Designing Energy-Efficient MPSoC with Untrustworthy 3PIP Cores

Yidan Sun, Guiyuan Jiang, Siew-Kei Lam, and Fangxin Ning

***Abstract*—The adoption of large-scale MPSoCs and the globalization of the IC design flow give rise to two major concerns: high power density due to continuous technology scaling and security due to the untrustworthiness of the third-party intellectual property (3PIP) cores. However, little work has been undertaken to consider these two critical issues jointly during the design stage. In this paper, we propose a design methodology that minimizes the energy consumption while simultaneously protecting the MPSoC against the effects of hardware trojans. The proposed methodology consists of three main stages: 1) Task scheduling to introduce core diversity in the MPSoC in order to detect the presence of malicious modifications in the cores, or mute their effects at runtime, 2) Vendor assignment to the cores using a novel heuristic that chooses vendor-specific cores with operating speed that minimizes the total energy consumption of the MPSoC, and 3) Explore optimization opportunities for further energy savings by minimizing idle periods on the cores, which are caused by the inter-task data dependencies. Experimental results show that our solutions consume only 1/3 energy of existing solutions without increasing schedule length while satisfying the security constraints.**

***Index Terms*—3PIP cores, untrustworthy, energy efficient, scheduling, vendor assignment, speed optimization.**

I. Introduction

EMBEDDED computing systems will continue to demand shorter development cycles without compromising power efficiency, performance, programmability, and cost. The widening gap between the power demands and the limited power source of mobile devices can be mitigated by exploiting heterogeneous Multiprocessor System-on-Chip (MPSoC). As such, there is an increasing trend to integrate third-party intellectual property (3PIP) cores to build heterogeneous MPSoC, which allows designers to quickly respond to the increasing demands in energy consumption, functionality as well as programmability without sacrificing design productivity [1].

At the same time, increasing number of real-time applications in finance, military, transportation and medical systems (such as the signaling control in transportation systems, medical electronics systems) necessitates that the systems are secure to assure integrity of information and confidentiality. However, the integration of 3PIP cores and the outsourcing of fabrication and testing have led to major security concerns such as functionality change, performance degradation, and denial of service, as the 3PIPs are not 100% trustworthy [2]. Heterogeneous MPSoCs designed using 3PIP cores are vulnerable to malicious modifications, also known as hardware trojans, which can cause system failures or create backdoors to leak confidential information back to the attacker [3]. Rogue foundries could insert disguised malicious modifications during the fabrication process, and the situation becomes more complicated when the 3PIP cores from the same vendor collude with each other. That is, spiteful vendors distribute trojans on different 3PIP cores and activate them through secret communication between these cores.

Detecting and mitigating the trojans in 3PIPs is extremely challenging due to the limited gate level visibility inside a 3PIP core since most 3PIP cores are commonly delivered as "black boxes" in order to protect the intellectual property of the third parties [4]. Moreover, the effects of trojans may be neglected because a healthy core could also generate similar performance fluctuation due to process variations and non-deterministic architectural events. Therefore, it is time-consuming and costly to thoroughly analyze or test a system for the presence of trojans using traditional methods such as functional testing [5], side-channel analysis [6], etc.

The work in [3] investigated security-driven MPSoC task scheduling to account for the untrustworthiness of the 3PIP cores. Since it is impossible to guarantee trustworthiness (i.e., 100% trojan freeness) of 3PIP cores, the authors propose approaches to enable the detection of trojans or mute their effects during runtime. Contrary to existing trojan detection and prevention techniques [7], [8], the authors incorporate vendor diversity into MPSoC task schedules to reduce false negatives in the detection stage. This is achieved by duplicating each task and mapping them on 3PIP cores of different vendors to detect trojans that maliciously alter task outputs. In addition, vendor diversity also mutes potential trojan effects by preventing collusion between malicious 3PIP cores from the same vendor that send triggers through communication paths. However, [3] focuses on incorporating security constraints into MPSoC design without concerning energy consumption. Also, it imposes a restriction that each vendor can provide only single processor operating speed, while a more practical scenario is that a task can be executed at different speeds through clock frequency scaling. The provision for each core to operate at different speeds presents new opportunities to improve the energy efficiency.

The above mentioned works mainly worked on security policy enforcement in modern MPSoC designs [9]. On the other hand, there have been considerable research efforts devoted to minimizing the energy consumption of heterogeneous MPSoC systems at different levels, i.e. algorithm, system, architecture, and circuit levels [10], [11]. These works

Y. Sun, G. Jiang, S.K. Lam and F. Ning are with the School of Computer Science and Engineering, Nanyang Technological University, 639798 Singapore. e-mail: (ysun014@e.ntu.edu.sg, {gyjiang,fangxin_ning}@ntu.edu.sg, siewkei_lam@pmail.ntu.edu.sg).

Manuscript received April 19, 2005; revised August 26, 2015.

mainly address the problem of task allocation and scheduling of energy-efficient MPSoCs with dynamic voltage frequency scaling (DVFS). However, there is very little work that jointly considers energy consumption and security for MPSoC design despite the importance of these two critical issues in modern embedded computing systems, especially for heterogeneous MPSoCs built from 3PIP cores which are untrustworthy.

In this paper, we propose a design methodology that aims to minimize energy consumption of heterogeneous MPSoCs under two security-driven diversity constraints as well as schedule length constraint. The joint consideration of task scheduling, DVFS and security constraints lead to a complex and heterogeneous structure of the design space, making the existing methods inefficient for searching optimal solutions. The contributions of this paper are summarized as follows:

1) To the best of our knowledge, this is the first framework that jointly considers energy consumption and security constraints in MPSoC design. We mathematically formulate security constraints by defining suitable binary decision variables, and we formulate the optimization problem using a mixed integer programming (MIP) model with the objective of minimizing the overall energy consumption under constraints of both security constraints and maximum delay constraint (i.e. the schedule length must not exceed). The proposed formulation can be solved using optimization tools (e.g. CPLEX), which enable us to evaluate our proposed method by comparing with the optimal solutions.
2) We propose a design methodology that solves the problem in three-stages, i.e., 1) security-driven scheduling of tasks to processor cores, 2) vendors to cores assignment based on the supply-demand speed deviation to minimize energy consumption, and 3) execution speed optimization for further reducing energy consumption while guaranteeing that the schedule length constraint is met. *In Stage 1*, we propose an efficient algorithm to schedule tasks onto a given number of cores under two security-driven diversity constraints. The scheduling algorithm relies on two efficient techniques, i.e. a task priority queue and a candidate core selection mechanism, to minimize the total schedule length while balancing the required number of vendors under security constraints. A Core Conflict Graph (CCG) is constructed and colored using the minimum number of colors during scheduling to capture the conflicting relationship among cores under the security constraints. *In Stage 2*, to assign suitable vendors to cores with the task schedules, we theoretically estimate the optimal speeds required by the tasks on each core, and develop an efficient heuristic to assign vendors to cores with the objective of minimizing the deviation between required speeds of the tasks and the available speeds provided by vendors. *In Stage 3*, after vendor assignment, we propose a novel optimization algorithm to further reduce energy consumption by exploiting the idle periods on cores due to the inter-task data dependency.
3) We demonstrate the effectiveness of our proposed approach by comparing them with the optimal solution obtained by CPLEX [46] and two existing methods. Extensive simulations have been conducted on both synthetic and real application task sets. Experimental results show that the proposed methodology leads to only 1/3 of the energy required by existing solutions, without an increase in schedule length while satisfying the security constraints. We show that CPLEX becomes intractable for solving large-scale problems. For problem instances with relatively small scales, our approach can be computed an order of magnitude faster than CPLEX, and provide solutions that are much closer to the optimal solutions than the baseline methods.

The remainder of the paper is organized as follows. Related works are discussed in Section II. Section III introduces important definitions and the problem formulation. Section IV discusses the proposed method consisting of three stages, i.e. 1) security-driven tasks scheduling, 2) vendors to cores assignment, and 3) execution speed refinement to improve energy consumption. We evaluate the performance of the proposed method in Section V, and Section VI concludes the paper.

II. Related Works

The existing works that are closely related to the problem considered in this paper can be classified into the following two categories, i.e. trojans detection in 3PIPs and security-aware MPSoC design.

A. Trojans Detection in 3PIPs

Existing methods in this category can be generally grouped into three approaches, i.e. code/structural analysis, security property verification techniques, and vendor diversity-based approach [13]. For *code/structural analysis*, code coverage analysis on Register Transfer Level (RTL) codes have been undertaken to identify suspicious signals due to the presence of a trojan [14], [15] since 3PIPs are typically delivered as RTL VHDL/Verilog codes. However, even with 100 percent coverage of the RTL code, it still cannot guarantee a fault-free code [16]. Usually, semi-manual approach is utilized, which first marks suspicious signals using tools (e.g. FANCI [17] marks gates with low activation probability as suspicious while VeriTrust [18] marks gates that are not driven by functional inputs as suspicious), and then manually analyzes the small number of suspicious gates to determine if they are part of a trojan. Rajendran et al. [12] focused on automated 3PIP trojans detection that compromise registers which store critical data, e.g. a stack pointer of a cores, or a secret key of a cryptographic design. The drawbacks of the code/structural analysis techniques are that they cannot guarantee trojan detection [19], burden the designer with manual analysis and can only analyze combinational parts of the entire design [12].

