IET Computers & Digital Techniques

Special Issue: Designing with Uncertainty - Opportunities & Challenges

IET Journals
The Institution of Engineering and Technology

Yield-driven design-time task scheduling techniques for multi-processor system on chips under process variation: a comparative study

ISSN 1751-8601
Received on 8th June 2014
Accepted on 16th December 2014
doi: 10.1049/iet-cdt.2014.0126
www.ietdl.org

Mahmoud Momtazpour[1] ✉, Omid Assare[2], Negar Rahmati[3], Amirali Boroumand[4], Saeid Barati[5], Maziar Goudarzi[6]

[1]*Electrical Engineering, Sharif University of Technology, Tehran, Iran*
[2]*University of California, San Diego, San Diego, California, USA*
[3]*Stanford University, Santa Clara, California, USA*
[4]*Sharif University of Technology, Tehran, Iran*
[5]*University of Chicago, Chicago, Illinois, USA*
[6]*Department of Computer Engineering, Sharif University of Technology, Tehran, Iran*
✉ *E-mail: momtazpour@gmail.com*

Abstract: Process variation has already emerged as a major concern in design of multi-processor system on chips (MPSoC). In recent years, there have been several attempts to bring variability awareness into the task scheduling process of embedded MPSoCs to improve performance yield. This study attempts to provide a comparative study of the current variation-aware design-time task and communication scheduling techniques that target embedded MPSoCs. To this end, the authors first use a sign-off variability modelling framework to accurately estimate the frequency distribution of MPSoC components. The task scheduling methods are then compared in terms of both the quality of the final solution and the computational complexity of the scheduling algorithm. Experimental results on a wide range of benchmarks show that ILP-based task scheduling technique, while guaranteeing the optimality of the solution, can be costly for large application task graphs. On the other hand, one-pass heuristic method is 795 times faster than ILP-based method on average, but is ineffective to find reasonable solutions in the case of large task graphs. Finally, metaheuristic approaches can produce near-optimal schedules within 1–2% of the optimal solutions on average, with up to 7.8 times faster execution time compared with ILP-based approach.

1 Introduction

The ever-increasing process variability in CMOS technology has made it inevitable to incorporate variability analysis into system-level design techniques when dealing with today's multi-processor system on chips (MPSoC). The variation in key process parameters such as dopant concentration, channel length and oxide thickness of transistors directly affect total performance and power consumption of each core of a system-on-chip, rendering them as essentially random variables. Traditional worst-case corner-based approach has proved too pessimistic as transistor size shrinks, making it essential to move towards statistical analysis to obtain detailed and accurate information on the behaviour of system components.

In recent years, there have been several attempts to adopt variability analysis in design-time task scheduling for embedded MPSoCs. Wang *et al.* [1] was the first to address this problem by introducing a constructive (one-pass) heuristic static task scheduling method for performance yield maximisation. The proposed technique starts from the root node of the task graph and selectively assigns tasks to MPSoC cores until all of the tasks are scheduled. In each step, the selection of the task to be scheduled is based on a metric called dynamic priority (DP) of tasks. They later extended their work in [2] by adding an NoC-based communication backbone in the variability-aware mapping scheme. As another work, Singhal *et al.* [3] introduced a dynamic programming-based variation-aware task scheduling and configuration selection algorithm, that is limited to applications with only series or parallel task graphs. Recently, Mirzoyan *et al.* [4] developed some heuristic variation-aware task scheduling methods that are limited to real-time streaming applications with synchronous data flow graph (SDFG) model.

There have been several attempts to incorporate metaheuristic approaches such as genetic algorithm (GA) or simulated annealing (SA) to solve the variation-aware task scheduling problem. For example in [5], Chon *et al.* presented a GA-based task scheduling method to maximise performance yield while taking resource sharing into account. Recently, Huang *et al.* [6] presented a quasi-static SA-based task scheduling method that is followed by a clustering-based performance yield enhancement procedure. In our previous work [7, 8], we presented GA-based and SA-based variation-aware task and communication scheduling algorithms for power yield maximisation under performance yield constraint.

Integer linear programming (ILP) is also used to find the optimal solution of the variation-aware task scheduling problem. Singhal *et al.* [9] recently developed an ILP-based variation-aware task replication and load balancing technique for pipelined configurable MPSoC. Bhardwaj *et al.* [10] also introduced a mixed ILP-based task scheduling technique for power-performance yield optimisation for MPSoCs. Finally, Ghorbani [11] proposed an ILP-based variation-aware task scheduling technique for power yield optimisation under performance constraints.

Table 1 summarises the aforementioned work and classifies them based on some of their characteristics. As shown in this table, most of the previous work either use a simplistic model of process variation, target specific types of application task graph or does not consider the timing overhead of the underlying on-chip interconnection network (communication scheduling). The main objective of this paper is to provide a comparative study of the current variation-aware design-time task and communication

Table 1 Classification of the most recent design-time variation-aware task scheduling techniques

Authors	Optimisation goal	Application model	Process variation model	Communication scheduling
Heuristic Algorithms				
Wang [2]	performance yield	wDAG[1]	Gaussian frequency distribution	✓
Singhal [3]	performance yield	SPDAG[2]	NA, timing yields of cores are given	×
Mirzoyan [4]	performance yield	SDFG[3]	Discretised Gaussian frequency distribution	×
Metaheuristic-based Algorithms				
Chon [5]	performance yield	DAG	Discretised Gaussian frequency distribution	×
Huang [6]	performance yield	DAG	VARIUS [12], An approximative model to propagate variability in L_{eff} and V_{th} to the frequency of cores	×
Momtazpour [7]	power yield, performance yield	wDAG	Gaussian frequency distribution	✓
Momtazpour [8]	power yield, performance yield	wDAG	an approximative model to propagate variability in L_{eff}, V_{th} and T_{ox} to the frequency of cores	✓
ILP-based Algorithms				
Singhal [9]	throughput, performance yield	DAG	NA, timing yields of cores are given	×
Bhardwaj [10]	power-performance yield	DAG	Gaussian frequency distribution	×
Ghorbani [11]	power yield, performance yield	wDAG	Gaussian frequency distribution	✓

1 Weighted directed acyclic graph.
2 Serial-parallel directed acyclic graph.
3 Synchronous data flow graph.

scheduling techniques that target embedded MPSoCs, by using a more realistic model of process variation, and under a generalised model of application task graph. To this end, and as the main contributions of this paper:

(i) We used our previously developed sign-off variability modelling framework [13] to accurately estimate the frequency distribution of MPSoC components. This enabled us to have a more realistic model of process variation in our analysis.
(ii) We also proposed a DP -based heuristic task and communication scheduling method (called modified heuristic task scheduling [MHTS]) which can provide a good compromise between the quality of solutions and execution time.

