RAPIDITAS: <u>RAPI</u>d <u>D</u>esign-space-exploration <u>I</u>ncorporating <u>T</u>race-based <u>A</u>nalysis and <u>S</u>imulation

Amit Kumar Singh, Anup Das and Akash Kumar
Department of Electrical and Computer Engineering, National University of Singapore, Singapore
Email: {eleaks, akdas, akash}@nus.edu.sg

***Abstract*—Simulation-based Design Space Exploration (DSE) to evaluate all possible mappings for a given application and Multiprocessor-System-on-Chip (MPSoC) platform is computationally costly for large problems. Even using efficient exploration methodologies to evaluate the mappings cannot overcome the evaluation time bottleneck. This paper presents a novel DSE methodology that analyzes the execution trace to prune the vast design space. Simulations are employed only on the pruned design points (mappings), hence reducing the number of simulations. The methodology performs iterative exploration and provides premier mappings requiring different number of processors, which can be used at run-time subject to desired performance and available platform processors. We evaluate our methodology by using models of real-life multimedia applications and demonstrate that the DSE time is reduced by 72% while generating high quality mappings.**

***Keywords*-Multiprocessor-System-on-Chip (MPSoC); Design Space Exploration; Run-time Mapping;**

I. Introduction

Embedded system designers need to handle several new challenges with the increasing complexity of modern embedded applications. In particular, the applications need efficient mapping on MPSoC platform in order to satisfy their performance constraints [1] [2]. Moreover, in order to support dynamism, multiple applications have to run concurrently on the MPSoC platform with the capability to accept new applications at run-time [1]. The run-time mapping can be assisted by DSE results (mappings explored at design-time) but computational complexity of the DSE needs to be reduced in order to have an acceptable exploration time. This necessitates the need to develop novel DSE methodologies towards performing rapid exploration.

Traditional DSE methodologies fall short as the number of possible mappings increases exponentially with the complexity (number of tasks) of the applications. For an application containing 14 tasks, a total of 190,899,322 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. Therefore, there is a challenge to finish the evaluation within a limited time without missing the efficient mappings when the number of tasks in the application is high. As an outcome, recently, there has been focus on devising efficient and effective DSE methodologies to identify the premier mappings[1]. The mappings contain allocation of application tasks on the MPSoC resources. The DSE methodologies can have one or multiple objectives, like energy consumption, compute performance etc. If the complete explored space pertaining to an objective is given, then one can select the candidates that meet the performance constraints. However, for application with high complexity, in practice, it might be infeasible to explore the complete design space within an acceptable (limited) time.

[1]Premier implies maximizing objective function (e.g. throughput) for given resources

State-of-the-art DSE methodologies explore a finite number of design points (mappings) and retain the Pareto-optimal points in terms of performance or power consumption. The best point amongst the Pareto-optimal points can be selected to design a system depending upon the user requirements or to facilitate for efficient run-time mapping depending upon the available system resources. The best point may refer to the Pareto-optimal point having the maximum performance. Since only a finite number of design points are evaluated, there is no guarantee to find the absolute optimum in the design space. However, the design space is reduced to a set of design points that are close to the optimum. Further, most of the current DSE methodologies use simulation-based approaches that are computationally costly [3] [4] [5]. The simulation time to evaluate the design points forms the real bottleneck in the DSE. In order to accelerate the DSE, analytical estimations can be considered [6] [7]. However, accuracy of the estimations is restricted as sufficiently required system behavior cannot be captured. Therefore, estimation based approaches are fast but less accurate, whereas simulation based approaches are accurate but slower. In order to cope with vast design space, some accuracy can often be traded for performing faster exploration.

Contribution: This paper addresses the shortcomings of simulative & analytical approaches and proposes a methodology for performing efficient DSE. To evaluate mappings, the proposed methodology combines simulations and analytical estimations in order to perform accurate and fast exploration. The methodology performs iterative evaluations. In each iteration, first analytical estimations are used to compute throughput of different possible mappings, then simulation is performed only for the best mapping. The mapping having maximum throughput has been referred to as the best mapping. The best mapping in the current iteration is simulated as it is used to estimate throughput of mappings in the next iteration. These intermediate simulations facilitate for near accurate estimations. The analytical estimation step analyzes execution traces of application tasks and edges for the best mapping and estimates throughput of mappings that can be generated by allocating tasks from two different resources to the same resource. Simulative evaluation is employed on the best generated mapping to get accurate throughput and to ensure for better estimation accuracy in the next iteration. Therefore, our flow limits the number of simulations to the number of iterations and explores the best mappings accurately. We evaluate the proposed methodology and demonstrate that by properly using simulative and analytical estimations, the exploration time can be greatly reduced while providing similar quality of mappings as that of simulation-based exploration.

The remainder of this paper is organized as follows. Section II provides an overview of state-of-the-art DSE methodologies. Section III introduces the preliminaries. Section IV presents the proposed methodology. The results to evaluate our methodology are presented in Section V. Finally, Section VI concludes the paper along with some future research directions.