On the other hand, some approaches have been reported to detect data leakage [21] and malicious modifications to registers [22] using *security property verification techniques*. That is, 3PIP vendors and MPSoC integrators agreed on a predefined set of security properties that the IP should satisfy,

and then MPSoC integrators will check whether a design honors these properties by converting the target design into a proof-checking format (for example, Coq) for verification using existing models [20]. One of the drawbacks is that it is hard (or impossible) to include all kinds of vulnerabilities as pre-defined properties [22]. In addition, even if a Coq representation of a design is considered trustworthy, it does not necessarily mean that the corresponding VHDL/Verilog representation is trustworthy [12]. Functional testing [23] is a common approach to detect manufacturing faults by comparing the outputs of the genuine and the Trojan circuits using the same inputs. The work in [24] presents a method that detects the presence of Trojans in third party IP cores of MPSoCs. It focuses on streaming applications and takes advantage of the specific MPSoC architecture to not only detect and identify hardware Trojans but also recover pipelined MPSoCs from such infections. In addition, side-channel analyses [25], [26], [27], [28], [29] are also widely used to analyze the signals generated by electrical activity, based on which the state of the device and the data it processes can be obtained. These signals enable the detection of trojans. Formal verification techniques only work efficiently on small-scale instances, as it does not directly detect trojans but attempts to evaluate the trustworthiness of IP cores with a much larger search space.

The *vendor diversity-based approach* detects trojans using several IP cores from different vendors, based on the assumption that vendors do not collude meaning that trojans that may be inserted into these IP cores are different. There is a high probability that the outputs of IP cores from different vendors will be different when the trojans are activated. The work in [30] proposed to discover suspicious IP cores using a majority voting circuit. Farag et al. [31] compared the outputs of an operating core with a reference core at runtime, which has the obvious drawback that a golden reference is required. Rajendran et al. [32] works at RTL level to build trustworthy systems during high-level synthesis on a given configuration of untrusted and potentially Trojan-infected 3PIPs. The authors achieve this by identifying design constraints for Trojan detection to achieve detection, collusion prevention, and isolation of Trojan-infected 3PIP. The focus of our work differs from [32], as we investigate design-for-trust techniques to determine the configuration of the MPSoC system that meets the performance requirements and reduce energy. To reduce the runtime cost, Reece et al. proposed a technique for identifying hardware trojans with logic-based payloads at design time [33]. However, it cannot be applied to large circuits due to the limited software capacity [13]. In summary, the above mentioned works typically focus on detecting the attack patterns and the effects of a trojan, without attempting to eliminate false negatives. Moreover, these works did not attempt to improve the system performance (such as hardware resources, energy consumption, task execution time etc.) when the security measures are deployed.

B. Security-aware MPSoC Design

The work in [34] investigated the problem of scheduling periodic tasks of sensitive applications in embedded systems subject to security and timing constraints. It designed a necessary and sufficient feasibility check, based on which a scheduling algorithm was proposed to distribute slack times among a variety of security services for a set of periodic tasks, with the objective of optimizing security for embedded systems without sacrificing schedulability. The work in [35] investigated a traditional real-time scheduling algorithm with consideration of a wide variety of security requirements, where different computation tasks have different security services/types. A security-aware algorithm based on earliest deadline first strategy is developed to integrate the group-based security model, to optimize the combined security value of the selected services while guaranteeing the schedulability of the real-time tasks.

For protecting MPSoC, the work in [36] proposed an online trojan detection and prevention scheme for protecting homogeneous systems against malicious modifications. It exploits redundancy by executing computation tasks on three or more cores simultaneously, and the results are verified through voting before being written to memory. By partitioning an application into computation tasks that are executed on different sets of cores, this technique limits the data access capability of each cores thus improving the security level. However, the redundancy-based strategy obviously incurs an extra delay, energy consumption and hardware cost. Liu et al. [3] used a vendor diversity-based approach to solve a similar problem which requires less redundancy, by imposing security-driven diversity constraints into task scheduling process of the MPSoC design. The diversity-based security constraints can either detect the existence of malicious modifications or even mute their effects during application execution. The authors also developed algorithms to satisfy the proposed security constraints without compromising much on performance and hardware. The work in [37] proposed a high-level synthesis (HLS) methodology for secure scheduling of loop-based control data flow graphs, with the objective of reducing hardware cost. In particular, the work in [38] investigated the problem of resource-constrained task scheduling where the security threats are taken into consideration. The objective is to jointly minimize the number of security constraint violations and minimize the schedule length (or the number of required cores to satisfy certain schedule length) for a given application. In contrast to [38], our works takes security constraints as hard constraints such that no violation is allowed. In addition, we aim to jointly minimize the overall energy consumption while meeting the security constraints and schedule length requirements. In addition, to the best of our knowledge, neither of them has tried to optimize the overall energy consumption in the presence of security measures.

Many existing works have investigated the problem of optimizing energy consumption for MPSoCs based on Dynamic Voltage Frequency Scaling (DVFS). For example, the work in [39] presents a general concept for a power management controller for per-core DVFS in heterogeneous MPSoC, which allows switching between several on-chip supply voltage levels. It does not pay attention to the schedule of computation tasks. The work in [40] studies the problem of jointly optimizing DVFS and Dynamic Power Management (DPM) with

scheduling to minimize the overall energy consumption. Due to the joint consideration of DPM, their optimization model is significantly different from ours, thus their method cannot be simply adapted to the problem considered in this paer. In this paper, we optimize the DVFS for reducing energy consumption while jointly taking into account security constraints. It is worth mentioning that the work in [40] is complementary to our work and thus incorporating the techniques proposed in [40] can potentially further improve the performance of our method.

III. Preliminaries

Let $P = \{p_1, p_2, \cdots, p_{|P|}\}$ be the set of available processor cores, where $|P|$ is the upper bound on available cores. The $|P|$ cores are provided by $|VE|$ ($|VE| \leq |P|$) vendors, and the set of available vendors is denoted as $VE = \{ ve_1, ve_2, \cdots, ve_{|VE|}\}$, where VE is an input set determined by user or designer. We assume that a vendor provides multiple cores of the same type, and each of the core can run at multiple speeds. In other words, cores of the same vendor have the same list of operating speeds. Let $SP_k = \{s_{k,1}, s_{k,2}, \cdots, s_{k,n_k}\}$ be the set of speeds provided by vendor ve_k, where n_k is the number of speeds ve_k can provide and speeds are sorted in increasing order. We assume that each vendor provides one type of core. This can be easily accommodated to cases where a single vendor provides different types of cores, by considering the speeds of different cores separately and dynamically selecting the most suitable type of core for the vendor during vendor assignment process.

We denote the input application task graph as a directed acyclic graph (DAG), $G = (V, E)$, where $V = \{v_1, v_2, v_3, \cdots, v_n\}$ is the set of tasks, $n = |V|$, $E = \{e(i, j)\}$ is the set of edges in task graph that captures data dependencies among tasks, and $m = |E|$. There exists an edge $e(i, j)$ from task v_i to v_j if task v_j depends on task v_i in terms of data dependency. Let ST_i and FT_i be the start time and end time of task v_i, then $FT_i = ST_i + t_i$, where t_i is the execution time of task v_i. The data dependency $e(i, j)$ between tasks v_i and v_j requires that $ST_j \geq FT_i$. Therefore, the schedule length of the given application can be represented as $\max_{v_i \in V}\{FT_i\}$.

Notation $x_{i,j} = 1$ indicates that task v_i is scheduled to core p_j, and $x_{i,j} = 0$ otherwise. Similarly, $y_{j,k} = 1$ indicates that core p_j is provided by the vendor ve_k, while notation $z_{i,u} = 1$ indicates that the task v_i is executed at the u-th speed provided by its associated core.

A. Threat Model and Security Constraints

We adopt the same threat model in [3], wihch mainly focuses on detecting (or muting) malicious modifications. Specifically, an attacker in a third party house may insert a hardware trojan into the 3PIP core to modify function, or create a backdoor to leak confidential information, etc. The trojan may cause the task running on the malicious 3PIP to either produce incorrect output or generate additional output to trigger trojans in another 3PIP core from the same vendor. As a result, the following two cases can occur at runtime: 1) Due to the insertion of the malicious logic into a 3PIP core, the outputs of the infected cores will be altered at some undetectable points in the cores, 2) Trojans distributed on multiple cores (in order to reduce the chance of being detected) can also form malicious communication paths where a malicious logic in one core triggers the trojans in another core using a secret communication channel (e.g., when some illegal values are written to certain memory space).

In this work, we target embedded platforms which executes application-specific tasks with tight performance and energy constraints. Such platforms are widely-used in automotive, safety-critical systems, or real-time edge computing platforms (e.g. smart sensors). In such systems, the designer has the prior knowledge of the application and its runtime constraints, which allows him/her to perform security-aware customizations to meet performance requirements and reduce energy consumption. Thus, we assume that the defender is in the design house, has knowledge of the task graph, and the ability to bind tasks to cores, choose cores from different vendors, set the operating speeds to the cores, and design glue logic to improve security.

