impact of it is expected to be low, as explained in Section 4.2. Therefore, we expect similar improvement numbers in reality for such systems. We proceed to describing our experimental setup.

6.1. Experimental setup

The case studies in this section are based on a synthetic application and a set of six SDF graphs that model real DSP and multimedia applications. From the DSP domain, the set contains a Modem [Bhattacharyya et al. 1999] and from the multimedia domain an H.263 encoder [Oh et al. 2004], an H.263 decoder [Stuijk et al. 2008], an MP3 playback [Wiggers et al. 2007] and an MP3 decoder [Stuijk et al. 2008]. An overview of the SDF graphs is shown in Table Ia. The table reports the number of actors, the number of cycles (feedback loops) in each graph and the available parallelism in terms of the minimum number of processing elements that can fully exploit it. The reason we include the synthetic application is that it has a higher level of parallelism compared to the real applications. The topologies of the applications, together with the execution times (in cycles) of actors and the size of data tokens (in bytes) sent across dependency edges, are given in [Mirzoyan 2013]. These application SDF graphs are the resource-aware application graphs in our formal framework. The described real applications are allocated to a NoC-based MPSoC platform consisting of three homogeneous processing elements (most of the applications have a parallelism of three processing elements). The MPSoC platform, described by a platform graph in our modeling framework, is shown in Figure 3a. The three processing elements and the interconnect (NoC) are placed in separate voltage-frequency islands; as such, there are four islands. Note that we do not analyze the impact of VFI partitioning on the number of good dies due to limited space. This is evaluated in [Mirzoyan 2013]. The number of clock-frequency levels provided by clock-generation units (not shown in Figure 3a) to each of the four voltage-frequency islands is chosen to be five. The interconnect consists of two routers, three network interfaces and multiple links, which connect the processing elements to each other. The net bandwidth (i.e., the overall bandwidth capacity) of each connection in the interconnect is assumed to be $\beta_c = 4$ bytes per cycle. The flit size is set to $sz_{tb} = 12$ bytes, and a TDM table size of $sz_{fl} = 20$ slots for each network interface is assumed. Note that the net bandwidth may be affected by possible overheads, such as inserted headers. To account for this, we assume a worst-case scenario, where four out of twelve bytes in a flit is a header. Therefore, the net bandwidth used for transferring data becomes two-thirds of the original bandwidth. These numbers correspond to the Æthereal network [Goossens et al. 2010]. The synthetic application has a parallelism that can be fully exploited by a minimum number of seven processing elements. Thus, this application is mapped to a multiprocessor platform consisting of seven homogeneous processing elements.

Table Ib illustrates the target frequency and the variation-related parameters for the random variables describing the maximum supported frequency of hardware resources in the multiprocessor platforms. The target frequency for each processing element is 300 MHz. For routers and network interfaces, a 500 MHz target frequency is chosen. The choice of the target frequency for links, compared to the target frequency for routers, is made according to a NoC implementation given in [Hernandez et al. 2012], where the authors design a link slightly faster than the routers not to limit the performance of the NoC and to preserve power. We select a target frequency of 560

MHz for links. Note that the target frequencies for the routers, network interfaces and links are higher than that of the processing elements. In a high performance 80-core TeraFLOPS processor, both the routers and cores operate at equal 4 GHz frequencies [Vangal et al. 2008]. However, we choose a faster interconnect to avoid any bottleneck in the communication. The mean frequencies μ_g^r of the random variables describing the maximum supported frequency of hardware resources due to global variation are computed based on a guard-band reduction value (Equation (14)). 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 at 45 nm technology in the frequency range of interest are provided in [Pang et al. 2008]. Based on this data, we assume standard deviation to mean ratios of 4% and 3.3% for within-die and die-to-die variations, respectively (Table Ib). We assume a 5% reduction in mean frequency for processing elements and links due to multiple critical paths and wires, respectively. For routers and network interfaces, no reduction in mean frequency is assumed. These choices correspond to the discussion presented in Section 4.2.

Table I. Application SDF graph overview (a). Target frequency and variation-related parameters assumed for hardware resources (b).

