Automatic Generation of Heterogeneous SoC Architectures with Secure Communications

Mattia Tibaldi, Christian Pilato, *Senior Member, IEEE*, and Fabrizio Ferrandi, *Member, IEEE*

***Abstract*—Heterogeneous Multiprocessor System-on-Chip (MPSoC) architectures allow designers to achieve energy efficiency and high performance, especially when tailored for the target applications. This specialization requires to select the MPSoC components and define their interconnections (on the architecture side), and also to determine the mapping and scheduling of tasks and communications (on the application side). This problem is exacerbated by the growing privacy and security concerns when these architectures elaborate sensitive data. We propose a co-design solution based on Ant Colony Optimization that also protects the exchange of private data among components. Our solution generates architectures with secure communications with systematically better performance than architectures with pre-defined security solutions.**

I. Introduction and Related Work

With the end of Moore's law and Dennard scaling, *heterogeneous multiprocessor System-on-Chip* (MPSoC) architectures are becoming the reference solution to achieve energy efficiency and high performance [1]. Fig. 1 shows the design process of a classic heterogeneous MPSoC architecture. The target MPSoC includes several general-purpose processors and specialized hardware accelerators. To reduce design costs, these components are usually intellectual property (IP) cores deployed as part of the chip or on FPGA devices [2]. The resulting architectures allow designers to exploit task-level parallelism and hardware acceleration [3]. So, the designer has to determine the best assignment (*mapping*) and ordering (*scheduling*) of application tasks and communications considering the available hardware resources, maximizing the performance, and respecting resource constraints (e.g., the maximum available area) [4]. Since these problems are NP-complete [5], exploration methods are usually preferred [4], [6], [7]. Often, these algorithms are based on the observation of natural behaviors: for example, *simulated annealing* (SA) is inspired by annealing in metallurgy to create perturbations and move around the design space, while *ant colony optimization* (ACO) is based on the behavior of the ants when searching for food to identify good solutions and avoid unfeasible ones.

Security is becoming a critical issue in the design of embedded systems [8]. Secure SoCs are hard to generate because there is no clear metric for evaluating hardware security. Most of designers focus on the generation of secure components [9], while the integration of insecure components mostly targets attacks on the supply chain [10]. While several methods aim at optimizing also data communications on standardized bus systems (e.g., AMBA, LIN, CAN, etc.) [4], [6], none of them takes into account security concerns. Designing unprotected MPSoCs force to exchange data and instructions in plain form, making the system vulnerable to eavesdropping attacks (e.g., with bus probing) to steal private information. However, protecting the exchange of private data is an open challenge [8], [11], [12], demanding efficient methods for co-optimizing architecture and application.

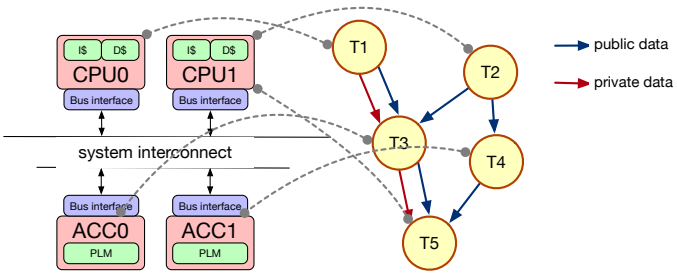

Fig. 1. Example of heterogeneous MPSoC architecture (on the left) and partitioned application with private and public data (on the right).

We propose an automatic method to efficiently secure communications in heterogeneous MPSoCs. Our exploration method is based on Ant Colony Optimization (ACO) [13] and efficiently solves the combined problem: *it refines the target architecture (i.e., it selects both processing and communication components) and determines the mapping and scheduling of tasks and communications to (1) improve the performance of the given application and (2) respect security constraints.*

After formalizing the problem addressed in this work (Section II), we present our main contributions:

- a complete solution that considers, for the first time, security implications in the mapping and scheduling of tasks and communications (Section III);
- an ACO-based heuristic for co-optimizing also secure communications on the available bus protocols (Section IV);
- an evaluation of the security-enhanced MPSoC architectures generated by our method (Section V).

Our approach is a first step towards the automatic generation of secure SoCs with several layers of protection.

II. Problem Definition

We provide security extensions to a state-of-the-art solution for mapping and scheduling in heterogeneous MPSoCs [4].

A. Architecture Model

Fig. 1 shows an example of target architecture that is created starting from a set P of available processing elements

Manuscript received October 30, 2019; revised February 7, 2020 and March 18, 2020; accepted June 15, 2020. This manuscript was recommended for publication by A. Shrivastava. *Contact author*: C. Pilato (christian.pilato@polimi.it).