We also applied a few modifications on the previously proposed task scheduling methods to provide fair comparison. The task scheduling methods are then compared with respect to both the quality of the final solution and the computational complexity of the scheduling algorithm. Experimental results on a wide range of benchmarks show that ILP-based task scheduling technique, while guaranteeing the optimality of the solution, can be costly for large application task graphs. On the other hand, one-pass heuristic method is 795 times faster than ILP-based method on average, but is ineffective to find reasonable solutions in the case of large task graphs. Moreover, our developed heuristic method shows a good compromise between the quality of solutions (within 2.5% of the optimal solution) and execution time (25.3 times faster than the ILP-based method). Finally, metaheuristic approaches can produce near-optimal schedules within 1–2% of the optimal solutions on average, with up to 7.8 times faster execution time on average.

The rest of the paper is organised as follows. Section 2 describes our process variation model. In Section 3, we formally define and formulate the MPSoC task scheduling problem. In Section 4, we briefly discuss various heuristic, metaheuristic and ILP-based task and communication scheduling approaches that are later used in the comparative study. The computational complexity of the mentioned approaches are then discussed in Section 5. Section 6 provides experimental results of evaluating and comparing the mentioned task and communication scheduling algorithms. The conclusion is then followed in Section 7.

2 Process variation model

As stated in the previous section, most of the variation models used previously at MPSoC-level either assume simple Gaussian distribution for frequency of cores [2, 4, 5, 7, 10, 11] or use approximative models to propagate the variations in transistor-level parameters to system-level components [6, 8]. For example, in VARIUS [12], the variation model assumes uniform distribution of identical critical paths across die, where each of the critical paths contains similar number of identical FO4 Inverter. Assuming normal distribution for within-die (WID) and die-to-die (D2D) variations in L_{eff} and V_{th}, it then propagates these variations to the delay of each gate, delay of each critical path and finally the frequency of the processing core. However, with the ever increasing amount of process variation, and the growing complexity of MPSoCs, we believe that the above assumptions may not be satisfactorily accurate for current and future technologies. Therefore to have an accurate and fair evaluation of the current variation-aware system-level techniques, more sophisticated and realistic models of process variation should be incorporated in the design process.

In this paper, we used our developed variability modelling framework *VAREX*, which was previously introduced and evaluated in [13, 14], to estimate the frequency distribution of MPSoC components. VAREX tries to minimise the aforementioned simplifying assumptions to produce the most accurate estimations while being fast enough to be included in system level VLSI CAD tools. This section tries to provide a brief overview of VAREX for being self-contained, yet the readers can refer to [13, 14] for more details.

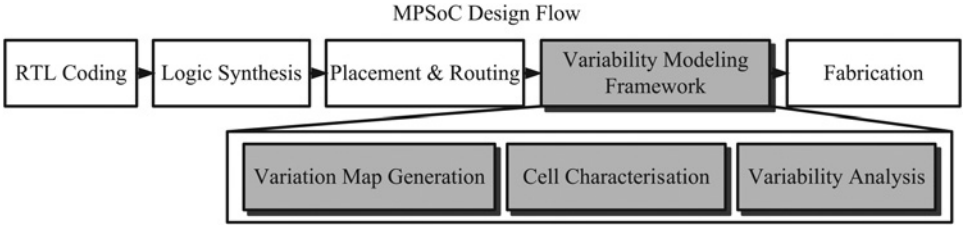

Fig. 1 *Overview of VAREX variability modelling framework [13, 14]*

Fig. 1 shows an overview of VAREX variability modelling framework. It has three components which are added to the traditional design steps: variation map generation, cell characterisation and variability analysis. We briefly review these steps in the following section. For more details, see [13, 14].

2.1 Variation map generation

This step is responsible for simulating the variability in the manufactured chips. Similar to [12, 15, 16], the random and systematic components of both D2D and WID variations are modelled as zero-mean normal distributions. Therefore each process parameter (V_{th} and L_{eff}) is modelled as a random variable with four Gaussian independent components that correspond to systematic D2D variation, random D2D variation, systematic WID variation and random WID variation. As in [17], we assume that each of the mentioned components contribute equally to the total variation. We use R statistical analysis package [18] to generate 10 000 different sets of WID and D2D variation maps for each of the process parameters while taking spatial correlation of the effective channel length into account. A variation map reflects the amount of variation in a process parameter across the surface of the die. Similar to [12], we use the spherical function (1) to incorporate spatial correlation of the effective channel length

$$\rho(n) = \begin{cases} 1 - \dfrac{3r}{2\Phi} + \dfrac{r^3}{2\Phi^3} & r \le \Phi \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Here, $\rho(r)$ is the correlation function, r is the distance between two points on the chip and Φ is the range, the distance between two points where they will be no longer correlated.

2.2 Cell characterisation

The goal of this step is to accurately model the effect of process variation on the delay of each type of cell. As discussed in [12], with a pin transition defined as its passing of $V/2$, and measuring the propagation delay as the mean of the elapsed times between all low-to-high and high-to-low transitions on the gate inputs that would toggle the gate output and their corresponding output transition, the propagation delay of sub-micron cells is proportional to

$$T_g \propto \frac{L_{eff} \cdot V}{\mu(V - V_{th})^{\alpha}} \quad (2)$$

where α is typically 1.3 and μ is the mobility of carriers. We inspect the effect of V_{th} and L_{eff} variations on the delay of each cell of the standard cell library using Monte-Carlo-based HSPICE simulations. Then, we find the best mathematical representation of the delay of each cell as a function of V_{th} and L_{eff} by fitting the gathered data to (2). We then use these results in the variability analysis stage to produce final frequency distributions.

2.3 Variability analysis

We finally use the following procedure to estimate the frequency distribution of the MPSoC components. First, using a static timing analyser, the path information including the number, type, delay and location of the cells in all of the paths of the design are extracted from the final post-P&R layout of the MPSoC. Then, for each variation map (that simulates a manufactured chip), and based on the delay model generated in the cell characterisation step, the variation-induced delay of each path is calculated and the frequency of the MPSoC is determined. Repeating this calculation for a large number of variation maps will provide us with an accurate estimate of the frequency distribution of each core.