Similar to [3], we introduce vendor diversity as security constraints in the MPSoC design stage to address the above-mentioned two types of Trojans, i.e. to enable the detection of trojans or mute their effects during runtime. The first constraint is called duplication-with-diversity, which duplicates each computation task on two cores from different vendors and compare the outputs of the two cores at runtime. The presence of malicious logic is detected when there is a mismatch in the outputs. The rationality lies in the fact that different vendors do not tend to collude, and thus their cores will not contain the same hardware trojan. As such, the cores will not produce the same incorrect output under the same input if malicious logic is present. Once mismatch of outputs is observed, a security flag will be raised and all the subsequent tasks are terminated. Thus after duplication of graph $G = (V, E)$, $V = \{v_1, v_2, v_3, \cdots, v_{2n}\}$ and E are also duplicated based on G. Formally, the security constraint duplication-with-diversity can be formulated as follows.

$$(x_{i,j} \cdot y_{j,k}) \cdot (x_{i+n,h} \cdot y_{h,k}) = 0, \quad p_j, p_h \in P, ve_k \in VE, 1 \leq i \leq n \quad (1)$$

where $v_i, v_{i+n} \in V$ and task v_{i+n} is the duplicated copy of v_i, and $x_{i,j} \cdot y_{j,k} = 1$ indicates that task v_i is scheduled to core p_j whose vendor is ve_k. Thus, formula (1) restricts that tasks v_i and v_{i+n} cannot be scheduled to cores of the same vendor.

Isolation-with-diversity addresses trojans that are split into several segments which are distributed in multiple 3PIP cores in order to reduce the chance of being detected. The trojans are distributed across different cores and are triggered by illegal communication between these cores. Isolation-with-diversity mutes the effects of the Trojan segments collusions by preventing communication of IP cores from the same vendors since malicious cores from different vendors cannot collude with each other. We adopt the same online verification scheme as in [3], wherein if two cores from the same vendor access the same data object through a secret communication channel, a security flag will be raised indicating an invalid communication path. For example, such checks can be achieved using an ownership vector for shared memory MPSoCs as discussed in

[3]. To mute malicious communication paths, the constraint is incorporated into the task scheduling in the following way: for a given task graph, any task pairs which have data dependency cannot be scheduled to two cores from the same vendor. In this way, the isolation-with-diversity constraint can mute the effect of Trojans distributed in multiple cores. Formally, the constraint can be formulated as:

$$(x_{i,h} \cdot y_{h,k}) \cdot (x_{j,u} \cdot y_{u,k}) = 0, \quad e(i,j) \in E, u \neq h \tag{2}$$

It is worth mentioning that the second constraint allows two tasks with data dependency to be scheduled on the same core, as this will not result in collusion of multiple cores since there are no inter-core communication.

B. Energy Model and Speed Model

The dynamic power of CMOS circuits is characterized by $p = aCv^2 f$ where a is an activity factor, C is the loading capacitance, v is the supply voltage, and f is the clock frequency [41]. The core speed s is linearly proportional to the clock frequency f ($s \propto f$), and the supply voltage v is related to f in the following manner: $f \propto v^\gamma$ ($0 < \gamma \leq 1$), which implies that $v \propto f^{1/\gamma}$ [42]. As such, we know that the power consumption p satisfies that $p \propto f^\alpha$ and $p \propto s^\alpha$, where $\alpha = 1 + 2/\gamma \geq 3$. For the ease of discussion, we assume $v = bf^{1/\gamma}$ and $s = cf$, where b and c are constants.

Let r_i ($1 \leq i \leq n$) be the computation requirement (e.g., the number of CPU cycles or the number of instructions) of task v_i. If v_i is executed at speed s_i, then the execution time t_i is calculated as $t_i = r_i/s_i$, and the energy consumption is $\varepsilon_i = p_i t_i = r_i(aCv_i^2 f_i/s_i) = (aC/c)r_i(bf_i^{1/\gamma})^2 = (ab^2C/c^\alpha)r_i s_i^{\alpha-1}$. Since constants a, b, c, and C only contribute to the effect of scaling [44], we can simply assume that

$$\varepsilon_i = r_i \cdot s_i^{\alpha-1} = r_i \cdot s_i^2 \tag{3}$$

Similar to existing work [44], [45], we set $\alpha = 3$ during simulation. Generally, processor cores have two types of speed models: 1) continuous model, where the cores can have arbitrary speeds which can be any value in the interval $[s_{min}, s_{max}]$, 2) discrete model, where cores have a finite number of speeds in the set $\{s_1, s_2, s_3, \cdots, s_d\}$. In this paper, discrete speed model is used so that each vendor only provide a few discrete speeds for user to choose.

Problem Description: Given the upper bound on the number of cores, a set of vendor-specific cores with multiple discrete operating speeds, an upper bound of the schedule length DL, and the application task graph $G(V, E)$, where each task v_i has computation requirement r_i and E represents the data dependency among tasks, determine: 1) the optimal schedule of all tasks for the cores, 2) the vendor of each core and 3) the execution speed of each task, with the objective of minimizing the total energy consumption under the schedule length constraint DL as well as the two security-driven diversity constraints. In other words, the problem needs to determine the values of $x_{i,j}$, $y_{j,k}$ and $z_{i,u}$ for any i, j, u with respect to $v_i \in V$, $p_j \in P$ and $ve_u \in VE$. Formally, the considered problems can be formulated as the following optimization problems.

$$Min \sum_{v_i \in V} \varepsilon_i \tag{4}$$

s.t.

(1)

(2)

$$\max_{v_i \in V} FT_i < DL \tag{5}$$

$$ST_j \geq FT_i, \quad v_i, v_j \in V, e(i,j) \in E \tag{6}$$

$$ST_i + \sum_{p_j \in P, ve_k \in VE, s_{i,u} \in SP_k} \frac{r_i \cdot (x_{i,j} \cdot y_{j,k} \cdot z_{i,u})}{s_{k,u}} \leq FT_i, \ v_i \in V \tag{7}$$

$$x_{i,j} \cdot x_{i',j} \cdot (F_i - S_{i'})(F_{i'} - S_i) \leq 0, \ v_i, v_{i'} \in V, i \neq i', p_j \in P \tag{8}$$

$$\sum_{p_j \in P} x_{i,j} = 1, \quad v_i \in V \tag{9}$$

$$\sum_{ve_k \in VE} y_{j,k} = 1, \quad p_j \in P \tag{10}$$

$$\sum_{s_u \in S_k} x_{i,j} \cdot y_{j,k} \cdot z_{i,u} = 1, \quad v_i \in V \tag{11}$$

Constraints (1) and (2) are the the previously defined security constraints, formula (5) is schedule length constraint and (6) captures the dependency constraints among consecutive tasks. Constraint (7) imposes that the finish time FT_i of task v_i must be no less than the task start time ST_i plus the execution time $\sum_{p_j \in P, ve_k \in VE, s_{k,u} \in SP_k} \frac{r_i \cdot (x_{i,j} \cdot y_{j,k} \cdot z_{i,u})}{s_{k,u}}$, where $(x_{i,j} \cdot y_{j,k} \cdot z_{i,u}) = 1$ indicates that the task v_i is executed by core p_j at the u-th speed of vendor ve_k. The constraint (8) ensures that any two tasks scheduled to the same core cannot be executed in parallel. The constraint (9) requires that each task is scheduled to single core, and constraint (10) ensures that each core is provided by a single vendor. The constraint (11) indicates that if task v_i is scheduled to a core whose vendor is ve_k ($x_{i,j} \cdot y_{j,k}$), the speed for running task v_i can only be one value from the set S_k, i.e. the speed options that vendor ve_k provides.

IV. Methodology

The proposed approach, which consists of three stages, is illustrated in Fig. 1. In the first stage, two security constraints are embedded into task scheduling and Core-Conflict-Graph (CCG) coloring to protect task execution from trojan attacks. That is, all tasks (including duplicated ones) are scheduled to a given number of cores such that 1) original task and its duplicate are not scheduled on cores of the same color (security constraint 1), 2) inter-core communication only happens among cores of different colors (security constraint 2). In the second stage, we determine the vendor of each core by considering the computation requirement of tasks on each core, the distribution of available speeds of each vendor, as well as the dependency among tasks. In the last stage, we assign discrete speed to each task based on the available speeds provided by the corresponding vendor, with the objective of minimizing energy consumption without violating schedule length constraint.

A. Stage 1: Security-Constrained Task Scheduling

In this section, we present our proposed approach for task scheduling with the two security constraints. The security constraints will be guaranteed by employing vendor diversity during scheduling, i.e., tasks are scheduled on cores from different vendors. Specifically, the security diversity DUPLICATION-WITH-DIVERSITY is achieved by duplicating each computation task and placing tasks of original copy and the duplicated copy on cores from different vendors. The security diversity ISOLATION-WITH-DIVERSITY that prevents illegal communication is achieved by allowing communication only between cores from different vendors to prevent collusion of trojans in different cores. Note that if two dependent tasks are located on the same core, it can be regarded that there is no communication among the two tasks. In contrast to the work in [1], we aim at minimizing the total energy cost while guaranteeing security constraints and schedule length.