M. Tibaldi, C. Pilato and F. Ferrandi are with the Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy.

(e.g., programmable cores or specialized-hardware accelerators) and a set C of available communication components (e.g., networks-on-chip or busses). Thanks to the abstraction of hardware and software services [14], these architectures can be composed without taking care of the low-level implementation details. Specialized accelerators are data-intensive and loosely coupled with hardware-based DMA bursts to exchange data between components [15]. In this case, accelerators are known to achieve high-throughput by adopting a non-coherent memory model [16]. On the contrary, tasks executing on the same processing element exchange data through data caches or private local memories, without requiring external data transfers. Each component offers a set of resources R that can be used by the application tasks (e.g., data memory space or physical hardware resources).

Security Extensions. We provide a security-related version for each communication component based on the algorithm used for securing the data transfer (e.g., cryptographic function). For example, it is possible to implement a secure and trusted execution with mutual authentication and secure encryption over a Controlled Area Network (CAN) bus based on a Physical Unclonable Function (PUF) [17]. So, each component $c_i \in C$ is characterized not only by its capacity (transmission rate) and latency (transmission time), but also in terms of area/time overhead to implement the corresponding secure data transfers. The overhead depends on the processing elements attached to the component.

B. Application Model

We model the partitioned input application as a Directed Acyclic Graph (DAG). A DAG is a finite directed graph $G = (T, C)$ with no cycles. The vertices T are the tasks, i.e., parts of the application, while the edges $C \subseteq T \times T$ are the data transfers among the tasks. An edge $c(t_1, t_2) \in C$ implies that task t_2 cannot start until task t_1 is terminated and the related amount of data $\alpha(c)$ has been transferred. We characterize each *computational element* k (either task or communication) with respect to the available elements of the architecture. An *implementation point* $i_{k,a}$ is defined as the cost $\theta(i)$ required to execute the computation element k on the component a. The cost is defined as the specific combination of hardware resources (e.g., silicon area for ASIC devices or heterogeneous resources for FPGA devices) and time required for execution. Implementation points are defined only when the computational element can be implemented and executed by the component. For example, when an application task cannot be synthesized in hardware, the corresponding implementation point is not generated. On the contrary, a computation element can have multiple implementation points (trade-offs) for the same processing element [18].

Security Extensions. We mark each data transfer with appropriate tags to represent the security level of the data. In particular, pvt_c and pub_c represent the amount of *private* and *public* data necessary to complete the transfer $c \in C$, respectively ($\alpha(c) = pvt_c + pub_c$). Private data represent sensible information to protect. We also specify an implementation point for each available method of transferring secure data. This allows us to explore and combine alternative solutions.

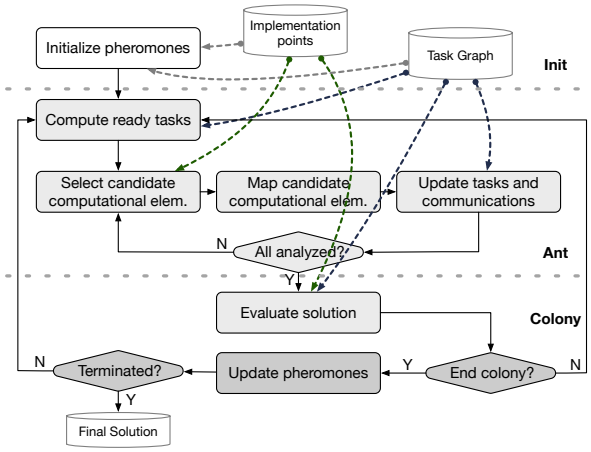

Fig. 2. Overall organization of the exploration methodology.

C. Security-aware mapping and scheduling

The *mapping and scheduling problem with secure communications* can be decomposed into the following interdependent sub-problems: (1) selection of the processing and communication elements; (2) mapping of the computational elements onto the selected components; (3) scheduling of the computational elements. Solutions have to *satisfy the design constraints*, *avoid resource conflicts*, and *maximize the performance*. The solution is defined as a *trace*, i.e., a sequence of mapping decisions. Each decision associates a computational element (either task or communication) with an admissible implementation point. The order in the trace represents the scheduling decisions, i.e., the precedences among the computational elements. A task can be assigned to a component as long as its resource requirements can be satisfied by the component at every execution time. The performance is defined as the *latency* of the application DAG, i.e., the total execution time of all computational elements.