Table 2 Symbols and definitions

System Parameters:	
n_s	Number of scenarios in Monte–Carlo simulation
n_t	Number of tasks in the task graph
n_e	Number of edges in the task graph
n_p	Number of processors in the MPSoC
$ct_{i,p}$	Execution cycle count of the ith task on the pth processor
ce_j	Transfer cycle count of the jth edge on the shared bus
System Constraints:	
$\lceil T \rceil$	Timing deadline of the task graph
Objective:	
Y_T	Timing yield: Percentage of chips that meet (are able to execute the whole task graph not later than) $\lceil T \rceil$ for a given task assignment and schedule

3 Problem formulation

In this paper, we use timing yield (Y_T), which was first introduced in [1], as our optimisation goal. It is defined as the percentage of manufactured chips that meet the timing constraint, $\lceil T \rceil$, for a given task schedule, as shown in (3)

$$Y_T = \text{Prob}(T < \lceil T \rceil) \quad (3)$$

Using the notation in Table 2, we formally define the task and communication scheduling problem as follows:

Problem: Assume that the target application is modelled as a weighted directed acyclic graph (wDAG) $\boldsymbol{G}$ with n_t tasks and n_e communication edges. Moreover, assume that the target MPSoC has n_p processor core and a shared bus, with given execution cycle count $ct_{i,\,p}$ of each task on each processor ($1 \le i \le n_t$, $1 \le p \le n_p$) and transfer cycle count ce_j of each edge on the shared bus ($1 \le j \le n_e$). The problem is to find a proper schedule/assignment of tasks and communication edges on MPSoC components, such that the timing yield (Y_T) is maximised.

This problem can actually be viewed as a statistical counterpart of a deterministic MPSoC task scheduling problem.

4 Variation-aware MPSoC task and communication scheduling

For our comparative study, we use five different approaches to solve the variation-aware MPSoC task and communication scheduling problem: (i) An ILP-based task scheduling method, previously proposed in [11] to find the optimal solution, (ii) a fast heuristic task scheduling technique, previously used in [2] to find a reasonable local solution efficiently, (iii) another heuristic method that is proposed in this work and is inspired from the work in [2], (iv) a GA-based task scheduling technique [7] and (v) an SA-based task scheduling technique [8]. This section provides a brief overview of the mentioned approaches, as well as minor modifications applied to them to provide fair comparison.

4.1 ILP-based task and communication scheduling

ILP is used to find the optimal allocation and scheduling of tasks and communication edges on MPSoC resources in [9–11]. To the best of our knowledge, the work in [11] addressed the most general form of this problem by using wDAGs as the target application model and providing communication scheduling along with the task scheduling process. However, the mentioned work used a simplistic model of process variation where the frequency of cores follow Gaussian distribution. Moreover, the objective was to maximise power yield while taking performance yield at the desired level. To provide fair comparison, we modified the proposed ILP formulation to address performance yield maximisation. The following parameters, variables and constraints are used to formulate the ILP-based task scheduling method:

4.1.1 Parameters: The *a priori* information of the given application task graph and MPSoC architecture is modelled using the following constant parameters:

n_p number of processors in the MPSoC architecture;
n_t number of tasks in the application task graph;
n_e number of communication edges in the application task graph;
$\boldsymbol{G}$ an $n_t \times n_t$ adjacency matrix representing the graph structure;
$\boldsymbol{C}_t$ an $n_t \times n_p$ matrix denoting the execution cycle count of tasks on MPSoC processors;
$\boldsymbol{C}_e$ a $1 \times n_e$ vector denoting the transfer cycle count of communication edges on MPSoC shared bus;
$\boldsymbol{D}_p$ an $n_p \times n_s$ matrix denoting the clock period of MPSoC processors in all n_s test chips;
$\boldsymbol{D}_c$ a $1 \times n_s$ vector denoting the clock period of MPSoC shared bus in all n_s test chips;
K a large enough constant.

4.1.2 Variables: The following variables are used to model the ILP-based task and communication scheduling problem:

$\boldsymbol{S}_t$ An $n_t \times n_s$ real matrix that represents the start time of tasks in all n_s test chips.
$\boldsymbol{S}_e$ An $n_e \times n_s$ real matrix that represents the start time of communication edges in all n_s test chips.
$\boldsymbol{X}_t$ An $n_t \times n_p$ boolean matrix that stores the task assignment variables; the entry $\boldsymbol{X}_t(i, p)$ is true if the ith task is allocated on the pth processor.
$\boldsymbol{X}_e$ A $1 \times n_e$ boolean vector that stores the task assignment variables; the entry $X_e(j)$ is true if the jth edge is scheduled on the shared bus.
$\boldsymbol{O}_t$ An $n_t \times n_t$ boolean matrix that stores the task order variables; the entry $\boldsymbol{O}_t(i, i')$ is true if task i is scheduled to run before task i', and both are allocated on the same processor.
$\boldsymbol{O}_e$ An $n_e \times n_e$ boolean matrix that stores the communication order variables; the entry $\boldsymbol{O}_e(j, j')$ is true if communication edge j is scheduled to run before communication edge j', and if both are allocated on the same communication route.
$\boldsymbol{M}$ A $1 \times n_s$ boolean vector that represents timing constraint satisfaction variables; the entry $\boldsymbol{M}(k)$ is true if the system meets timing constraint in the kth test chip.

4.1.3 Constraints: The constraints are formulated as follows:

c_1: A task must be assigned to only one processor

$$\sum_{p=1}^{n_p} \boldsymbol{X}_t(i, p) = 1 \tag{4}$$

where $i \in [1, 2, \ldots, n_t]$.
c_2: The order of execution of any two tasks that run on the same processor must be unique

$$\boldsymbol{O}_t(i, i') + \boldsymbol{O}_t(i', i) = 1 \tag{5}$$

where $i, i' \in [1, 2, \ldots, n_t]$ and $i < i'$.
c_3: The order of execution of any two data transfers that run on the shared bus must be unique

$$\boldsymbol{O}_e(j, j') + \boldsymbol{O}_e(j', j) = 1 \tag{6}$$

where $j, j' \in [1, 2, \ldots, n_e]$ and $j < j'$.
c_4: Eliminate intra-processor communications: do not assign a data transfer on the shared bus if both its source and sink tasks are assigned to the same processor

$$\boldsymbol{X}_e(j) + \boldsymbol{X}_t(i, p) - \boldsymbol{X}_t(i', p) \geq 0 \tag{7}$$

where $i, i' \in [1, 2, \ldots, n_t]$ and $i \neq i'$, $p \in [1, 2, \ldots, n_p]$, $j \in [1, 2, \ldots, n_e]$ and j refers to the index of the data transfer between ith and i'th tasks.
c_5: The execution of any two tasks that run on the same processor cannot be overlapped

