A Hybrid Strategy for Mapping Multiple Throughput-constrained Applications on MPSoCs

Amit Kumar Singh
Nanyang Technological University
Singapore 639798
amit0011@ntu.edu.sg

Akash Kumar
National University of Singapore, Singapore 117583;
Eindhoven University of Technology, The Netherlands
akash@nus.edu.sg

Thambipillai Srikanthan
Nanyang Technological University
Singapore 639798
astsrikan@ntu.edu.sg

ABSTRACT

Modern embedded systems are based on Multiprocessor-Systems-on-Chip (MPSoCs) to meet the strict timing deadlines of multiple applications. MPSoC resources must be utilized efficiently by mapping the applications in throughput-aware manner in order to meet throughput constraints for each of them. A design-time methodology is applicable only to predefined set of applications with static behavior, which is incapable of handling dynamism in applications. On the other hand, a run-time approach can cater to the dynamism but cannot provide timing guarantees for all the applications due to large computation requirements at run-time. This paper presents a hybrid flow which performs compute intensive analysis at design-time to derive multiple resource-throughput trade-off points and selects one of these at run-time subject to available resources and desired throughput. Experimental results show that the design-time analysis is faster by 39%, provides better trade-off points and the run-time mapping is speeded up by 93% when compared to state-of-the-art techniques.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems; J.6 [**Computer-Aided Engineering**]: Computer-aided design (CAD)

General Terms

Algorithms, Design, Management, Performance

Keywords

Multiprocessor, Synchronous Dataflow, design-time analysis, run-time mapping, throughput

CASES'11, October 9–14, 2011, Taipei, Taiwan.

1. INTRODUCTION

Modern embedded systems such as smart phones, PDAs, tablet PCs etc. often support multiple applications concurrently. For example, in a smart phone, one might want to listen to music using an MP3 decoder while viewing an image using a JPEG decoder over the internet. In order to meet the increasing computational demands of these applications, systems are increasingly relying on MPSoCs, for example, ST Nomadik, NXP Nexperia [6] and IBM Cell [7]. Consumers expect that all the applications running in the system have a robust behavior and their performance is appeased [3]. This requires that each application to be supported in the system has a predictable timing nature. The timing property of an application depends on its system resource usage and the timing is analyzed at design-time that is incapable of handling dynamism (run-time aspect), i.e., a new application cannot be supported. Synchronous Dataflow Graphs (SDFGs) are used to model time-constrained multimedia applications and provide predictability [21]. Further, analysis techniques to find throughput of an SDFG exist [4].

To support new applications in the system at run-time, the applications' tasks need to be mapped onto the system resources such that the performance requirements are satisfied. Existing mapping strategies are based on either design-time analysis [15] [1] [22] or on run-time mapping [20] [25]. The design-time strategies are unable to handle dynamism in applications incurred at run-time as they are applicable only to predefined set of applications with static behavior. The run-time strategies start the application mapping without any previous analysis and thus cannot guarantee for schedulability, i.e., for the strict timing deadlines due to limited computational resources at run-time. An approach that can perform compute intensive analysis at design-time and can use the analyzed results at run-time needs to be developed to overcome the above mentioned problems.

Contribution. This paper presents a hybrid approach that performs design-time analysis on a generic platform and produces resource-throughput trade-off points. A resource is referred to as a tile that essentially contains a processor along with other resources, e.g., memory. These points are then used by a proposed run-time approach to select the best point depending upon the available tiles and desired throughput. The analysis takes less run-time as compared to the existing analysis approaches and always provides largest throughput mapping. Our run-time strategy provides better timing guarantees as the compute intensive analysis is performed at design-time.

Existing design-time analysis strategies do not always provide the largest throughput mapping, are applicable only to fixed architecture platform and don't scale well with the number of tiles in the platform. These strategies map the potentially parallel executing tasks on the same tile while optimizing for some performance metrics like energy and resource optimization, and thus forcing their execution in sequence. This often reduces the available parallelism in the application thereby reducing throughput. Further, these strategies take a fixed architecture platform as input and the number of evaluated mappings depends upon the number of tiles in the platform. The analysis time is determined by the number of evaluated mappings. Thus, the existing analysis approaches require a lot of time in exploration for advanced available commercial platforms containing large number of tiles [24] [23]. Existing run-time mapping strategies start the mapping without any previous analysis of the application and thus do not perform well.

Our design-time analysis strategy considers a generic multiprocessor platform where the number of tiles is exactly equal to the number of tasks in the application. The considered platform can exploit all the parallelism present in the application. The number of used tiles in a mapping is called as tile count. At each tile count, our technique analyzes a number of mappings and stores the best mapping in terms of throughput. The stored mappings provide a trade-off between throughput and the number of tiles used. Our run-time strategy selects one of the stored mapping depending upon the number of available tiles without performing any computation to evaluate mappings and thus performs faster run-time mapping. The considered generic platform 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 distance from each other, for example, a 2×2 grid of tiles platform has a few tiles separated by a hop_distance of 1 while others at hop_distance of 2. The analysis is performed by considering maximum separation between the tiles in the future expected platform. The analysis results can be used for any platform so long the chosen tiles for mapping are separated by less than or equal to the fixed distance for which the analysis was performed. Design-time analysis is performed for all the applications that might need to be supported onto the platform at run-time. For supporting required applications at run-time, their analyzed results are used to identify a suitable mapping that satisfies the throughput constraint. This concept has been explained in detail in later sections. In the analysis for H.263 decoder modeled with 4 tasks, a platform containing 4 tiles is considered and has a run-time of less than 3 seconds to evaluate 11 mappings for the maximum possible separation between the tiles. However, to get the 11 best mappings on a grid of 2×2, 3×3 and 4×4 tiles platform, it takes a run-time of 12, 19 and 25 seconds respectively, when the existing approach presented in [22] is employed.

Overview. Sec. 2 discusses related work in the field of design-time analysis and run-time mapping. Sec. 3 introduces application model and multiprocessor model used in this work. The hybrid mapping flow that first performs design-time analysis of applications and then map an application on a multiprocessor platform at run-time is presented in Sec. 4. The experimental results are presented in Sec. 5. Sec. 6 concludes the paper and provides direction for future work.

2. RELATED WORK

Most of the design-time analysis strategies provide single mapping for the application and some are presented in [13], [10], [2] and [18]. The analysis is performed in view of some optimization parameters such as computational performance and energy. The optimization approaches are very time consuming. The strategies target fixed MPSoC platforms and do not provide mapping having maximum throughput in some cases. Further, they are unable to handle dynamism in resource availability and throughput requirement at run-time. In contrast, our strategy is applied to a generic MPSoC platform and generates a number of mappings with different resource requirements and throughput, which helps to handle dynamism at run-time.