In the proposed scheduling algorithm, denoted as SCSA, the tasks are processed in a particular order, which is maintained using a priority queue. As shown in Algorithm 1, the proposed scheduling algorithm iteratively selects the task with highest priority and identify the best candidate core to host the task using a dynamic priority mechanism. The process repeats until the priority queue becomes empty.

In the algorithm SCSA, the security constraints are imposed at the stage of selecting a candidate core for hosting a task. The first security constraint, DUPLICATION-WITH-DIVERSITY, requires that the task v and its duplicated task v' cannot be hosted by cores from the same vendor. The second security constraint, ISOLATION-WITH-DIVERSITY, requires that, if two tasks v_i and v_j with dependency relationship are hosted by two cores p_i and p_j respectively, then p_i and p_j cannot be from the same vendor. A Core-Conflict-Graph CCG is constructed to capture the conflicting relationship among different cores. If two cores have an edge in CCG, this indicates that one of the cores hosts a task, say v, and the other core hosts the duplicated task v', or there exists communication between the two cores. In CCG, any adjacent node pair (conflicted cores) cannot be of the same vendor in order to meet the security constraints. If we color adjacent nodes in CCG with different colors, then the minimum number of required colors indicates the minimum number of required vendors. CCG is initialized as a set of isolated cores without any links. During the scheduling of a task, conflicting links may be created between pair of cores, e.g., p_i and p_j, if p_i is assigned a task that conflicts with a task on p_j. An Upper Bound of Maximum Clique Size (UB) checks whether the solution violates the vendor number constraint [3], i.e. there will be insufficient vendors if the UB of the graph CCG exceeds the vendor upper bound $|VE|$. After CCG is obtained, we use the standard coloring algorithm [43] to color the CCG, where each color represents a vendor, and cores of the same color are from the same vendor.

1) Determining Priority of Tasks to Schedule: For a given non-trivial task graph, the scheduling order of tasks are determined based on four metrics, i.e., task layer, task criticality, number of its successors, and the number of its critical successors. The layer of task v is defined as the number of hops on the longest path from the start node to v, where the start node is v's predecessor that has no parent node. A task is a critical task if it is on a critical path of the graph. Based on above metrics, the following rules are utilized to calculate the priority of a task: (1) A task in the i-th layer has higher priority than any task in the $(i+1)$-th layer. In fact, by taking layer as a priority indicator, the data dependencies between the tasks can be guaranteed. (2) A critical task has higher priority than the less critical tasks in the same layer, as critical tasks affect the schedule length. (3) If rules (1) and (2) cannot differentiate the priority of two tasks, the task v that has more critical successors is given higher priority, because executing v earlier will enable more critical task to be executed sooner. (4) If all the above rules cannot differentiate the priority of two tasks, the task that has more successors is given higher priority.

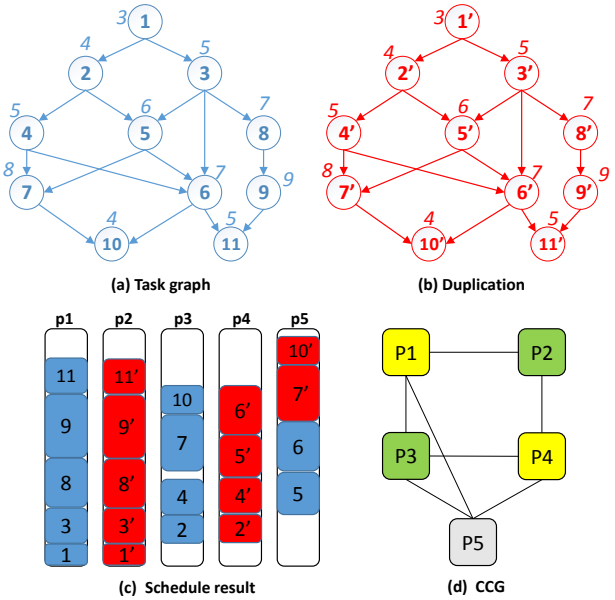


Fig. 1. Example of security constrained scheduling, (a) original graph, (b) duplicated graph, (c) the schedule results and (d) CCG which is colored with three colors. The priority queue is: 1, 1', 3, 3', 2, 2', 8, 8', 4, 4', 5, 5', 9, 9', 6, 6', 7, 7', 11, 11', 10, 10'.

2) Candidate Core Selection: Now we introduce how to select a core for hosting a given task v from core candidate set (P_{cs}). For each task to be scheduled, its P_{cs} is calculated based on the two security constraints: if no security constraint is violated by placing task v_i on core p_j, then p_j will be put into v_i's P_{cs}. Let $S_{v_i} = \{v_{i_1}, v_{i_2}, \cdots, v_{i_{|S_{v_i}|}}\}$ be the set of v_i's parent nodes, which are hosted on core $p_{j_1}, p_{j_2}, \cdots, p_{j_{|S_{v_i}|}}$ (can have repeated cores) respectively. Let $P_v = \{p_{j_1}, p_{j_2}, \cdots, p_{j_{|S_{v_i}|}}\}$, tp_{j_k} be the earliest available time for running new task on core p_{j_k}, and T_{ready} be the latest end time of v_i's parent tasks, i.e. $T_{ready} = \max_{v_i \in S_{v_i}}\{FT_i\}$.

The proposed method to select a core for hosting a given task v from candidate set (P_{cs}) is based on the following three steps.

(1) Calculate P'_{cs} as a subset of P_{cs} such that, for each $p \in P'_{cs}$, placing v to p won't increase maximum clique size MCS of CCG. If $MCS \geq UB$, we reduce set P_{cs} to P'_{cs},

meaning that we consider cores in P'_{cs} with higher priority in order to guarantee the constraint on the number of available vendors while minimizing the scheduling length.

(2) If $T_e = \min_{p_i \in P_v \cap P_{cs}} \{tp_i\} \leq T_{ready}$, which indicates that at least one core in P_v can process task v at time T_{ready}, then task v will be allocated to the core $p_i = \arg\max_{p_i \in P_v \cap P_{cs}, tp_i \leq T_{ready}} \{tp_i\}$ with start time $ST_v = T_{ready}$. That is, among all cores p_i that can allow task v to start at time T_{ready} (i.e., $tp_i \leq T_{ready}$), we select the core with latest available time so as to minimize the idle time. Also, cores with earlier available time can serve other tasks without incurring idle time in order to minimize the overall schedule length.

(3) If $T_e = \min_{p_i \in P_v \cap P_{cs}} \{tp_i\} > T_{ready}$, we select the core with the earliest available time T_e as the core to host task v, so that task v can start at time T_e. We consider cores that are not within the set P_v only when there is no core P_v that allows task v to start earlier than T_{ready}, i.e., $T_e > T_{ready}$. This is undertaken to increase the chance of placing neighboring tasks on the same core with the aim of reducing the communication overhead as well as reduce the overall schedule length.

The above procedure first considers $P_v \cap P_{cs}$ to select a candidate core as in step 1), and then search the entire P_{cs} if step 1) cannot find a valid core that can execute current task v with start time no later than T_{ready}. Both T_{ready} and P_v are dynamically updated whenever a task is mapped to a core. Fig.1 shows an example of security constrained scheduling. Calculating priority queue before scheduling runs in $O(n^2)$, then, selecting the candidate core for each task runs in $O(n)$, thus it needs $O(n^2)$ to select cores for all tasks. Therefore, time complexity of Algorithm 1 is $O(n^2)$, where n indicates the number of computation tasks in a given application.

After task scheduling, CCG is constructed to represent the conflicting relationship among cores. If two cores have an edge in CCG, this indicates one of the core hosts a task, say v, and the other core hosts the duplicated task v', or there exists communication between the two cores. Two conflicted cores must come from different vendors in order to meet the security constraints. After CCG is obtained, we use the standard coloring algorithm [5] to color the CCG, where each color represents a vendor, and cores of the same color are from the same vendor.

Algorithm 1: SCSA (Security Constrained Scheduling Algorithm)

Input: Task graph G (original and duplicated), number of cores $|P|$, number of vendors K.
Output: Schedule of tasks to cores $x_{i,j}$, and core conflict graph CCG.

```
begin
  Step 1: Determine priority queue Q for task scheduling
  Step 2: Scheduling
  Initialize CCG as isolated cores;
  while Q ≠ ∅ do
      v ← task of highest priority in Q;
      Compute set P_cs for task v;
      P'_cs ← ∅;
      for each p in P_cs do
          if placing v to p won't increase UB of CCG then
              P'_cs ← P'_cs ∪ {p};
      If P'_cs ≠ ∅ then P_cs ← P'_cs;
      P ← best candidate from P_cs based on core selection
       rules;
      Schedule task v to core p;
      Update CCG;
  end
```

B. Stage 2: Vendor Assignment

The previous stage partitions all cores into several groups such that cores of the same group are of the same color (i.e. cores of the same group are from the same vendor). In this stage, the vendor for each core group (color) will be determined, i.e. which color corresponds to which vendor. Due to the heterogeneous computing capability of cores from different vendors, and unbalanced computation requirement of different cores, the vendor assignment problem which assigns vendors to core groups/colors should be carefully designed so as to maximize the energy efficiency. Intuitively, a core with large volume of computation requirement should be from a vendor that can provide stronger computing capability. We first analyze the optimal speeds that can minimize the total energy consumption for processing the tasks using continuous speed model. Next, an efficient heuristic is developed to assign vendors to cores with the objective of minimizing the deviation between the required speeds of the tasks and the available operating speeds provided by vendor-specific cores.