II. Related Work

Several DSE flows that explore multiple mappings for an application have been reported in [4], [5], [8], [9], [10] and [11]. They perform exploration aiming to optimize some performance metrics such as compute and energy efficiency. The explored mappings can be used to design a system or to handle dynamism in resource availability at run-time depending upon different throughput requirements. However, the explored mappings may not be optimal and will be applicable only to the fixed platform considered during DSE. In [12], exploration is performed for a set of platforms and thus the mappings' applicability gets extended. In [13], exploration is performed for a generic platform, extending applicability of mappings to variety of platforms. Most of the aforementioned DSE flows use pure simulative evaluations and thus exhibit high exploration time that might not be acceptable.

The DSE flows that use analytical estimations along with the simulative evaluations are presented in [6], [7] and [12]. These approaches perform DSE first by using analytical models to rapidly identify the points of interest in the design space, then by using simulative evaluations on the interesting points to find the premier design points more accurately. Thus, for most of the design points (mappings), these flows try to use analytical estimations in place of the simulative evaluations, resulting in acceleration of the overall DSE process. However, if the analytical estimations are not accurate enough due to insufficient capturing of the system behavior, then the best set of mappings will not get simulated, resulting in wrong set of premier mappings. In [14], a hybrid approach that integrates estimation and simulation phases is presented. This approach uses analytical estimations to compute throughput for majority of the mappings and interleave the estimations with simulative evaluations in order to ensure that premier mappings are explored accurately. The accuracy of the results depends upon the percentage of simulations in the whole DSE process.

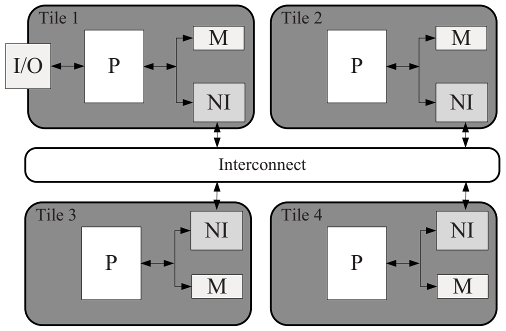

Figure 1. Tile-based example multiprocessor platform.

In contrast to above approaches, our DSE flow uses a fixed minimum number of simulations for a given application and explores premier mappings. Our flow performs iterative evaluations. In each iteration, first analytical estimations are used to compute throughput of different possible mappings, then simulation is performed only for the best mapping. Thus, our flow limits the number of simulations to the number of iterations and explores the premier mappings accurately.

III. Preliminaries

In this section, we provide a brief overview of the MPSoC platform and application models, and the main problems encountered during DSE. This overview is necessary for proper understanding of the contributions presented in Section IV.

A. *Multiprocessor Platform Model*

The multiprocessor platform used in this work is modeled as the tile-based platform template described in [15]. Fig. 1 shows an example platform containing four tiles. The tiles are connected through a network interface (NI) to an interconnection network that provides point-to-point connections between the tiles. These connections can be implemented through a network-on-chip (NoC) by providing latencies of connections between tiles according to the NoC [16]. Therefore, any type of interconnection network can be modeled so long as the latencies between tiles are provided. Each tile contains a processor (P), a local memory (M) and the NI that is accessed by the interconnect and local processor. The NI has a fixed number of input/output connections that provide maximum incoming/outgoing bandwidth when used fully. Multiprocessor systems such as StepNP [17] and PROPHID [18] fit nicely into this platform model.

B. *Application Model*

The applications considered are multimedia applications with timing constraints and they are modeled as Synchronous Dataflow Graphs (SDFGs) [19]. Fig. 2 shows

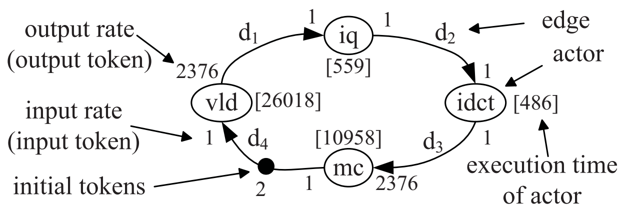

Figure 2. SDFG model of an H.263 decoder.

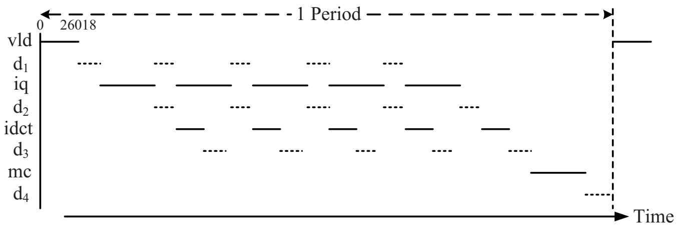

Figure 3. Execution trace of H.263 decoder for one periodic execution.