A few design-time analysis strategies that generate multiple mappings for the application have recently been reported in [22] and [11], which perform analysis in view of optimizing power and resource usage, respectively. The generated mappings can be used to handle dynamism at run-time but these approaches have several drawbacks such as applicable only to fixed platform, don't provide optimal mappings from throughput point of view, evaluate large number of mappings for relatively larger platforms including some duplicate mappings and don't scale with the platform size. The duplicate mappings just differ in placement of tasks on different tiles with the same tasks to tiles binding and provide the same throughput. Additionally, the analysis needs to be performed again with any changes in the platform. In contrast, our strategy considers a generic platform that contains the same number of tiles as the number of tasks in the application and provides the mappings having largest throughput at different tile counts. The mappings generated with our approach are applicable to any platform so long the chosen tiles are separated by less than or equal to the maximum hop distance considered during analysis, which avoids repetition of the analysis for a new platform. The generation of duplicate mappings is avoided by not considering a bigger platform than required.

To map the application tasks on the platform tiles at run-time, one can start the mapping with or without previously analyzed results for the application obtained at design-time. Most of the work presented in literature start mapping without any previous analysis and thus cannot guarantee for schedulability and strict timing deadlines due to limited computational resources at run-time [20], [25], [14], [12]. A few strategies using design-time analysis results are presented in [18] and [11]. The analysis results are applicable to a fixed platform only. In [18], analysis result includes only a single mapping having minimum average power consumption, so, the mapping may not be optimized from throughput point of view. In [11], analysis result includes multiple mappings having trade-off in terms of target power consumption and performance. The result does not include mappings satisfying the constraints in case of limited resources. At run-time, this case forces the application to be put into a relaxed application set and it is not mapped immediately, which may result in missing the strict timing deadline.

Table 1 shows a comparison of the approaches reported in literature those consider design-time analysis and then analyzed results for run-time mapping, and where our approach

Table 1: Comparison of various approaches for performing design-time analysis and run-time mapping

Properties	Ref. [18]	Ref. [22]	Ref. [11]	Our strategy
Platform	Fixed	Fixed	Fixed	Generic
Evaluation	Non-scal.	Non-scal.	Non-scal.	Scal.
Mappings	Single	Multiple	Multiple	Multiple
Applicability	Fix-plat.	Fix-plat.	Fix-plat.	Any-plat.
Run-time	Yes	No	Yes	Yes
Guarantee	No	No	No	Yes

is different. As can be seen, all the existing approaches perform design-time analysis on a fixed platform, are not scalable (Non-scal.) with platform size and evaluate mappings that are applicable only to the fixed platform (Fix-plat.). They have large evaluation time for larger 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 (Any-plat.). Our strategy has a support for run-time mapping that always tries to provide timing guaranty.

3. PRELIMINARIES

This section covers some preliminaries necessary to explain our proposed hybrid mapping flow.

3.1 Application Model

The Synchronous Dataflow Graphs (SDFGs) [8] are used to model multimedia applications with timing constraints. The analysis techniques to calculate throughput and storage requirements for an SDFG have been described in [4]. Throughput is an important constraint for multimedia applications and it refers to how often the tasks of an application need to finish their execution, which is determined by the cycles in the SDFG.

The SDFG model of H.263 decoder is shown in Fig. 1. The nodes model the tasks and are referred to as *actors*, which communicate with *tokens* that are sent from one actor to another through the edges modeling dependencies. The H.263 decoder is modeled with four actors *a1, a2, a3* & *a4* and four edges *d1, d2, d3* & *d4*. 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 in Fig. 1.

In the application model description, resource requirements of the actors and edges are clearly specified. For each actor, the execution time (in time-units) and memory needed (in bits) when allocated to a tile, are specified. For each edge, the size of the token (in bits), the memory needed (in tokens) when connected actors are allocated to the same tile, memory needed (in tokens) in source and destination tiles, and bandwidth needed (in bits/time-unit) when connected actors are allocated to different tiles, are specified. The application model also specifies a throughput-constraint that must be satisfied when the application is mapped onto the platform.

Execution pattern for the SDFG model of H.263 decoder (consisting of 4 actors) mapped on a 4-tile MPSoC platform

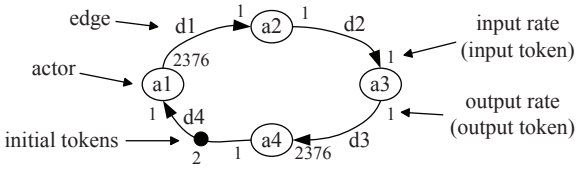

Figure 1: SDFG model of an H.263 decoder.

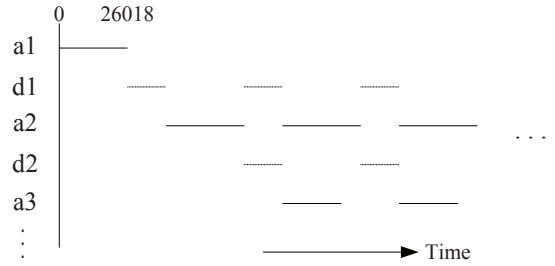

Figure 2: Execution trace of H.263 decoder.

such that each actor is mapped on a different tile, is shown in Fig. 2. It is clearly seen that actors *a2* and *a3* have 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, the best produced mapping contains actors *a2* and *a3* on the same tile while optimizing for some performance metrics such as power and resource optimization. This forces their execution sequentially, resulting in reduced throughput. However, our approach finds the best mapping which has the maximum throughput. The best mapping doesn't contain actors *a2* and *a3* on the same tile, but actors like *a1* and *a2* which execute sequentially, 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.

3.2 Multiprocessor Platform Model

The multiprocessor platform model used in this work uses a tile-based architecture where an interconnection network is used to connect the tiles as shown in an example platform of Fig. 3. The platform has four tiles t_1, t_2, t_3 & t_4. Point-to-point connections (c) with fixed latency between tiles are used to connect the tiles. Each connection may have different latency and thus the latency of connections through a network-on-chip (NoC) can be taken into account [5], i.e., 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, size in bits), a set of communication buffers, called network interface (NI) that are accessed both by the interconnect and the local processor, and maximum number of input/output connections to connect with the NI that provide maximum incoming/outgoing bandwidth (in bits/time-unit). Multiprocessor systems such as StepNP [16], PROPHID [9] and Eclipse [17] fit nicely into this platform model.

