

Accelerating Throughput-Aware Runtime Mapping for Heterogeneous MPSoCs

AMIT KUMAR SINGH, Nanyang Technological University and National University of Singapore
AKASH KUMAR, National University of Singapore and Eindhoven University of Technology
THAMBIPILLAI SRIKANTHAN, Nanyang Technological University

Modern embedded systems need to support multiple time-constrained multimedia applications that often employ multiprocessor-systems-on-chip (MPSoCs). Such systems need to be optimized for resource usage and energy consumption. It is well understood that a design-time approach cannot provide timing guarantees for all the applications due to its inability to cater for dynamism in applications. However, a runtime approach consumes large computation requirements at runtime and hence may not lend well to constrained-aware mapping.

In this article, we present a hybrid approach for efficient mapping of applications in such systems. For each application to be supported in the system, the approach performs extensive design-space exploration (DSE) at design time to derive multiple design points representing throughput and energy consumption at different resource combinations. One of these points is selected at runtime efficiently, depending upon the desired throughput while optimizing for energy consumption and resource usage. While most of the existing DSE strategies consider a fixed multiprocessor platform architecture, our DSE considers a generic architecture, making DSE results applicable to any target platform. All the compute-intensive analysis is performed during DSE, which leaves for minimum computation at runtime. The approach is capable of handling dynamism in applications by considering their runtime aspects and providing timing guarantees.

The presented approach is used to carry out a DSE case study for models of real-life multimedia applications: H.263 decoder, H.263 encoder, MPEG-4 decoder, JPEG decoder, sample rate converter, and MP3 decoder. At runtime, the design points are used to map the applications on a heterogeneous MPSoC. Experimental results reveal that the proposed approach provides faster DSE, better design points, and efficient runtime mapping when compared to other approaches. In particular, we show that DSE is faster by 83% and runtime mapping is accelerated by 93% for some cases. Further, we study the scalability of the approach by considering applications with large numbers of tasks.

Categories and Subject Descriptors: D.4.7 [Operating Systems]: Organization and Design—*Real-time systems and embedded systems*; J.6 [Computer Application]: Computer-Aided Engineering: Computer-aided design (CAD)

General Terms: Algorithms, Design, Management, Performance

Additional Key Words and Phrases: Multiprocessor systems-on-chip, embedded systems, multimedia applications, design-space exploration, runtime mapping, synchronous data-flow graphs, throughput, energy consumption

ACM Reference Format:

Singh, A. K., Kumar, A., and Srikanthan, T. 2012. Accelerating throughput-aware runtime mapping for heterogeneous MPSoCs. *ACM Trans. Des. Autom. Electron. Syst.* 18, 1, Article 9 (December 2012), 29 pages. DOI = 10.1145/2390191.2390200 <http://doi.acm.org/10.1145/2390191.2390200>

Author's addresses: A. K. Singh, A. Kumar, Department of Electrical and Computer Engineering, National University of Singapore, 21 Lower Kent Ridge Road, Singapore 119077; email: {eleaks,akash}@nus.edu.sg; T. Srikanthan, School of Computer Engineering, Nanyang Technological University, 50 Nanyang Avenue, Singapore 639798; email: astsrikan@ntu.edu.sg.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.

© 2012 ACM 1084-4309/2012/12-ART9 \$15.00

DOI 10.1145/2390191.2390200 <http://doi.acm.org/10.1145/2390191.2390200>

1. INTRODUCTION

Multiprocessor systems-on-chip (MPSoCs) consist of multiple processing elements (PEs) connected by a communication infrastructure. Heterogeneous MPSoCs contain different type of PEs. The distinct features of the different type of PEs can be exploited in order to achieve high computation performance and energy efficiency. Modern embedded systems are based on MPSoCs to meet high performance demands, for example, ST Nomadik, NXP Nexperia [Kim et al. 2008], and IBM Cell [Kistler et al. 2006].

Modern embedded systems (e.g., smart phones, PDAs, tablet PCs) often support a number of multimedia applications concurrently, and this number is increasing faster than ever. For example, a smart phone might be used to view an image using a JPEG decoder over the internet and, at the same time, to listen to music using an MP3 decoder. Users expect that all applications running in the system should satisfy their timing (throughput) constraints and have a robust behavior [Gangwal et al. 2005]. Thus, the supported applications should have a predictable timing nature that depends upon the system resource usages. Synchronous data flow graphs (SDFGs) can be used to model time-constraint multimedia applications and provide predictability [Lee and Messerschmitt 1987]. The timing is often analyzed at design time, which, is incapable of handling runtime aspects, such as supporting a new application.

To support a new application in the system at runtime, the application tasks need to be mapped onto the system resources such that the throughput constrained is satisfied, and energy consumption and resource usage are optimized. The timing nature depends upon how the mapping is performed. Most of the existing mapping strategies are based on either design-time analysis [Palermo et al. 2005; Stuijk et al. 2007; Ascia et al. 2007] or on runtime mapping [Ykman-Couvreur et al. 2006; Moreira et al. 2007; Nollet et al. 2008; Carvalho and Moraes 2008; Singh et al. 2009]. The design-time strategies are unable to handle dynamism in applications incurred at runtime, as they are applicable only to predefined sets of applications with static behavior. However, the runtime strategies cannot guarantee schedulability, that is, meeting the strict timing deadlines due to lack of any prior analysis and limited compute power at runtime. Thus, there is a need to devise a hybrid strategy that should perform compute-intensive analysis at design time and should use the analyzed results at runtime to overcome these problems. While there are some efforts in the hybrid strategy direction [Schranzhofer et al. 2010; Ykman-Couvreur et al. 2011; Yang et al. 2002], their analysis results are not optimized from the throughput point of view. Further, they are applicable for the analyzed platform only. The strategy in Singh et al. [2011] does consider a generic platform and provides throughput-optimized analysis results, but they are limited to *homogeneous* platforms and do not consider energy consumption.

We present a hybrid strategy for *heterogeneous* platforms containing different types of processing tiles. A processing tile essentially contains a processor, for example, general purpose processor (GPP), digital signal processor (DSP), accelerator, reconfigurable hardware (RH), etc., along with other elements, for example, memory. The RH can be configured as a processor. The processor type determines the tile type. The presented strategy considers energy consumption as well.

The presented hybrid strategy has several new design challenges. First, the design space to be explored becomes μ -dimensional with μ number of tile types, whereas it is linear (one-dimensional) for the homogeneous case. Figure 1 shows the number of possible design points (mappings) at different tile combinations for an application with five tasks, when each task can be mapped on two types of tiles. The five tasks can use a maximum of five tiles at a time. If we take the number of tiles of 2nd type as zero, then we get the one-dimensional (homogeneous) design space, shown by the blue color bars in Figure 1. We get a total of 52 such mappings. For 14 tasks, a total of 190,899,322

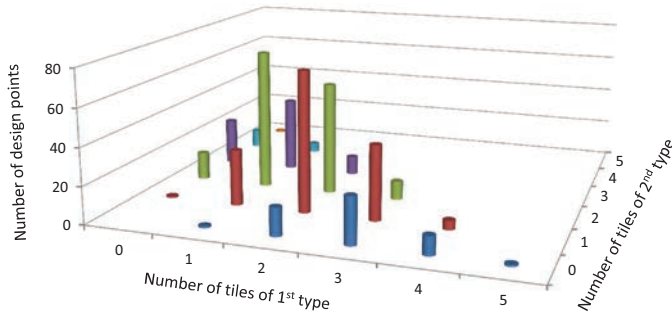


Fig. 1. Number of mappings at different combination of used tiles.

mappings are obtained, which will take approximately 220 days in evaluation if we assume 100 milliseconds (ms) to evaluate one mapping. Thus, evaluation of all the possible mappings is not always feasible. The strategy proposed in Singh et al. [2011] adopts a pruning technique to discard evaluation of inefficient mappings and performs the evaluation in a limited time. Therefore, the presented strategy adopts the technique proposed in Singh et al. [2011] for evaluating homogeneous tile mappings. In Figure 1, we consider two types of tiles and get a total of 454 design points in the two-dimensional design space. For 14 tasks, more than 20 billion mappings are obtained, which is going to take many years to evaluate. The number of mappings increases exponentially with the number of tile types and in turn, the evaluation time. Therefore, the presented strategy has the challenge of finishing the evaluation within a limited time without missing the efficient mappings when the number of tasks and tile types is increased.

Another challenge for heterogeneous platforms is to avoid time-consuming evaluation of mappings using non-supported tile combinations in case all tasks cannot be supported on all the tile types. The next challenge is to accurately measure energy consumption for all the mappings to be evaluated. The presented hybrid strategy also has the challenge of finding the Pareto-optimal points at different tile combinations from a large number of design points and of selecting the best one at runtime, depending upon the different types of available tiles.

Key Contributions. In this article, these challenges have been addressed through the presented hybrid strategy by providing the following main contributions.

- A design-time DSE strategy for a generic MPSoC platform computing throughput and energy consumption at different resource combinations. The platform may contain different types of processors, such as GPPs, DSPs, accelerators, etc.
- An optimization technique to accelerate the DSE when the maximum number of tiles in the platform is known in advance.
- A memory optimization technique based on Pareto algebra to be applied on the available design points to keep only Pareto-optimal points, thereby reducing the evaluation overhead at runtime.
- An efficient runtime strategy for selecting the best point from the Pareto-optimal points subject to the desired throughput, while optimizing for energy consumption and resource usage at runtime.

Existing design-time DSE strategies are applicable only to a fixed architecture platform; they do not scale well with the number of tiles in the platform and do not always provide the largest throughput mapping. These strategies perform optimization for some performance metrics, like energy, resource optimization, etc., and in turn map the potentially parallel executing tasks on the same tile, forcing their execution in

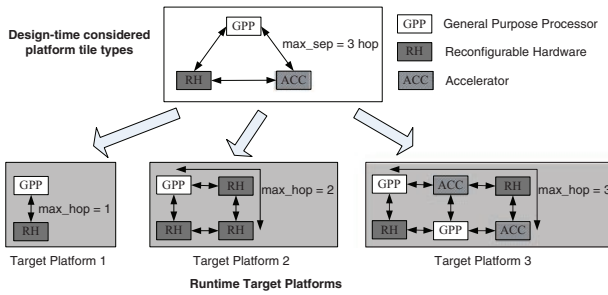


Fig. 2. Analyze once & run everywhere demonstration.

sequence. This often reduces the available parallelism, thereby reducing throughput. Further, the number of evaluated mappings by existing DSE strategies depends upon the number of tiles in the platform. The number of evaluated mappings determines the exploration time. Thus, the existing DSE strategies require a lot of time in exploration for advanced available commercial platforms containing hundreds of tiles [Vangal et al. 2007; TILE-Gx100 2009], and even more time for anticipated MPSoCs containing thousands of tiles [Borkar 2007]. Existing runtime mapping strategies start mapping without any previous analysis of the application and thus do not perform well.

Our design-time DSE strategy considers a generic multiprocessor platform that contains tiles depending upon the number of tasks and their implementation alternatives provided in the applications. Implementation alternatives of a task determine the processor types onto which it can be supported, such as on a GPP, DSP, and RH block. The number of used tiles in a mapping is referred to as the *tile count*. At each tile count, our technique analyzes a number of mappings at different processing resource combinations and stores the best mapping in terms of throughput and energy consumption for each combination. Our runtime strategy selects one of the stored mappings depending upon the desired throughput, without performing any computation to evaluate mappings and thus performs fast runtime mapping.

The considered generic platform, during design-time DSE, contains tiles that are separated by a fixed distance from each other, referred to as *hop distance* in this work. A real-life platform might have tiles at varying distances from each other, for example, a 2×2 grided tile platform has a few tiles separated by a hop distance of 1, while others at a hop distance of 2. The DSE is performed by considering maximum separation between the tiles in the expected future target platform. The DSE results can be used for any target platform as long as (i) the target platform tile types are a subset of the analyzed tile types, and (ii) the maximum distance between the chosen target platform tiles for mapping is less than or equal to the maximum separation for which the DSE was performed. Thus, no additional design-time analysis is needed in case of such different target platforms. This approach is analogous to *analyze once & run everywhere*, which is similar to Java's *write-once-run-everywhere* capability. Figure 2 shows a demonstration for three types of tiles (GPP, RH, ACC) analyzed during DSE with maximum separation between the tiles equal to 3 hop (hop distance). The analyzed results will be applicable to the three shown target platforms, as their tile types and max hop distance are a subset of the tile types and maximum separation considered during DSE.