1) Estimating the Required Speed for Tasks: The work in [44] investigated the power (speed) assignment to minimize the overall energy consumption under schedule length constraint for processing a set of dependent tasks on multicores. It analyzed the optimal speed assignment for level by level scheduling algorithm LL-A. The input task graph is partitioned into h layers, where the tasks in set VS_l (consists of tasks in the l-th layer) cannot start until all tasks in VS_{l-1} have finished. The algorithm further partitions VS_l ($1 \leq l \leq h$) into p disjoint sub-sets to run on the p cores, i.e., $VS_l = \{VS_{l,1}, VS_{l,2}, \cdots, VS_{l,i}, \cdots, VS_{l,p}\}$ where $VS_{l,i}$ indicates the sub-set that has been scheduled to core p_i. Note that $VS_{l,i}$ could be empty, i.e. no tasks of layer VS_l has been assigned to core p_i.

In algorithm LL-A, each layer VS_l is assigned with an unknown time period T_l ($1 \leq l \leq h$ and $\sum_{l=1}^{h} T_l = DL$), thus the total energy consumption can be represented as

$$\varepsilon = \sum_{l=1}^{h} \left(\frac{R_{l,1}^{\alpha} + R_{l,2}^{\alpha} + \cdots + R_{l,p}^{\alpha}}{T_l^{\alpha-1}}\right) \tag{12}$$

where $\alpha = 3$, $R_{l,i}$ represents the total computation requirement of tasks in set $VS_{l,i}$, which is known in advance. Then we can obtain the assignment $T_1, T_2, \cdots, T_h$ ($\sum_{l=1}^{h} T_l = DL$) which minimizes ε

$$T_l = \left(\frac{A_l^{1/\alpha}}{A_1^{1/\alpha} + A_2^{1/\alpha} + \cdots + A_h^{1/\alpha}}\right) \cdot DL \tag{13}$$

where $A_l = R_{l,1}^{\alpha} + R_{l,2}^{\alpha} + \cdots + R_{l,p}^{\alpha}$, for $1 \leq l \leq h$, and DL is known as a constraint, so the minimized energy consumption

is

$$\varepsilon = \frac{(A_1^{1/\alpha} + A_2^{1/\alpha} + \cdots + A_h^{1/\alpha})^\alpha}{DL^{\alpha-1}} \quad (14)$$

by using the above time assignment for each level. The performance ratio is

$$\beta \leq (1 + \frac{p \cdot h \cdot r*}{VS})^\alpha \quad (15)$$

where the input task graph is partitioned into h layers, $r^* = \max(r_1, r_2, \cdots, r_n)$ is the maximum task execution requirement, VS is the total computing requirement of all tasks. For a fixed p, the performance ratio β can be arbitrarily close to 1 as $VS/(hr^*) \to \infty$.

Based on the obtained $T_1, T_2, \cdots, T_h$ we can easily calculate the *required speed* of each task. Specifically, a task in $VS_{l,i}$ should be executed at speed $R_{l,i}/T_l$.

2) Vendor Assignment via Minimizing Speed Deviation: At this stage, each core contains a set of tasks, and all cores are partitioned into many groups such that each group is associated with a color. We use a color to represent a group of cores instead of considering each core separately. For the ease of presentation, we define the required speeds of a color (a group of cores) as the required speeds of tasks that are associated with the color. In this section, we propose an approach to determine which color corresponds to which vendor, based on the required speeds of each color and the provided operating speeds of each vendor-specific core.

Note the required speeds of each color, obtained through the theoretical analysis method presented in the previous section, is under continuous speed model meaning that any speed can be available when required, and the vendors' available speeds are under discrete speed model indicating that only several discrete speeds are available. Vendor assignment maps the available vendors to each group (color) of cores and find a speed for each task from the provided speeds of the assigned vendor with the objective of minimizing the overall deviation D of the assigned speeds from the available speeds. Formally, the vendor assignment tries to minimize

$$D = \sum_{v_i \in V, ve_u \in VE} ((s_i' - bs_{i,u})^2 \cdot x_{i,j} \cdot y_{j,u} \cdot \frac{r_i}{\overline{r}}) \quad (16)$$

where s_i' is the required speed of task v_i obtained using the approach in the previous section, $x_{i,j}y_{j,u} = 1$ if v_i is associated with vendor ve_u, $bs_{i,u}$ is the best speed for task v_i from ve_u's available operating speeds, $\overline{r}$ is the average computation requirement of all tasks and is used for normalization. Clearly, small D leads to better performance.

Suppose task v_i is assigned to vendor ve_u, then $bs_{i,u}$ is calculated as the minimum speed that is larger than s_i' from vendor ve_u's speed set. If none of ve_u's speeds is larger than s_i', ve_u's maximum speed will be selected. Thus, after the vendor assignment, all the variables $x_{i,j}$, $y_{j,u}$, $bs_{i,u}$ as well as D can be determined. The assignment that minimizes D is the best solution.

An example of vendor assignment is given in Table I-III, where Table I shows the vendors' provided operating speeds, and Table II shows the required speeds by different core groups (colors), i.e., the required speeds of all tasks associated with each color, obtained based on theoretical analysis. Table III illustrates two assignment (assignment 1: color1 $\leftarrow ve_1$, color2 $\leftarrow ve_2$, color3 $\leftarrow ve_3$; assignment 2: color1 $\leftarrow ve_1$, color2 $\leftarrow ve_3$, color3 $\leftarrow ve_2$). Assignment 1 is a better solution due to lower D, assuming all tasks are of the same computation requirement.

TABLE I
Vendors and their provided speeds

vendors speed level	s_1	s_2	s_3	s_4
ve_1	3	4	5	6
ve_2	5	7	9	11
ve_3	4	6	8	12

TABLE II
Colors and their required speeds

color	required speeds
color 1	2.31, 3.45, 4.37, 4.38, 6.59
color 2	4.27, 4.45, 5.66, 5.69, 6.01, 6.03, 8.11, 9.12
color 3	5.77, 5.98, 6.10, 8.79, 8.91, 10.02, 11.99

Due to the NP-hardness of the vendor assignment, we develop a heuristic, denoted as Algorithm 2, to generate approximate solutions. To reduce computation redundancies, we define $D_{a,b}$ as the deviation of $color_a$'s required speeds from vendor ve_b's available speeds.

$$D_{a,b} = \frac{1}{|V_a|} \sum_{v_i \in V_a} (s_i' - bs_{i,b})^2 \cdot \frac{r_i}{\overline{r}} \quad (17)$$

where V_a is the set of tasks of $color_a$, s_i' is the required speed by task v_i, $\overline{r}$ is the average computation requirement of tasks and is used for normalization, $bs_{i,b}$ is the best candidate speed for v_i from ve_b's speeds.

Algorithm 2: `Vendor Assignment`

Input: Number of vendors K, vendor set VE, color $color_j$ for $j = 1, 2, ..., K$, speed set SP_k of vendor ve_k.
Output: $y_{i,k}$ and $z_{i,u}$ for $v_i \in V, ve_k \in VE, sk_u \in SP_k$
begin
Calculate $D_{a,b}$ for $1 \leq a, b \leq |K|$;
Rank all the colors (rows) according to their criticality;
while array $D_{a,b}$ is not empty **do**
  $color_j \leftarrow$ the most critical row in $D_{a,b}$;
  $k \leftarrow \arg\min_{1 \leq b \leq K}\{D_{j,b}\}$;
  Assign vendor ve_k to color $color_j$;
  Set $y_{i,k} = 1$ for all cores that are colored with $color_j$, set $z_{i,u}$ based on the above mentioned methods;
  Remove the j−th row and k-th column from array $D_{j,b}$;
end

In our second algorithm, denoted as Algorithm 2, we first calculate $D_{a,b}$ for $1 \leq a, b \leq K$ where K is the number of vendors/colors. Then we rank all the colors according to their criticality. Next, we iteratively find the most critical color (that has most computation overhead), $color_j$, and determine the minimum $D_{j,k}$ from the j-th row of array $\{D_{a,b} | 1 \leq a, b \leq K\}$. Vendor ve_k is assigned to color $color_j$, so that $y_{i,k}$ is set to 1 for all cores that are colored with color $color_j$, and the

TABLE III
Comparison of two assignments in terms of deviation between colors' required speeds and vendors' provided speeds.

Color	assignment 1: deviation D = 39.9		assignment 2: deviation D = 44.1	
	Vendor	Assigned speeds	Vendor	Assigned speeds
Color1	ve_1	3, 4, 5, 5, 6	ve_1	3, 4, 5, 5, 6
Color2	ve_2	5, 5, 7, 7, 7, 7, 9, 11	ve_3	6, 6, 6, 6, 8, 8, 12, 12
Color3	ve_3	6, 6, 8, 12, 12, 12, 12	ve_2	7, 7, 7, 9, 9, 11, 11

speeds for running corresponding tasks on these cores, i.e., $z_{i,u}$, are determined based on the above mentioned methods. Then we remove the j-th row as well as the k-th column from array $\{D_{a,b}\}$ and repeat the process until all rows and columns are removed.