In Fig. 3, the communication network is arranged in a 2-D mesh topology. The distance between two tiles is referred to as hop_distance. Tiles t_1 & t_2 are at hop_distance of 1 (just adjacent) and t_1 & t_4 at hop_distance of 2 as com-

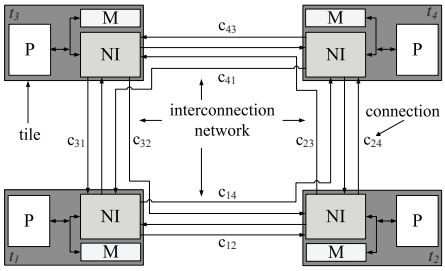

Figure 3: Example multiprocessor platform.

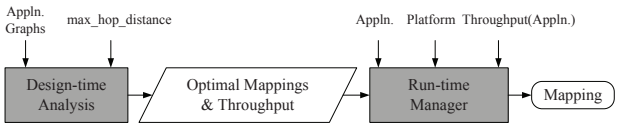

Figure 4: Hybrid mapping flow.

munications are via tile t_2 (1 hop in X-direction to reach tile t_2 and then 1-hop in Y-direction to reach tile t_4). The hop_distance between the tiles determines the latency of the connections connecting the tiles. The application edges are mapped onto the connections between tiles. Each edge occupies one connection between the tiles at its full bandwidth and the occupied connection always serves only to the assigned edge. Other connections can be used for remaining edges when required. Thus, the latency is directly proportional to the hop_distance that determines the length of the connection. 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 facilitates for finding mappings when the tiles are further apart in the actual platform.

4. HYBRID MAPPING FLOW

This section details our hybrid mapping flow. The flow is presented in Fig. 4. It consists of two main steps: first, analysis of applications at design-time (*Design-time Analysis*), and then mapping of an application on a platform by using the analysis results (*Optimal Mappings & Throughput*) with the help of a platform manager (*Run-time Manager*) at run-time.

4.1 Design-time Analysis

The *Design-time Analysis* step evaluates a number of mappings for each application to be supported onto a hardware platform. The applications are evaluated one after another. The evaluation considers finding different mappings and their throughput. For each mapping, actors and edges of the application graph are bound to tiles and connections between two tiles or the memory inside a tile in the platform graph. This binding gives a resource allocation for the application on the platform with the following constraints for each tile:

1. (memory imposed by actors and edges bound on the tile) $\leq$ (memory of the tile),
2. (allocated input/output connections on the tile) $\leq$ (maximum input/ouput connections of the tile),
3. (allocated incoming/outgoing bandwidth on the tile) $\leq$ (maximum incoming/outgoing bandwidth of the tile).

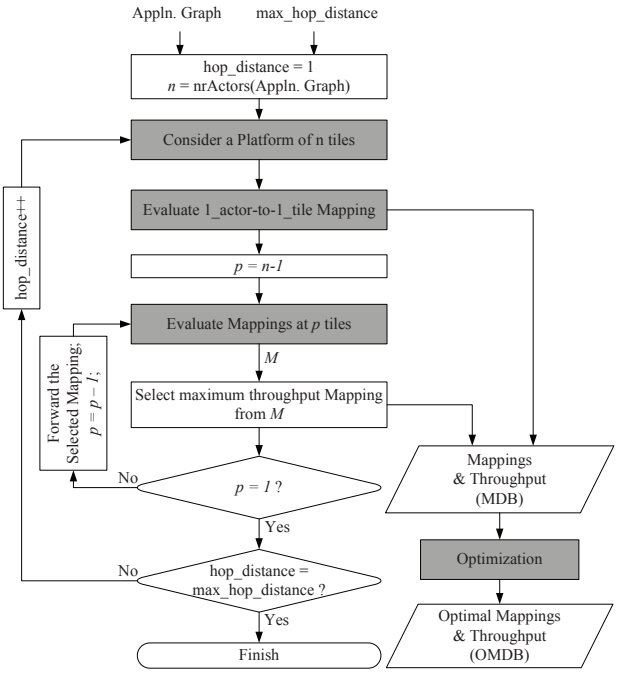

Figure 5: Design-time analysis flow.

Throughput Computation

The throughput for a mapping is computed by taking the resource allocations into account. 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. Then, all the binding and scheduling decisions are modeled in a graph called binding-aware SDFG. Finally, the throughput is computed by self-timed state-space exploration of the binding-aware SDFG [4].

The *Design-time Analysis* flow to evaluate a number of mappings for an application (*Appln. Graph*) has been presented in Fig. 5. The main steps of the analysis flow are highlighted in Fig. 5 and are described subsequently.

4.1.1 Considering a Suitable Platform Graph

This step of the analysis flow considers a platform graph containing n tiles (*Consider a Platform of n tiles*), where n is the number of actors in the application graph. This platform is capable of covering all the potential mappings for the application and considering any bigger platform wouldn't provide better performance as the considered one can exploit all the parallelism present in the application. The potential mappings are those providing maximum throughput.

Initially, the considered platform contains tiles with separation between them as one hop_distance (*hop_distance = 1*), which provides a minimum fixed latency for all the connections between the tiles. The analysis flow is repeated by considering a similar platform that contains tiles with separation of one hop_distance more (*hop_distance++*) between them, i.e., with increased latency for connections, till the *hop_distance* reaches to *max_hop_distance* (one of the input to the analysis flow). The designers can choose an appropriate value of *max_hop_distance* depending upon the expected hardware platform at run-time, where, maximum

ALGORITHM 1: Proc/RH tiles Combination Mappings Evaluation

Input: Best *mapping* α using $(p + 1)$ tiles.
Output: Mappings using p tiles.
Initialize the mapping set M, i.e., $M = \{\ \}$;
Select $p + 1$ 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* β;
  Compute *throughput* of β;
  Add β with its *throughput* to set M;
end

hop_distance between two tiles can be up to *max_hop_distance*. For a higher value of *max_hop_distance*, the design-time analysis evaluates larger number of mappings. This requires more evaluation time but the mappings have more applicability. For example, evaluated mappings with *max_hop_distance* value of 6 are applicable to any platform of size up to 4×4 mesh of tiles.

By considering varying hop_distances, 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 us to cater for the run-time aspects when the available tiles are at different hop_distances. A strategy to find the best mapping in such run-time scenarios is presented in Section 4.2. We have considered generic tile architecture so any type of interconnection network can be modeled.