The rest of the article is organized as follows. Section 2 reviews the related work in the direction of design-time DSE and runtime mapping. Section 3 introduces the multiprocessor and application model used in this work. The hybrid mapping flow that first performs design-time analysis of applications and then maps the applications on a multiprocessor platform at runtime is presented in Section 4. The experimental results

to evaluate our methodology are presented in Section 5. Section 6 concludes the article and provides directions for future work.

2. RELATED WORK

Most of the design-time DSE strategies reported in the literature provide a single mapping for the application (e.g., see Moreira et al. [2007]; Ahn et al. [2008]; Keinert et al. [2009]; Liu et al. [2008]; Bonfietti et al. [2009]; Stuijk et al. [2007]). They perform DSE in view of some optimization parameters, such as computational performance and energy, and the optimization is very time consuming. These strategies target fixed MPSoC platforms and do not provide mappings for optimal throughput and energy consumption, in some cases. Further, they are unable to handle dynamism in resource availability and throughput requirement at runtime. In contrast, our DSE strategy is applied to a generic MPSoC platform and generates a set of mappings with different resource requirements, throughput, and energy consumption, which helps to handle dynamism at runtime.

Design-time DSE strategies that generate multiple mappings for the application have recently been reported in [Mariani et al. 2010; Stuijk et al. 2010; Giovanni et al. 2010; Zamora et al. 2007; Angiolini et al. 2006; Lukasiewicz et al. 2008]. The generated mappings can be used to handle dynamism in resource availability and throughput requirement at runtime, but these approaches have several drawbacks, such as being applicable only to fixed platform, not providing optimal mappings in some cases, evaluating large numbers of mappings for relatively larger platforms (including some duplicate mappings), and not scaling well with the platform size. The duplicate mappings just differ in placement of tasks on different tiles, with the same task-to-tile bindings, and provide the same performance. Further, the time-consuming DSE needs to be repeated with any changes in the platform. Jia et al. [2010] perform exploration for multiple platforms. The applicability of generated mappings is limited to the set of explored platforms. In contrast, our strategy considers a generic platform that contains tiles, depending upon the tasks and their implementation alternatives provided in the application, and provides the mappings having the largest throughput and minimum energy consumption at different processing resource combinations. The mappings generated by our approach are applicable to any target platform so long as the target platform tile types and maximum separation between the tiles are a subset of the tile types and maximum separation considered during DSE. Thus, repetition of the DSE for a new platform is avoided. The generation of duplicate mappings is avoided as well by not considering a bigger platform than required.

There has been quite a lot of research on multiple application DSEs. Some researchers focus on scenario-based approaches where multiple application mapping scenarios are explored at design time in order to handle dynamism in the number of active applications at runtime [van Stralen and Pimentel 2010; Stuijk et al. 2010; Palermo et al. 2008]. A *scenario* contains a set of simultaneously active applications. The scenarios also have been referred to as *use-cases* [Kumar et al. 2008; Benini et al. 2008]. The scenario-based approaches are not scalable, as the number of scenarios increases exponentially with the number of applications, which might become intractable. In order to support multiple active applications at runtime, the applications can be mapped one after another. We map the applications one after another and thus avoid the overhead of handling large numbers of use-cases.

To map the application tasks on the platform tiles at runtime, one can start the mapping with or without previously analyzed results. Most of the work presented in the literature start mapping without any previous analysis and thus cannot guarantee for schedulability and strict timing deadlines, due to limited computational resources at runtime [Singh et al. 2010; Carvalho and Moraes 2008; Ykman-Couvreur et al. 2006;

Table I. Comparison of Various Approaches for Performing Design-Time Analysis and then Runtime Mapping

Reference	Platform	Applicability	Mappings	Evaluation	Runtime
[Mariani et al. 2010]	Fixed	Homogeneous	Multiple	Non-scalable	Yes
[Stuijk et al. 2010]	Fixed	Homogeneous	Multiple	Non-scalable	No
[Giovanni et al. 2010]	Fixed	Homogeneous	Multiple	Non-scalable	No
[Ykman-Couvreur et al. 2011]	Fixed	Homogeneous	Multiple	Non-scalable	Yes
[Singh et al. 2011]	Generic	Homogeneous	Multiple	Scalable	Yes
[Yang et al. 2002]	Fixed	Heterogeneous	Multiple	Non-scalable	Yes
[Angiolini et al. 2006]	Fixed	Heterogeneous	Multiple	Non-scalable	No
[Zamora et al. 2007]	Fixed	Heterogeneous	Multiple	Non-scalable	No
[Lukasiewicz et al. 2008]	Fixed	Heterogeneous	Multiple	Non-scalable	No
[Schranzhofer et al. 2010]	Fixed	Heterogeneous	Single	Non-scalable	Yes
[Jia et al. 2010]	Flexible	Heterogeneous	Multiple	Non-scalable	No
Our strategy	Generic	Heterogeneous	Multiple	Scalable	Yes

Nollet et al. 2008; Moreira et al. 2007]. A few strategies using design-time analysis results have been presented [Schranzhofer et al. 2010; Ykman-Couvreur et al. 2011; Yang et al. 2002; Singh et al. 2011]. In Schranzhofer et al. [2010], analysis results include only a single mapping having minimum average power consumption; so, the mapping may not be optimized from a throughput point of view. Analysis results in Ykman-Couvreur et al. [2011] and Yang et al. [2002], include multiple mappings having trade-offs in terms of target power consumption and performance. The results do not include mappings that satisfy the constraints in the case of limited resources. At runtime, this case forces the application to be put into a relaxed application set, so it is not mapped immediately, which may result in missing the strict timing deadline. The analysis results of Schranzhofer et al. [2010], Ykman-Couvreur et al. [2011] and Yang et al. [2002] are applicable to a fixed platform only. In Singh et al. [2011], analysis results include mappings optimized from the throughput point of view for the limited resources case, but are applicable to homogeneous platforms and do not include energy consumption. Our presented strategy considers energy consumption and heterogeneous platforms. At runtime, the strategy efficiently uses the analyzed results and always tries to provide a timing guarantee.

Table I shows a comparison of the approaches reported in the literature which consider design-time analysis and the analyzed results for runtime mapping—where our approach is different. As can be seen, most of the existing approaches perform design-time analysis on fixed or flexible (multiple) platforms, are not scalable (Non-scalable) with application and platform size, and evaluate mappings that are applicable only to fixed homogeneous, fixed heterogeneous, or a set of heterogeneous (Flexible heterogeneous) platforms. However, in our approach, design-time analysis considers a generic platform and is scalable, while providing multiple mappings that are applicable to any platform. Our strategy has support for runtime mapping that uses design-time analysis results.

3. PRELIMINARIES

This section covers some preliminaries necessary to explain our proposed hybrid mapping flow. We describe the hardware platform and the application model with the underlying assumptions and terminology.

3.1. Multiprocessor Platform Model

The multiprocessor platform model used in this work is a tile-based architecture, as shown in the example platform of Figure 3, where an interconnection network is used to connect the tiles. The platform has four tiles: t_1 , t_2 , t_3 and t_4 . End-to-end connections (c) with fixed latency between tiles are used to connect the tiles. Each connection may have a different latency, so the latency of connections through a network-on-chip (NoC)

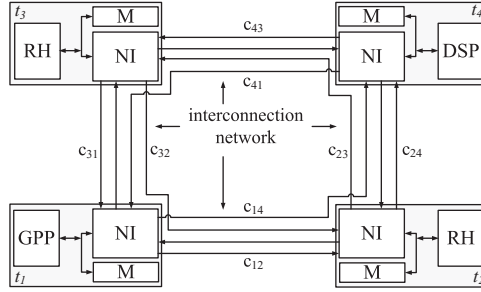


Fig. 3. Example multiprocessor platform.

Table II. Properties of the Example Platform

tile	p_{type}	m	ci	co	$i\omega$	$o\omega$	pwr	connection	L
t_1	GPP	1000000	8	8	12	12	1000	$c_{12}, c_{24}, c_{43}, c_{31}$	3
t_2	RH	1000000	8	8	12	12	4.60	$c_{14}, c_{23}, c_{41}, c_{32}$	6
t_3	RH	1000000	8	8	12	12	4.60		
t_4	DSP	1000000	8	8	12	12	330		

can be taken into account [Grecu et al. 2005], that is, any type of interconnection network can be modeled so long as the latencies between tiles are provided. Each tile contains a processor (e.g., general purpose processor (GPP), digital signal processor (DSP), or reconfigurable hardware (RH), as shown in Figure 3), a local memory (M), and a network interface (NI) containing a set of communication buffers that is accessed both by the interconnect and the local processor.

Definition 3.1 (Platform Graph (PG)). A PG is represented as (T, C, L) , which contains a set T of tiles, a set C of connections, and a latency function L that provides latency (in time units) of a connection ($L(c)$). A tile $t \in T$ is a 7-tuple $(p_{type}, m, ci, co, i\omega, o\omega, pwr)$, where $p_{type} \in PT$ (PT is a set of processor types), m is the memory size (in bits), ci and co are the maximum number of input and output connections supported by the NI, $i\omega$ and $o\omega$ are the maximum incoming and outgoing bandwidth (in bits/time-unit), and pwr is the power consumption (in milliwatts) of the processor type p_{type} .

Table II shows the values of all the elements in the example platform graph (Figure 3). Multiprocessor systems, such as StepNP [Paulin et al. 2004], PROPHID [Leijten et al. 1997] and Eclipse [Rutten et al. 2002], fit nicely into this platform model.

In Figure 3, the latencies of end-to-end connections are modeled according to a 2D mesh network in order to model a mesh interconnection network. The latency of a connection depends upon the distance between the connecting tiles, and this distance is referred to as the hop_distance. In Figure 3, tiles t_1 and t_2 are at a hop_distance of 1 (just adjacent) and t_1 and t_4 at a hop_distance of 2, as communications are via tile t_2 (1 hop in the x-direction to reach tile t_2 and then 1 hop in the y-direction to reach tile t_4). The latencies of connections are modeled according to the hop_distance, as can be seen in Table II. The application edges can get mapped onto the connections between tiles. Each such edge occupies one connection between the tiles at its full bandwidth, and the occupied connection always serves only the assigned edge. Therefore, the latency between tiles remains constant. Examples of such NoCs are circuit-switched networks, AEthereal [Goossens et al. 2005] and Spatial Division Multiplexing (SDM) [Yang et al. 2010], which provide guaranteed throughput that implies constant latency. However, latency will not be constant in a packet-switched network as it depends upon the traffic present in the network. In such cases, a reasonable upper bound can be calculated using

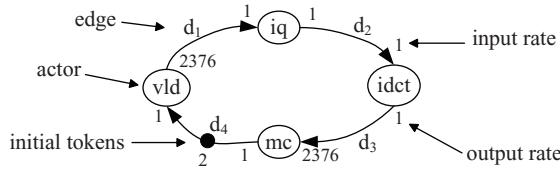


Fig. 4. SDFG model of an H.263 decoder.

traffic patterns. To incorporate that two tiles are at higher hops, we change the latency of the connections between the tiles according to the hops. This incorporation helps in finding mappings when the tiles are further apart in the actual platform.

3.2. Application Model

Synchronous data flow graphs (SDFGs) [Lee and Messerschmitt 1987] are used to model concurrent multimedia applications with timing constraints. The SDFG model of the H.263 decoder is shown in Figure 4. The nodes model the tasks and are referred to as *actors*, which communicate with *tokens* sent from one actor to another through the edges modeling dependencies. The H.263 decoder is modeled with four actors, *vld*, *iq*, *idct*, and *mc*, and four edges, d_1 , d_2 , d_3 , and d_4 . An actor fires (executes) when there are sufficient input tokens on all of its input edges and sufficient buffer space on all of its output channels. Every time an actor fires, it consumes a fixed amount of tokens from the input edges and produces a fixed amount of tokens on the output edges. These token amounts are referred to as *rates*. The rates determine how often actors have to fire with respect to each other. The edges may contain *initial tokens* indicated by a bullet point, as in Figure 4.