	Nr. actors	Nr. cycles	Parallelism (Nr. PEs)
Synthetic	17	3	7
H.263 decoder	4	0	3
H.263 encoder	5	1	3
MP3 playback	4	1	2
Modem	16	5	3
MP3 decoder	14	0	3

(a)

	pe	*rt*	*ni*	*lk*
f_{tg}^r (MHz)	300	500	500	560
σ_g^r (% of μ_g^r)	4	4	4	4
δ^r (% of μ_g^r)	5	0	0	5
σ_l^r (% of μ_g^r)	3.3	3.3	3.3	3.3

(b)

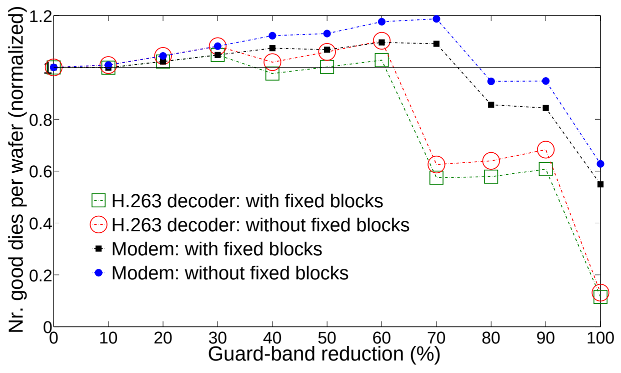
The area of a processing element is assumed to be 0.7 mm^2, based on the area of an ARM Cortex-A5 processor at 45 nm technology. The area of the interconnect is 3.1 mm^2. We assume that a typical fine-grained clock-generation unit has area of 0.03 mm^2 [Rylyakov et al. 2008]. We neglect the area of clock-generation units with 0% guard-band reduction, as a simple clock-generation unit is required for providing a single (target) clock frequency for each voltage-frequency island. We assume that 70% of the area of processing elements and the interconnect consists of standard logic cells, and 30% of embedded SRAM and IO cells. Two scenarios are distinguished: (1) guard-band reduction results in an overall decrease in die area (i.e., $v_{sc} = 1$), (2) only the area of logic cells is reduced when reducing guard-bands (i.e., $v_{sc} = 0.7$) (Equation (12)).

The throughput requirement for each application is decided such that it is just satisfied with full guard-bands (i.e., the target clock frequencies of the hardware resources are guaranteed in manufactured chips). This means that the hardware platform is dimensioned just enough for the particular application with full guard-bands. Selecting a more relaxed throughput requirement would lead to having performance slack. Thus, guard-band reduction would still result in sufficiently fast dies for the given throughput requirement (i.e., a large number of good dies). This would not fairly demonstrate the benefits of reducing guard-bands. To determine the throughput requirement, each application is mapped to the platform with the specified target frequencies such that the throughput is maximized. The heuristic mapping algorithm presented in [Mirzoyan et al. 2014] is used for finding a mapping with maximized throughput. The result of the mapping is a throughput value, which is taken as the requirement for each application. The application yield corresponding to a $u\%$ guard-band reduction is evaluated by Definition 5.1. The set of bindings provided for estimating the application yield is determined using the variation-aware heuristic mapping algorithm given by [Mirzoyan et al. 2014]. There are two phases in the heuristic mapping algorithm, initial resource allocation and allocation optimization. In the initial resource allocation, an initial binding of application actors to processing elements is derived such that the execution load on the processing elements is balanced. The actors whose execution times are likely to have a large impact on the throughput are considered first. This initial binding later undergoes an optimization stage where the allocation of each actor is reconsidered to improve the throughput. The mapping stages are performed for each chip-frequency vector, resulting in a set of bindings for all chips. More details on the algorithm can be found in [Mirzoyan et al. 2014].