4.1.2 Evaluating 1_actor-to-1_tile Mapping

Our considered platform contains n tiles. Therefore, n actors of the application are mapped onto n tiles so that each tile contains exactly one actor and the edges are mapped onto connections. Next, throughput for the mapping is computed with the method described earlier. The mapping is stored in a mapping database as *Mappings & Throughput (MDB)* and the same is passed to evaluate mappings at reduced tile count (p = *n-1*).

4.1.3 Evaluating Mappings at Reduced Tiles

This step takes the best mapping using $(p + 1)$ tiles as input and evaluates mappings at reduced tile count, i.e., mappings using p tiles. The evaluation follows steps presented in ALGORITHM 1. First, $(p + 1)$ tiles containing actor(s) are selected. Then, for each pair of selected tiles, all the actors from one tile are moved to another to generate a new mapping. For each generated mapping, its throughput is computed and the mapping with its throughput is added to mapping set M.

For the selected $(p + 1)$ tiles containing actor(s), the algorithm finds $(p + 1)$-choose-2 ($^{(p+1)}C_2$) unique pairs and evaluates the same number of mappings using p tiles, where $0 < p < n$ (n is the number if actors in the application). At $p = n$, a single mapping is evaluated as discussed earlier in Section 4.1.2.

Out of all the evaluated mappings M using p tiles, the flow selects the maximum throughput mapping (*Select maximum throughput Mapping*) to store it into the mapping database *MDB* and to forward it (*Forward the Selected Mapping*) to evaluate mappings at further reduced tile count (p = *p-1*) until the tile count reduces to one. Thus, the flow stores the maximum throughput mapping at each tile count in the database *MDB*.

4.1.4 Optimization

Amongst the stored mappings in the database *MDB*, it might be possible that some of them using less number of tiles as compared to others have the same or more throughput (performance). For example, a mapping using 2 tiles might have the same or more throughput as compared to mappings using 4 tiles and 3 tiles. The mappings using more number of tiles and providing the same or less throughput are referred as non-pareto-optimal mappings. These mappings sometimes have to cater for larger communication overhead without much gain in parallel processing and thus provide less throughput. There is no point in keeping such non-optimal mappings. So, we perform an *Optimization* to discard them in order to store only the pareto-optimal mappings into an optimized mapping database as *Optimal Mappings & Throughput (OMDB)*, which is used at run-time by a run-time mapping strategy. Keeping only the optimal mappings reduces memory requirement to store them and reduces overhead in selecting the best mapping at run-time as the mapping strategy needs to select from a relatively smaller set of mappings. The run-time mapping strategy is discussed later in Section 4.2.

Design-time Analysis: Complexity

The complexity of the design-time analysis strategy (Fig. 5) in terms of number of actors in the application graph 'n' and maximum hop_distance 'h' has been evaluated. The complexity (C) is determined by the number of evaluated mappings in the analysis flow presented in Fig. 5. For a given value of 'n' and 'h', the number of mappings evaluated over the design-time analysis loops is calculated as follows:

$$C = h \times \left[1 + \sum_{p=1}^{n-1} {}^{(p+1)}C_2\right] = h \times \left[1 + \sum_{p=1}^{n-1} \left(\frac{p^2}{2} + \frac{p}{2}\right)\right] = h \times \left[1 + \frac{n^3 - n}{6}\right] \quad (1)$$

In the equation 1, $^{(p+1)}C_2$ is the number of unique pair of tiles at a tile count of $p+1$, i.e., number of mappings using p tiles. For an application with n actors and given value of h, the analysis evaluates a total of [*1 + (*n^3 *- n)/6*] mappings at each hop_distance that varies from 1 to h. Thus, the complexity of our analysis strategy is $O(h.n^3)$.

For an expected platform containing n tiles, the existing strategies evaluate more number of mappings as compared to our strategy and thus have complexity of more orders than this work. The mappings evaluated by other strategies are discussed in Section 5 and compared with our strategy.

4.2 Run-time Mapping

The *Design-time Analysis* step performs all the compute intensive analysis, thereby leaving for minimum computation at run-time. Run-time mapping of throughput-constrained multimedia applications onto a platform is handled by the *Run-time Manager* (Fig. 4). The *Run-time Manager* maps the applications one after another, i.e. after accomplishing mapping for one application it goes on to map the next

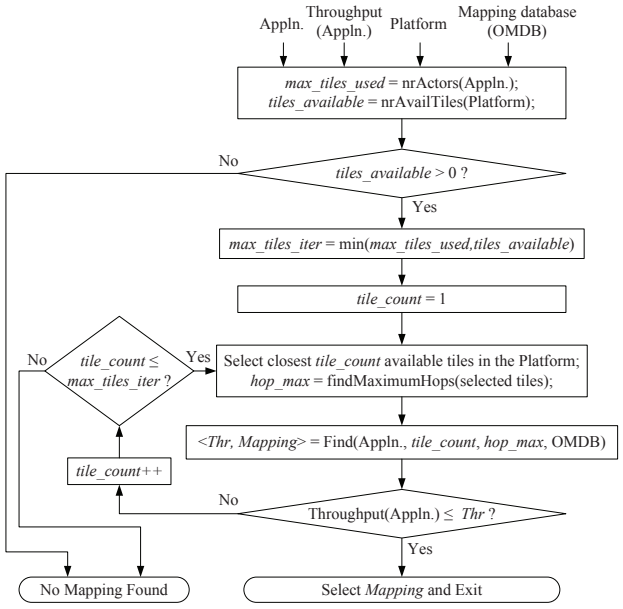

Figure 6: Run-time mapping strategy.

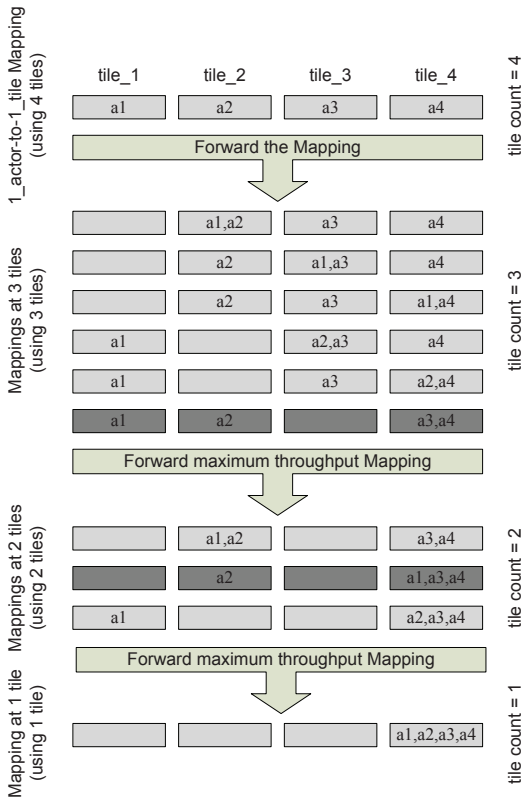

Figure 7: Design-time analysis for H.263 decoder.

application till all the applications are mapped. A run-time strategy is presented in Fig. 6. The strategy takes an application, its desired throughput, platform with updated resources' status and the optimized mapping storage *OMDB* as input and selects the best mapping from *OMDB* depending upon the desired throughput and available tiles. The same strategy is applied to all the applications to be mapped.