SDFG model of H.263 decoder. The nodes (*vld, iq, idct* & *mc*) and edges (d_1, d_2, d_3 & d_4) model tasks and dependencies respectively. The nodes are referred to as *actors* that communicate with *tokens* sent from one actor to another through the edges. An actor has following attributes: execution time and memory requirement when mapped on a tile. An edge has following attributes: size of a token, memory needed when connected actors are allocated to the same tile, memory needed in source and destination tiles when connected actors are allocated to different tiles and respective bandwidth requirements. The edges may contain *initial tokens* indicated by a bullet point as in Fig. 2. 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. At each firing, the actor 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 application model specifies a throughput-constraint as well, which is important for multimedia applications. Throughput is determined as the inverse of the long term period, i.e., the average time needed for one iteration of the application. An iteration is defined as the minimum non-zero execution such that the original state of the SDFG is obtained. For the example H.263 decoder, period is equal to the summation of ExecTime(*vld*), 2376×ExecTime(*iq*), 2376×ExecTime(*idct*) and ExecTime(*mc*), where ExecTime is the execution time of respective actors. This period is just for demonstration and does not include network and memory access delays. It should be noted that actors *iq* and *idct* have to execute 2376 times in one iteration and the number of executions for each actor is referred to as *repetition vector* of the actor. The rate 2376 is pertaining to the used video frames that have a resolution of 348 by 288 pixels. An SDFG with a throughput of 1000 Hz takes 1 millisecond to complete one iteration.

C. Design Space Exploration of Applications

The DSE process evaluates a number of design points (mappings) for each multimedia application to be supported on a hardware platform. The evaluation considers finding different mappings and their throughput. For each mapping, actors are bound to tiles and edges to memory inside tiles or to connections in the platform. The binding is considered valid if memory imposed, allocated input/output connections and allocated incoming/outgoing bandwidth are less than or equal to the maximum available on each tile. In the DSE, only valid bindings are considered and throughput for the same is computed.

An exhaustive DSE flow (e.g., [20]) evaluates all the possible actors to tiles combinations, i.e. mappings. However, since the number of mappings increases exponentially with the complexity of the application, the complete evaluation might not be feasible within an acceptable time when employing simulative evaluations. For example, a total of 190,899,322 mappings needs to be evaluated exhaustively for an application containing 14 tasks and it will take close to 220 days in evaluation if we assume 100 milliseconds (ms) to evaluate one mapping. Even pruning the design space with existing DSE strategies does not lead to acceptable evaluation time. This necessitates the need to employ analytical estimations along with simulative evaluations in order to provide fast and accurate results.

Further, existing DSE strategies employing simulative evaluations do not provide premier mappings in some cases. This can be realized by examining the execution trace of the H.263 decoder (consisting of 4 actors) mapped on a 4-tile MPSoC platform such that each actor is mapped on a different processor tile, as shown in Fig. 3. For easier realization, the execution trace is shown for one period while considering rates as 5 in places of 2376. Thus, actors *vld, iq, idct* & *mc* fire 1, 5, 5 & 1 times respectively. It can be observed that actors *iq* and *idct* are executing in parallel. However, when existing DSE strategies are applied to evaluate mappings using 3 tiles, in some cases, the best mapping contains actors *iq* and *idct* on the same tile while optimizing for power, resource usage etc. For example, the same happens when strategy in [5] is applied, which optimizes for load balancing on three used tiles. This forces execution of actors *iq* and *idct* sequentially, resulting in reduced throughput. In contrast, the best (maximum throughput) mapping by our strategy contains sequentially executing actors like *vld* and *iq* on the same tile.

IV. Proposed RAPIDITAS Methodology

This section describes our DSE methodology (RAPIDITAS: RAPId Design-space-exploration Incorporating Trace-based Analysis and Simulation). In contrast to conventional existing DSE methodologies, our methodology differs in following aspects: 1) provides mappings using different number of tiles for handling run-time resource availability issues, 2) uses a fixed minimum number of simulations for faster DSE, and 3) uses iterative estimations on execution traces and simulation for the best mapping in order to perform design space pruning and to obtain premier results accurately.

An overview of our DSE flow is presented in Fig. 4. The DSE flow evaluates applications one after another and provides a number of mappings for each application. The flow takes an application (*Application Model*) & a platform (*Platform Model*) as input and performs DSE to evaluate the mappings (*Mappings* & *Throughput*). A platform containing *n* tiles (same as the number of actors in the application) is considered as it can exploit all the parallelism present in the application under evaluation and is capable of covering all potential mappings. Considering any bigger platform wouldn't provide better performance. However, if a small size platform is considered then all the parallelism cannot be exploited as tasks executing in parallel may get mapped on the same tile.

The DSE flow first finds 1_actor-to-1_tile mapping where *n* actors of the application are mapped onto *n* processor tiles such that each tile contains exactly one actor and the edges are mapped onto connections. Then, the mapping is simulated (*Simulate Mapping*) to compute its throughput and to capture execution traces of actors and edges of the application. Thereafter, execution traces are analyzed (*Analyze Execution Trace*) to estimate throughput of the mappings using one lower number of tiles ($n-1$ tiles) and the best mapping is selected (*Select Maximum Throughput Mapping*) for further simulation. The simulation and analysis process is repeated until the number of used tiles in the current simulated mapping is equal to one ($p = 1$?). The best mapping using p tiles is simulated to capture its throughput and execution traces that is used to estimate throughput of mappings using $(p-1)$ tiles. Further, such intermediate simulation guarantees for accurate execution traces, which facilitates for more accurate estimations. The flow stores each simulated mapping into the mapping database *MTDB*. Thus, the database *MTDB* contains maximum throughput mappings using different number of tiles ranging from 1 tile to *n* tiles. Now, we discuss the simulation and analytical estimation strategy that is used for simulation and execution trace analysis of the mapping, respectively.