6.2. Evaluation results

Figure 8a illustrates the change in the number of good dies per wafer against guard-band reduction for the H.263 decoder and Modem applications. Normalized values against the number of good dies with 0% guard-band reduction are used. The graphs are presented for two different assumptions: a design with and without fixed blocks. The design with fixed blocks refers to having IPs that have predefined layouts (SRAM, IO, analog cells), the area of which does not change with guard-band reduction. The design without fixed blocks refers to not having IPs with predefined layouts or designing IPs specifically for each guard-band value. The application yield corresponding to different $u\%$ guard-band reduction values is shown in Figure 8b (application yield computation times for the different applications can be found in [Mirzoyan 2013]). Figure 8a shows that for the H.263 decoder application and the design with fixed blocks, the number of good dies is maximized at a 30% guard-band reduction. This results in a 4.8% more dies that satisfy the throughput requirement imposed on the application. When there are no fixed blocks, the increase in the number of good dies is 10.3% at a 60% guard-band reduction. For the Modem application, significant improvements of 9.6% (with fixed blocks) and 18.8% (without fixed blocks) in the number of good dies are illustrated at a 70% guard-band reduction. This is due to the relatively high application yield provided at a 70% reduction in guard-bands, as shown in Figure 8b. Note that a 9.6% increase in the number of good dies is significant. For example, if 4 K wafers are required to produce 30 M good dies, a 9.6% larger number of good dies per wafer translates into 350 fewer wafers for the same 30 M good dies. For a wafer cost of €3000, the cost saving is \$ 1,050,000. For each application, the number of good dies gradually decreases after a certain guard-band value (e.g., beyond 70% for the Modem application). This is because the application yield becomes considerably lower and the gain in the number of gross dies due to smaller die area is not large enough to offset the loss in the application yield, resulting in less good dies, which is the product of the number of gross dies and application yield. For the Modem application at 80% guard-band reduction, the application yield is 71%, while the increase factor in number of gross dies for without fixed blocks is 1.32 (Figure 8b). The product of these two leads to a lower number of good dies. The number of good dies for each guard-band value can be computed using the data in Figure 8b.

Table II illustrates the increase in the number of good dies per wafer and the associated reduction in guard-bands for the rest of the applications considering designs with and without fixed blocks. The results for the H.263 encoder and MP3 decoder applications are identical. A 30% reduction in guard-bands results in 3.7% and 7% more good dies for designs with and without fixed blocks, respectively. For the MP3 playback application and considering the design with fixed blocks, the number of good



(a)

u%	y% (I)	y% (II)	n_g	n'_g
0	99	99	1.00	1.00
10	99	99	1.00	1.01
20	99	99	1.02	1.04
30	99	99	1.05	1.08
40	90	99	1.07	1.12
50	90	96	1.10	1.17
60	90	96	1.13	1.21
70	49	93	1.16	1.26
80	49	71	1.19	1.32
90	49	68	1.23	1.38
100	9	43	1.26	1.45

(b)

Fig. 8. (a) Number of good dies per wafer against guard-band reduction for the H.263 decoder and Modem applications. Designs with and without fixed blocks are considered. (b) Application yield for a u% reduction in guard-bands for the (I) H.263 decoder and (II) Modem applications. The increase factor in the number of gross dies for with and without fixed blocks is given by n_g and n'_g, respectively.

dies is maximized at a 30% reduction in guard-bands, resulting in a 4.8% more good dies. For a design without fixed blocks, a 60% guard-band reduction provides a 11.5% more good dies. Finally, considerable improvements of 6.1% and 12.3% are seen for the synthetic application. These results show that a higher number of good dies with reduced guard-bands is obtained, increasing profit. Note that these results are given for an architecture where five clock-frequency levels are provided to each of the voltage-frequency islands. It has been shown in [Mirzoyan et al. 2014] that a larger number of clock-frequency levels provided to islands is more likely to result in a higher application yield. Therefore, increasing the number of clock-frequency levels that provided to each island is more likely to result in an increased number of good dies.

Table II. i% Increase in the number of good dies for a u% reduction in guard-bands (I) with and (II) without fixed blocks for the rest of the applications.

		H.263 encoder	MP3 playback	MP3 decoder	Synthetic
(I)	i% (u%)	3.7 (30)	4.8 (30)	3.7 (30)	6.1 (50)
(II)	i% (u%)	7 (30)	11.5 (60)	7 (30)	12.3 (50)