Definition 3.2 (SDFG). An SDFG (A, E) consists of a set A of actors and a set E of edges. An edge $e = (a_1, a_2, tk_1, tk_2)$ represents a dependency of actor a_2 on a_1 . When a_1 fires, it generates tk_1 tokens on e , and when a_2 fires, it consumes tk_2 tokens from e . Initial tokens on edges are defined as $TokIn : E \rightarrow$ natural numbers including 0.

Analysis techniques for calculating throughput and storage requirements for an SDFG already exist [Ghamarian et al. 2006]. Throughput is an important constraint for multimedia applications and is defined as the inverse of the long-term period, that is, the average time needed for one iteration of the application. An iteration is defined as the minimum nonzero execution such that the original state of the SDFG is obtained. For example, in Figure 4, a period of the H.263 decoder is $(ExecTime(vld) + 2376.ExecTime(iq) + 2376.ExecTime(idct) + ExecTime(mc))$, where $ExecTime$ is the execution time. It should be noted that actors *iq* and *idct* have to execute 2,376 times in one iteration and that the number of executions for each actor is referred to as the *repetition vector* of the actor. This period is just for demonstration and does not include network and memory access delays. An SDFG with a throughput of 100 Hz takes 10 ms to complete one iteration.

For modeling an application, the resource requirements of the actors and edges are clearly specified. The application model also specifies a throughput constraint that must be satisfied when the application is mapped onto the platform.

Definition 3.3 (Application Graph (AG)). An AG is represented as (A, E, AP, EP) which is derived from SDFG (A, E) . AP and EP provide the resource requirements of actors and edges on the platform, respectively. For each actor $a \in A$, AP provides a tuple (ET, mem) for each implementation alternative $(\in p_{types})$, where p_{types} represents the implementation alternatives of the actor, and ET and mem represent the execution time (in time units) and memory needed (in bits) on the implementation

Table III. Resource Requirement of Actors and Edges of H.263 Decoder

actors	p_{types}	GPP(ET,mem)	ACC(ET,mem)	RH(ET,mem)
<i>vld</i>	GPP,ACC	(26018,10848)	(13009,10848)	(-, -)
<i>iq</i>	GPP,ACC	(559,400)	(450,400)	(-, -)
<i>idct</i>	GPP	(486,400)	(-, -)	(-, -)
<i>mc</i>	GPP,RH	(10958,8000)	(-, -)	(5479,8000)

edges	sz	$mreq_t$	$mreq_{src}$	$mreq_{dst}$	ω
d_1	512	2376	2376	1	5
d_2	512	1	1	1	5
d_3	512	2376	1	2376	5
d_4	1216512	3	1	1	5

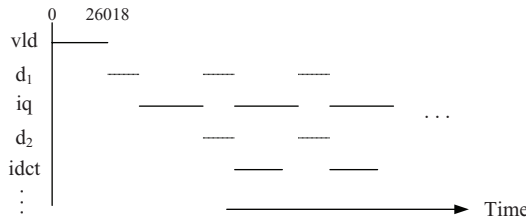


Fig. 5. Execution trace of an H.263 decoder.

alternative, respectively. AP provides null values for ET and mem for unsupported implementation alternatives. For each edge $e = (a_1, a_2, tk_1, tk_2) \in E$, EP provides a 5-tuple $(sz, mreq_t, mreq_{src}, mreq_{dst}, \omega)$, where sz is the size of a token (in bits), $mreq_t$ are the memory (in tokens) needed when a_1 and a_2 are allocated to the same tile, $mreq_{src}$ and $mreq_{dst}$ are the memory (in tokens) needed in source and destination tile, respectively, and ω is the bandwidth (in bits/time unit) needed when a_1 and a_2 are allocated to different tiles. The throughput constraint of the AG is represented as τ .

Table III represents the values of AP and EP for actors and edges of an H.263 decoder application. The execution pattern of the H.263 decoder (consisting of four actors) mapped on a 4-tile MPSoC platform such that each actor is mapped on a different GPP (ARM7TDMI processor) tile is shown in Figure 5. It is clearly seen that actors iq and $idct$ have the potential to execute in parallel. It has been observed that when the existing strategies are applied to perform design-time analysis for the H.263 decoder on a 3-tile platform, in some cases, the best produced mapping contains actors iq and $idct$ on the same tile while optimizing for some performance metrics, such as power and resource usage. For example, Stuijk et al. [2007] maps actors iq and $idct$ on the same tile while optimizing for load balancing on the three used tiles for the application. This forces execution of actors iq and $idct$ sequentially, resulting in reduced throughput. However, our approach finds the best mapping, which has the maximum throughput where actors iq and $idct$ are not allocated on the same tile, but sequentially executing actors like vld and iq are on the same tile. Mapping the connected and sequentially executing actors on the same tile results in reduced communication overhead between the actors, which may maximize the throughput, even on smaller tile counts.

4. HYBRID MAPPING STRATEGY

This section details our hybrid mapping strategy. The strategy is presented in Figure 6. It has two main steps: (1) analysis of applications at design time (*design-time analysis*), and (2) mapping of the applications on a platform by using the analysis results (*optimal mappings with throughput and energy*) with the help of a platform manager (*runtime manager*) at runtime.

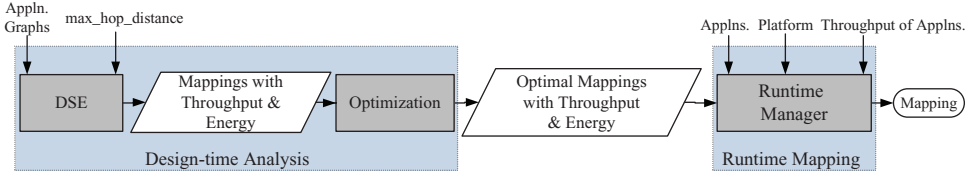


Fig. 6. Hybrid mapping strategy.

4.1. Design-Time Analysis

The *design-time analysis* step evaluates a number of mappings for each application to be supported on a hardware platform. The applications are evaluated one after another. The evaluation considers finding different mappings and their throughput and energy consumption. For each mapping, actors (A) and edges (E) of the application graph AG are bound to tiles (T) and connections (C) between two tiles or the memory inside a tile in the platform graph PG . This binding gives a resource allocation for the application graph AG on the platform graph PG with the following constraints for each tile $t \in T$.

- (1) (memory imposed by actors and edges bound on t) \leq (memory (m) on t).
- (2) (allocated input connections on t) \leq (maximum input connections (ci) on t).
- (3) (allocated output connections on t) \leq (maximum output connections (co) on t).
- (4) (allocated incoming bandwidth on t) \leq (maximum incoming bandwidth ($i\omega$) on t).
- (5) (allocated outgoing bandwidth on t) \leq (maximum outgoing bandwidth ($o\omega$) on t).

Throughput and Energy Consumption Computation. The throughput for a mapping is computed by taking the resource allocations into account. First, a static-order schedule for each tile is constructed that orders the execution of bound actors. A list scheduler is used to construct the static-order schedules for all the tiles at once. Then, all the binding and scheduling decisions are modeled in a graph called binding-aware SDFG. Finally, throughput is computed by self-timed state-space exploration of the binding-aware SDFG [Ghamarian et al. 2006].

The energy consumption for a mapping is computed as the sum of the communication and computation energy for all the tasks for one iteration of the application. Communication energy is required to transfer data from the source tile to the destination tile through a connection when actors mapped on the two tiles need to communicate with each other. The communication energy is estimated as the product of the number of bits to be transferred, the number of hops to be traversed between the two tiles, and the energy required to transfer one bit through one hop, for each edge (e) mapped to a connection ($conn$) from Equation (1). The transferred bits through a connection are calculated as the product of the number of tokens to be transferred and the token size for the edge mapped on the connection. The number of tokens for an edge (e) is computed as the product of the repetition vectors of the source (or destination) actor and the source (or destination) port rate of the edge from Equation (2). The energy required to transfer one bit through one hop is denoted as E_{Lbit} [Palma et al. 2005; Hu and Marculescu 2004]. Computation energy is required to process the transferred token on the destination tile after it is received and able to fire (execute) the mapped actor. The computation energy for each actor (a) mapped to tile (t) is estimated as the product of the number of executions of actor a , and the execution time and power consumption on tile t from Equation (3). Total energy consumption is measured as the sum of the communication and computation energy.

$$E_{comm} = \sum [e \rightarrow nrTokens) \times (e \rightarrow tokenSize) \times hopCount \times E_{Lbit}]. \quad (1)$$

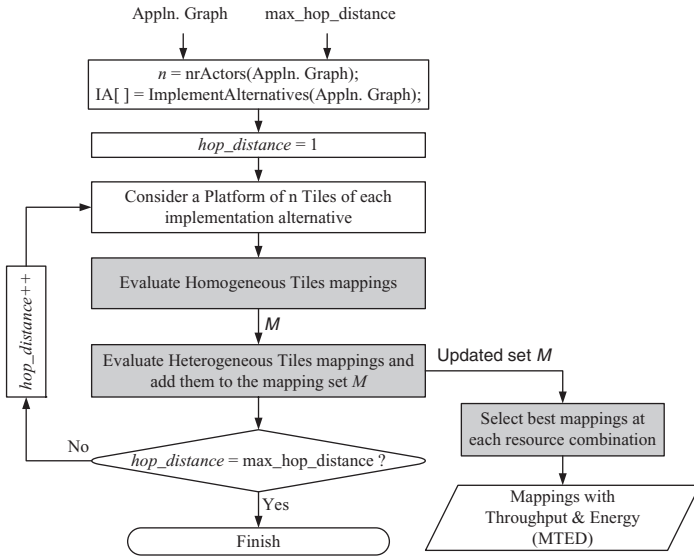


Fig. 7. Design-time DSE flow.

$$e \rightarrow nrTokens = repVector[e \rightarrow srcActor] \times (e \rightarrow srcPortRate). \quad (2)$$

$$E_{comp} = \sum [repVector[a] \times (a \rightarrow execTime(t \rightarrow procType)) \times procPower]. \quad (3)$$

The total energy consumption does not include static energy. In our approach, we focus on mapping of applications on the architecture after it is designed. So, we cannot do much with the static energy and focus only on dynamic energy consumption that can be optimized.

The design-time analysis for an application (Appl. Graph) first performs design space exploration (DSE) to obtain design points that contain mappings and their corresponding throughput and energy consumption (mappings with throughput and energy). Then, an optimization on the explored design points to get only the Pareto-optimal design points is performed (optimal mappings with throughput and energy), providing throughput and energy consumption at different resource combinations (Figure 6). The DSE flow is presented in Figure 7.

The presented DSE flow first considers a suitable platform graph (T, C, L) that can cover all the possible mappings for the application graph (A, E, AP, EP) to be analyzed. A platform is considered, containing n tiles of each implementation alternative provided in the application, where n is the number of actors in the application. This platform is capable of covering all the potential mappings. Considering a bigger platform wouldn't provide better performance, as the considered one can exploit all parallelism present in the application. However, all the parallelism might not be exploited if a small-size platform is considered, where concurrent executing tasks will get mapped on the same tile.

Initially, the considered platform contains tiles with a separation between them of one hop.distance ($hop_distance = 1$), which provides a minimum fixed latency for all the connections between the tiles. The DSE flow is repeated by considering a similar platform that contains tiles with separation of one hop.distance more ($hop_distance++$) between them, that is, with increased latency for connections, until the $hop_distance$ reaches $max_hop_distance$ (one of the inputs to the DSE flow). The

Algorithm 1: GPP Tiles Mappings Evaluation at Reduced Tile Count

Input: Best mapping α using $(p + 1)$ GPP tiles.

Output: Mappings using p GPP tiles.

Select $p + 1$ GPP tiles ($\in T$) containing actor(s);

for each unique pair of selected tiles **do**

 Move actor(s) from one GPP tile to another to generate a new mapping β ;

 Compute *throughput* and *energyConsumption* of β ;

 Add β with its *throughput* and *energyConsumption* to set M ;

end

designers can choose an appropriate value of *max_hop_distance* depending upon the expected hardware platform at runtime, where maximum separation between the tiles can be up to *max_hop_distance*. For a higher value of *max_hop_distance*, the design-time DSE evaluates larger numbers of mappings. This requires more evaluation time, but the applicability of mappings increases. For example, evaluated mappings with a *max_hop_distance* value of 6 are applicable to any platform where the maximum separation between the tile is less than or equal to 6 hops, such as a mesh of 2×2 , 2×3 , 3×3 and 4×4 tile platforms. However, when platforms are very large (say 10×10), it is unrealistic to expect tasks of an application to be mapped on extreme ends of the platform. The maximum hop distance in such cases is the maximum separation of the tiles on which various tasks of an application are mapped.