$$\boldsymbol{S}_t(i, s) + \boldsymbol{C}_t(i, p)\boldsymbol{D}_p(p, s) + K(\boldsymbol{X}_t(i, p) + \boldsymbol{X}_t(i', p) + \boldsymbol{O}_t(i, i') - 3) \leq \boldsymbol{S}_t(i', s) \tag{8}$$

where $i, i' \in [1, 2, \ldots, n_t]$ and $i \neq i'$, $p \in [1, 2, \ldots, n_p]$ and $s \in [1, 2, \ldots, n_s]$.
c_6: The execution of any two data transfers that run on the shared bus cannot be overlapped

$$\boldsymbol{S}_e(j, s) + \boldsymbol{X}_e(j)\boldsymbol{C}_e(j)\boldsymbol{D}_c(s) + K(\boldsymbol{X}_e(j) + \boldsymbol{X}_e(j') + \boldsymbol{O}_e(j, j') - 3) \leq \boldsymbol{S}_e(j', s) \tag{9}$$

where $j, j' \in [1, 2, \ldots, n_e]$ and $j \neq j'$ and $s \in [1, 2, \ldots, n_s]$.
c_7, c_8: Constraints that hold precedences imposed by the application task graph: A data transfer cannot be started until the execution of the task that provides its data ends. Similarly, the execution of a task cannot be started until the data transfer that provides its data ends

$$\boldsymbol{S}_t(i, s) + \sum_{p=1}^{n_p} \boldsymbol{C}_t(i, p)\boldsymbol{D}_p(p, s)\boldsymbol{X}_t(i, p) - \boldsymbol{S}_e(j, s) \leq 0 \tag{10}$$

$$\boldsymbol{S}_e(j, s) + \boldsymbol{C}_e(j)\boldsymbol{D}_c(s)\boldsymbol{X}_e(j) - \boldsymbol{S}_t(i', s) \leq 0 \tag{11}$$

where $j \in [1, 2, \ldots, n_e]$ and j refers to the index of the data transfer between ith and i'th tasks, $i, i' \in [1, 2, \ldots, n_t]$ and $i \neq i'$ and $s \in [1, 2, \ldots, n_s]$.
c_9: Set the value of $\boldsymbol{M}$, if the execution of all tasks ends before the time limit $\lceil T \rceil$

$$\boldsymbol{S}_t(i, s) + \sum_{p=1}^{n_p} \boldsymbol{C}_t(i, p)\boldsymbol{D}_p(p, s)\boldsymbol{X}_t(i, p) + K(\boldsymbol{M}(s) - 1) \leq \lceil T \rceil \tag{12}$$

where $i \in [1, 2, \ldots, n_t]$ and $s \in [1, 2, \ldots, n_s]$.

4.1.4 Objective function: The goal is to maximise the performance yield Y_T. Therefore we have

$$Y_T = \sum_{s=1}^{n_s} \boldsymbol{M}(s) \tag{13}$$

4.2 Heuristic task and communication scheduling

In this section, we present two variation-aware heuristic methods for task and communication scheduling for MPSoCs. The first one, which is called heuristic task scheduling (HTS) throughout this paper, was proposed in [2] and to the best of our knowledge, is the only heuristic variation-aware task and communication scheduling method in the field of MPSoCs that use weighted directed acyclic task graphs as the application model. However, as we will show in the experimental results section, this method becomes inefficient as the size of the task graph increases. This motivated us to provide a modified version of HTS (called MHTS) that can produce more efficient results by sacrificing execution time of the algorithm over the quality of the solutions. We briefly review the mentioned methods in the rest of this section.

4.2.1 Heuristic task scheduling: As proposed in [2], HTS is a one-pass dynamic-list-scheduling-based task and communication scheduling algorithm, that starts from the root node of the task graph and selectively assigns ready tasks (for which the precedent tasks have already been scheduled) to MPSoC resources until all of the tasks are scheduled. The corresponding pseudo code for HTS is given in Algorithm 1 (see Fig. 2).

IET Comput. Digit. Tech., 2015, Vol. 9, Iss. 4, pp. 221–229

Algorithm 1

Input: task graph, MPSoC architecture, system constraints ($\lceil T \rceil$)

Output: final schedule

1: Initialize Ready Task Set (RTS), a set of unscheduled tasks for which the precedent tasks have already been scheduled.
2: **while** RTS is not empty **do**
3: Select N most critical tasks and form Critical Task Set (CTS).
4: **for** Each task $task_i$ in CTS **do**
5: Tentatively schedule $task_i$ to MPSoC resources to obtain the best and second best schedule, and compute $DP(task_i)$.
6: **end for**
7: Select the schedule with maximum DP and perform it.
8: Add new ready tasks to RTS.
9: **end while**
10: Compute Performance Yield.

Fig. 2 *Pseudo code of HTS, a variation-aware task and communication scheduling heuristic proposed in [2]*

In each step, the selection of the task to be scheduled is guided through the definition of a metric called DP of tasks, given as follows [2]

$$\mathrm{DP}(\mathrm{task}_i) = \mathrm{MaxYield}(\mathrm{task}_i) + \Delta\mathrm{Yield}(\mathrm{task}_i) \tag{14}$$

where $\mathrm{MaxYield}(\mathrm{task}_i) = \max(\mathrm{Yield}(\mathrm{task}_i, p))$, for $p \in [1, 2, \ldots, n_p]$ and $\Delta\mathrm{Yield}(\mathrm{task}_i)$ is the difference between the highest yield and the second highest yield, which stands for the yield loss if the ith task is not scheduled onto its preferred MPSoC resource.

4.2.2 Modified heuristic task scheduling: As we will show in the experimental results section, HTS becomes inefficient as the size of the task graph grows. This is because of the one-pass nature of the algorithm, which cannot benefit from rescheduling of tasks for improving the quality of the solution. Moreover, the computation of DP is only performed for the ready tasks, ignoring the criticality of the unready ones. This motivated us to provide a more sophisticated DP-based heuristic method called MHTS, as described as below.

Algorithm 2 (see Fig. 3) shows the pseudo code of the proposed MHTS heuristic task and communication scheduling algorithm. We first compute the DP of all paths of the task graph and select the N most critical paths (MCP). We define the DP of a path as

$$\mathrm{DP}(\mathrm{path}_i) = 1 - \mathrm{Yield}(\mathrm{path}_i) \tag{15}$$

where $\mathrm{Yield}(\mathrm{path}_i)$ is defined as the probability of the path to meet the timing deadline of the system $\lceil T \rceil$. This definition enables us to focus on the rescheduling of paths that contribute to more yield loss. Then, starting from the MCP, we compute the DP of all tasks within this path (using the same approach as in [2]), and choose the task (communication) reschedule with highest DP. We commit this reschedule if it improves the total timing yield. Then, we update the task graph and proceed with the next iteration of the algorithm. If the timing yield is not improved by this reschedule, we discard MCP and try the next critical path. If all paths are traversed and no improvement is possible, the algorithm stops.