Security Extensions. From the application viewpoint, communications must be implemented in a way that prevents bus probing attacks on private data. From the architectural viewpoint, resource utilization must be optimized to achieve the best performance without compromising system security.

III. Methodology Overview

Fig. 2 shows our exploration methodology, which extends the approach proposed in [4] with security information. The exploration is based on the ACO heuristic search methodology. Ants start from their nest going in random directions, depositing a trail of pheromone that motivates other ants to follow the same path. The ants moving on the shortest paths will reach the food and come back faster than the others, proportionally depositing more pheromones and reinforcing the trails. As time passes, the oldest trails evaporate, and, at some point, only the shortest path will remain with a strong reinforcement.

The exploration algorithm receives as input the task graph G, and the set I of implementation points for both tasks and communications. The fully-software solution is used to initialize the values of the pheromone matrix to the same initial probability. The pheromone matrix represents the probability, for each decision, to lead to a good solution. Each ant is

launched to explore the search space, taking a sequence of decisions to compose a trace, i.e., an alternative solution. At each step, the sets of available computational elements (tasks and communications) are updated based on the decisions taken in the previous steps. As in [4], tasks are added when all predecessors have been mapped, while communications are added when both source and target tasks have been mapped in order to verify whether the data transfer is effectively required. The procedure is repeated until the entire solution is built, i.e., all tasks and communications have been analyzed. At the end of each generation, all solutions (i.e., all traces generated by the corresponding ants) are evaluated and ranked, and the ACO reinforces each decision in the pheromone matrix proportionally to the quality of the corresponding solution. In this work, we target performance optimization of the resulting task graph, so we schedule each solution to estimate its total execution time. Also, the current best solution is replaced if a better is found. A new ant colony is launched until the exploration is terminated (i.e., the maximum number of generations is reached).

IV. ACO Extensions for Securing Communications

At each step x, an ant generates the set of admissible decisions (i.e., pair of computational elements and implementation points that satisfy the constraints). Each admissible decision (i.e., combination of combinational element and implementation point) is generally associated with a probability [13] that is computed as follows:

$$p_{x,y} = \frac{[\tau_{x,y}]^\alpha * [\eta_{x,y}]^\beta}{\sum_{l \in \Omega_x} [\tau_{x,l}]^\alpha * [\eta_{x,l}]^\beta} \quad (1)$$

where y is the candidate decision, η is a problem-related local heuristic (i.e., calculated every time a probability is generated), and τ is the global heuristic associated with the pheromone and stored in the pheromone matrix. Both contributions, weighted through α and β, influence the decision of the ant. Ω_x contains all choices (i.e., candidate decisions) at the current step. This approach has a two-fold objective. On one hand, generating probabilities only for admissible choices allows us to take only admissible moves. On the other hand, the specialization of the local heuristic η allows us to include problem-related information to better drive the exploration process.

To include the security extensions considered in this work, we revise the decision process of each ant by specializing the problem-related local heuristic for both tasks and communications. The former allows us to select mapping configurations that aim at reducing the exchanges of private data on the bus, while the latter aims at selecting the proper communication protocols. The *local heuristic for tasks* is computed as:

$$\eta_{x,y} = \frac{1 + indegree_{pvt}(t_y)}{latency(i_y)} \quad (2)$$

where $indegree_{pvt}(t_y)$ represents the number of edges with private data incoming to the task t associated with the given candidate decision y. On the contrary, the *local heuristic for communications* is computed as

$$\eta_{x,y} = \frac{M_y}{latency(i_y)} \quad (3)$$

where i_y is the implementation point associated with the given candidate decision (i.e., the specific implementation for the communication), while M_y is the number of secure communications already added between the same source and target tasks. By considering the inverse of the latency, this metric prefers short communications which are easier to accomodate in the schedule. By considering the already-existing secure communications, the procedure prefers to reuse the same secure communication element also for public communications. While this method allows for resource reduction, it can introduce a performance overhead since parallel communications must be serialized. However, the solution will be discarded (i.e., the corresponding choices will be penalized) if a better and feasible solution with parallel communications is found.

V. Experimental Evaluation

We evaluated our methodology in the following scenarios:

- `S0`: the DSE is performed by excluding secure implementations for the communications. This solution achieves the *best performance* but is *vulnerable to attacks*.
- `S1`: the DSE is performed by pre-selecting the implementation of the bus to be used for secure communications. We created three variants (`B0`, `B1`, and `B2`), one for each available bus (CAN, LIN, and AMBA, respectively).
- `S2`: the DSE selects also the communication infrastructure for secure communications. This corresponds to the complete solution proposed in this work.