After vendor assignment, it is possible (with very low probability) that a task v_i's required speed (through analysis using previously mentioned approach) is larger than the maximum speed that its core can provide. This can potentially increase the total schedule length if v_i is on the critical path, and thus leads to violation of schedule length constraint. To address this problem, we refine the initial speeds to meet schedule length by accelerating tasks on critical path, i.e., assign higher speeds to these tasks. Specifically, if schedule length is violated, we iteratively accelerate a task until the schedule length constraint is met. Assume that task v_i can be accelerated to speed s' from s, we calculate the benefit as $bf(v_i)$ as the ratio of reduction in schedule length over the increase in energy consumption, caused by accelerating task v_i. In each iteration, the task with maximum benefit will be selected for acceleration. The process repeats until the schedule length constraint is met. Note we only consider cases where the schedule length constraint can be met if all tasks run at the highest speed on the corresponding cores.

In algorithm 2, calculating D runs in $O(n \cdot K)$ and vendor assignment by row/column deletion runs in $O(K^2)$, where n is the number of tasks, K is the vendor number and $n > K$. In addition, in case of violating schedule length, the algorithm needs to iteratively accelerate critical tasks, and a loose upper bound on the number of iterations is $n \cdot \delta$, where δ is the maximum number of core operating speeds.

C. Stage 3: Speed Refinement to Improve Energy Efficiency

Due to the data dependency among tasks, there exist idle periods on cores as a task has to wait for all its parent tasks to complete before commencing execution, especially when task operating speeds are not properly assigned. This provide opportunities to further reduce the total energy consumption by slowing down the execution speeds of some uncritical tasks, which are usually followed by idle periods. In this section, we propose methods to refine the speeds in order to guarantee the schedule length as well as further reducing the total energy consumption.

In order to fully exploit the optimization opportunities, we iteratively decelerate the most profitable task by assigning lower speed to it, as long as the schedule length constraint is satisfied. All the tasks are maintained in two groups, where

Algorithm 3: `SR(Speed Refinement)`

Input: Application graph G, variables $x_{i,j}$, $y_{j,h}$, and $z_{i,u}$, bound DL on schedule length.
Output: refined $z_{i,u}$ for all tasks.

begin
Calculate current SL based on dependency constraint;
while $(SL \leq DL)$ **do**
  for each task v_i **do**
    for each candidate speed s_u **do**
      Calculate ΔE and ΔSL for accelerating v_i;
      if decelerating $\Delta SL = 0$ **then**
        Put $< v_i, s_u >$ into $ListA$;
      else if $SL + \Delta SL \leq DL$ **then**
        Put $< v_i, s_u >$ into $ListB$;
  if $ListA \neq \phi$ **then**
  Decelerate the task in $ListA$ with $\max_{v_i \in ListA}\{\Delta E\}$;
  else Decelerate the task in $ListB$ with
  $\max_{v_i \in ListB}\{\frac{\Delta E}{\Delta SL}\}$;
  Update SL;
end

$ListA$ contain all tasks whose deceleration will not increase the overall schedule length, while the $ListB$ contains the remaining tasks. Both groups are dynamically updated in each iteration. If $ListA$ is not empty, the task with maximum energy saving ΔE will be selected from $ListA$, otherwise, the task with maximum $\Delta E/\Delta SL$ will be selected from $ListB$, where ΔE is the energy saved by decelerating the task, and ΔSL is the increase in the schedule length. This process repeats until no more task can be decelerated without violating the constraint of overall schedule length. The running time of Algorithm 3 depends on whether energy savings can be achieved. Better results can be obtained if more iterations are executed in the algorithm.

V. Experimental Results and Analysis

The two approaches reported in [3], i.e. the straightforward scheduling approach (denoted as `B1`) and the cluster-based with clique constraint (denoted as `B2`), are used as the baseline. The differences between the two algorithms are as follows. Algorithm B1 (straightforward scheduling) aims to meet the security constraints at the finest granularity, where the duplication-with-diversity constraint is added to each node in the task graph and isolation-with-diversity constraint is added to each edge in the task graph. As a result, secured communications (edges) are enforced between the 3PIP cores of different vendors. Algorithm B2 (cluster-based schedule with clique constraint) allows the scheduler to assign dependent

tasks on the same core to reduce the number of inter-core communications while satisfying the security constraints. This is achieved by grouping dependent tasks on critical paths into a cluster and scheduling the entire cluster to a single core, thus dependent tasks are scheduled either to the same core (for the intra-cluster cases) or across different vendor cores (for the inter-cluster cases). Please refer to [3] for details on the two algorithms. Let SL_{B1} and SL_{B2} be the schedule length obtained by B1 and B2. In order to make a fair comparison with the baseline algorithms, we set $\min\{SL_{B1}, SL_{B2}\}$ as the schedule length bound for our method and evaluate the energy consumption of each algorithm. In addition, we solved the Mixed-Integer Linear Programming (MILP) formulation of security-constrained scheduling problem using CPLEX [46] to obtain the optimal solutions, on small-scale instances of 25 tasks using 8 processing cores provided by at most 4 vendors. CPLEX is a widely-used optimization software package which forms complex optimization problems as mathematical models and uses advanced optimization algorithms to find solutions. CPLEX is well known for its high performance in solving large, real-world optimization problems. We compare the optimal solution computed by CPLEX with the results of the proposed algorithms. The time limit was set to 24 hours per run using 20 cores on a workstation. CPLEX 12.5 with default settings was used to solve the problems.

The test set is composed of both realistic application task graphs and synthetic task sets generated using TGFF-3.5. The three real applications include sparse matrix solver (98 nodes), robot control (90 nodes) and a part of fppp (336 nodes) in the SPEC benchmarks [47]. TGFF is used to generate non-trivial instances of task sets that are commonly used, where the number of tasks are set to 50, 100, 150, 200, 250, 300 respectively. For each graph size, graphs with indegree/outdegree of 3/3, 4/4, 5/5, 6/6, 7/7 are generated. We set the maximum speed of all vendors as $s_1^m > s_2^m, \cdots, s_K^m$ with $s_j^m = (1-(j-1)\delta)s_1^m$, where we set $\delta = 10\%$ when $K \leq 6$ and $\delta = 7.5\%$ when $7 \leq K \leq 10$. We set the vendor ve_k's speeds $s_{k,1} > s_{k,2} > \cdots > s_{k,n_k}$ with $s_{k,j} = (1-(j-1)\sigma)s_{k,1}$, where $s_{k,1} = s_k^m$, $\sigma = 10\%$, $n_k = 5$ for $1 \leq k \leq K$. We assumed that the underlying MPSoC platform can accommodate up to 12 cores.

Table IV compares the three methods, i.e. B1, B2 and our method, on three real application graphs. The CPLEX method failed to produce the result within 24 hours for each of the three real applications. In order to obtain the results, we first run the two baseline algorithms, i.e. B1, B2 and then set the minimum schedule length of the two algorithms as the bound of our algorithm. The vendor count upper bound is set to 4.

It can be observed from the table, even with the same (or shorter) schedule length, our algorithm significantly outperforms the baseline approaches in terms of energy consumption. For example, in the case of 'fppp', our schedule length is the same as the algorithm B1 and shorter than B2, while we achieve an improvement in energy consumption of 60% and 38.9% over B1 and B2 respectively. Even algorithm B1 often produce shorter schedule length than B2, it leads to more energy consumption and fails to produce feasible solutions for 'robot', as it runs into a situation where no core can be selected to host the current task being scheduled. For a few cases, such as 'sparse', our algorithm obtained longer SL than B1. This is because B1 uses only 3 vendors and it chooses the 3 vendors of highest speeds to minimize SL, while our algorithm uses 8 vendors, which enables it to choose slower-speed cores that leads to longer SL but with lower energy consumption. In terms of algorithm runtime, all the algorithms can produce solution within several second for the cases of 'sparse' and 'robot'. Our method requires nearly 3 mins to output the solution for the case 'fppp' which has 336 tasks. This is because our method performs more complicated techniques with the aim of achieveing better performances.

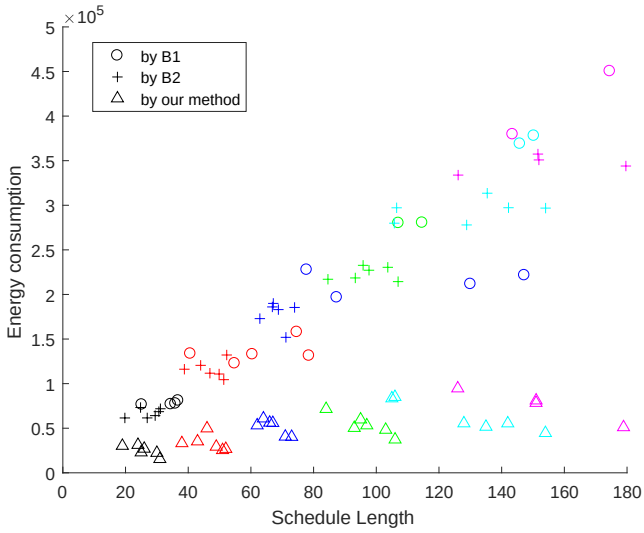


Fig. 2. Comparison on synthetic graphs. The results of B1, B2 and our algorithm are illustrated as circles, pluses and triangles, respectively. Results of the same color are collected on graphs of the same size, where the colors black, red, blue, green, light blue and pink correspond to graph size of 50, 100, 150, 200, 250, and 300 respectively.