4.3 Metaheuristic task and communication scheduling

In this paper, we use previously proposed GA-based [7] and SA-based [8] techniques to solve variation-aware task and communication scheduling problem. This section briefly reviews the mentioned methods for being self-contained.

4.3.1 GA-based task scheduling: We chose [7] as our candidate for GA-based scheduling methods since it is, to the best of our knowledge, the only variation-aware method that targets both task and communication scheduling. As described in [7], each solution of the scheduling problem is modelled as a chromosome with $2n_t + n_e$ genes: Part-1 with n_t integer values representing the task assignment (i.e. the index of processors to which the tasks are assigned), and Part-2 with $n_t + n_e$ integer values representing the order of execution of tasks and communications (i.e. the index of tasks and data transfers in the order specified in the solution).

For the crossover function, we used a simple two-point crossover operator for Part-1 of the solution where the genes between two randomly chosen locations of the parent chromosomes are combined to form the child chromosome. For Part-2, a specialised form of single-point crossover operator is used to preserve the uniqueness of indexes of tasks and communications in the reordered chromosome [7]. To summarise it, given two parent chromosomes parent_1 and parent_2, a crossover point k, $1 < k < n_p + n_e$, is selected randomly. The genes $[1, k]$ of parent_2 are copied to the genes $[1, k]$ of child chromosome. To fill the remaining genes $[k+1, n_t + n_e]$ of the child chromosome, chromosome parent_1 is scanned from the first to the last gene and each node that is not yet in the child is added to the next empty position of the child, while preserving its order in parent_1.

For the mutation function, a randomly picked gene from Part-1 is randomly mutated to a new value in the range $[1, 2, \ldots, n_p]$. For Part-2, and to preserve the uniqueness of indexes of tasks and communications in the reordered chromosome, the values of two randomly picked genes are swaped.

4.3.2 SA-based task scheduling: As described in [8], the only SA-based variation-aware task and communication scheduling method in the literature, the solution is composed of $2n_t + n_e$ integer values similar to our GA-based method. The only difference is that it uses an annealing procedure that works as follows: For Part-1 of the solution (the task assignment part), it alters the value of a randomly picked node. For Part-2, it first traverses the task graph to find a set of all tasks (and communications) that can be reordered. Then it simply picks two random tasks and two random communications from the set and reorder them to form a new solution.

Algorithm 2

Input: task graph, MPSoC architecture, system constraints ($\lceil T \rceil$)

Output: final schedule

```
1: while not converged do
2:   Traverse task graph to find all paths from the root to the leafs of the task graph
3:   Compute the dynamic priority of each path, and form Critical Path Set (CPS), a sorted
     list of N most critical paths.
4:   while CPS is not empty do
5:     Select the most critical path (MCP) among the ones in CPS as the candidate path.
6:     for Each task task_i in MCP do
7:       Tentatively schedule task_i to MPSoC resources to obtain the best and second best
         schedule, and compute DP(task_i).
8:     end for
9:     Select the schedule with maximum DP and perform it.
10:    if fitness is improved then
11:      Update the task graph, and continue with the next iteration (go to step 1).
12:    else
13:      Remove MCP from CPS, and try the next candidate path (go to step 4).
14:    end if
15:  end while
16: end while
17: Compute Performance Yield.
```

Fig. 3 *Pseudo code of MHTS, the proposed variation-aware task and communication scheduling heuristic*

5 Complexity analysis

In this section, we analyse the computational complexity of the mentioned approaches in terms of number of tasks (n_t), number of processors (n_p), and number of sample MPSoC chips (n_s).

5.1 ILP-based approach

As suggested by Schrijver [19], the average-case computational complexity of the Simplex method (which we use here as our solver) is $O(\min(m, n) \times m \times (n+m))$ in practice, were n is the number of variables and m is the number of constraints. According to the proposed ILP problem in Section 4.1, we have

$$n = n_t^2 + n_e^2 + n_t(n_s + n_p) + (n_e + 1)(n_s + 1) \tag{16}$$

and

$$\begin{aligned} m = {} & (n_t^2 + n_e^2 - n_t - n_e)\left(n_s n_p + \frac{1}{2}\right) \\ & + n_e(2n_s + n_p) + n_t(n_s + 1) + 1 \end{aligned} \tag{17}$$

Assuming $\hat{n} = n_t + n_e$ and $n_s \gg \hat{n}$, the overall complexity of the ILP problem will be $O(\hat{n}^5 n_s^3 n_p^2)$.

5.2 Heuristic approaches

As illustrated in Algorithm 1 (Fig. 2), there are two loops in HTS. The outer loop runs $n_t + n_e$ times. The number of iterations of the inner loop depends on the structure of the task graph, varying from 1 to $n_t + n_e$. To compute DP of a task, we inspect all $n_t + n_p$ possible locations in the task queues of processors (or n_e locations of the shared bus queue in case of data transfers), for which we compute timing yield using depth first search method with the complexity of $O(n_s(n_t + n_e))$. Therefore the overall complexity of HTS is $O(\hat{n}^4 n_s)$.

As illustrated in Algorithm 2 (Fig. 3), there are three loops in MHTS. The innermost loop has the computational complexity of $O(\hat{n}^2 n_s)$ similar to HTS. However, the number of iterations of the two while loops, n_i, depends on the task graph structure. Therefore the overall computational complexity will be $O(\hat{n}^2 n_s n_i)$.

5.3 Metaheuristic approaches

As suggested by Brucker [20], the computational complexity of a GA-based multiprocessor task scheduling algorithm can be described as $O(n_i n_{ch} O(\text{Fitness}))$, where n_i and n_{ch} are the number of iterations and the number of chromosomes in the population, respectively. Our GA-based method uses a depth-first-search-based fitness function with the complexity of $O(n_s(n_t + n_e))$ to compute timing yield for a given chromosome. It is also suggested in [21] that the optimal value of n_{ch} is somewhere between n and $2n$ (n is the variable size), rendering its complexity as $O(n)$. This leads to an overall computational complexity of $O(\hat{n}^2 n_s n_i)$.

For the SA-based approach and similar to GA, the computational complexity can be described as $O(n_i O(\text{Fitness}))$ since $n_{ch} = 1$. Using the same fitness function as our GA-based method, the overall complexity of the SA-based method will be $O(\hat{n} n_s n_i)$. Note that for our GA-based and SA-based methods, the crossover, mutation and annealing operators are designed such that their computational complexity does not scale with problem size [7, 8].