A. Simulation Strategy

The simulation of a mapping involves its throughput computation and execution trace capturing, which are briefed subsequently.

1) Throughput Computation: The throughput for a mapping is computed by taking the resource allocations of the application on the platform into account. The resource allocations are derived by the binding of actors and edges of the application to the tiles and connections between two tiles or the memory inside a tile in the platform.

In order to compute the throughput, first, static-order schedule for each tile is constructed, which orders the execution of bound actors. A scheduling function using a list-scheduler is used to construct the static-order schedules for all the tiles at once. Thereafter, all the binding and

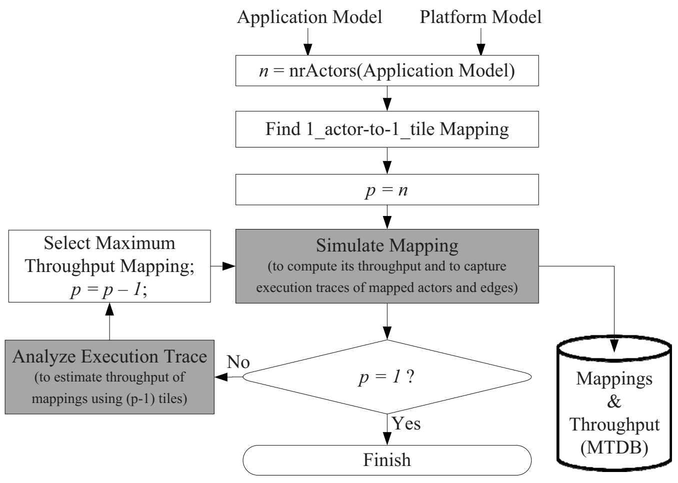
Application Model
Platform Model
n = nrActors(Application Model)
Find 1_actor-to-1_tile Mapping
p = n
Select Maximum Throughput Mapping; p = p – 1;
Simulate Mapping (to compute its throughput and to capture execution traces of mapped actors and edges)
Analyze Execution Trace (to estimate throughput of mappings using (p-1) tiles)
No
p = 1 ?
Yes
Finish
Mappings & Throughput (MTDB)

Figure 4. Proposed RAPIDITAS flow.

scheduling decisions are modeled in a graph called binding-aware SDFG. Lastly, the throughput is computed by self-timed state-space exploration of the binding-aware SDFG [21]. Towards this, states visited during self-timed execution are examined and stored until a recurrent state is found. Then, throughput is computed from the periodic part of the state-space.

2) Execution Trace Capturing: The execution traces of actors and edges of the application are captured based on their execution pattern for a given mapping during one periodic execution. For example, Fig. 3 shows execution pattern of actors and edges of H.263 decoder mapped on a 4-tile MPSoC platform such that each actor and edge is mapped on a different tile and connection between tiles, respectively. The execution traces for each actor and edge is captured as the start and end time of their active executions (firings) in the whole period. For example, in Fig. 3, actor *iq* has five active executions with different start and end times, which will get captured. The captured execution traces are used for analytical estimations as described subsequently.

B. Analytical Estimation Strategy

The analytical estimation strategy is presented in Algorithm 1. The strategy takes captured execution traces of actors/edges for the best mapping α using p tiles as input and estimates throughput of mappings using $(p-1)$ tiles. First, p tiles containing actor(s) are selected. Then, for each unique pair of selected tiles, actors of one tile are moved to another to generate a new mapping that uses $(p-1)$ tiles. For each generated mapping, its throughput (1/period) is estimated and the mapping with its throughput is added to mapping set M. Period of the mapping using $(p-1)$ tiles ($period_\beta$) is estimated by utilizing period of the mapping using p tiles ($period_\alpha$) as follows:

$$period_\beta = period_\alpha + gain_{\alpha,\beta} + loss_{\alpha,\beta} \qquad (1)$$

In Equation 1, $gain_{\alpha,\beta}$ and $loss_{\alpha,\beta}$ are the increase and decrease in the period of the mapping α using p tiles when

Algorithm 1: Analytical Estimation

```
Input: Execution trace for the best mapping $\alpha$ using $p$ tiles.
Output: Mappings & their throughput, using $(p-1)$ tiles.
Initialize the mapping set $M$, i.e., $M = \{ \}$;
Select $p$ tiles containing actor(s);
for each unique pair of selected tiles do
    Move actor(s) from one tile to another to generate a new mapping $\beta$ using $(p-1)$ tiles;
    Estimate throughput of $\beta$;
    Add $\beta$ with its throughput to set $M$;
end
```

the new mapping β is generated by moving actors from one tile to another in α. The period increases when parallel executing actors mapped on selected pair of tiles in mapping α are forced to execute sequentially by mapping the actors on the same tile in mapping β. For example, in Fig. 3, period will increase when parallel executing actors *iq* and *idct* are mapped on the same tile. The period decreases when execution of the edge(s) between the selected pair of tiles is not in parallel with other actors and edges. For example, in Fig. 3, period will decrease due to elimination of edge d_1 trace in the first firing when actors *vld* and *iq* are mapped on the same tile. The $gain_{\alpha,\beta}$ and $loss_{\alpha,\beta}$ are calculated by adhering to the following set of rules:

1) $gain_{\alpha,\beta}$ is calculated by assuming sequential execution of the actors mapped on the selected pair of tiles. The non-parallel executions of the actors (with executions of other actors/edges) contribute to $gain_{\alpha,\beta}$.
2) $loss_{\alpha,\beta}$ is calculated by considering execution traces of edge(s) mapped between the selected pair of tiles. The non-parallel executions of the edge(s) (with executions of other actors/edges) contribute to $loss_{\alpha,\beta}$.

In Algorithm 1, for the selected p tiles containing actor(s), p-choose-2 ($^{p}C_2$) unique pairs are found. Each unique pair provides a mapping that uses $(p-1)$ tiles. Out of all the mappings M using $(p-1)$ tiles, the DSE flow (Fig. 4) selects the maximum throughput mapping to perform its simulation. Similar process is repeated to evaluate mappings using lower number of tiles until the number of used tiles reaches one. Thus, the number of simulations is limited to n (number of actors in the application), where the best (maximum throughput) mapping using 1 tile to n tiles are simulated.

Run-time Mapping

The DSE flow stores the maximum throughput mapping using different number of tiles for all the applications that are expected to be supported (mapped) on a platform at run-time. The stored mappings can be used to perform efficient run-time mapping. A run-time platform manager (RTPM) that keeps the updated resources status handles the mapping

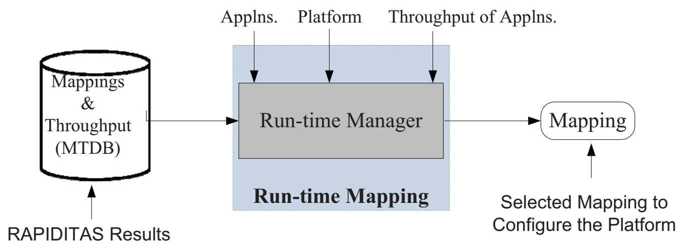

Figure 5. Run-time Mapping.

process by assigning the platform resources to the required applications one after another till all the applications are mapped. For mapping each application, the RTPM takes its desired throughput, platform with updated resources status and the mapping storage MTDB as input and selects a throughput satisfying mapping using minimum number of tiles from the MTDB as shown in Fig. 5. The platform is then configured based on the actors to tiles allocations provided in the selected mapping. If RTPM does not find a throughput satisfying mapping then the application cannot be supported with available platform resources. The run-time mapping process gets accelerated as the time consuming throughput computation is avoided (done at design-time).

In the DSE flow, the considered platform contains tiles that are connected by some fixed latency connections. However, the hardware platform at run-time may contain a larger number of tiles and latencies of connections between the available tiles may be higher than ones considered during DSE. In order to cope with such scenarios, the DSE flow can be repeated by considering a similar platform as earlier but with increased latencies of connections between the tiles till it reaches to maximum latency of the connection in the hardware platform to be considered at run-time. The repeated DSE provides mappings where edges are mapped to connections having varying latencies. This facilitates us to cater for the run-time aspects when the available tiles are connected by varying latency connections. Further, we have considered generic tile architecture and thus any type of interconnection network can be modeled.

V. Performance Evaluation

The proposed DSE strategy that uses analytical estimations and simulations has been implemented as an extension of the publicly available SDF^3 tool set [22]. In order to evaluate the execution time and quality of the strategy, models of real-life multimedia applications H.263 decoder (4 actors), H.263 encoder (5 actors), JPEG decoder (6 actors), sample rate converter (6 actors) and MP3 decoder (14 actors) are considered. Experiments are performed on a Quad Core processor at 2.4 GHz.

The same generic platform model consisting of identical tiles is considered to evaluate different DSE strategies. Each tile contains ARM7TDMI processor. In particular, we present results obtained from our DSE strategy referred to as RAPIDITAS and compare them to that of simulation-based exhaustive exploration (EDSE) strategy adopted in [20] and

Table I
NUMBER OF SIMULATED MAPPINGS BY EDSE, REF. [5], REF. [13] AND RAPIDITAS AT DIFFERENT NUMBER OF ACTORS

Number of Actors	EDSE Flow	Ref. [5] Flow	Ref. [13] Flow	RAPIDITAS Flow
1	1	1	1	1
2	2	6	2	2
3	5	39	5	3
4	15	100	11	4
5	52	180	21	5
6	203	282	36	6
7	877	406	57	7
8	4,140	552	85	8
9	21,147	720	121	9
10	115,975	910	166	10
14	190,899,322	1,834	456	14

strategies pruning the design space in [5] and [13]. We implemented the DSE strategies EDSE, in [5] and [13] with steps similar to our DSE in order to make a fair comparison. The strategies chosen for comparison with RAPIDITAS also perform exploration to evaluate mappings using different number of tiles and providing different throughput values, which can be used to facilitate efficient run-time mapping.