Since CPLEX cannot produce feasible solutions for large-scale instances in 24 hours, we first evaluate our approach on small-scale instances (i.e. with 25 tasks) by comparing with baselines B1, B2, and the CPLEX method. In this experiment, 8 cores are utilized and at most 4 vendors are allowed. The upper bound on schedule length of our algorithm and the CPLEX approach is set to the minimum schedule length of the B1 and B2. The heuristic baselines, CPLEX and our algorithm do not obtain exactly the same schedule length (even when using the same bound), because discrete speed model is used (each processor only operates at a few discrete speeds). In general, the obtained schedule lengths of our approach as well as the CPLEX are shorter than both the B1 and B2 as shown in Table V. On the other hand, the schedule length of CPLEX is closer to the schedule bound than our algorithm (our/CPLEX < 1), as it can obtain optimal schedule and DVFS configurations. From the table, it can be observed that methods B1 and B2 produce solutions that lead to energy consumptions that are 2-4 times of the optimal solutions. In contrast, our method achieved performance that are much closer to the optimal energy consumption. Also from the table, we observe that the CPLEX method needs several hours to

TABLE IV
Results on three real application graphs.

Application	#.vendor	schedule length			energy consumption (10^5)			number of vendors			average runtime (s)		
		B1	B2	Our	B1	B2	Our	B1	B2	Our	B1	B2	Our
fppp	4	509.1	426.1	425.9	11.1	9.85	5.31	4	4	4	2.34	2.65	162.2
sparse		125.8	94.3	93.9	2.9	2.6	1.2	3	4	4	2.32	2.3	3.38
robot		INF	154	153.9	INF	3.4	1.5	INF	4	4	2.32	2.30	3.39

#.vendor: the upper bound on the number of vendors;
INF: failed to produce feasible solution;

TABLE V
Comparison with optimal solutions on small-scale instances.

taskNo-In/Out	average Schedule Length (SL)			average Energy Consumption (EC)			average running time (s)			
	B1/CPLEX	B2/CPLEX	our/CPLEX	B1/CPLEX	B2/CPLEX	our/CPLEX	B1	B2	our	CPLEX
25-2/2	1.019	1.003	0.998	4.570	4.003	1.071	2.56	2.30	2.70	26325
25-3/3	1.201	1.010	0.994	2.808	2.544	1.069	1.14	0.91	0.13	27379
25-4/4	1.519	1.007	0.997	2.142	1.942	1.106	1.33	0.52	0.15	35509
25-5/5	1.541	1.016	0.997	2.158	2.124	1.081	1.50	0.33	2.23	63873

[1] B1/CPLEX (or B2/CPLEX, our/CPLEX) indicate the ratio between the algorithm B1 (or B2, our) and the CPLEX method;
[2] taskNo-In/Out: taskNo indicates the number of tasks in the application graph, In/Out indicate the maximum in-degree and out-degree of the graph.

compute the solutions while the other three approaches can output the solutions in very short time. Our method is slightly slower than the baselines B1 and B2 because our algorithm employs more complicated techniques which lead to much better performances.

TABLE VI
Results on synthesized graph with varying In/Out

In/Out	Schedule Length (SL)		Energy Consumption (EC)	
	Our/B1	Our/B2	Our/B1	Our/B2
3/3	0.967	0.993	0.198	0.262
4/4	0.848	0.994	0.215	0.297
5/5	0.543	0.992	0.386	0.355
6/6	0.741	0.999	0.334	0.408
7/7	0.564	0.989	0.341	0.412

Fig. 2 compares the three algorithms in terms of schedule length and energy consumption on synthetic graphs with size ranging from 50 to 300. For each size, graphs with Indegree/Outdegree of 3/3, $\cdots$, 7/7 are generated. In this experiment, there exists a certain number of instances that require 5 vendors for both algorithm B2 and our algorithm. This is required to accommodate the large number of computation tasks and large In/Out-degree of some computation tasks. Thus, we set the vendor bound to 5 for these instances and 4 for all other instances. For each instance, the three algorithms are tested under the same upper bound of vendors. The CPLEX method cannot produce feasible solutions for most cases within 24 hours, thus no result is shown for CPLEX method. The results show that even when our method restricts its schedule length to be smaller than both B1 and B2, it still significantly outperforms the two baseline algorithms in terms of energy consumption EC. In addition, B1 cannot always produce feasible solutions (about 51% instances failed). Similar improvements of our approach over B1 and B2 are observed in Table VI, which shows the comparison on graphs with varying Indegree/Outdegree. The results reveal that our approach significantly improves energy consumption EC, with average improvement of 71% over B1 and 65% over B2, while simultaneously achieving shorter schedule length SL. In addition, smaller Indegree/Outdegree tends to achieve better energy improvement of the proposed algorithm over the baselines.

VI. Conclusion

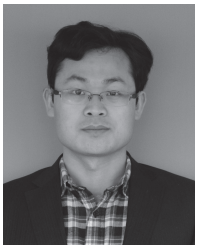
We presented a novel MPSoC design methodology to minimize the energy consumption while jointly considering security constraints when using untrusted 3PIP cores. The security constraints are embedded in task scheduling and a core conflict graph is utilized to model conflicting cores which are assigned to different vendors. We theoretically estimated the optimal required speeds for minimizing the total energy consumption. The optimal speed estimates are utilized in an efficient heuristic for vendor to core assignment. We further reduced the energy consumption via operating speed optimization by exploiting idle periods on cores.

References

[1] W. Wolf, "The future of multiprocessor systems-on-chips," in *Proc. 41st Design Autom. Conf. (DAC)*, pp. 681-685, 2004.
[2] R. Karri, J. Rajendran, K. Rosenfeld, M. Tehranipoor, "Trustworthy hardware: identifying and classifying hardware Trojans," *Computer*, vol. 43, no. 10, pp. 39-46, 2010.
[3] Liu, C., Rajendran, J., Yang, C., Karri, R., "Shielding heterogeneous MP-SoCs from untrustworthy 3PIPs through security-driven task scheduling," *IEEE Trans. Emerg. Top. Comput.*, vol. 2, no. 4, pp. 461-472, 2014.
[4] A. Al-Anwar, Y. Alkabani, M.W. El-Kharashi, H. Bedour, "Hardware Trojan protection for third party IPs," in *Proc. Euromicro Conf. Digit. Syst. Design (DSD)*, pp. 662-665, 2013.
[5] M. Banga, M. S. Hsiao, "A novel sustained vector technique for the detection of hardware Trojans," in *Proc. 22nd Int. Conf. VLSI Design*, pp. 327-332, 2009.
[6] F. Koushanfar, A. Mirhoseini, "A unified framework for multimodal submodular integrated circuits Trojan detection," *IEEE Trans. Inf. Foren. Sec.*, vol. 6, no. 1, pp. 162-174, 2011.
[7] M. Beaumont, B. Hopkins, and T. Newby, "SAFER PATH: Security architecture using fragmented execution and replication for protection against Trojaned hardware," in *Proc. Design Autom. Test Eur. (DATE)*, pp. 1000-1005, 2012.
[8] A. Waksman and S. Sethumadhavan, "Silencing hardware backdoors," in *Proc. 32nd Int. Symp. Security Privacy (SP)*, pp. 49-63, 2011.