By considering varying values of hop_distance, we get mappings where each edge of the application is mapped to a connection at hop_distance of one (to account for minimum latency) to *max_hop_distance* (to account for maximum latency). This facilitates catering to the runtime aspects when the available tiles are at different hop_distances. A strategy to find the best mapping in such runtime scenarios is presented in Section 4.2. We have considered generic tile architecture so that any type of inter-connection network can be modeled. The main steps of the DSE flow (highlighted in Figure 7) and the optimization technique are described next.

4.1.1. Evaluating Homogeneous Tile Mappings. The mappings using only GPP tiles are generated by using the DSE strategy proposed in Singh et al. [2011]. This strategy discards evaluation of inefficient mappings (providing less throughput) and performs faster evaluation without missing the efficient mappings. Therefore, the same strategy has been adopted to generate homogeneous tile mappings. For each generated mapping, the strategy in Singh et al. [2011] computes throughput only, whereas the presented strategy computes energy consumption as well. The strategy first evaluates 1_actor-to-1_GPP_tile mapping, where n actors of the application are mapped onto n GPP tiles so that each GPP tile contains exactly one actor and the edges are mapped onto connections. Then, mappings at reduced tile count ($p = n - 1$), that is, mappings using $(n - 1)$ GPP tiles are evaluated by Algorithm 1. The algorithm takes the best mapping using $(p + 1)$ GPP tiles as input and evaluates mappings using p GPP tiles. First, $(p + 1)$ GPP tiles containing actor(s) are selected. Then, for each pair of selected tiles, all the actors from one GPP tile are moved to another to generate a new mapping. Each generated mapping is added to a mapping set M after computing its throughput and energy consumption. For the selected $(p + 1)$ GPP tiles, the algorithm finds $(p + 1)$ -choose-2 $\binom{p+1}{2}$ unique pairs and thus evaluates the same number of mappings using p GPP tiles, where $0 < p < n$.

Out of all the evaluated mappings using p GPP tiles, the maximum throughput mapping is selected as the best mapping to evaluate mappings at further-reduced tile count, that is, mappings using $(p - 1)$ GPP tiles by following the steps of Algorithm 1. The same process is repeated until the tile count reduces to one. Thus, all the

Algorithm 2: Heterogeneous Tile-Combinations Mappings Evaluation**Input:** GPP Tiles Mappings M .**Output:** Heterogeneous tile-combinations mappings to be added to set M .

```

for  $tileCount = n$  (number of actors in the AG);  $tileCount \geq 1$ ;  $tileCount --$  do
  Select maximum throughput mapping  $\gamma$  using  $tileCount$  GPP tiles from set  $M$ ;
   $maxNrTileTypesUsed = 1$  // only GPP tiles used;
  repeat
    Initialize the mapping set  $S$ , i.e.,  $S = \{ \}$ ;
    for each GPP tile  $t (\in T)$  in the current mapping  $\gamma$  do
      for each implementation alternative  $\kappa$  (e.g., DSP, ACC, RH tiles) do
        if  $t$  contains actor(s)  $\in A$  and all have their implementation alternatives as  $\kappa$  then
          Move actor(s) to a  $\kappa$  having no previous actor to generate a new mapping  $\delta$ ;
          Compute throughput and energyConsumption of  $\delta$ ;
          Add  $\delta$  with its throughput and energyConsumption to set  $S$  and to global set  $M$ ;
          Move actor(s) back on the initial tile  $t$  to reset  $\gamma$ ;
        end
      end
    end
    Select maximum throughput mapping from set  $S$  and assign as current mapping  $\gamma$ ;
     $maxNrTileTypesUsed ++$ ;
  until  $maxNrTileTypesUsed \leq tileCount$ ;
end

```

mappings using different numbers of GPP tiles get stored into the mapping set M . We have assumed that a GPP implementation alternative is available for all the actors. However, this assumption can easily be removed by allocating the actors to their first available implementation alternatives.

4.1.2. Evaluating Heterogeneous Tile Mappings. The heterogeneous tile-combinations mappings are evaluated by using the GPP tiles mappings (M) obtained in the previous step. Such mappings are possible only when implementation alternatives other than GPP tiles are also available. The mappings are evaluated by following the steps in Algorithm 2 and added to M . At each tile count ($tileCount$), the maximum throughput mapping using GPP tiles is selected to generate mappings at different processing tile-combinations. This type of selection facilitates evaluation of efficient mappings (providing maximum throughput) at heterogeneous tiles as well. For the selected mapping, the actors on each GPP tile are moved to another tile type (implementation alternative) in order to generate a new mapping, provided that all the actors on the GPP tile can be supported on the other tile type. Such movement of actors avoids the evaluation of mappings using non-supported tile combinations. The generated mapping, with its throughput and energy consumption, is added to set M and temporary set S . The mappings at the next possible tile combinations are evaluated by selecting the maximum throughput mapping from the temporary set S . By selecting the maximum throughput mapping at different places in the algorithm, evaluation of inefficient mappings is discarded. The gain in evaluation time is described in experiments in Section 5.

4.1.3. Selecting and Storing Best Mappings at Each Processing Resource Combination. At each possible processing tile combination, we get a number of mappings. This step selects the maximum throughput mapping and minimum energy consumption mapping at each tile combination and stores them into the *mappings with throughput and energy database (MTED)* (Figure 7). In cases when both the maximum throughput and minimum energy consumption mapping are the same, only one mapping is stored. The stored mappings are sorted by the number of tiles used by them in increasing order. The number of used tiles are referred to as the tile count. At each tile count, the mappings

get stored in increasing order of heterogeneity, as explained in Algorithm 2. Increasing heterogeneity implies the use of more numbers of tile types.

Storing the mappings in such order facilitates runtime selection from lower tile count to higher tile count and in increasing order of heterogeneity at each tile count. The runtime approach finds a throughput-satisfying mapping using the minimum number of tiles (tile count) and having minimum energy consumption. While evaluating mappings by Algorithm 2, may all the possible tile combinations may not be covered because the pruning consideration speeds up exploration. In such cases, we need to look for a combination that is a subset of the covered combination at runtime. The runtime algorithm is described later in Section 4.2.

4.1.4. Optimization. Amongst the stored mappings in the database *MTED*, some of them might be sub-optimal. The sub-optimal mappings require a greater number of processing tiles compared to others and have less throughput (performance) and higher energy consumption. For example, for an application, a mapping requiring three GPP and one ACC tiles might have less throughput and higher energy consumption compared to a mapping requiring only two GPP and one ACC tiles. The former mapping is sub-optimal, and it has to cater to larger communication overhead without much gain in parallel processing, thus providing less throughput and consuming higher energy. There is no point in keeping such sub-optimal mappings, so we perform an optimization on *MTED* to discard all such mappings, storing Pareto-optimal mappings (*optimal mappings with throughput and energy (OMTED)*), see Figure 6.

The concepts of Pareto algebra have been used to find the Pareto-optimal mappings [Geilen et al. 2005]. In the optimization process, we compare throughput and energy consumption of mappings that require higher numbers of tiles to ones that require lower numbers of tiles. If the throughput of a mapping using a higher number of tiles is the same as or smaller than the throughput of a mapping using a lower number of tiles, the energy consumption in the latter mapping is the same as or lower than the former mapping, and the processing tiles in the latter mapping are a subset of the processing tiles in the former mapping, then the former mapping is discarded. The same process is performed for each processing tile combination to discard all the sub-optimal mappings. The optimization result includes Pareto-optimal mappings, and each such mapping is better than another in terms of throughput, energy consumption, or resource usage. Keeping only the optimal mappings reduces the memory required to store them and the overhead in selecting the best mapping, since the runtime mapping strategy needs to select from a relatively small set of mappings.

Design-Time Analysis: Complexity. The design-time analysis complexity in terms of the number of actors n , the number of implementation alternatives μ , and max_hop_distance h has been computed. The worst-case complexity (C) is determined by the total number of evaluated mappings (M) in the DSE flow (Figure 7) when all the actors have μ implementation alternatives. For a given value of n , μ , and h , the total number of mappings evaluated over the DSE loops is calculated by Equation (4). The number of homogeneous and heterogeneous tiles mappings are evaluated by Equations (5) and (6), respectively.

$$C = h \times [nr \text{ Homogeneous Tiles Mappings} + nr \text{ Heterogeneous Tiles Mappings}]. \quad (4)$$

$$nr \text{ Homogeneous Tiles Mappings} = 1 + \sum_{p=1}^{n-1} \binom{(p+1)}{2} C_2 = 1 + \sum_{p=1}^{n-1} \left(\frac{p^2}{2} + \frac{p}{2} \right) = 1 + \frac{n^3 - n}{6}. \quad (5)$$

$$\begin{aligned}
nrHeterogeneousTilesMappings &= (\mu - 1) \sum_{p=1}^n \{p + (p - 1) + (p - 2) + \dots + 2 + 1\} \\
&= (\mu - 1) \sum_{p=1}^n \left(\frac{p^2}{2} + \frac{p}{2} \right) = (\mu - 1) \left(\frac{n^3}{6} + \frac{n^2}{2} + \frac{2n}{6} \right).
\end{aligned} \tag{6}$$

Thus, the total number of mappings can be calculated as follows.

$$C = h \times \left[1 + \frac{n^3 - n}{6} + (\mu - 1) \left(\frac{n^3}{6} + \frac{n^2}{2} + \frac{2n}{6} \right) \right] = h \times \left[\frac{\mu n^3}{6} + \frac{(\mu - 1)n^2}{2} + \frac{(2\mu - 3)n}{6} + 1 \right]. \tag{7}$$

In Equation (5), ${}^{(p+1)}C_2$ is the number of unique pairs of GPP tiles at a tile count of $p + 1$. Each pair forms a mapping using p GPP tiles. Heterogeneous tiles mappings are possible only when an actor has more than one implementation alternative, that is, $\mu > 1$, so Equation (6) is valid for $\mu > 1$. At each tile count p , heterogeneous tiles mappings are evaluated by selecting the best mapping using p GPP tiles. The total number of mappings can be calculated from Equation (7), which has a complexity of $O(h\mu n^3)$. The existing strategies evaluate a greater number of mappings compared to our strategy and thus have complexities of higher orders. The mappings evaluated by existing strategies are discussed and compared with our strategy in Section 5.

Design-Time Analysis for a Given Platform Size. The DSE strategy presented in Figure 7 considers a generic MPSoC platform, so the generated mappings are applicable to any target MPSoC platform. For a given platform (PG) containing a smaller number of tiles than the number of actors in the application (AG), the mappings with a greater number of tiles than present in the given platform will never be used. Such mappings have been referred to as infeasible mappings for the given platform. The DSE process can be speeded up by discarding the evaluation of these infeasible mappings.

Evaluation of infeasible mappings is discarded by extending the DSE flow presented in Figure 7. The number of tiles ($nrTiles$) in the given platform is taken as one additional input to the DSE flow. Homogeneous tile mappings are evaluated in a similar manner. While evaluating heterogeneous tile-combination mappings, Algorithm 2 is modified to start the mappings evaluation from the tile-count value of $nrTiles$ in order to discard evaluation of infeasible mappings. All the mappings using a maximum of $nrTiles$ tiles are then selected and stored, which will be applicable to the given platform.

4.2. Runtime Mapping

The design-time analysis step performs all compute-intensive analysis, thus leaving minimum computation for runtime. Runtime mapping of throughput-constrained multimedia applications onto a platform is handled by the runtime manager (Figure 6). Out of the many available processors on the platform, one is used as a manager processor that is responsible for actor mapping, actor scheduling, platform resource control, and configuration control. The resource status is updated at runtime when an actor is loaded on the platform in order to provide the manager processor with accurate knowledge of resource occupancy, which is required for taking the mapping decision based on available resources at runtime. The runtime manager (RTM) maps the applications on the platform one after another, that is, after mapping one application, it goes on to map the next application, and so forth, until all the applications are mapped. The sequential mapping is scalable, because we need not to worry about the large number of scenarios containing different simultaneously active applications, as described in Section 2. The strategy adopted by the RTM to map an application is presented in Algorithm 3.