First, the run-time manager (RTM) finds the maximum number of tiles that might get used (*max_tiles_used*) by the application, which is the same as the number of actors in the application, and then the number of available tiles (*tiles_available*) in the platform. Next, the RTM checks if the platform has any available tile (*tiles_available* > 0). If no tile is available, then no mapping is found. If tiles are available, then maximum tiles iteration value (*max_tiles_iter*) up to which the mappings need to be searched in the database OMDB is calculated based on the values of *tiles_available* and *max_tiles_used*. The value of *max_tiles_iter* is equal to the minimum of *tiles_available* & *max_tiles_used*. Then, the *mapping* satisfying the throughput constraint of the application (*Throughput(Appln.)* $\leq$ *Thr*) is found from the *OMDB* by iterating from tile count one (*tile_count* $= 1$) to *max_tiles_iter* (*tile_count* $\leq$ *max_tiles_iter*) as shown in Fig. 6. At each tile count (*tile_count*), first, the RTM selects closest *tile_count* available tiles, then finds maximum hop_distance (*hop_max*) between the selected tiles, and finally, a mapping with its throughput (<*Thr, Mapping*>) is found from the *OMDB* that stores mappings for multiple applications. The found throughput *Thr* is checked against the desired throughput (Throughput(Appln.)) and the corresponding mapping is selected if the throughput constraint is satisfied.

The strategy tries to find a mapping satisfying the throughput-constraint at the lowest tile count, resulting in improved resource utilization. The mapping contains the set of actors allocated on different tiles. The actors of the application are mapped onto the hardware platform tiles according to the mapping. If no mapping satisfying the throughput-constraint of the application is found then the application cannot be supported on the platform.

Hybrid Mapping Flow: Example Application

The hybrid mapping flow has been applied onto the example application H.263 decoder (Fig. 1) to show how the flow first performs design-time analysis of the application, and then map it onto a platform at run-time.

Design-time Analysis

The design-time analysis step evaluates multiple mappings at different tile counts. First, a platform containing 4 tiles (same as the number of actors in H.263 decoder) is considered and 1_actor-to-1_tile mapping is evaluated where each tile contains exactly one actor as shown in Fig. 7. The four tiles are named as tile_1, tile_2, tile_3 and tile_4, where the four actors a1, a2, a3 and a4 get mapped, respectively. The edges are mapped on connections between the tiles and we have not shown mapping for edges as we want to focus only on the number of mappings evaluated that depends upon the placement of actors. Here, the tiles are shown as linearly arranged as we just want to illustrate the analysis flow, whereas in the actual flow the separation between the tiles can be any fixed value of hop_distance.

After evaluating 1_actor-to-1_tile mapping that uses four tiles, mappings at a reduced tile count, i.e., mappings using 3 tiles are evaluated by using the ALGORITHM 1. First, the four tiles of 1_actor-to-1_tile mapping are selected to find all the unique pair of tiles and a total of 6 ($^4C_2$) such pairs are found. Then, mappings are generated for each of the pairs

by moving the actors from one tile to another. Thus, we get a total of 6 mappings as shown in Fig. 7. Now, to evaluate mappings at further reduced tile count, the maximum throughput mapping at current tile count is selected and forwarded for further evaluation. We have considered the darken shades mapping (Fig. 7) as the maximum throughput one so the same is forwarded. For mappings using 2 tiles, we select the three tiles containing actor(s) from the forwarded mapping and thus get a total of 3 ($^3C_2$) pair of tiles. The mappings generated by these pair of tiles are shown in the figure. Similarly, mapping using one tile is generated from the maximum throughput mapping (darken shaded) using one higher tile count. Thus, the flow evaluates a total of 11 mappings at each considered hop_distance (1 to max_hop_distance). The number of evaluated mappings at each hop_distance (in each loop for hop_distance 1 to max_hop_distance) is same as the ones calculated from equation 1, i.e., [*1 + (4*^{3} *- 4)/6*]. Only the maximum throughput mapping at each tile count is stored and a further optimization is performed to discard the non-optimal mappings. Here, we assume that the optimal storage OMDB contains the best mapping for each tile count.

Let the future expected platform on which H.263 decoder will be mapped is a 4×4 grid of tiles as shown in Fig. 8. For this platform, the value of max_hop_distance is 6, so the design-time analysis is repeated 6 times by considering platform tiles separated from hop_distance of 1 to 6.

Run-time Mapping

The run-time mapping of the analyzed H.263 decoder on the 4×4 platform is handled by the run-time strategy presented in Fig. 6. The strategy selects the best mapping from the OMDB satisfying the desired throughput depending upon the available tiles in the platform. First, the run-time platform manager (RTM) finds values of the maximum number of tiles that might get used and the number of available tiles as 4 (number of actors) and 5 (number of white tiles in Fig. 8), respectively. Then, maximum tiles iteration value as 4 (minimum of 4 and 5). Next, the mapping using a single tile stored in OMDB is checked if it satisfies the throughput constraint. Let us assume that this mapping does not satisfy the constraint, so, the stored mappings using more number of tiles are checked. In order to find a throughput satisfying mapping using 2 tiles, the RTM first selects two closest available tiles, let us say t_5 & t_6, and then finds maximum hop_distance between them as 1. Next, a mapping using 2 tiles separated by hop_distance of 1 is searched in the OMDB. If the found mapping does not satisfy the desired throughput, then a mapping using 3 tiles is searched where the tiles are separated by a hop_distance of 2 that is the maximum hop_distance between the closest 3 available tiles (t_2, t_5 & t_6) in the platform. Let us say this mapping satisfies the desired throughput. The mapping contains set of actors allocated on 3 tiles as allocated in the maximum throughput mapping at tile count of 3 (shaded), shown in Fig. 7.

The actors a1, a2, a3 & a4 of H.263 decoder are mapped onto the closest 3 available tiles t_2, t_5 & t_6 based on the allocations provided in the found mapping from OMDB as shown in Fig. 8. In the found mapping, all the edges are separated by a hop_distance of 2. So, mapping the actors on the available tiles as shown in Fig. 8 will satisfy the throughput constraint for sure as some edges will be mapped at lower hop_distances (lower latencies). The edges are mapped on the connections between the tiles.