6 Experimental results

In this section, we provide the experimental results of applying our variation-aware task and communication scheduling algorithms on real world benchmarks.

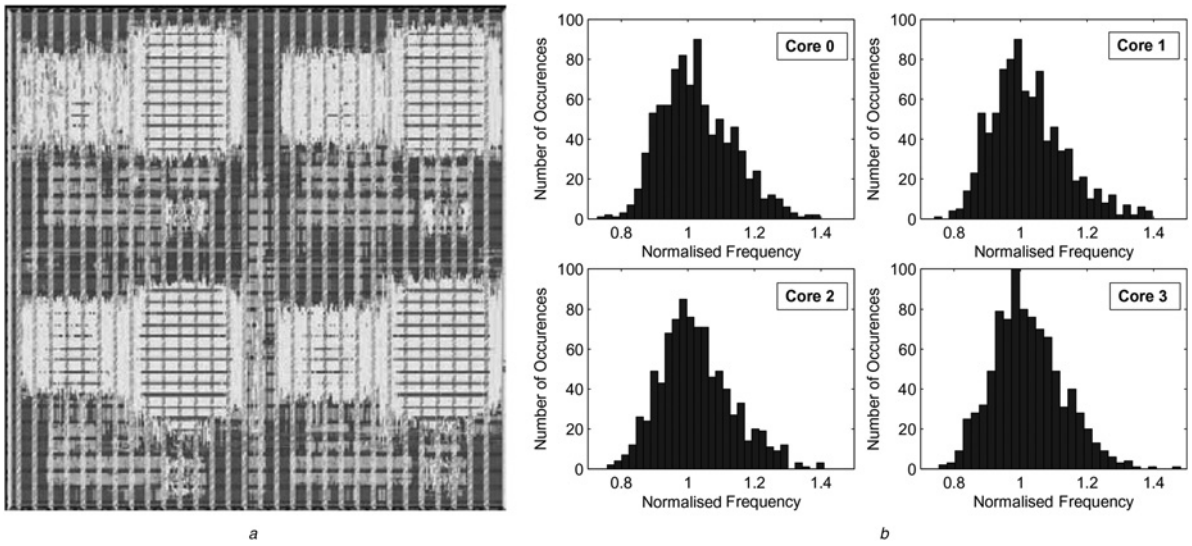

Fig. 4 *LEON MP final post-P&R layout along with the normalised frequency distribution of cores*

a Post-P&R layout of LEON MP system
b Normalised frequency distribution of its processor cores

6.1 Experimental setup

Our target MPSoC is LEON MP, a configurable LEON3 multi-processor system with four identical LEON3 processors [22]. LEON MP is a shared-memory and shared address-space multiprocessor system. The LEON3 processor is a synthesisable VHDL model of a 32-bit processor compliant with the SPARC V8 architecture. Its integer unit consists of a seven stage pipeline; and there are hardware multiply, divide and MAC units and separate instruction and data caches. The processor is particularly designed for system-on-a-chip designs. It is also highly configurable and the designer can configure functional units, register windows and instruction and data caches.

In our LEON MP system, LEON processors are attached to a shared AMBA high performance bus. The source code of the multiprocessor was synthesised using synopsys design compiler [23] and NanGate 45 nm open cell library [24], and then placed and routed by using Cadence SoC Encounter [25] to produce the final layout. Then, the number of paths along with the location, type and delay of all gates of the design are extracted using Synopsys PrimeTime [23]. The extracted information are then feed into our variability modelling framework to generate the frequency distributions of MPSoC cores. Fig. 4 shows the LEON MP final post-P& R layout along with the normalised frequency distribution of cores obtained using the above approach.

We evaluate the performance of the variation-aware task scheduling methods using two sets of embedded system benchmarks from E3S [26] and MiBench [27]. To this end, we first cross-compiled the source code of benchmarks using LEON BCC cross-compiler and then measured the execution cycle count of tasks using LEON cycle-true simulator, TSIM [28]. The task scheduling methods were implemented using Matlab and its GA and SA optimisation toolboxes [29]. The GA and SA parameters are shown in Table 3 and 4, respectively. The simulations for GA-based and SA-based methods were run 100 times and the average values are reported. The ILP model was solved using IBM CPLEX optimiser [30]. The simulations were conducted on a 3 GHz quad-core computer with 12 GB of RAM.

Table 3 GA parameters

Parameters	Values
population size	$2N$
variable size	N
scaling function	rank
selection function	stochastic uniform
elite count	$0.2N$
crossover fraction	0.8
stopping criteria	stall limit = 10 000, or when it converges to the optimal value obtained by ILP-based method

6.2 Results

Table 5 shows the timing yields obtained by applying our task and communication scheduling algorithms on E3S and MiBench benchmarks, at various timing constraint scenarios. The number of nodes in each task graph, $\hat{n} = n_t + n_e$, is also shown besides the name of each task graph. Three different timing constraint scenarios are used for our comparative study. We first measured the best achievable execution time of each benchmark assuming 100% timing yield. Then we used 70, 80 and 90% of this value as our timing constraints.

As shown in Table 5, HTS is able to find good solutions in the case of small task graphs. However, as the size of task graph grows, it becomes ineffective in finding a proper solution. In case of telecom as our biggest task graph, HTS results in schedules with zero timing yield in all three scenarios, rendering it ineffective in case of large task graphs. On the other hand, the quality of the solutions produced by MHTS surpasses HTS, because of its iterative nature and the use of path DP to guide the task selection process. Our metaheuristic approaches are also effective in finding near-optimal solutions, even for large task graphs.