Algorithm 3: Runtime Mapping Strategy

Input: Application AG , Required throughput τ , Platform PG , Optimized mapping database $OMTED$.
Output: The best *mapping* satisfying the throughput-constraint τ .
 $Max_Tiles_Used = nrActors(AG)$; $Tiles_Available = nrAvailTiles(PG)$; $Max_Tiles_Iter = 0$; $tile_count = 1$;
if $Tiles_Available > 0$ **then**
 $Max_Tiles_Iter = \min(Max_Tiles_Used, Tiles_Available)$;
 repeat
 for each *mapping* ϕ using $tile_count$ tiles in $OMTED$ **do**
 Select closest available $tile_count$ tiles used by ϕ in PG ;
 $hop_max = \text{findMaximumHop}(\text{selected tiles})$;
 $thrMapping = \text{Find}(OMTED, AG, tile_count, \phi, hop_max)$;
 if $\tau \leq thrMapping$ **then**
 $Mapping_list = \text{Find all throughput satisfying mappings using the same resource combination as of } \phi \text{ from } OMTED$;
 Select the *mapping* having minimum *energyConsumption* from $Mapping_list$ and exit;
 end
 end
 $tile_count++$;
 until $tile_count \leq Max_Tiles_Iter$;
 No mapping found;
else
 Application can't be supported, i.e., no mapping found;
end

The strategy takes the application, its desired throughput, the platform with updated resource statuses and the optimized mapping storage $OMTED$ as input and selects the best mapping from the $OMTED$ depending upon the desired throughput and available platform tiles. The selected best mapping satisfies the throughput requirement, uses minimum resources, and has minimum energy consumption. The platform is then configured based on the actors to tiles allocations provided in the selected mapping.

The RTM first finds the maximum number of tiles that might get used (Max_Tiles_Used) by the application and then the number of available tiles ($Tiles_Available$) in the platform. A *mapping* satisfying the throughput constraint of the application ($\tau \leq thrMapping$) and having minimum *energyConsumption* is selected from the $OMTED$ by iterating from the tile count one to Max_Tiles_Iter . The maximum tile iteration value Max_Tiles_Iter is calculated as the minimum of $Tiles_Available$ and Max_Tiles_Used in order to restrict unnecessary search in the $OMTED$. For each mapping using $tile_count$ tiles, the RTM first selects the closest available tiles in the platform, then finds the maximum hop_distance (hop_max) between the selected tiles, and finally the throughput of the mapping ($thrMapping$) is checked against the throughput constraint τ . As soon as a throughput-satisfying mapping ϕ is found ($\tau \leq thrMapping$), all the throughput-satisfying mappings using the same resource combination as ϕ are found and added into a mapping list ($Mapping_list$). Thereafter, the mapping having minimum energy consumption is selected from the mapping list, and the platform is configured based on the selected mapping. If a throughput-satisfying mapping is not found, then the application cannot be supported on the platform with the available resources. In such a case, the application mapping may be tried with relaxed throughput requirements in order to support it on the platform.

Throughput computation for a mapping takes much more time than the time it takes to find the mapping, that is, tasks-to-tiles allocations. Our RTM just selects the best mapping without involving throughput computation at runtime, thus accelerating the runtime mapping process. Further, the selected throughput-satisfying mapping uses a minimum number of tiles, as the search is performed from lower tile count to higher

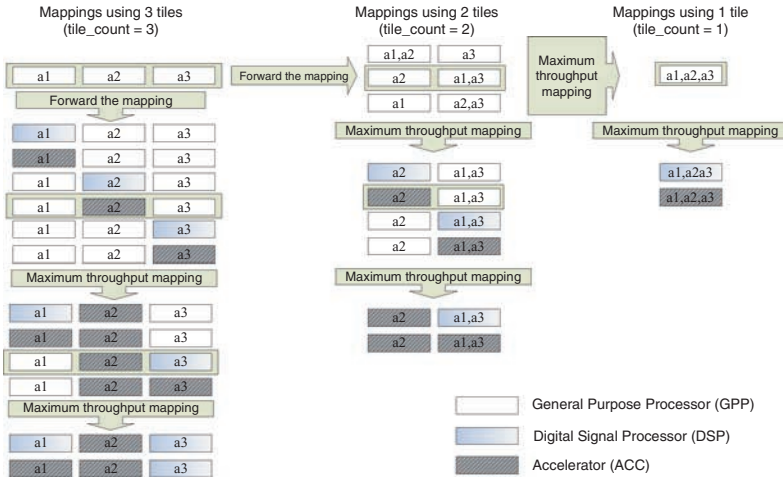


Fig. 8. Design space exploration for an application modeled with three actors: a1, a2, and a3.

tile count. The selected mapping has minimum energy consumption as well. Therefore, the RTM performs effective and efficient mapping.

Hybrid Mapping Flow: Example Demonstration. The hybrid mapping flow has been applied to applications to demonstrate how the flow first performs design-time analysis and then maps the required applications onto a platform at runtime.

Design-Time Analysis. The DSE step of design-time analysis evaluates multiple mappings for an application. Let us consider an application modeled with three actors (a1, a2, and a3), each having implementation alternatives (GPP, DSP, and ACC tiles). The DSE flow first considers a platform containing three tiles of each implementation alternative and then evaluates mappings using GPP tiles, followed by mappings using combinations of GPP, DSP, and ACC tiles. The GPP, DSP, and ACC tiles are represented in different shades, as shown in Figure 8.

Mappings using only GPP tiles are evaluated by the method described in Section 4.1.1. First, 1_actor-to-1_GPP_tile mapping is evaluated, where each GPP tile contains exactly one actor, as shown in Figure 8 (top-left mapping). Only the used tiles of the mapping are shown. The edges are mapped on connections between the tiles, which we have not shown, as we want to focus only on the number of mappings that depend upon the placement of the actors. Here, the tiles are shown in a linear arrangement for each mapping, as we just want to illustrate the DSE flow, whereas in the actual flow, the separation between the tiles can be any fixed value of hop_distance. Next, mappings at a reduced tile count, that is, mappings using two GPP tiles are evaluated by Algorithm 1. The algorithm finds three (3C_2) unique pairs of tiles containing actor(s) from 1_actor-to-1_GPP_tile mapping, as shown in Figure 8. The maximum throughput mapping at each tile count is selected and forwarded to evaluate mappings at reduced tile count. We have considered the highlighted mapping as the maximum throughput one, so the same is forwarded. We get one mapping using one GPP tile. The flow evaluates a total of five mappings, which is the same as the ones calculated from Equation (5), that is, $[1 + (3^3 - 3)/6]$.

Mappings using combinations of GPP, DSP, and ACC tiles are evaluated by Algorithm 2, described in Section 4.1.2. At each tile count, Algorithm 2 takes a maximum throughput mapping using GPP tiles as input and evaluates mappings using combinations of tiles, as shown in Figure 8. Each task is moved from GPP tiles to DSP and ACC

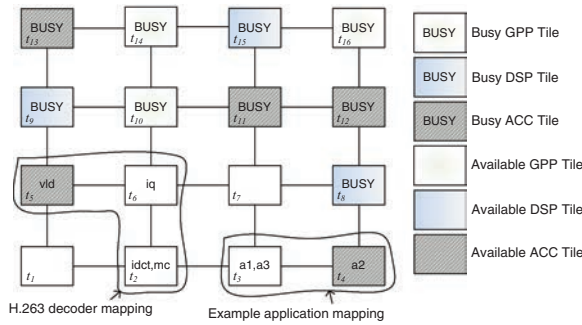


Fig. 9. Runtime mapping of an H.263 decoder (four actors) and the example application (three actors: a1, a2, and a3).

tiles in order to generate mappings. The maximum throughput mapping (highlighted) is selected and forwarded to evaluate mappings at further tile combinations, by moving only the tasks of GPP tiles to DSP and ACC tiles. The same process is repeated until all the tasks of GPP tiles are moved to DSP or ACC tiles. The algorithm evaluates a total of 20 mappings, which is the same as the ones calculated from Equation (6) by setting n and μ equal to 3.

Similarly, DSE can be demonstrated for multimedia applications—H.263 decoder, H.263 encoder, JPEG decoder, and MP3 decoder—modeled with 4, 5, 6, and 14 actors, respectively. Let us assume that the target platform on which the applications need to be mapped is a 4×4 grid of tiles, as shown in Figure 9. For this platform, the value of `max_hop_distance` is six, so the DSE is repeated six times by considering platform tiles separated by `hop_distance` of one to six.

Runtime Mapping. The runtime mapping of the analyzed applications on the target platform is handled by Algorithm 3. The applications are mapped one after another. For each application, the strategy selects the best mapping from the OMTEd subject to the desired throughput and available platform tiles. Let the H.263 encoder, JPEG decoder, and MP3 decoder be mapped already on the platform using the busy tiles (Figure 9). Runtime mapping of the H.263 decoder (Figure 4) and DSE demonstrated application (modeled with three actors a1, a2, and a3) on the available tiles is shown in Figure 9. Let us assume that for the H.263 decoder and the demonstrated application, throughput-satisfying mappings using three and two tiles, respectively, are found, which use different tile-type combinations.

The four actors vld , iq , $idct$, and mc of the H.263 decoder (Figure 4) are mapped onto the three closest available tiles t_2 , t_5 , and t_6 , based on the allocations provided in its mapping, as shown in Figure 9. In the found mapping, all the edges are separated by a `hop_distance` of two. Therefore, mapping the actors on the available tiles (as shown in Figure 9) will satisfy the throughput constraint, as some edges will be mapped at lower `hop_distances` (i.e., lower latencies). Edges are mapped on the connections between the tiles. Similarly, three actors a1, a2, and a3—of the demonstrated application are mapped onto the two closest available tiles, t_3 and t_4 , based on its found mapping, as shown in Figure 9.

5. PERFORMANCE EVALUATION

The proposed hybrid mapping strategy has been implemented as an extension of the publicly available SDF³ tool set [Stuijk et al. 2006]. As a benchmark for evaluating the runtime and quality of the strategy, models of real-life multimedia applications—H.263 decoder (four actors), H.263 encoder (five actors), MPEG-4 decoder (five actors),

JPEG decoder (six actors), sample rate converter (6 actors), MP3 decoder (14 actors), and models of synthetic applications containing varying numbers of actors—have been considered. Experiments are performed on a Core 2 Duo processor at 3.16 GHz.

The same generic platform graph is considered in evaluating the different strategies for an application. In the platform, the number of tiles and their types depend upon the number of actors and their implementation alternatives provided in the application. We consider tile-based architecture, but any other type of architecture can also be considered based on the known latencies between the tiles, as previously discussed. For an actor, the implementation alternative could be GPP, DSP, accelerator, RH, etc. ARM7TDMI [Segars 1997] and Texas Instruments TMS320C6412 [Texas Instruments 2010] are used as the GPP and DSP, respectively. The accelerator for each actor is different, as it is customized for a specific task to be performed by the actor. The RH can be configured to support actors according to their requirements and as an accelerator. The considered applications contain some common actors, and we have considered the same RH for an actor. The common actors' *video length decoding (vld)* [Cho et al. 1999], *inverse quantization (iq)* [Hentati et al. 2011], *inverse discrete cosine transform (idct)* [Sung et al. 2006], *motion compensation (mc)*, *motion estimation (me)*, and *deblocking* [Ren and Kehtarnavaz 2007] are considered to have RH as one of their implementation alternatives. While performing simulation, execution time and power consumption are considered based on the actors' mappings on different types of tiles.

Particularly, we present results obtained from our design-time analysis flow referred to as heuristic analysis (HDSE) flow, and compare them to that of the flows presented in Stuijk et al. [2007], [Stuijk et al. 2010] and an exhaustive analysis (EDSE) flow, adopted in Yang et al. [2002]. We implemented these flows with steps similar to that of our flow in order to make a fair comparison. The flow in Stuijk et al. [2007] performs exploration aimed at load balancing on used tiles. The flow in Stuijk et al. [2010] is applied to scenarios in which each scenario contains a different version of the same application. The different versions model different behaviors of an application at different times. We consider a single scenario, that is, a single version of the application that always has the same behavior. Therefore mappings obtained with this flow can be fairly compared with the HDSE flow. The flow in Yang et al. [2002] performs exploration aimed at power consumption reduction of tiles. We have compared the HDSE flow with previously mentioned flows, as they all perform exploration to evaluate mappings providing different throughput values, and we target throughput-aware runtime mapping. The results from our runtime technique are compared to those of runtime techniques presented in Carvalho and Moraes [2008] and Singh et al. [2010].