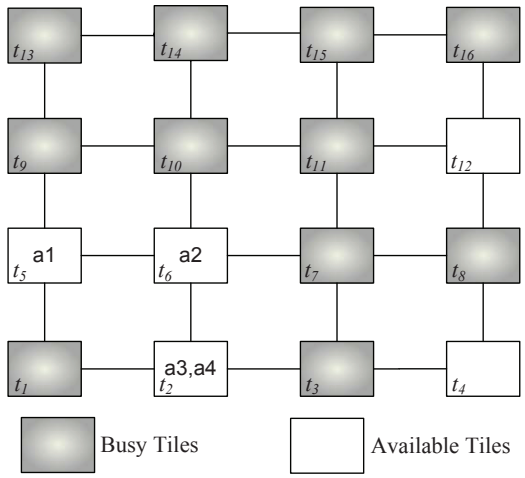

Figure 8: Run-time mapping of H.263 decoder.

5. EXPERIMENTAL RESULTS

The hybrid mapping flow has been implemented as an extension of the publicly available SDF^3 tool set [21]. A benchmark is needed to evaluate the run-time and quality of the flow. As a benchmark, models of multimedia applications H.263 decoder (4 actors), H.263 encoder (5 actors) and MP3 decoder (14 actors) have been considered. The experiments are performed on a Core 2 Duo processor at 3.16 GHz.

The same generic platform graph is considered to evaluate the different flows for an application, where all the tiles in the platform are identical, i.e., it is a homogeneous platform. We have considered tile-based architecture but any other type of architecture can be considered based on the known latencies between the tiles as discussed earlier.

In particular, we have presented the results obtained from the design-time analysis flow referred as heuristic analysis flow and have compared them to that of an exhaustive analysis flow and the flow presented in [22]. We implemented the exhaustive analysis flow with similar steps in order to make a fair comparison. The flow of [22] is applied to scenarios, where each scenario contains a different version of the same application. The different versions model different behavior of the application at different times. We considered a single scenario, i.e., a single version of the application that has always the same behavior, so that mappings obtained with this flow can be fairly compared with the heuristic analysis flow. The results from our run-time strategy are compared to that of a run-time strategy presented in [19].

5.1 Design-time Analysis

Table 2 shows the design-time analysis results for the H.263 encoder (5 actors) at max_hop_distance value of 6. The analysis flow runs 6 times starting from hop_distance of 1 (hop_1) to 6 (hop_6). At each run, the number of mappings evaluated at different tile counts (5tiles to 1tile) is shown as *nrMappings*. At 5 tiles (the same as number of actors n), we get a single mapping and at other lower tile counts p ($0 < p < n$), a total of $^{(p+1)}C_2$ mappings are evaluated as ex-

Table 2: Design-time analysis for H.263 encoder

		The Best mappings throughput ($\times 10^{-12}$/time-units)						
	nrMappings	hop_0	hop_1	hop_2	hop_3	hop_4	hop_5	hop_6
5tiles	1	×	778362	777744	777128	776512	775898	775284
4tiles	10	×	**941289**	**940395**	**939630**	**939262**	**938924**	**937670**
3tiles	6	×	794199	793569	792940	792311	791684	791058
2tiles	3	×	**794199**	**793569**	**792940**	**792311**	**791684**	**791058**
1tile	1	**534068**	×	×	×	×	×	×

plained earlier. Thus we get a total of 21 mappings, which is the same as calculated from equation 1, i.e, [*1 + (5*^{3} - *5)/6*]. For each hop, the best mapping's throughput at different tile counts is shown. The last row (1tile) is empty for hop_1 through hop_6 as hop_distance between mapped actors on the single tile will be zero. This hop_distance has been referred as hop_0 and it is not applicable when actors will mapped on more than one tile, denoted as ×. By storing throughput of the best mappings at different tile counts at varying hops as shown in the Table 2, at run-time, the best mapping can be selected depending upon the number of available tiles and maximum hop_distance between them. Similar analysis results have been obtained for other multimedia applications. We can easily extend the results for higher hops by taking a large value for max_hop_distance that can cater for larger future platforms.

The optimal mappings for the H.263 encoder are highlighted in Table 2. The optimal mappings are those requiring less number of tiles and providing the same or better performance. Similar optimization results have been obtained for other multimedia applications. Storing only the optimal mappings reduces memory requirement for storage and facilitates for faster run-time selection since the run-time strategy needs to select from fewer mappings.

We have applied an exhaustive analysis for the multimedia applications, where, all possible mappings at each tile count are evaluated and the best ones are stored in the similar manner as in the heuristic analysis flow. The exhaustive flow starts as the heuristic flow but while evaluating mappings at reduced tile counts, all the possible mappings are evaluated unlike in the heuristic flow where the best mapping at a higher tile count is taken as input and the number of evaluated mappings depends upon the unique pair of tiles containing actors. The exhaustive analysis has been performed to verify if the heuristic approach misses evaluation of any potential mapping providing better throughput. For the H.263 encoder (5 actors), the exhaustive flow evaluates 1 mapping, 10 mappings, 25 mappings, 15 mappings and 1 mapping at 5tiles, 4tiles, 3tiles, 2tiles and 1 tile, respectively. Thus, a total of 52 mappings are evaluated, whereas the heuristic flow evaluates only 21 mappings. It has been observed that the best stored mappings at different tile counts are the same as obtained by the heuristic analysis flow. Similarly, for H.263 decoder, the exhaustive and heuristic flow evaluates a total of 15 and 11 mappings respectively, and the best mappings at different tile counts are the same. The heuristic flow decreases the chances of missing the best mappings at different tile counts as at each tile count, the flow finds mappings from the best mapping at a higher tile count, which increases the possibility of covering all the best mappings.

Next, we analyzed multimedia applications for given platforms that may contain any arbitrary number of tiles. Table 3 shows the analysis results for multimedia applications H.263 decoder/encoder for platforms containing 1×2, 2×2, 3×3 and 4×4 grid of tiles. For each platform, analysis time (milliseconds) and the best mapping throughput ($\times 10^{-12}$/time-units) has been tabulated when the analysis approach of [22], exhaustive (Exhst.) and heuristic (Heurt.) approach has been employed.