Table 6 also shows the execution time of the mentioned methods in seconds. As illustrated in this table, HTS is multiple order of magnitude faster than the other algorithms since it is a one-pass

Table 4 Simulated annealing parameters

Parameters	Values
variable size	N
reannealing interval	100
initial temperature	100
temperature update function	initial temperature $\times 0.95^{\text{iteration}}$
stopping criteria	stall limit = 20 000, or when it converges to the optimal value obtained by ILP-based method

Table 5 Timing yield (%) obtained from our task and communication scheduling algorithms

		Benchmarks													Average
		Auto (43)	Consumer (29)	Network (24)	Office (14)	Telecom (50)	Basicmath (17)	Bitcount (25)	Dijkstra (5)	FFT (7)	Quicksort (7)	Stringsearch (25)	Susancorner (9)	Susanedge (13)	Susansmooth (7)	
Deadline 1																
approaches	ILP	83.3	73.3	75.4	91.0	58.3	76.0	53.7	72.5	78.5	76.5	80.6	79.2	80.3	72.5	75.1
	GA	81.8	73.2	74.9	76.5	51.2	76.0	52.5	72.5	78.5	76.5	77.0	79.2	80.3	72.5	73.1
	SA	82.1	73.2	74.7	76.5	49.7	76.0	53.7	72.5	78.5	76.5	77.0	79.2	80.3	72.5	73.0
	HTS	73.3	70.2	38.2	74.2	0.0	75.5	50.5	72.5	78.2	76.5	75.2	78.0	79.0	72.5	65.3
	MHTS	82.2	73.0	71.8	75.5	49.0	76.0	53.7	72.5	78.5	76.5	76.7	79.2	80.3	72.5	72.7
Deadline 2																
approaches	ILP	96.5	93.4	100.0	100.0	82.3	93.0	76.1	86.5	95.7	91.8	100.0	96.7	97.0	86.8	92.6
	GA	96.5	91.7	94.2	91.5	76.0	93.0	71.5	86.5	95.7	91.8	92.3	96.7	97.0	86.8	90.1
	SA	96.5	91.7	94.2	91.5	75.7	93.0	75.0	86.5	95.7	91.8	92.8	96.7	97.0	86.8	90.4
	HTS	94.3	85.5	63.0	89.0	0.0	92.5	73.7	86.5	93.2	91.5	91.8	95.5	95.0	86.8	81.3
	MHTS	96.5	90.7	94.2	90.5	75.3	93.0	75.0	86.5	95.7	91.7	92.8	96.7	96.7	86.8	90.1
Deadline 3																
approaches	ILP	100.0	99.1	100.0	100.0	90.4	98.5	100.0	97.0	99.5	99.0	100.0	99.5	100.0	97.0	98.6
	GA	99.7	98.5	98.2	98.5	88.3	98.5	91.5	97.0	99.5	99.0	98.2	99.5	99.5	97.0	97.4
	SA	99.7	98.5	98.5	98.5	89.7	98.5	92.5	97.0	99.5	99.0	99.0	99.5	99.8	97.0	97.6
	HTS	98.5	97.5	85.5	98.0	0.0	98.0	91.5	97.0	98.5	99.0	98.7	99.0	99.2	97.0	89.8
	MHTS	99.7	98.5	98.2	98.0	78.5	98.5	92.7	97.0	99.5	99.0	98.5	99.5	99.8	97.0	96.7

Table 6 Execution time (s) of running our task and communication scheduling algorithms

		Benchmarks													Average
		Auto (43)	Consumer (29)	Network (24)	Office (14)	Telecom (50)	Basicmath (17)	Bitcount (25)	Dijkstra (5)	FFT (7)	Quicksort (7)	Stringsearch (25)	Susancorner (9)	Susanedge (13)	Susansmooth (7)	
Deadline 1																
approaches	ILP	16 578	4749	1699	309	41 250	1147	8402	8.9	17.6	12.2	3749.8	22	364	13.2	5594
	GA	1718.5	251.6	4.7	1.4	1674	627.1	883.2	364.68	392.47	384.69	12.5	518.7	617.8	369.9	558.6
	SA	1484.8	46.1	9.5	3.8	1544	950.3	1792.2	920.7	969.4	950.69	487.97	995.5	949.9	979.6	863.2
	HTS	8.3	2.5	1.78	0.15	64.7	0.85	3.88	0.01	0.02	0.02	2.36	0.06	0.12	0.04	6.1
	MHTS	356.6	59.9	41.4	5.67	2048	16.79	148.1	0.69	1.49	1.6	116.3	2.47	5.77	1.3	200.4
Deadline 2																
approaches	ILP	13 810	607	251	117	39 760	201	13 959	20.8	31.7	24.1	778	35	90.8	28	4979
	GA	1781.3	2.6	1322.7	0.29	1742	692.3	939.7	630.8	542.9	542.6	797.4	386.5	0.80	350.1	695.1
	SA	1025.3	67.8	1183.1	0.6	1623	923.5	882.8	1150.5	970.1	700.19	716	667.9	5.76	649.33	754.7
	HTS	7.8	2.2	2.6	0.12	70.7	0.83	3.6	0.01	0.02	0.02	1.66	0.03	0.05	0.02	6.4
	MHTS	683.7	106.1	85.4	4.7	1345	22.8	294.3	0.91	1.8	1.87	152	3.5	7.08	1.36	193.6
Deadline 3																
approaches	ILP	16 480	1233	399.9	124.6	39 910	976.6	2353	23.1	18.7	22	704.6	25	74	34	4456
	GA	1643	904.8	759.6	1.11	1693	579.8	903.9	318	371.3	367.4	867.9	416.1	462.7	338.8	678.6
	SA	665.2	669.4	111.5	0.4	1612	626.1	608.2	627.6	655.8	654.8	75.2	668.9	672.3	643.2	592.2
	HTS	8.69	2.77	1.56	0.11	70.4	0.84	4.6	0.01	0.02	0.02	1.7	0.03	0.06	0.02	6.5
	MHTS	508	98.49	113.2	6.4	1798	20.8	117.9	0.67	1.46	1.57	107.7	3.0	6.3	1.2	198.9

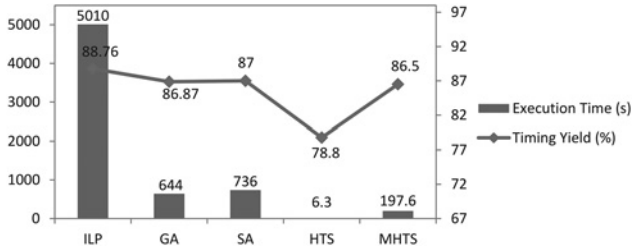

Fig. 5 *Overall comparison of the task scheduling methods in terms of quality of the results and execution time*

method. MHTS can take much longer to execute than HTS, but as shown in Table 5, outperforms HTS in terms of quality of the results. Another observation from Table 6 is that ILP-based approach runs slightly slower than the other approaches on average, as we also showed in the complexity analysis section. This is the case with the telecom and auto benchmark.

Fig. 5 shows the overall comparison of the mentioned methods in terms of quality of the results and execution time. The results are the average values of timing yield and execution time over all benchmarks and timing constraint scenarios. As illustrated in this figure and compared with our ILP-based method, HTS is 795 times faster, but provides solutions with 10% lower timing yield on average. SA-based and GA-based methods provide near-optimal solutions within 1–2% of the optimal solution, and are 7.8 and 6.8 times faster on average. Finally, MHTS yields in solutions within 2.5% of the optimal solutions, and is 25.3 times faster.