7. CONCLUSIONS AND FUTURE WORK

This work addresses the problem of designing real-time streaming applications constrained by a throughput requirement on a NoC-based multiprocessor system under better than worst-case design (i.e., with reduced design margins). With better than worst-case design, circuit area is reduced, resulting in more dies on a wafer. However, the target maximum supported frequency of hardware components in a multiprocessor system is not guaranteed anymore. The first contribution of the work is a complete formal framework to estimate the probability distribution of application throughput in a NoC-based multiprocessor system with voltage-frequency islands under the impact of die-to-die and within-die process variations. The second contribution is a methodology to use the proposed framework and evaluate the impact of reducing design margins on the number of good dies that satisfy the throughput requirement of a real-time streaming application. On both synthetic and real applications, we show that the proposed better than worst-case design methodology can increase the number of good dies by up to 18.8% compared to worst-case design. Assuming that 4 K wafers are required to produce 30 M good dies, a 18.8% larger number of good dies per wafer translates into 632 fewer wafers for the same 30 M good dies. For a wafer cost of $3000, the cost saving is $1,896,000 (i.e., $6,32 cents per die). While these results show clear benefits over the worst-case design approach, the disadvantage of the proposed better-than worst-case design is the longer design time due to iterative application yield estimation. Note that the improvement in the number of good dies depends on the application yield against guard-band reduction curve. The slower the application yield reduces, the higher the benefits are. This curve will differ from application to application. It depends on the topology of the application graph, its properties, as well as the hardware platform (e.g., the number of processing elements). For the synthetic and real applications, and different number of processing elements in the hardware platform, we experimentally proved that the number of good dies is maximized (i.e., the application yield reduces slower than die area).

In the proposed modeling framework, a single application is considered. In practice, multiple applications are mapped to a multi-processor platform. Possible future work is to extend the proposed framework such that multiple applications are considered. For example, this can be achieved by TDM virtualization, where each application is given a portion of a multi-processor platform such that there is no temporal interference between applications. Additionally, the models presented in Section 4.2 can be extended such that correlation between hardware components due to systematic within-die variation can be specified. This will provide a modeling framework that more accurately captures the physical phenomenon of process-induced variation.

REFERENCES

BAMAKHRAMA, M. A. ET AL. 2012. A methodology for automated design of hard-real-time embedded streaming systems. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*.

BHATTACHARYYA, S. S. ET AL. 1999. Synthesis of embedded software from synchronous dataflow specifications. *Journal of VLSI Signal Processing Systems (IJVSPA) 21*.

BOWMAN, K. ET AL. 2002. Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration. *Journal of Solid-State Circuits (JSSC) 37,* 2.

DALLY, W. ET AL. 2001. Route packets, not wires: on-chip interconnection networks. In *Proc. Design Automation Conference (DAC)*.

DIGHE, S. ET AL. 2011. Within-die variation-aware dynamic-voltage-frequency-scaling with optimal core allocation and thread hopping for the 80-core TeraFLOPS processor. *Journal of Solid-State Circuits (JSSC) 46,* 1.

FRIEDBERG, P. ET AL. 2005. Modeling within-die spatial correlation effects for process-design co-optimization. In *Proc. Quality of Electronic Design (ISQED)*.

GARG, S. ET AL. 2008. System-level throughput analysis for process variation aware multiple voltage-frequency island designs. *Transactions on Design Automation of Electronic Systems (TODAES) 13,* 4.

GHAMARIAN, A. ET AL. 2006. Throughput analysis of synchronous data flow graphs. In *Proc. Int'l Conference on Application of Concurrency to System Design (ACSD)*.

GOOSSENS, K. ET AL. 2010. The Æthereal network on chip after ten years: Goals, evolution, lessons, and future. In *Proc. Design Automation Conference (DAC)*.

GOOSSENS, K. ET AL. 2013. Virtual execution platforms for mixed-time-criticality systems: The CompSOC architecture and design flow. *Special Interest Group on Embedded Systems (SIGBED) Review 10,* 3.

HANSSON, A. ET AL. 2009. Enabling application-level performance guarantees in network-based systems on chip by applying dataflow analysis. *IET Computers & Digital 3,* 5.

HERNANDEZ, C. ET AL. 2012. On the impact of within-die process variation in GALS-based NoC performance. *Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) 31,* 2.

HUANG, L. ET AL. 2010. Performance yield-driven task allocation and scheduling for MPSoCs under process variation. In *Proc. Design Automation Conference (DAC)*.

JEONG, K. ET AL. 2009. Impact of guardband reduction on design outcomes: A quantitative approach. *Transactions on Semiconductor Manufacturing (SM) 22,* 4.

LEE, E. ET AL. 1987. Synchronous data flow. *Proceedings of the IEEE 75,* 9.

MARCULESCU, D. ET AL. 2008. Process-driven variability analysis of single and multiple voltage frequency island latency-constrained systems. *Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) 27,* 5.