We considered an architecture composed of hardware and software components. We designed ten task graphs used to model realistic applications. These graphs have from 5 to 500 nodes and from 4 to 602 edges, as reported in Fig. 3. Each task requires 800,000-3,000,000 cycles for hardware execution, while 800,000-3,100,000 cycles for software execution. Each task requires a different amount of hardware resources (up to 50,000 slices) representative of realistic hardware implementations, while the maximum number of available hardware resources is set to 70,000 slices. Each edge is annotated with a quantity of data to exchange and it is marked with PVT if it requires a secure implementation. We consider three communication protocols (CAN, LIN, and AMBA), each of them with a secure implementation. The corresponding performance overhead (50,000-80,000 cycles) and resource overhead (6,000-8,000 slices) are based on real implementations [17], [19], [20], [21].

To validate our method, we evaluated the scenarios mentioned above with two exploration strategies: the proposed ACO and the tabu search (TS) implemented also in [4]. The ACO colony is composed of 50 ants that iterate for 100 generations. We use large α values and small β values at the beginning of the exploration, updating the values every 20 generations so that the two factors are inverted towards the end. This favors a better exploration in the initial phase, progressively increasing the importance of the metric in the final phase. The evaporation rate has been set to $\rho = 0.015$. For the TS, at each iteration we generate 10 neighbors with a tabu list composed of 5 sets of solutions each. All experiments have been executed on an Intel CPU (i7-5500U at 2.4GHz) with 16GB of RAM running Windows 10.

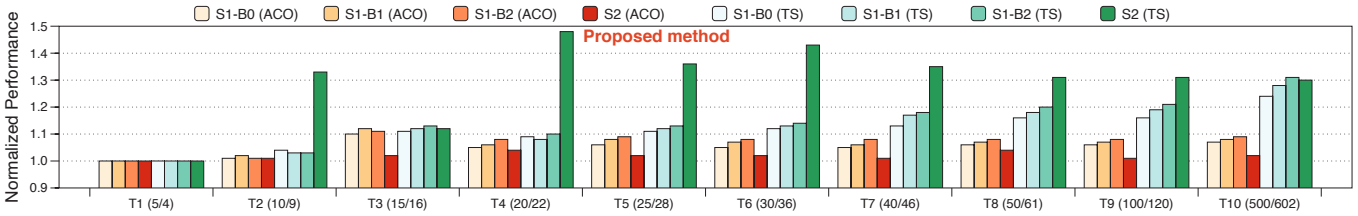

Fig. 3. Comparison of performance results, normalized with respect to the unprotected solutions `S0`. Each benchmark reports also the number of nodes/edges.

Performance Evaluation: Fig. 3 shows the performance results obtained in the different scenarios and with the different exploration methods. These results show that securing the communications with a predefined protocol requires an additional overhead compared to unprotected solutions (`S0`). This overhead is smaller when using ACO, but results with TS are comparable. Conversely, when exploring also the communication protocols, our approach (`S2` - ACO) is able to obtain significantly better solutions (more than 10% on average) than the case with pre-defined secure protocols for the communications (`S1`), while they are only 4% worse than insecure ones (`S0`). Also, TS is not able to efficiently search this highly-constrained design space and obtains solutions that are on average about 24% worse than the corresponding ACO ones. These results are consistent when increasing the size of the benchmarks, demonstrating the scalability of our approach. Concerning execution times, ACO is on average 4× slower than TS due to the constructive and stochastic method to generate solutions, requiring up to almost 2 hours for the largest graph. So, we performed an analysis on the largest graph (`T10`) where the two methods are execute for the same time. In this case, ACO already obtains a better solution (more than 14%) than TS, showing that our method is also faster to reach a good solution.

Analysis of Secure Communications: We also defined a metric to evaluate the *data protection* in the resulting solutions. Specifically, we assume that private data exchanged on the same processing element can be considered secure. So, we define a *communication security metric* as the ratio between the total amount of bytes that are still exchanged between processing elements (even if they are on a secure bus) over the total amount of exchange data. This metric ranges between 0 and 1, and lower values are preferred. `S2` solutions (i.e., the one proposed in this work) obtain an average *communication security metric* of 0.29, while the `S1` solutions obtain a metric of 0.63. This means that our solution is able to reduce the *communication security metric* by more than 50% on average. It is worth noting that, in case of `T1`, tasks exchanging private data are mapped onto the same processing elements, without exposing any private communications on the busses and thus obtaining a full protection of the communications.