7 Conclusion

This paper provides a comparative study of the current variation-aware static task scheduling algorithms for embedded MPSoCs. We used our developed variability modelling framework to extract MPSoC frequency distributions in the presence of both WID and D2D process variations. We compared the task scheduling algorithms in terms of both quality of the solutions and computational complexity. We showed that the ILP-based task scheduling technique, while guaranteeing the optimality of the solution, can be costly for large application task graphs. On the other hand, one-pass heuristic method is 795 times faster than ILP-based method on average, but is ineffective to find reasonable solutions in the case of large task graphs. Finally, metaheuristic approaches can produce near-optimal schedules within 1–2% of the optimal solutions on average, with up to 7.8 times faster execution time on average compared with ILP-based approach.

8 References

1 Wang, F., Nicopoulos, C., Wu, X., Xie, Y., Vijaykrishnan, N.: 'Variation-aware task allocation and scheduling for MPSoC'. IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD'07), November 2007, pp. 598–603

2 Wang, F., Chen, Y., Nicopoulos, C., Wu, X., Xie, Y., Vijaykrishnan, N.: 'Variation-aware task and communication mapping for MPSoC architecture', *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. (TCAD)*, 2011, **30**, (2), pp. 295–307

3 Singhal, L., Oh, S., Bozorgzadeh, E.: 'Yield maximization for system-level task assignment and configuration selection of configurable multiprocessors'. Sixth IEEE/ACM/IFIP IntConf. on Hardware/Software Codesign and System Synthesis (CODES + ISSS'08), 2008, pp. 249–254

4 Mirzoyan, D., Akesson, B., Goossens, K.: 'Process-variation aware mapping of best-effort and real-time streaming applications to MPSoCs', *To Appear In: ACM Trans. Embed. Comput. Syst. (TECS)*, 2013, **13**, (2s), pp. 1–24

5 Chon, H., Kim, T.: 'Timing variation-aware task scheduling and binding for MPSoC'. Asia and South Pacific Design Automation Conf. (ASPDAC'09), January 2009, pp. 137–142

6 Huang, L., Xu, Q.: 'Performance yield-driven task allocation and scheduling for MPSoCs under process variation'. 47th Design Automation Conf. (DAC'10), June 2010, pp. 326–331

7 Momtazpour, M., Goudarzi, M., Sanaei, E.: 'Variation-aware task and communication scheduling in MPSoCs for power-yield maximization', *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 2010, **E93.A**, (12), pp. 2542–2550

8 Momtazpour, M., Goudarzi, M., Sanaei, E.: 'Static statistical MPSoC power optimization by variation-aware task and communication scheduling', *Elsevier J. Microprocess. Microsyst., Spec. Issue Embed. Multicore Syst.*, 2012, **37**, (8), pp. 953–963

9 Singhal, L., Kooti, H., Bozorgzadeh, E.: 'Process variation-aware task replication for throughput optimization in configurable MPSoCS'. Electronic System Level Synthesis Conf. (ESLsyn), June 2012, pp. 44–49

10 Bhardwaj, K., Roy, S., Chakraborty, K.: 'Power-performance yield optimization for MPSoCs using MILP'. 13th Int. Symp. on Quality Electronic Design, March 2012, pp. 764–771

11 Ghorbani, M.: 'A variation and energy aware ILP formulation for task scheduling in MPSoC'. 13th Int. Symp. on Quality Electronic Design, March 2012, pp. 772–777

12 Sarangi, S., Greskamp, B., Teodorescu, R., Nakano, J., Tiwari, A., Torrellas, J.: 'VARIUS: A model of process variation and resulting timing errors for microarchitects', *IEEE Trans. Semicond. Manuf. (SM)*, 2008, **21**, (1), pp. 3–13

13 Assare, O., Rad, H.I., Momtazpour, M., Sanaei, E., Goudarzi, M.: 'VAREX: A post-P& R variability modeling framework for multiprocessor SoCs'. IEEE/ACM Workshop on Variability Modeling and Characterization (VMC), November 2011

14 Assare, O., Momtazpour, M., Goudarzi, M.: 'Accurate estimation of leakage power variability in sub-micrometer CMOS circuits'. Process of Publication in 15th Euromicro Conf. on Digital System Design (DSD'12), September 2012

15 Chandra, S., Lahiri, K., Raghunathan, A., Dey, S.: 'Considering process variations during system-level power analysis'. Int. Symp. on Low Power Electronics and Design (ISLPED'06), October 2006, pp. 342–345

16 Teodorescu, R., Torrellas, J.: 'Variation-aware application scheduling and power management for chip multiprocessors'. 35th Int. Symp. on Computer Architecture (ISCA'08), June 2008, pp. 363–374

17 Karnik, T., Borkar, S., De, V.: 'Probabilistic and variation-tolerant design: Key to continued moores law'. ACM/IEEE TAU Workshop on Timing Issues in the Specification and Synthesis of Digital Systems, February 2004

18 R Development Core Team: 'R: A language and environment for statistical computing' (R Foundation for Statistical Computing, Vienna, Austria, 2007)

19 Schrijver, A.: 'Theory of linear and integer programming' (Wiley, 1987)

20 Brucker, P.: 'Scheduling algorithms' (Springer, 2007, 5th edn.)

21 Alander, J.: 'On optimal population size of genetic algorithms'. Proc. CompEuro '92. 'Computer Systems and Software Engineering', May 1992, pp. 65–70

22 LEON3 multiprocessor: [Online]. Available at http://www.gaisler.com/index.php/products/processors/leon3

23 Synopsys Design Compiler: [Online]. Available at http://www.synopsys.com/tools/implementation

24 The NanGate 45 nm Open Cell Library: 'An open source standard cell library', [Online]. Available at http://www.nangate.com

25 Cadence Encounter Digital Implementation System: [Online]. Available at http://www.cadence.com/products/di/edi_system/

26 Dick, R.: 'Embedded Systems Synthesis Benchmark Suites (E3S)'. [Online]. Available at http://ziyang.eecs.umich.edu/~dickrp/e3s

27 MiBench Embedded System Benchmark Suite: [Online]. Available at http://www.eecs.umich.edu/mibench/

28 LEON3 Cycle-True Simulator: [Online]. Available at http://www.gaisler.com/index.php/products/simulators/tsim

29 MATLAB, The MathWorks Inc.: [Online]. Available at http://www.mathworks.com/products

30 CPLEX Optimization software package: [Online]. Available at http://www.cplex.com/