A number of experiments have been performed for extensive evaluation of our proposed strategy. First, design-time HDSE results are presented to show how the explored results are stored and used at runtime. Then, the number of mappings evaluated by EDSE (e.g., [Yang et al. 2002]), Stuijk et al. [2010] and HDSE flows are computed and compared to show that the HDSE flow is faster and provides high-quality mappings. All the DSE flows also have been applied on 100 randomly generated applications to show the quality of the mappings and speed-up obtained by the Stuijk et al. [2010] and HDSE flows over those of EDSE. Next, DSE of multimedia applications are performed for given platforms in order to compare the exploration time and best mapping throughput by different DSE flows. Thereafter, runtime mapping results are presented to show the efficiency of our runtime technique over existing techniques.

5.1. Design-Time Analysis

Table IV shows the design-time HDSE results for the H.263 decoder (four actors) at `max_hop_distance` of four when each actor has two implementation alternatives, ARM and RH. The DSE flow runs four times from `hop_distance` of one (`hop_1`) to four (`hop_4`).

For each run, the numbers of evaluated and best mappings at different tile combinations are shown as *nrMaps* and *nrBestMaps*, respectively. A total of 31 mappings are evaluated, which is the same as that calculated from Equation (7) with n and μ as 4 and 2, respectively. At each tile combination, the best mappings (*nrBestMaps*) excel in throughput or energy consumption that are chosen from the evaluated mappings (*nrMaps*). In case of throughput-energy trade-offs, more than one best mapping may need to be stored. At each hop, the best mappings' throughput and energy consumption are shown. When all the actors are mapped on a single tile, the hop_distance is referred to as hop_0, denoted as \times . Similar DSE results have been obtained for other multimedia applications. The results can easily be extended for higher hops by taking a large value of max_hop_distance, which can cater to future larger target platforms. At runtime, one can select a mapping having maximum throughput and minimum energy consumption, depending upon the available tiles and the maximum hop_distance between them.

The Pareto-optimal mappings are highlighted in Table IV. These mappings require fewer numbers of tiles and provide the same or better performance (throughput and energy consumption). It can be seen in Table IV that the mappings using three or four tiles have worse performance than mappings using only two tiles is because there is larger communication overhead when using a larger number of tiles and not much is gained in parallel processing. Similar optimization results have been obtained for other multimedia applications. For applications with a larger number of tasks, it has been observed that the performance increases with the number of tiles used by the mappings and saturates after some fixed number of used tiles.

We have applied an exhaustive analysis (i.e., EDSE, e.g., [Yang et al. 2002]), the flow in Stuijk et al. [2010], and the HDSE flow in order to find the number of mappings to be evaluated by them. The EDSE flow evaluates all the possible mappings at different tile-combinations. The number of mappings evaluated by the EDSE flow increases exponentially with the number of actors. Further, the number of mappings increases even more when the implementation alternatives of the actors, that is, number of tile types on which the actors can be supported, increases. For n actors, each having *nrTileTypes* implementation alternatives, the EDSE flow considers a platform containing n tiles of each implementation alternative and uses a maximum of n tiles in mappings to be evaluated. The total number of mappings is calculated as the number of ways of placing n labeled balls into n unlabeled (but *nrTileTypes*-colored) boxes, where balls and boxes represent tasks and tiles, respectively [OEIS 2012]. The number of mappings by the DSE flow in Stuijk et al. [2010] is limited by X times the number of actors times the number of tiles, where X is the maximum number of partial bindings that is carried over to the next iteration for evaluating the mappings. The HDSE flow considers pruning where maximum throughput mapping is selected for further evaluation, thus limiting the number of mappings to be evaluated.

Table V shows the number of mappings evaluated by the EDSE (e.g., Yang et al. [2002]), the DSE of Stuijk et al. [2010], and HDSE flow, as the numbers of actors (*nrActors*) increases at different numbers of available implementation alternatives (*nrTileTypes*) for each actor. At *nrTileTypes* = 1, the number of mappings evaluated by the EDSE at increasing values of *nrActors* follows bell numbers which represents the number of ways of placing *nrActors*-labeled balls into *nrActors*-indistinguishable boxes [OEIS 2012]. The number of mappings by the Stuijk et al. [2010] flow is shown for $X = 10$. The number of mappings increases with the value of X and may lead to an explosion in the number of mappings. The number of mappings evaluated by HDSE follows Equation (7). For *nrActors* and *nrTileTypes* equal to 14 and 3, respectively, the EDSE evaluates 461,101,962,108 mappings. Evaluating such a large number of

Table IV. DSE Results for an H.263 Decoder

Tile Count	Tile Combinations	nrMaps	nrBestMaps	Best mappings' throughput ($\times 10^{-10}$ /time-units) & energy consumption ($\times 10^{-3}$ mJ)				
				hop.0	hop.1	hop.2	hop.3	hop.4
4	4ARM	1	1	\times	28,655 & 393	28,616 & 515	28,578 & 638	28,540 & 761
	3ARM & 1RH	4	1	\times	29,170 & 301	29,130 & 424	29,090 & 547	29,050 & 670
	2ARM & 2RH	3	2	\times	29,170 & 249	29,130 & 372	29,090 & 495	29,051 & 618
	1ARM & 3RH	2	1	\times	29,612 & 268	29,571 & 390	29,530 & 513	29,489 & 636
		1	1	\times	29,612 & 216	29,571 & 369	29,530 & 461	29,489 & 584
		4RH	1	1	\times	29,612 & 194	29,571 & 317	29,530 & 439
3	3ARM	6	1	\times	62,669 & 352	62,665 & 434	62,661 & 515	62,657 & 597
	2ARM & 1RH	3	1	\times	67,387 & 227	67,382 & 309	67,378 & 390	67,373 & 472
	1ARM & 2RH	2	2	\times	67,387 & 175	67,383 & 257	67,378 & 338	67,374 & 420
		1	1	\times	69,970 & 205	69,965 & 287	69,960 & 369	69,956 & 450
2	3RH	1	1	\times	69,970 & 153	69,965 & 235	69,960 & 317	69,956 & 398
	2ARM	3	1	\times	91,585 & 311	91,583 & 352	91,580 & 393	91,578 & 434
	1ARM & 1RH	2	1	\times	123,230 & 164	123,228 & 205	123,227 & 246	123,226 & 287
1	2RH	1	1	\times	123,230 & 112	123,228 & 153	123,227 & 194	123,226 & 235
	1ARM	1	1	\times	73,961 & 270	\times	\times	\times
	1RH	1	1	\times	106,204 & 71	\times	\times	\times

Table V. Number of Mappings by Exhaustive DSE (EDSE), [Stuijk et al. 2010] DSE, and Heuristic DSE (HDSE) at Different Number of Actors (nrActors) and Their Available Implementation Alternatives (nrTileTypes)

nrActors	EDSE Flow			[Stuijk et al. 2010] Flow			HDSE Flow		
	nrTileTypes			nrTileTypes			nrTileTypes		
	1	2	3	1	2	3	1	2	3
1	1	2	3	1	2	3	1	2	3
2	2	6	12	6	20	42	2	6	10
3	5	22	57	39	102	180	5	15	25
4	15	94	309	100	132	372	11	31	51
5	52	454	1,866	180	410	615	21	56	91
6	203	2,430	12,351	282	612	918	36	92	148
7	877	14,214	88,563	406	854	1,281	57	141	225
8	4,140	89,918	681,870	552	1,136	1,704	85	205	325
9	21,147	610,182	5,597,643	720	1,458	2,187	121	286	451
10	115,975	4,412,798	48,718,569	910	1,820	2,730	166	386	606
14	190,899,322	20,732,504,062	461,101,962,108	1,834	3,668	5,502	456	1,016	1,576

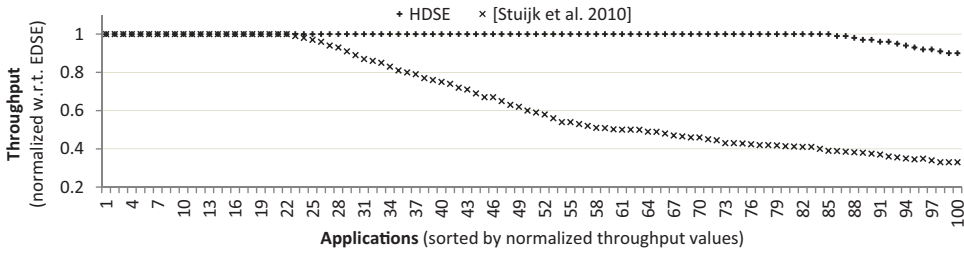


Fig. 10. Quality of mappings by the HDSE and the Stuijk et al. [2010] flow over that of the EDSE for 100 random applications.

mappings would take many years, even if we assume only a few milliseconds for a single mapping. Thus, the EDSE is not scalable, and evaluation is not always feasible.

The HDSE has been employed to speed up the exploration process while providing almost the same quality of mappings as that of EDSE. The flow in Stuijk et al. [2010] also speeds up the exploration process over the EDSE, but the quality of mappings is reduced. Figure 10 shows the quality (throughput) of the best mapping using three tiles for 100 random applications when the EDSE, Stuijk et al. [2010] flow, and the HDSE are employed. The applications are modeled as SDFGs with four, five, six, and seven actors having their implementation alternatives (as ARM and RH tiles) generated randomly. The tiles are assumed to be separated by a fixed hop_distance. The best mapping throughput obtained by HDSE and Stuijk et al. [2010] flow is normalized with respect to EDSE. The normalized throughput values are plotted after sorting them in descending order for each flow. It has been observed that loss in quality of mappings by the HDSE is greater when the number of actors increases and the same best mappings are obtained for more than 80% of the applications. The Stuijk et al. [2010] flow provides mappings having even worse quality than those of HDSE, and the same best mappings are obtained only for about 20% of the applications. Similar behavior is obtained at other resource combinations. For the applications where we don't get the same quality of mappings, the variation in quality with the HDSE is only 10%, whereas it varies largely with the Stuijk et al. [2010] flow, as shown in Figure 10.

Figure 11 shows the speedup obtained by the HDSE and the [Stuijk et al. 2010] flow over that of EDSE for the same set of applications. The applications are sorted by the number of actors within them. A couple of observations can be made from Figure 11. First, the HDSE is faster over the EDSE and Stuijk et al. [2010] flow for all applications.

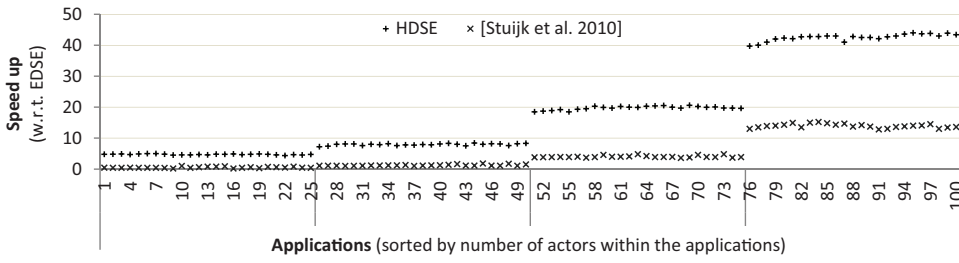


Fig. 11. Speed up obtained by HDSE and [Stuijk et al. 2010] flow over EDSE for 100 random applications.

Second, as the number of actors increases in the applications, the speedup obtained by Stuijk et al. [2010] flow increases, because the difference in the number of evaluated mappings by the EDSE and Stuijk et al. [2010] flow increases as shown in Table V. The HDSE shows further speedup as it evaluates even lower number of mappings. Third, Stuijk et al. [2010] flow is slower than the EDSE for some of the applications, as it evaluates more mappings (including some duplicates) than the speedup EDSE. Thus, exploration provides almost the same quality of mappings when employing the HDSE. We also have applied DSE flows on a number of multimedia applications: H.263 decoder (four actors), H.263 encoder (five actors), and JPEG decoder (six actors). It has been observed that for H.263 decoder/encoder, the best mapping at different tile-combinations is the same as those of the EDSE and HDSE, whereas for the JPEG decoder, the HDSE misses the best mapping at few tile-combinations.