MEIJER, M. ET AL. 2012. Body-bias-driven design strategy for area- and performance-efficient CMOS circuits. *Transactions on Very Large Scale Integration (VLSI) Systems 20,* 1.

MEINCKE, T. ET AL. 1999. Globally asynchronous locally synchronous architecture for large high-performance ASICs. In *Proc. Int'l Symposium on Circuits and Systems (ISCAS)*. Vol. 2.

MILLBERG, M. ET AL. 2004. The Nostrum backbone - a communication protocol stack for networks on chip. In *Proceedings of the International Conference on VLSI Design (VLSID)*.

MIRANDA, M. ET AL. 2009. Variability aware modeling of SoCs: From device variations to manufactured system yield. In *Proc. Quality of Electronic Design (ISQED)*.

MIRZOYAN, D. 2013. Better than worst-case design for streaming application under process variation. Ph.D. thesis, EEMCS Department, Delft University of Technology.

MIRZOYAN, D. ET AL. 2013. Throughput analysis and voltage-frequency island partitioning for streaming applications under process variation. In *Proc. Embedded Systems for Real-Time Multimedia (ESTIMedia)*.

MIRZOYAN, D. ET AL. 2014. Process-variation-aware mapping of best-effort and real-time streaming applications to MPSoCs. *Transactions in Embedded Computing Systems (TECS) 13,* 2s.

MUTTERSBACH, J. ET AL. 2000. Practical design of globally-asynchronous locally-synchronous systems. In *Proc. Int'l Symposium on Asynchronous Circuits and Systems (ASYNC)*.

OH, H. ET AL. 2004. Fractional rate dataflow model for efficient code synthesis. *Journal of VLSI Signal Processing Systems 37,* 1.

PANG, L.-T. ET AL. 2008. Measurement and analysis of variability in 45nm strained-Si CMOS technology. In *Custom Integrated Circuits Conference (CICC)*.

PANG, L.-T. ET AL. 2009. Measurement and analysis of variability in 45 nm strained-Si CMOS technology. *Journal of Solid-State Circuits 44,* 8.

POPLAVKO, P. ET AL. 2003. Task-level timing models for guaranteed performance in multiprocessor networks-on-chip. In *Proc. Int'l conference on Compilers, architecture and synthesis for embedded systems (CASES)*.

RYLYAKOV, A. ET AL. 2008. A wide tuning range (1 GHz-to-15 GHz) fractional-N all-digital PLL in 45nm SOI. In *Custom Integrated Circuits Conference (CICC)*.

SHABBIR, A. ET AL. 2010. CA-MPSoC: An automated design flow for predictable multi-processor architectures for multiple applications. *Journal of Systems Architecture (JSA) 56,* 7.

SRIRAM, S. ET AL. 2000. *Embedded Multiprocessors: Scheduling and Synchronization* 1st Ed.

STILIADIS, D. ET AL. 1998. Latency-rate servers: a general model for analysis of traffic scheduling algorithms. *Transactions on Networking (TON) 6,* 5.

STUIJK, S. ET AL. 2006a. Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs. In *Proc. Design Automation Conference (DAC)*.

STUIJK, S. ET AL. 2006b. SDF^3: SDF For Free. In *Proc. Int'l Conference on Application of Concurrency to System Design (ACSD)*.

STUIJK, S. ET AL. 2007. Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs. In *Proc. Design Automation Conference (DAC)*.

STUIJK, S. ET AL. 2008. Throughput-buffering trade-off exploration for cyclo-static and synchronous dataflow graphs. *Transactions on Computers (TC) 57,* 10.

VAN BERKEL, C. H. K. 2009. Multi-core for mobile phones. In *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*.

VANGAL, S. ET AL. 2008. An 80-tile sub-100-w TeraFLOPS processor in 65-nm CMOS. *Journal of Solid-State Circuits (JSSC) 43,* 1.

WASSEL, H. M. G. ET AL. 2013. SurfNoC: a low latency and provably non-interfering approach to secure networks-on-chip. *SIGARCH Computer Architecture News 41,* 3.

WIGGERS, M. H. ET AL. 2007. Efficient computation of buffer capacities for cyclo-static dataflow graphs. In *Proc. Design Automation Conference (DAC)*.

WILHELM, R. ET AL. 2008. The worst-case execution-time problem overview of methods and survey of tools. *Transactions on Embedded Compuing Systems (TECS) 7,* 3.