We have applied DSE strategies EDSE, in [5], in [13] and RAPIDITAS to find the number of simulated mappings by them. The EDSE strategy simulates all possible mappings (using different number of tiles) and the number of mappings increases exponentially with the number of actors. For n actors, EDSE considers a platform containing n tiles and total number of mappings is calculated as the number of ways of placing n labeled balls (actors) into n indistinguishable boxes (tiles), which follows bell numbers [23]. The number of mappings by the DSE strategy in [5] is limited by the product of *X*, number of actors and number of tiles, where *X* is the maximum number of partial bindings that is carried over to the next iteration for pruning and simulating the mappings. The strategy in [13] carries forward with the maximum throughput mapping in each iteration in order to prune the design space. The RAPIDITAS performs analytical estimations for most of the mappings and simulates only n mappings. Table I shows the number of mappings simulated by different DSE strategies as the number of actors (*nrActors*) increases. The number of mappings (including partial bindings) by [5] is shown for *X* equal to 10 and it increases with *X*. Thus, increase in *X* may lead to an explosion in the number of mappings. Existing strategies simulate large number of mappings for higher *nrActors* and thus impose large evaluation time that might not be acceptable.

Next, we have applied different DSE strategies to compute the overall exploration time, which consists of simulation and estimation times. During the exploration, a number of mappings are evaluated and we have chosen the best (maximum throughput) mapping to observe the quality of mappings produced by different exploration strategies. For an application containing n actors, all the DSE strategies consider a platform containing n tiles to perform the exploration. Table II shows exploration time (in milliseconds) and best mappings' throughput for multimedia applications H.263 decoder (4 actors), H.263 encoder (5 actors) and sample rate converter (6 actors) when different exploration strategies are employed. Exploration time depends upon the number of mappings to be evaluated by simulative and analytical evaluations. The strategies EDSE, [5] and [13] use simulations. For an application, the number of simulated mappings by different exploration strategies depends upon the number of actors in the application and follows Table I. The approach [5] simulates some duplicate mappings which differ in only placement of actors on different tiles providing the same throughput. Therefore, in some cases, it simulates more number of mappings (including duplicates) than the EDSE and in turn takes more time in the exploration as shown in Table II. The EDSE flow evaluates all the possible mappings without any duplicate ones and the flow in [13] prunes the exploration space to discard evaluation of inefficient mappings. The difference in the number of explored mappings by EDSE and [13] flows increases with the number of actors in the application and thus the difference in the exploration time. The RAPIDITAS simulates only n mappings (evaluates rest by analytical estimations), resulting in reduced exploration time as shown in Table II. On an average, RAPIDITAS reduces exploration time by 92% and 72% when compared to [5] and [13], respectively.

Further, the evaluation by existing exploration strategies is not feasible within a reasonable time for applications with larger number of actors, whereas RAPIDITAS converges fast. For example, the EDSE need to simulate 190,899,322 mappings for MP3 decoder (14 actors) (Table I) which will take more than a year that is unacceptable. The flow of [13] reduces the number of simulations to 456, but it will take longer evaluation time for applications with larger number of actors. In contrast, RAPIDITAS flow performs lesser simulations, resulting in reduced evaluation time.

It has been observed that EDSE, [13], and RAPITIDAS flows provide the same best mapping for H.263 decoder, H.263 encoder and sample rate converter as shown in Table II. Therefore, RAPIDITAS flow does not miss the best throughput mapping despite requiring much lower time for exploration. However, RAPIDITAS flow might provide lower quality (throughput value) mappings than EDSE flow for different applications, i.e., cannot always guarantee for premier solutions due to its heuristic (hill-climbing) behavior. The chances for missing the best quality mappings is

Table II
EXPLORATION TIME AND BEST MAPPING THROUGHPUT FOR MULTIMEDIA APPLICATIONS WHEN EMPLOYING DIFFERENT DSE STRATEGIES

	Exploration Time (milliseconds)				Best mappings' throughput ($\times$ 10^{-12}/time-units)	
Application	EDSE	Ref. [5]	Ref. [13]	RAPIDITAS	Ref. [5]	EDSE & [13] & RAPIDITAS
H.263 decoder	5,138	11,128	3,768	1,379	7396120	9158520
H.263 encoder	11,048	25,481	4,832	1,475	662473	941289
sample rate converter	242,551	204,072	43,014	7,179	410000000	410000000