Design Space Exploration for a Given Platform. We performed multimedia application DSE for given platforms that may contain any arbitrary number of tiles. Table VI shows the DSE results for the following multimedia applications: H.263 decoder, H.263 encoder, and sample rate converter for platforms containing 1×2 , 2×2 , 3×3 , and 4×4 grid of tiles. For each platform, the exploration time (milliseconds) and the best mapping throughput ($\times 10^{-12}$ /time units) have been tabulated when the exploration approaches of Stuijk et al. [2010], EDSE, and HDSE are employed. The different platform tiles are ARM tiles. The number of evaluated mappings by the Stuijk et al. [2010] approach depends upon the number of tiles present in the platform. So, for larger platforms (containing a greater number of tiles), the Stuijk et al. [2010] approach evaluates higher numbers of mappings, thus showing increased exploration time, as shown in Table VI. The approach evaluates some duplicate mappings which differ only in the placement of actors on different tiles providing the same throughput. So, in some cases, it evaluates a larger number of mappings (including duplicates) than the EDSE, thus taking more time than the EDSE, as shown in Table VI. The EDSE flow evaluates all the possible mappings without any duplicate ones, and the HDSE flow performs pruning to discard evaluation of inefficient mappings. The EDSE and HDSE flows are executed in a similar manner. Larger platforms are covered by executing the flow repeatedly by considering higher separation (hop_distance) between the tiles. For 1×2 , 2×2 , 3×3 , and 4×4 platforms, the maximum hop_distance between the tiles is 1, 2, 4, and 6 respectively, so the flow is repeated maximum hop_distance times by increasing the delay of connections according to the hop_distance. In each execution of the flow (for the H.263 decoder (four actors), the H.263 encoder (five actors), and the sample rate converter (six actors)), the EDSE flow evaluates a total of 15, 52, and 203 mappings, whereas the HDSE flow evaluates a total of 11, 21, and 36 mappings, respectively, as previously discussed earlier in Table V. Table VI shows the exploration time for complete execution of the EDSE and HDSE flows. The difference in the number of explored mappings by the EDSE and HDSE flows increases with the number of actors in the application and thus

Table VI. Multimedia Applications DSE at Different Platforms for Exploration Time (seconds) and Best Mappings' Throughput ($\times 10^{-6}$ /time-units) and Energy Consumption ($\times 10^{-3}mJ$)

Application	Platform	Exploration Time (seconds)			Best mappings' throughput & energy consumption	
		[Stuijk et al. 2010]	EDSE	HDSE	[Stuijk et al. 2010]	EDSE & HDSE
H.263 decoder (4 actors)	1×2	7.24	4.73	2.89	7.40 & 270	9.16 & 311
	2×2	11.47	9.47	5.78	7.40 & 270	9.16 & 311
	3×3	19.01	18.95	11.57	7.40 & 270	9.16 & 311
	4×4	24.73	28.44	17.35	7.40 & 270	9.16 & 311
H.263 encoder (5 actors)	1×2	10.57	18.55	5.81	0.54 & 4520	0.79 & 4261
	2×2	21.91	37.12	11.63	0.54 & 4520	0.79 & 4261
	3×3	38.65	74.15	23.26	0.54 & 4520	0.79 & 4261
	4×4	45.37	111.36	34.89	0.54 & 4520	0.79 & 4261
sample rate converter (6 actors)	1×2	59.41	111.58	19.78	410.00 & 4.87	410.00 & 4.87
	2×2	118.84	202.34	38.57	410.00 & 4.87	410.00 & 4.87
	3×3	236.77	426.72	78.16	410.00 & 4.87	410.00 & 4.87
	4×4	356.63	632.56	107.82	410.00 & 4.87	410.00 & 4.87

the percentage savings (difference) in the exploration time. Further, the evaluation by EDSE is not feasible within a reasonable time for applications with a larger number of tasks, whereas HDSE converges fast. For example, HDSE evaluates 456 mappings for the MP3 decoder (14 actors) in a single iteration, taking close to 103,167 milliseconds, whereas EDSE needs to evaluate 190,899,322 mappings (Table V) which will take more than a year (which is unacceptable).

It can be observed from Table VI that the HDSE flow does not miss the best throughput mapping despite requiring much less time for exploration. On the average, for H.263 decoder, H.263 encoder, and sample rate converter, exploration time of the HDSE flow is lowered by 39%, 35%, and 68% compared to the Stuijk et al. [2010] flow, and by 38%, 68%, and 83% compared to the EDSE flow, respectively. The best mapping throughput for the H.263 decoder, H.263 encoder, and sample rate converter is improved by 23%, 37%, and 38%, respectively, over the Stuijk et al. [2010] approach.

Figure 12 shows the throughput for the best mappings for multimedia applications where different numbers of ARM tiles are used for the HDSE flow and the flows presented in Stuijk et al. [2007] and Stuijk et al. [2010]. The throughput at each tile count (number of used tiles by the mappings) has been normalized with respect to the HDSE flow. It can be observed that the HDSE flow always provides better quality (throughput) of mappings at all tile counts. For each application, the same best mapping is obtained by all the flows at platforms containing one tile (1 tile) and platforms containing the same number of tiles as number of actors.

We also performed DSE of multimedia applications for given platforms containing different types of tiles, such as GPP, DSP, and RH tiles. For a given platform containing five GPP and five RH tiles, the HDSE flow explores a total of 56 mappings for the H.263 encoder (five actors) when each actor has implementation alternatives of GPP and RH tiles. The exploration took a runtime of 35,178 ms. For given platforms of two GPP and one RH tiles and one GPP and one RH tiles, the HDSE flow takes tile-count values of three and two respectively, and evaluates 31 and 25 mappings in a runtime of 19,683 and 15,825 ms, respectively. For given platforms containing tiles which are a subset of the total available implementation alternatives of actors in the application, the HDSE flow discards evaluations of infeasible mappings requiring more tiles than are available. Thus, the DSE process gets speed up in the case of smaller platforms.

5.2. Runtime Mapping

The results obtained from our runtime strategy have been compared with existing runtime strategies: Nearest Neighbor (NN) proposed in Carvalho and Moraes [2008] and

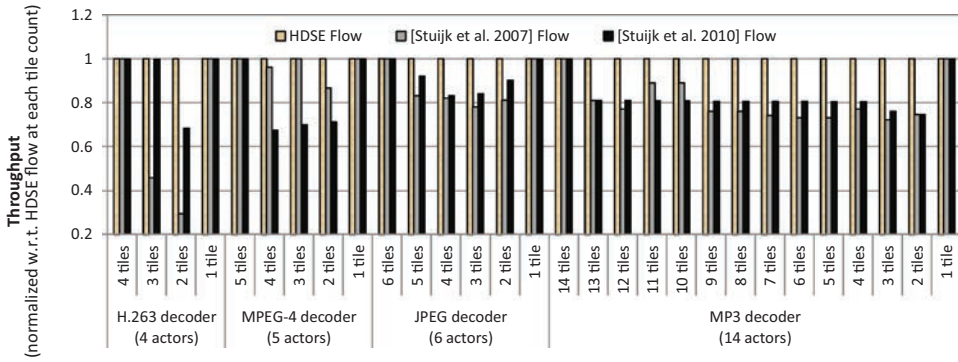


Fig. 12. The best mapping throughput comparison at different platforms for different applications.

Table VII. Time Required (in ms) to Map the Applications by Different Runtime Mapping Strategies

	NN	CNN	Our Runtime
H.263 decoder	27.98	27.96	2.47
H.263 encoder	29.98	29.97	2.84
JPEG decoder	35.74	35.32	3.21
MP3 decoder	771.87	771.83	3.93

Communication-aware Nearest Neighbor (CNN) proposed in Singh et al. [2010]. NN and CNN start mapping an application without any previous analysis and perform the required analysis at runtime. The NN strategy tries to map the communicating actors on the neighboring tiles, whereas the CNN strategy tries to map the maximum communicating pairs of actors on the same tile and throughput and energy consumption for the mapping are then computed at runtime. Throughput computation for a mapping takes much more time than the time to find the mapping. These strategies need to find a new mapping and then calculate throughput and energy consumption for the same, in case throughput-constraint is not fulfilled with the current mapping. Such strategies take time, first in finding a mapping and second in computing throughput and energy consumption for the mapping at runtime, whereas our strategy just selects the best mapping satisfying the throughput-constraint from the optimized mappings database. The selected mapping is used to configure the actors on the platform tiles. So, in our runtime strategy, the total time consists of selection and placement time only. Table VII shows the time required (in milliseconds) to map throughput-constrained multimedia applications on a 4×4 MPSoC platform when the NN, CNN, and our runtime mapping strategy are employed. On average, our runtime strategy is faster by about 93% compared to CNN (which requires less time than NN).

Penalty for Overestimation of hop_distances. Our flow evaluates mappings by assuming that all the platform tiles are separated by some fixed hop_distance. However, in real situations, it is quite possible that not all available tiles at runtime are at the same hop_distance. Thus, our flow enforces a penalty for estimating higher hop_distances. At runtime, we look for a throughput satisfying mapping from the explored design-time mappings, which contains tiles separated by the maximum possible hop_distance between the available tiles. So, mapping the actors on the available tiles based on the found mapping will definitely satisfy the throughput constraint, since latency of some connections will be smaller compared to ones considered during analysis. To map the H.263 decoder on four ARM tiles when all edges are mapped at a hop_distance of two, that is, tiles containing actors are separated by two hops, the throughput

is 2.86168×10^{-6} (1/time units) (Table IV), and when two edges are mapped at a hop_distance of one and the remaining edges at a hop_distance of two, the throughput is 2.86343×10^{-6} (1/time units). The two throughput values vary only by 0.0006%, thus there is very little penalty in overestimating hop_distances. Thus, the stored results from our design-time analysis are acceptable for use in runtime mapping. Further, we always get better throughput than the stored one, as actors are mapped on available tiles, making the approach suitable for real time situations.

6. CONCLUSIONS

It has been observed that most of the existing mapping strategies perform mapping either at design time or at runtime without any previous analysis of the applications. A design-time strategy is incapable of handling dynamism in applications, and a runtime approach can miss the timing deadline due to large computation requirements. This article describes a hybrid mapping strategy for efficient mapping of throughput-constrained applications on MPSoC platforms. The hybrid strategy first performs extensive design-time analysis of the applications, providing multiple design points. This is followed by a runtime mapping strategy to select the best point from the many available points (subject to available resources and desired throughput) in order to map an application. The best selected point satisfies the throughput constraint and has minimum energy consumption and resource usage.

Our flow considers a generic MPSoC platform while performing design-time analysis, so the generated design points are applicable to any MPSoC. The analysis strategy is scalable with the number of application tasks and platform tiles. During design-time analysis, an optimization is performed on the design points to discard sub-optimal points that results in reduced memory requirement to store them and facilitates faster runtime selection. Our design-time analysis is very fast and provides better-quality of solutions when compared to other approaches. Our runtime mapping strategy is very efficient as it uses the design-time analysis results in contrast to the conventional runtime approaches where the time-consuming analysis is performed at runtime.

In the future, we plan to develop more efficient runtime mapping strategies. We also plan to incorporate task migration to further optimize throughput and energy consumption at runtime by migrating tasks from one processor type to another processor type.

ACKNOWLEDGMENTS

The authors would like to thank the reviewers for their valuable feedback that has improved the quality of the article. The authors would also like to thank Dr. Sander Stuijk of the Eindhoven University of Technology for providing specifications about the application scenarios used in Stuijk et al. [2010].

REFERENCES

- AHN, Y., HAN, K., LEE, G., SONG, H., YOO, J., CHOI, K., AND FENG, X. 2008. SoCDAL: System-on-chip design accelerator. *ACM Trans. Des. Autom. Electron. Syst.* 13, 17, 1–38.
- ANGIOLINI, F., CENG, J., LEUPERS, R., FERRARI, F., FERRI, C., AND BENINI, L. 2006. An integrated open framework for heterogeneous MPSoC design space exploration. In *Proceedings of the Design, Automation and Test Conference in Europe*. 1–6.
- ASCIA, G., CATANIA, V., DI NUOVO, A. G., PALESI, M., AND PATTI, D. 2007. Efficient design space exploration for application specific systems-on-a-chip. *J. Syst. Archit.* 53, 733–750.
- BENINI, L., BERTOZZI, D., AND MILANO, M. 2008. Resource management policy handling multiple use-cases in MPSoC platforms using constraint programming. In *Proceedings of the International Conference on Logic Programming*. 470–484.
- BONFIETTI, A., LOMBARDI, M., MILANO, M., AND BENINI, L. 2009. Throughput constraint for synchronous data flow graphs. In *Proceedings of the International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. 26–40.