Table 3: Multimedia applications analysis at different platforms for analysis time (ms) and best mapping throughput ($\times$ 10^{-12}/time-units)

		Analysis Time (ms)			Best mapping throughput	
Appln	Platform	Ref. [22]	Exhst.	Heurt.	Ref. [22]	Exhst. & Heurt.
H.263 decoder (4actors)	1×2	7243	4739	2894	7352890	9138520
	2×2	11479	9476	5787	7396120	9158520
	3×3	19010	18956	11572	7396120	9158520
	4×4	24738	28444	17358	7396120	9158520
H.263 encoder (5actors)	1×2	10576	18559	5817	649689	794199
	2×2	21918	37128	11633	662473	941289
	3×3	38659	74154	23265	662473	941289
	4×4	45378	111367	34897	662473	941289

The number of mappings to be evaluated by the approach of Stuijk et. al [22] (Ref. [22]) depends upon the number of tiles present in the platform. For platforms containing more number of tiles, the approach evaluates larger number of mappings, where some mappings are duplicated differing in only placement of actors on different tiles and providing the same throughput. Thus, the analysis time increases with the number of tiles as shown in the Table 3.

The exhaustive flow works in the similar manner as of the heuristic flow. This flow too considers a generic platform containing the same number of tiles as the number of actors in the application and larger platforms are covered by executing the flow repeatedly by considering the higher separation between the tiles, i.e., higher hop_distance between tiles. For 1×2, 2×2, 3×3 and 4×4 platforms, 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 H.263 decoder and encoder, this flow evaluates a total of 15 and 52 mappings respectively, as discussed earlier. At each tile count, this flow evaluates all the possible actors to tiles unique combinations unlike the heuristic flow where a pruning is done by selecting the best mapping.

For H.263 decoder and encoder, in each execution, the heuristic flow evaluates a total of 11 and 21 mappings, respectively. The heuristic flow has been executed for MP3 decoder as well and for each hop_distance, a total of 456 mappings are evaluated in a run-time of 103167 milliseconds with best mapping throughput always better as compared to the flow of [22]. In all the approaches, the analysis time depends upon the number of evaluated mappings. In heuristics approach, the number of evaluated mappings for an application can be calculated from equation 1. It can be observed from the Table 3 that the heuristic (Heurt.) approach does not miss the best throughput mapping despite requiring much lower time for analysis. On an average, for the H.263 decoder, analysis time is improved by 39% and 38% as compared to Stuijk et. al [22] and exhaustive flow, respectively, and for H.263 encoder, it is improved by 35% and 68%. It has been observed that the difference in the number of evaluated mappings from exhaustive and heuristic flow increases with the number of actors in the application and thus the difference in the evaluation time. The best mapping throughput for H.263 decoder and encoder is

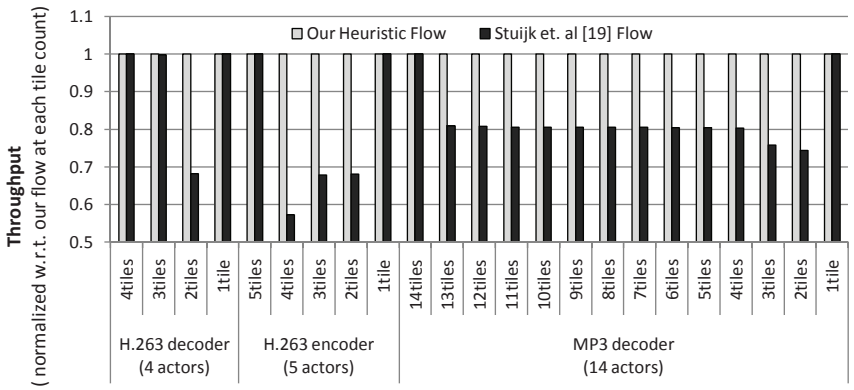

Figure 9: Throughput comparison for the best mapping at different platforms for different applications.

Table 4: Time required (ms) by run-time mapping strategies to map the multimedia applications

	H.263 decoder	H.263 encoder	MP3 decoder
SNN	27.96	29.97	771.83
Our Run-time	2.47	2.84	3.93

improved by 23% and 37% respectively over the approach of [22].

Fig. 9 shows the throughput for the best mappings for multimedia applications where different number of tiles is used for our heuristic design-time flow and the flow presented in [22]. The throughput at each tile count (number of used tiles by the mappings) has been normalized with respect to (w.r.t.) our heuristic flow. It can be observed that the heuristic flow provides better mappings at all the tile counts for each of the application. For all the applications, the same best mapping is obtained by both the flows at platforms containing one (1tile) and the same number of tiles as the number of actors.

5.2 Run-time Mapping

The results obtained from our run-time strategy have been compared with existing run-time strategies that start the application mapping without any previous analysis and perform the required analysis at run-time. Table 4 shows the time required (in milliseconds) to map the throughput-constrained applications on a 4×4 MPSoC platform when the Smart Nearest Neighbor (SNN) proposed in [19] and our run-time mapping strategies are employed. The SNN strategy tries to map the communicating actors on the same or neighboring tiles and then throughput for the mapping is computed at run-time, which is very time consuming. The strategy needs to find a new mapping and then to calculate throughput for the same if throughput-constraint is not satisfied with the current mapping. Such strategies firstly take time to find a mapping and secondly in computing throughput for the same, whereas our strategy just selects the best mapping satisfying the throughput-constraint from the optimized mappings database. The selected mapping is used to place the actors on the tiles and thus total time is contributed from selection and placement time only. On average, our run-time strategy is faster by about 93% as compared to SNN.

Hop_distance Overestimation Penalty

Our design-time flow evaluates mappings by assuming that all the actors are separated by some fixed hop_distance, whereas in real situation, the available tiles at run-time might not be at the same hop_distance. Thus, our flow imposes a penalty for overestimating higher hop_distances. For finding a throughput satisfying mapping from the design-time analyzed mappings, we look for a mapping where all the tiles are separated at a hop of maximum possible hop between the available tiles. So, if the found mapping satisfies the throughput constraint, then mapping the actors on the available tiles will definitely satisfy the constraint as latency of some connections will be smaller as compared to considered during analysis. To map H.263 encoder (5 actors) on 5 tiles, when actors are separated by 2 hops, i.e., all edges are mapped at a hop_distance of 2, then throughput is 777,744 $\times 10^{-12}$ (1/time-units) and when 2 actors are separated by 2 hops and rest are separated by 3 hops, then throughput is 777,328 $\times 10^{-12}$ (1/time-units). The two throughput differ only by 0.0005% and thus very less penalty in overestimating higher hop_distances. So, the stored results from our analysis are acceptable to use them for run-time mapping. Additionally, we never get worse throughput than the stored one as the actors are mapped on the available tiles, thus our approach is suitable for real-time scenarios.