[9] Ray, S., Jin, Y., "Security policy enforcement in modern SoC designs," in *Proc. IEEE/ACM Int. Conf. Comput. Aid. Design (ICCAD)*, pp. 345-350, 2015.
[10] Kim, H., Hong, H., Kim, H.S., Ahn, J.H., Kang, S., "Total energy minimization of real-time tasks in an on-chip multiprocessor using dynamic voltage scaling efficiency metric," *IEEE Trans. Comput. Aid. Design*, vol. 27, no. 11, pp. 2088-2092, 2008.
[11] Melani, A., Mancuso, R., Cullina, D., Caccamo, M. and Thiele, L., "Speed optimization for tasks with two resources," in *Proc. Design Autom. Test Eur. (DATE)*, pp. 1072-1077, 2016.
[12] J. Rajendran, V. Vedula, R. Karri, "Detecting malicious modifications of data in third-party intellectual property cores," in *Proc. 52nd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, pp. 1-6, 2015.
[13] Chen, X., Liu, Q., Yao, S., Wang, J., Xu, Q., Wang, Y., Liu Y., Yang, H., "Hardware Trojan detection in third-party digital intellectual property cores by multi-level feature analysis," *IEEE Trans. Comput. Aid. Design*, 2017, DOI: 10.1109/TCAD.2017.2748021.
[14] X. Zhang, M. Tehranipoor, "Case study: Detecting hardware Trojans in third-party digital IP cores," in *Proc. Int. Symp. Hardware Oriented Security Trust (HOST)*, pp. 67-70, 2011.
[15] M. Banga and M. Hsiao, "Trusted RTL: Trojan detection methodology in pre-silicon designs," in *Proc. Int. Symp. Hardware Oriented Security Trust (HOST)*, pp. 56-59, 2010.
[16] J. Jou, C. J. Liu, "Coverage analysis techniques for HDL design validation," in *Proc. Asia Pac. Chip Design Lang.*, pp. 48-55, 1999.
[17] A. Waksman, M. Suozzo, S. Sethumadhavan, "FANCI: Identification of stealthy malicious logic using boolean functional analysis," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, pp. 697-708, 2013.
[18] J. Zhang, F. Yuan, L. Wei, Z. Sun, Q. Xu, "VeriTrust: Verification for hardware trust," *IEEE Trans. Comput. Aid. Design*, vol. 34, no. 7, pp. 1148-1161, 2015.
[19] J. Zhang, F. Yuan, Q. Xu, "DeTrust: Defeating hardware trust verification with stealthy implicitly-triggered hardware Trojans," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, pp. 153-166, 2014.
[20] E. Love, Y. Jin, Y. Makris, "Proof-carrying hardware intellectual property: a pathway to trusted module acquisition," *IEEE Trans. Inf. Forensics Security*, vol. 7, no. 1, pp. 25-40, 2012.
[21] Y. Jin, Y. Makris, "Proof carrying-based information flow tracking for data secrecy protection and hardware trust," in *Proc. IEEE VLSI Test Symp.*, pp. 252-257, 2012.
[22] Jin, Y., Makris, Y., "A proof-carrying based framework for trusted microprocessor IP," in *Proc. IEEE/ACM Int. Conf. Comput. Aid. Design (ICCAD)*, pp. 824-829, Nov. 2013.
[23] M. Banga, M. S. Hsiao, "A novel sustained vector technique for the detection of hardware trojans," in *Proc. Int. Conf. VLSI Design*, pp. 327-332, Jan. 2009.
[24] Malekpour, A., Ragel, R., Ignjatovic, A., Parameswaran, S. "Trojanguard: Simple and effective hardware trojan mitigation techniques for pipelined mpsocs," in *Proc. 54th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, pp. 1-6, Jun. 2017.
[25] Narasimhan, S., Du, D., Chakraborty, R. S., Paul, S., Wolff, F. G., Papachristou, C. A., K. Roy, Bhunia, S., "Hardware Trojan detection by multiple-parameter side-channel analysis," *IEEE Trans. Comput.*, vol. 62, no. 11, pp. 2183-2195, 2013.
[26] Narasimhan, S., Wang, X., Du, D., Chakraborty, R. S., Bhunia, S., "TeSR: A robust temporal self-referencing approach for hardware Trojan detection," in *Proc. IEEE Int. Symp. Hardware Oriented Security Trust (HOST)*, pp. 71-74, Jun. 2011.
[27] Vincent, H., Wells, L., Tarazaga, P., Camelio, J., "Trojan detection and side-channel analyses for cyber-security in cyber-physical manufacturing systems," *Procedia Manuf.*, vol. 1, pp. 77-85, 2015.
[28] JS, R., Ancajas, D. M., Chakraborty, K., Roy, S., "Runtime detection of a bandwidth denial attack from a rogue network-on-chip," in *Proc. 9th Int. Symp. Networks-on-Chip (NOCS)*, Article No. 8, Sep. 2015.
[29] Kulkarni, A., Pino, Y., Mohsenin, T., "SVM-based real-time hardware Trojan detection for many-core platform," in *Proc. 17th Int. Symp. In Qual. Electron. Design (ISQED)*, pp. 362-367, Mar. 2016.
[30] A. Al-Anwar, Y. Alkabani, M. W. El-Kharashi, H. Bedour, "Hardware Trojan detection methodology for FPGA," in *Proc. IEEE Pac. Rim Conf. Comput. Signal Process. (PACRIM)*, pp. 177-182, Aug. 2013.
[31] M. M. Farag, M. A. Ewais, "Smart employment of circuit redundancy to effectively counter trojans (SECRET) in third-party IP cores," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, pp. 1-6, Dec. 2014.
[32] J. Rajendran, O. Sinanoglu, R. Karri, "Building trustworthy systems using untrusted components: a high-level synthesis approach," *IEEE Trans. Very Large Scale Integ. Syst.*, vol. 24, no. 9, pp. 2946-2959, 2016.
[33] T. Reece, W. H. Robinson, "Detection of hardware trojans in third-party intellectual property using untrusted modules," *IEEE Trans. Comput. Aid. Des.*, vol. 35, no. 3, pp. 357-366, 2016.
[34] Xie, T., Qin, X., "Improving security for periodic tasks in embedded systems through scheduling," *ACM Trans. Embed. Comput. Syst.*, vol. 6, no. 3, Article No. 20, 2007
[35] Lin, M., Xu, L., Yang, L.T., Qin, X., Zheng, N., Wu, Z., Qiu, M., "Static security optimization for real-time systems," *IEEE Trans. Ind. Informat.*, vol. 5, no. 1, pp.22-37, 2009
[36] M. Beaumont, B. Hopkins, T. Newby, "SAFER PATH: Security architecture using fragmented execution and replication for protection against Trojaned hardware," in *Proc. Design Autom. Test Eur. (DATE)*, pp. 1000-1005, Mar. 2012.
[37] Sengupta, A., Bhadauria, S. and Mohanty, S.P., "TL-HLS: Methodology for low cost hardware Trojan security aware scheduling with optimal loop unrolling factor during high level synthesis," *IEEE Trans. Comput. Aid. Design*, vol. 36, no. 4, pp.655-668, 2017.
[38] Wang, N., Chen, S., Ni, J., Ling, X., Zhu, Y., "Security-aware task scheduling using untrusted components in high-level synthesis," *IEEE Access*, vol. 6, pp. 15663-15678, 2018.
[39] Höppner, S., Shao, C., Eisenreich, H., Ellguth, G., Ander, M., Schüffny, R., "A power management architecture for fast per-core DVFS in heterogeneous MPSoCs," in *Proc. IEEE Int. Symp. Circuit. Syst. (ISCAS)*, pp. 261-264, 2012.
[40] Chen, G., Huang, K., Knoll, A., "Energy optimization for real-time multiprocessor system-on-chip with optimal DVFS and DPM combination," *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 3s, Article No. 111, pp. 15663-15678, 2014.
[41] A.P. Chandrakasan, S. Sheng, R.W. Brodersen, " Low-power CMOS digital design," *IEEE J. Solid-State Circuits*, vol. 27, no. 4, pp. 473-484, 1992
[42] B. Zhai, D. Blaauw, D. Sylvester, K. Flautner, "Theoretical and practical limits of dynamic voltage scaling," in *Proc. Design Autom. Test Eur. (DATE)*, pp. 868-873, 2004.
[43] Brélaz, D., "New methods to color the vertices of a graph," *Commun. ACM*, vol. 22, no. 4, pp. 251-256, 1979.
[44] K. Li, "Scheduling precedence constrained tasks with reduced processor energy on multiprocessor computers," *IEEE Trans. Comput.*, vol. 61, no. 12, pp. 1668-1681, 2012.
[45] K. Li, "Scheduling parallel tasks with energy and time constraints on multiple manycore processors in a cloud computing environment," *Future Gener. Comput. Syst.*, vol. 82, pp. 591-605, 2018.
[46] CPLEX package, https://www.ibm.com/products/ilog-cplex-optimization-studio, 2018.
[47] Standard Task Graph Set, http://www.kasahara.elec.waseda.ac.jp/schedule.

Yidan Sun received the B.Eng. degree from the School of Computer Science and Technology, Tianjin University, China. She is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering, Nanyang Technological University, Singapore. Her research interests include energy efficient MPSoC design, big data analytics in transportation, and urban computing.



Guiyuan Jiang received the B.S. degree from Northwest University for Nationalities, China, the M.Eng. degree from Tianjin Polytechnic University, China, and the doctoral degree from Tianjin University (TJU), all in Computer Science and Technology. Currently, he works as a research fellow at School of Computer Science and Engineering, Nanyang Technological University, Singapore. His research interests include data analytics for citywide transportation modeling and optimization, design methodologies for reconfigurable computing systems, resource optimization for datacenter and sensor networks.

Lam Siew Kei received his BASc, MEng and PhD from School of Computer Science and Engineering (SCSE), NTU. He was a Visiting Research Fellow in the Imperial College of London, University of Warwick, and RWTH Aachen, Germany. He is currently an Assistant Professor in SCSE and his research focuses on devising custom computing solutions to meet the challenging demands of energy-efficiency, reliability and security in embedded systems. His current projects include architecture-aware algorithms for vision-enabled sensing, design methodologies for secure and reliable embedded systems, and transportation analytics.

Fangxin Ning received the B.S. degree in Navigation Technology, and M.Eng. degree in Transportation Information Engineering and Control, both from Wuhan University of Technology, China. Currently, he works as a research associate at School of Computer Science and Engineering, Nanyang Technological University, Singapore. His research interests include design algorithms for citywide transportation modeling and optimization, design methodologies for intelligent transportation system.