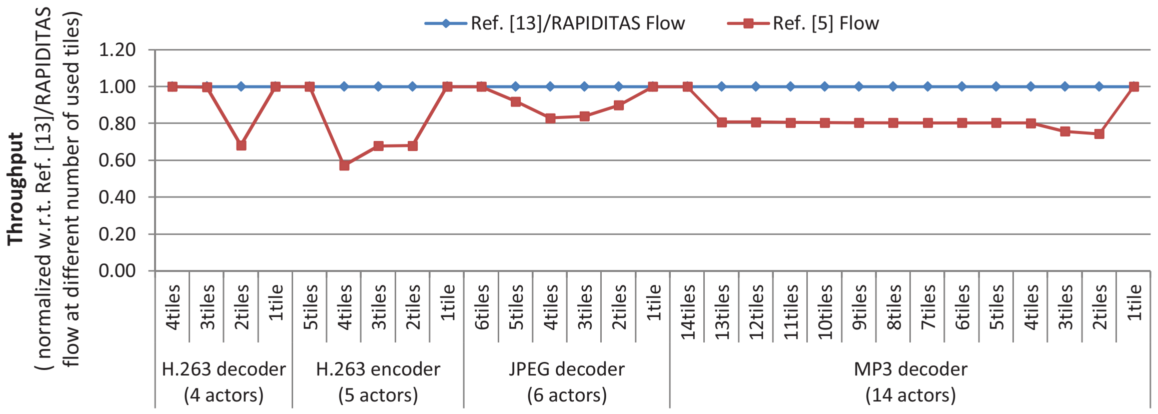

Figure 6. Throughput comparison for the best mappings using different number of tiles.

expected to increase for applications with large number of actors as the pruning of the exploration space gets increased.

Fig. 6 shows throughput of the best mappings using different number of tiles when DSE strategies [13], [5] and RAPIDITAS are employed for different multimedia applications. It has been observed that [13] and RAPIDITAS provide the same best mappings using different number of tiles. The shown throughput values are normalized with respect to (w.r.t.) the throughput obtained by [13] and RAPIDITAS flows. At different number of used tiles, it can be observed that RAPIDITAS provides the same or better quality of mappings as compared to existing strategies. It can also be observed that the quality of the best mappings using one tile and the same as the number of actors in the application is the same by all the flows for each application. This is because the same mapping is obtained by all the flows at these numbers of used tiles.

We also have verified the accuracy of our analytical estimation step. In analytical estimations, the throughput values for a number of mappings are estimated and the best (maximum throughput) mapping is chosen for simulation. The difference between the throughput of the best mapping by simulation and estimation has been referred to as error of the estimation. It has been observed that the quality (throughput) of best mappings using different number of tiles by analytical estimations and simulations is almost the same. One an average, the analytical estimation error is less than 0.033%, which is calculated based on the difference between estimated and simulated throughput values.

The DSE results are stored into MTDB to facilitate for efficient run-time mapping. Our run-time mapping approach utilizes the MTDB (mappings and their throughput when utilizing different number of tiles) and has been compared with existing run-time strategies that start the application mapping without any previous analysis. Such strategies need to perform all the compute intensive analysis at run-time [24] [25]. Fig. 7 shows time required (in milliseconds) to map the different multimedia applications on a 4×4 MPSoC platform when the run-time mapping strategies Nearest Neighbor (NN) proposed in [25] and ours are employed. The NN strategy tries to map the communicating actors on neighboring tiles and then throughput for the mapping is computed at run-time, which is a very time consuming process. These strategies first consume time to find a mapping and then in computing throughput for the mapping. Therefore, they incur large overhead for the run-time mapping. In contrast, our approach just needs to select the best mapping satisfying the throughput-constraint from the MTDB. The selected mapping is used to configure the platform. Therefore, in our approach the mapping time is contributed from selection and placement time only, resulting in faster run-time mapping as shown in the figure.

VI. CONCLUSION

It was observed that most of the existing DSE strategies employ simulations to evaluate the design points. The simulation-based approaches are computationally costly and thus impose large evaluation time that might not be acceptable. Few strategies employ analytical estimations to accelerate the exploration process but estimations are not accurate enough. This paper described a DSE methodology that uses analytical and simulative evaluations iteratively in order to perform fast and accurate exploration. Analytical estimations are used to evaluate the design points while per-

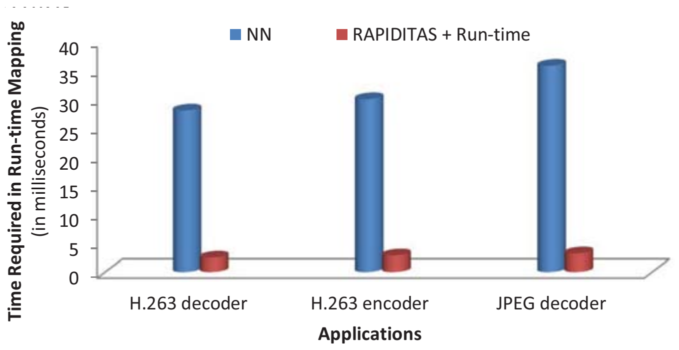

Figure 7. Time required (in milliseconds) to map the applications by different run-time mapping strategies.

forming simulations only on Pareto-optimal points in order to reduce the number of simulations. Experimental results have demonstrated that our methodology provides similar quality solutions as that of simulation-based approaches but at a fraction of the exploration time. In future, we plan to replace required simulations with accurate analytical estimations in order to further accelerate the exploration process.

Acknowledgment

This work is supported by Singapore Ministry of Education Academic Research Fund Tier 1 under grant number R-263-000-655-133.