6. CONCLUSIONS AND FUTURE WOK

This paper presents a hybrid approach for efficient mapping of throughput-constrained applications on MPSoC platforms. The hybrid strategy first performs design-time analysis of the applications providing multiple resource-throughput trade-off points, and then uses a run-time mapping strategy to select the best point subject to available resources and desired throughput in order to map an application. The trade-off points are applicable to varying MPSoC platforms. The design-time analysis strategy is scalable, faster and provides better quality of solutions when compared to existing analysis strategies. The analysis results are used by the run-time strategy making it faster over the existing run-time strategies that start the mapping without any previous analysis of the applications and perform time consuming analysis at run-time.

In future, we plan to develop and automate more ways of faster design-time analysis, where we might need to evaluate even lesser number of mappings without missing evaluation

of any optimal mapping. Thus, we can provide more acceleration over the existing run-time strategies. We also plan to extend the hybrid strategy when a tile needs to be shared by different applications. By sharing, we expect to get better resource utilization albeit at the cost of context switching for the shared applications.

7. REFERENCES

[1] G. Ascia, V. Catania, A. G. Di Nuovo, M. Palesi, and D. Patti. Efficient design space exploration for application specific systems-on-a-chip. *J. Syst. Archit.*, 53:733–750, October 2007.

[2] A. Bonfietti, M. Lombardi, M. Milano, and L. Benini. Throughput constraint for synchronous data flow graphs. In *Proc. CPAIOR '09*, pages 26–40. Springer-Verlag, 2009.

[3] O. Gangwal, A. Rădulescu, K. Goossens, S. González Pestana, and E. Rijpkema. Building predictable systems on chip: An analysis of guaranteed communication in the Æthereal network on chip. *Dynamic and Robust Streaming in and between Connected Consumer-Electronic Devices*, pages 1–36, 2005.

[4] A. H. Ghamarian, M. C. W. Geilen, S. Stuijk, T. Basten, B. D. Theelen, M. R. Mousavi, A. J. M. Moonen, and M. J. G. Bekooij. Throughput analysis of synchronous data flow graphs. In *Proc. ACSD'06*, pages 25–36. IEEE, 2006.

[5] C. Grecu, P. Pande, A. Ivanov, and R. Saleh. Timing analysis of network on chip architectures for mp-soc platforms. *Microelectronics J.*, 36(9):833 – 845, 2005.

[6] M. Kim, S. Banerjee, N. Dutt, and N. Venkatasubramanian. Energy-aware cosynthesis of real-time multimedia applications on mpsocs using heterogeneous scheduling policies. *ACM Trans. Embed. Comput. Syst.*, 7:9:1–9:19, January 2008.

[7] M. Kistler, M. Perrone, and F. Petrini. Cell multiprocessor communication network: Built for speed. *IEEE Micro*, 26:10–23, May 2006.

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

[9] J. Leijten, J. van Meerbergen, A. Timmer, and J. Jess. Prophid: a heterogeneous multi-processor architecture for multimedia. In *Proc. ICCD'97*, pages 164–169. IEEE, 1997.

[10] W. Liu, M. Yuan, X. He, Z. Gu, and X. Liu. Efficient sat-based mapping and scheduling of homogeneous synchronous dataflow graphs for throughput optimization. In *Proc. RTSS'08*, pages 492–504. IEEE, 2008.

[11] G. Mariani, P. Avasare, G. Vanmeerbeeck, C. Ykman-Couvreur, G. Palermo, C. Silvano, and V. Zaccaria. An industrial design space exploration framework for supporting run-time resource management on multi-core systems. In *Proc. DATE'10*, pages 196–201, 2010.

[12] O. Moreira, J. J.-D. Mol, and M. Bekooij. Online resource management in a multiprocessor with a network-on-chip. In *Proceedings of the 2007 ACM symposium on Applied computing*, SAC '07, pages 1557–1564, New York, NY, USA, 2007. ACM.

[13] O. Moreira, F. Valente, and M. Bekooij. Scheduling multiple independent hard-real-time jobs on a heterogeneous multiprocessor. In *Proc. EMSOFT'07*, pages 57–66. ACM, 2007.

[14] V. Nollet, P. Avasare, H. Eeckhaut, D. Verkest, and H. Corporaal. Run-time management of a mpsoc containing fpga fabric tiles. *IEEE Trans. Very Large Scale Integr. Syst.*, 16:24–33, January 2008.

[15] G. Palermo, C. Silvano, and V. Zaccaria. Multi-objective design space exploration of embedded systems. *J. Embedded Comput.*, 1:305–316, August 2005.

[16] 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 *Proc. DATE'04*, pages 58–63. IEEE, 2004.

[17] M. J. Rutten, J. T. J. van Eijndhoven, E. G. T. Jaspers, P. van der Wolf, E.-J. D. Pol, O. P. Gangwal, and A. Timmer. A heterogeneous multiprocessor architecture for flexible media processing. *IEEE Des. Test*, 19:39–50, 2002.

[18] A. Schranzhofer, J.-J. Chen, and L. Thiele. Dynamic power-aware mapping of applications onto heterogeneous mpsoc platforms. *Industrial Informatics, IEEE Transactions on*, 6(4):692 –707, November 2010.

[19] A. K. Singh, W. Jigang, A. Kumar, and T. Srikanthan. Run-time mapping of multiple communicating tasks on mpsoc platforms. *Procedia Computer Science*, 1(1):1019 – 1026, 2010.

[20] A. K. Singh, T. Srikanthan, A. Kumar, and W. Jigang. Communication-aware heuristics for run-time task mapping on noc-based mpsoc platforms. *J. Syst. Archit.*, 56:242–255, 2010.

[21] S. Stuijk, M. Geilen, and T. Basten. SDF^3: SDF For Free. In *Proc. ACSD'06*, pages 276–278. IEEE, 2006.

[22] S. Stuijk, M. Geilen, and T. Basten. A predictable multiprocessor design flow for streaming applications with dynamic behaviour. In *Proc. DSD'10*, pages 548–555. IEEE, 2010.

[23] First 100-core processor with the new tile-gx family, 2009. http://www.tilera.com/products/processors/TILE-Gx_Family.

[24] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar. An 80-tile 1.28tflops network-on-chip in 65nm cmos. In *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International*, pages 98–589, Feb. 2007.

[25] C. Ykman-Couvreur, V. Nollet, F. Catthoor, and H. Corporaal. Fast multi-dimension multi-choice knapsack heuristic for mp-soc run-time management. In *Proc. SoC'06*, pages 1 –4, 2006.