- BORKAR, S. 2007. Thousand core chips: A technology perspective. In *Proceedings of the Annual Design Automation Conference*. 746–749.
- CARVALHO, E. AND MORAES, F. 2008. Congestion-aware task mapping in heterogeneous MPSoCs. In *International Symposium on System-on-Chip (SoC)*. 1–4.
- CHO, S. H., XANTHOPOULOS, T., AND CHANDRAKASAN, A. 1999. A low power variable length decoder for MPEG-2 based on nonuniform fine-grain table partitioning. *IEEE Trans. Very Large Scale Integ. (VLSI) Syst.* 7, 2, 249–257.
- GANGWAL, O. P., RADULESCU, A., GOOSSENS, K., PESTANA, S. G., AND RIJPKEMA, E. 2005. Building predictable systems on chip: An analysis of guaranteed communication in the Æthereal network on chip. In *Dynamic and Robust Streaming in and between Connected Consumer-Electronic Devices*, vol. 3, Springer, 1–36.
- GEILEN, M., BASTEN, T., THEELEN, B., AND OTTEN, R. 2005. An algebra of Pareto points. In *Proceedings of the International Conference on Application of Concurrency to System Design*. 88–97.
- GHAMARIAN, A. H., GEILEN, M. C. W., STULJK, S., BASTEN, T., THEELEN, B. D., MOUSAVI, M. R., MOONEN, A. J. M., AND BEKOOLJ, M. J. G. 2006. Throughput analysis of synchronous data flow graphs. In *Proceedings of the International Conference on Application of Concurrency to System Design*. 25–36.
- GIOVANNI, B., FOSSATI, L., AND SCIUTO, D. 2010. Decision-theoretic design space exploration of multiprocessor platforms. *IEEE Trans. Comput. Aided Des. Integ. Cir. Sys.* 29, 1083–1095.
- GOOSSENS, K., DIELISSSEN, J., AND RADULESCU, A. 2005. Æthereal network on chip: Concepts, architectures, and implementations. *IEEE Des. Test* 22, 5, 414–421.
- GRECU, C., PANDE, P., IVANOV, A., AND SALEH, R. 2005. Timing analysis of network on chip architectures for mp-soc platforms. *Microelectronics J.* 36, 9, 833–845.
- HENTATI, M., AOUNI, Y., NEZAN, J., ABID, M., AND DEFORGES, O. 2011. FPGA dynamic reconfiguration using the RVC technology: Inverse quantization case study. In *Proceedings of the Conference on Design and Architectures for Signal and Image Processing*. 1–7.
- HU, J. AND MARCULESCU, R. 2004. Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints. In *Proceedings of the conference on Design, automation and Test in Europe (DATE'04)*.
- JIA, Z. J., PIMENTEL, A., THOMPSON, M., BAUTISTA, T., AND NUNEZ, A. 2010. NASA: A generic infrastructure for system-level MP-SoC design space exploration. In *Proceedings of the Workshop on Embedded Systems for Real-Time Multimedia*. 41–50.
- KEINERT, J., STREUBÜHR, M., SCHLICHTER, T., FALK, J., GLADIGAU, J., HAUBELT, C., TEICH, J., AND MEREDITH, M. 2009. SystemCoDesigner—an automatic ESL synthesis approach by design space exploration and behavioral synthesis for streaming applications. *ACM Trans. Des. Autom. Electron. Syst.* 14, 1, 1–23.
- KIM, M., BANERJEE, S., DUTT, N., AND VENKATASUBRAMANIAN, N. 2008. Energy-aware cosynthesis of real-time multimedia applications on MPSoCs using heterogeneous scheduling policies. *ACM Trans. Embed. Comput. Syst.* 7, 1, 1–19.
- KISTLER, M., PERRONE, M., AND PETRINI, F. 2006. Cell multiprocessor communication network: Built for speed. *IEEE Micro* 26, 10–23.
- KUMAR, A., FERNANDO, S., HA, Y., MESMAN, B., AND CORPORAAL, H. 2008. Multiprocessor systems synthesis for multiple use-cases of multiple applications on FPGA. *ACM Trans. Des. Autom. Electron. Syst.* 13, 40, 1–27.
- LEE, E. A. AND MESSERSCHMITT, D. G. 1987. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Trans. Comput.* 36, 24–35.
- LELTEN, J., VAN MEERBERGEN, J., TIMMER, A., AND JESS, J. 1997. PROPHID: A heterogeneous multi-processor architecture for multimedia. In *Proceedings of the International Conference on Computer Design*. 164–169.
- LIU, W., YUAN, M., HE, X., GU, Z., AND LIU, X. 2008. Efficient SAT-based mapping and scheduling of homogeneous synchronous dataflow graphs for throughput optimization. In *Proceedings of the Real-Time Systems Symposium*. 492–504.
- LUKASIEWYCZ, M., GLASS, M., HAUBELT, C., AND TEICH, J. 2008. Efficient symbolic multi-objective design space exploration. In *Proceedings of the Asia and South Pacific Design Automation Conference*. 691–696.
- MARIANI, G., AVASARE, P., VANMEERBEECK, G., YKMAN-COUVREUR, C., PALERMO, G., SILVANO, C., AND ZACCARIA, V. 2010. An industrial design space exploration framework for supporting run-time resource management on multi-core systems. In *Proceedings of the Conference on Design, Automation and Test in Europe*. 196–201.
- MOREIRA, O., MOL, J. J.-D., AND BEKOOLJ, M. 2007. Online resource management in a multiprocessor with a network-on-chip. In *Proceedings of the Symposium on Applied Computing*. 1557–1564.

- MOREIRA, O., VALENTE, F., AND BEKOOLJ, M. 2007. Scheduling multiple independent hard-real-time jobs on a heterogeneous multiprocessor. In *Proceedings of the International Conference on Embedded Software*. 57–66.
- NOLLET, V., AVASARE, P., EECKHAUT, H., VERKEST, D., AND CORPORAAL, H. 2008. Run-time management of a MPSoC containing FPGA fabric tiles. *IEEE Trans. Very Large Scale Integr. Syst.* 16, 24–33.
- OEIS. 2012. Encyclopedia of integer sequences. <http://oeis.org/>.
- PALERMO, G., SILVANO, C., AND ZACCARIA, V. 2005. Multi-objective design space exploration of embedded systems. *J. Embed. Comput.* 1, 305–316.
- PALERMO, G., SILVANO, C., AND ZACCARIA, V. 2008. Robust optimization of SoC architectures: A multi-scenario approach. In *Proceedings of the Workshop on Embedded Systems for Real-Time Multimedia*. 7–12.
- PALMA, J., MARCON, C., MORAES, F., CALAZANS, N., REIS, R., AND SUSIN, A. 2005. Mapping embedded systems onto NoCs—The traffic effect on dynamic energy estimation. In *Proceedings of the Symposium on Integrated Circuits and Systems Design*. 196–201.
- PAULIN, P. G., PILKINGTON, C., BENSODANE, E., LANGEVIN, M., AND LYONNARD, D. 2004. Application of a multi-processor SoC platform to high-speed packet forwarding. In *Proceedings of the Conference on Design, Automation and Test in Europe*. 58–63.
- REN, J. AND KEHTARNAVAZ, N. 2007. Comparison of power consumption for motion compensation and deblocking filters in high definition video coding. In *Proceedings of the International Symposium on Consumer Electronics*. 1–5.
- RUTTEN, M. J., VAN ELJNDHOVEN, J. T. J., JASPERS, E. G. T., VAN DER WOLF, P., POL, E.-J. D., GANGWAL, O. P., AND TIMMER, A. 2002. A heterogeneous multiprocessor architecture for flexible media processing. *IEEE Des. Test* 19, 39–50.
- SCHRANZHOFER, A., CHEN, J.-J., AND THIELE, L. 2010. Dynamic power-aware mapping of applications onto heterogeneous MPSoC platforms. *IEEE Trans. Ind. Inf.* 6, 4, 692–707.
- SEGARS, S. 1997. ARM7TDMI power consumption. *IEEE Micro* 17, 4, 12–19.
- SINGH, A. K., JIGANG, W., PRAKASH, A., AND SRIKANTHAN, T. 2009. Efficient heuristics for minimizing communication overhead in noc-based heterogeneous MPSoC platforms. In *Proceedings of the International Symposium on Rapid System Prototyping*. 55–60.
- SINGH, A. K., KUMAR, A., AND SRIKANTHAN, T. 2011. A hybrid strategy for mapping multiple throughput-constrained applications on MPSoCs. In *Proceedings of the International Conference on Compilers, Architectures and Synthesis of Embedded Systems*.
- SINGH, A. K., SRIKANTHAN, T., KUMAR, A., AND JIGANG, W. 2010. Communication-aware heuristics for run-time task mapping on NoC-based MPSoC platforms. *J. Syst. Archit.* 56, 242–255.
- STULJK, S., BASTEN, T., GEILEN, M. C. W., AND CORPORAAL, H. 2007. Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs. In *Proceedings of the 44th Annual Design Automation Conference*. 777–782.
- STULJK, S., GEILEN, M., AND BASTEN, T. 2006. SDF³: SDF for free. In *Proceedings of the 6th International Conference on Application of Concurrency to System Design*. 276–278.
- STULJK, S., GEILEN, M., AND BASTEN, T. 2010. A predictable multiprocessor design flow for streaming applications with dynamic behaviour. In *Proceedings of Euromicro Conference on Digital System Design*. 548–555.
- SUNG, T.-Y., SHIEH, Y.-S., YU, C.-W., AND HSIN, H.-C. 2006. High-efficiency and low-power architectures for 2-D DCT and IDCT based on CORDIC rotation. In *International Conference on Parallel and Distributed Computing, Applications and Technologies*. 191–196.
- Texas Instruments. 2010. TMS320C6412 DSP. <http://www.ti.com/product/tms320c6412>.
- TILE-Gx100 2009. First 100-core processor with the new TILE-Gx family. <http://www.tilera.com/products/processors/TILE-Gx-Family>.
- VAN STRALEN, P. AND PIMENTEL, A. 2010. Scenario-based design space exploration of MPSoCs. In *International Conference on Computer Design*. 305–312.
- VANGAL, S., HOWARD, J., RUHL, G., DIGHE, S., WILSON, H., TSCHANZ, J., FINAN, D., IYER, P., SINGH, A., JACOB, T., JAIN, S., VENKATARAMAN, S., HOSKOTE, Y., AND BORKAR, N. 2007. An 80-tile 1.28TFLOPS network-on-chip in 65nm CMOS. In *Proceedings of the International Solid-State Circuits Conference*. 98–589.
- YANG, P., MARCHAL, P., WONG, C., HIMPE, S., CATTHOOR, F., DAVID, P., VOUNCKX, J., AND LAUWEREINS, R. 2002. Managing dynamic concurrent tasks in embedded real-time multimedia systems. In *Proceedings of the International Symposium on System Synthesis*. 112–119.
- YANG, Z., KUMAR, A., AND HA, Y. 2010. An area-efficient dynamically reconfigurable spatial division multiplexing network-on-chip with static throughput guarantee. In *Proceedings of the International Conference on Field-Programmable Technology*. 389–392.

- YKMAN-COUVREUR, C., AVASARE, P., MARIANI, G., PALERMO, G., SILVANO, C., AND ZACCARIA, V. 2011. Linking run-time resource management of embedded multi-core platforms with automated design-time exploration. *Computers Digital Techniques, IET* 5, 2, 123–135.
- YKMAN-COUVREUR, C., NOLLET, V., CATTLOOR, F., AND CORPORAAL, H. 2006. Fast multi-dimension multi-choice knapsack heuristic for MP-SoC run-time management. In *Proceedings of the International Symposium on System-on-Chip*. 1–4.
- ZAMORA, N. H., HU, X., AND MARCULESCU, R. 2007. System-level performance/power analysis for platform-based design of multimedia applications. *ACM Trans. Des. Autom. Electron. Syst.* 12, 2, 1–29.

Received September 2011; revised April 2012; accepted August 2012