References

[1] L. Karam, I. AlKamal, A. Gatherer, G. Frantz, D. Anderson, and B. Evans, "Trends in multicore dsp platforms," *Signal Processing Magazine, IEEE*, vol. 26, no. 6, pp. 38 –49, 2009.

[2] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: survey of current and emerging trends," in *Proceedings of the Design Automation Conference*, 2013, pp. 1:1–1:10.

[3] A. D. Pimentel, C. Erbas, and S. Polstra, "A systematic approach to exploring embedded system architectures at multiple abstraction levels," *IEEE Trans. Comput.*, vol. 55, no. 2, pp. 99–112, 2006.

[4] G. Mariani et al., "A correlation-based design space exploration methodology for multi-processor systems-on-chip," in *Proceedings of Design Automation Conference*, 2010, pp. 120–125.

[5] S. Stuijk, M. Geilen, and T. Basten, "A Predictable Multiprocessor Design Flow for Streaming Applications with Dynamic Behaviour," in *Proceedings of the Digital System Design*, 2010, pp. 548–555.

[6] S. Mohanty, V. K. Prasanna, S. Neema, and J. Davis, "Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation," *SIGPLAN Not.*, vol. 37, no. 7, pp. 18–27, 2002.

[7] J. Kim and M. Orshansky, "Towards formal probabilistic power-performance design space exploration," in *Proceedings of ACM Great Lakes symposium on VLSI*, 2006, pp. 229–234.

[8] B. Giovanni, L. Fossati, and D. Sciuto, "Decision-theoretic design space exploration of multiprocessor platforms," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 29, pp. 1083–1095, 2010.

[9] N. H. Zamora, X. Hu, and R. Marculescu, "System-level performance/power analysis for platform-based design of multimedia applications," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 12, pp. 2:1–2:29, 2007.

[10] F. Angiolini et al., "An Integrated Open Framework for Heterogeneous MPSoC Design Space Exploration," in *Proceedings of Design, Automation and Test in Europe*, 2006, pp. 1 –6.

[11] M. Lukasiewycz, M. Glaß, C. Haubelt, and J. Teich, "Efficient symbolic multi-objective design space exploration," in *Proceedings of the Asia and South Pacific Design Automation Conference*, 2008, pp. 691–696.

[12] Z. J. Jia, A. Pimentel, M. Thompson, T. Bautista, and A. Nunez, "NASA: A generic infrastructure for system-level MP-SoC design space exploration," in *Proceedings of Embedded Systems for Real-Time Multimedia*, 2010, pp. 41 –50.

[13] A. K. Singh, A. Kumar, and T. Srikanthan, "A Hybrid Strategy for Mapping Multiple Throughput-constrained Applications on MPSoCs," in *Proceedings of Compilers, Architectures and Synthesis of Embedded Systems*, 2011, pp. 175–184.

[14] R. Piscitelli and A. Pimentel, "Design space pruning through hybrid analysis in system-level design space exploration," in *Design, Automation Test in Europe Conference Exhibition*, 2012, pp. 781 –786.

[15] D. Culler, J. Singh, and A. Gupta, *Parallel computer architecture: a hardware/software approach*, 1999.

[16] C. Grecu, P. Pande, A. Ivanov, and R. Saleh, "Timing analysis of network on chip architectures for MP-SoC platforms," *Microelectronics J.*, vol. 36, no. 9, pp. 833 – 845, 2005.

[17] P. G. Paulin, C. Pilkington, E. Bensoudane, M. Langevin, and D. Lyonnard, "Application of a Multi-Processor SoC Platform to High-Speed Packet Forwarding," in *Proceedings of the conference on Design, automation and test in Europe*, 2004, pp. 58–63.

[18] J. Leijten, J. van Meerbergen, A. Timmer, and J. Jess, "PROPHID: a heterogeneous multi-processor architecture for multimedia," in *Proceedings of the International Conference on Computer Design*, 1997, pp. 164–169.

[19] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Trans. Comput.*, vol. 36, pp. 24–35, 1987.

[20] P. Yang et al., "Managing dynamic concurrent tasks in embedded real-time multimedia systems," in *Proceedings of International Symposium on System Synthesis*, 2002, pp. 112–119.

[21] A. H. Ghamarian et al., "Throughput Analysis of Synchronous Data Flow Graphs," in *Proceedings of Application of Concurrency to System Design*, 2006, pp. 25–36.

[22] S. Stuijk, M. Geilen, and T. Basten, "SDF^3: SDF For Free," in *Proceeding Application of Concurrency to System Design*, 2006, pp. 276–278.

[23] "Encyclopedia of Integer Sequences," http://oeis.org/.

[24] A. K. Singh, W. Jigang, A. Kumar, and T. Srikanthan, "Run-time mapping of multiple communicating tasks on mpsoc platforms," *Procedia Computer Science*, pp. 1019 – 1026, 2010.

[25] E. Carvalho and F. Moraes, "Congestion-aware task mapping in heterogeneous MPSoCs," in *Proceedings of the International Symposium on System-on-Chip*, 2008, pp. 1–4.