VI. Conclusions

We propose a hardware/software co-design methodology for heterogeneous MPSoCs that considers also the problem of secure communications. Our exploration methodology is based on Ant Colony Optimization (ACO) and identifies the best architecture and application implementation to 1) satisfy the design constraints, 2) optimize the execution time, and 3) perform each transfer of private data on the properly secured solution. Our solutions achieve better performance compared to architectures with pre-defined security solutions.

References

[1] H. Esmaeilzadeh *et al.*, "Power challenges may end the multicore era," *Commun. ACM*, vol. 56, no. 2, pp. 93–102, Feb. 2013.
[2] K. Bertels *et al.*, "Hartes: Hardware-software codesign for heterogeneous multicore platforms," *IEEE Micro*, vol. 30, no. 5, pp. 88–97, Sep. 2010.
[3] P. Ellervee and J. Nurmi, "Guest editorial: Implementation issues in system-on-chip," *Journal of Signal Processing Systems*, vol. 87, no. 3, pp. 269–270, Jun. 2017.
[4] F. Ferrandi *et al.*, "Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems," *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 29, no. 6, pp. 911–924, Jun. 2010.
[5] D. Bernstein, M. Rodeh, and I. Gertner, "On the complexity of scheduling problems for parallel/pipelined machines," *IEEE Transactions on Computers*, vol. 38, no. 9, pp. 1308–1313, Sep. 1989.
[6] K. Huang *et al.*, "A scalable and adaptable ILP-based approach for task mapping on MPSoC considering load balance and communication optimization," *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 38, no. 9, pp. 1744–1757, Sep. 2019.
[7] S. Chen, Z. Li, B. Yang, and G. Rudolph, "Quantum-inspired hyper-heuristics for energy-aware scheduling on heterogeneous computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 6, pp. 1796–1810, Jun. 2016.
[8] E. Peeters, "SoC security architecture: Current practices and emerging needs," in *Proc. of DAC*, Jun. 2015, pp. 1–6.
[9] C. Pilato, S. Garg, K. Wu, R. Karri, and F. Regazzoni, "Securing hardware accelerators: A new challenge for high-level synthesis," *IEEE Embedded Systems Letters*, vol. 10, no. 3, pp. 77–80, Sep. 2018.
[10] J. Rajendran, O. Sinanoglu, and R. Karri, "Building trustworthy systems using untrusted components: A high-level synthesis approach," *IEEE Transactions on VLSI Systems*, vol. 24, no. 9, pp. 2946–2959, Sep. 2016.
[11] S. Ravi, A. Raghunathan, and S. Chakradhar, "Tamper resistance mechanisms for secure embedded systems," in *Proc. of the Int.l Conf. on VLSI Design*, Jan. 2004, pp. 605–611.
[12] F. Koeune and F.-X. Standaert, *A Tutorial on Physical Security and Side-Channel Attacks*. Springer, 2005, pp. 78–108.
[13] M. Dorigo and G. Di Caro, "Ant colony optimization: a new meta-heuristic," in *Proc. of CEC*, vol. 2, Jul. 1999, pp. 1470–1477 Vol. 2.
[14] L. P. Carloni, "Invited - the case for embedded scalable platforms," in *Proc. of DAC*, 2016, pp. 1–6.
[15] P. Mantovani *et al.*, "Handling large data sets for high-performance embedded applications in heterogeneous systems-on-chip," in *Proc. of CASES*, Oct. 2016, pp. 1–10.
[16] D. Giri, P. Mantovani, and L. P. Carloni, "Accelerators and coherence: An SoC perspective," *IEEE Micro*, vol. 38, no. 6, pp. 36–45, Nov. 2018.
[17] A. S. Siddiqui, Y. Gui, J. Plusquellic, and F. Saqib, "Secure communication over CANBus," in *Proc. of MWSCAS*, Aug. 2017, pp. 1264–1267.
[18] C. Pilato *et al.*, "Improving evolutionary exploration to area-time optimization of FPGA designs," *Journal of Systems Architecture - Embedded Systems Design*, vol. 54, no. 11, pp. 1046–1057, 2008.
[19] S. Muhlbach and S. Wallner, "Secure and authenticated communication in chip-level microcomputer bus systems with tree parity machines," in *Proc. of SAMOS*, Jul. 2007, pp. 201–208.
[20] J. M. Ernst and A. J. Michaels, "LIN bus security analysis," in *Proc. of IECON*, Oct. 2018, pp. 2085–2090.
[21] D. Kim, Y. Jeon, and J. Kim, "A secure channel establishment method on a hardware security module," in *Proc. of ICTC*, Oct. 2014, pp. 555–556.