Incorporating Energy and Throughput Awareness in Design Space Exploration and Run-time Mapping for Heterogeneous MPSoCs

Nam Khanh Pham[1,2], Amit Kumar Singh[1], Akash Kumar[1], Khin Mi Mi Aung[2]
[1]*Department of Electrical and Computer Engineering, National University of Singapore, Singapore*
[2]*Data Storage Institute, A*STAR, Singapore*
{*phamnamkhanh,amit.singh,akash*}*@nus.edu.sg*
Mi_Mi_AUNG@dsi.a-star.edu.sg

***Abstract*—The advancement in process technology has enabled integration of different types of processing cores into a single chip towards creating heterogeneous Multiprocessor Systems-on-Chip (MPSoCs). While providing high level of computation power to support complex applications, these modern systems also introduce novel challenges for system designers, like managing a huge number of mappings (application tasks to processing cores allocations) that increases exponentially with the number of cores and their types. This paper presents a mapping approach that computes multiple energy-throughput trade-off points (mappings) at design-time and uses one of these points at run-time based on desired throughput and current resource availability while optimizing for the overall energy consumption. While significantly reducing the complexity of the design space exploration (DSE) to compute mappings at design-time, the proposed strategy still evaluates mappings for all the resource combinations of the platform, providing efficient mapping solutions for all the scenarios of system architecture at run-time. Moreover, the proposed approach performs energy-aware mapping at run-time while utilizing the DSE results. Experimental results show that proposed strategy achieves better energy-throughput trade-off points, covers all the resource combinations and reduces energy consumption up to 24.93% at design-time and additionally 17.8% at run-time when compared to state-of-the-art techniques.**

***Keywords*-design space exploration; heterogeneous MPSoC; mapping algorithm;**

I. Introduction

Heterogeneous MPSoCs have recently become a promising production trend of chip vendors due to their high computation potential and energy-savings ability [11]. These systems contain different types of processing elements (PEs) such as General Purpose Processor (GPP), hardware accelerators and programmable hardware for dedicated computation. Examples of such MPSoCs are Texas Instruments OMAP [5], ST Microelectronics Platform 2012 [3], etc. Multiple applications are expected to run on these systems concurrently. Each application shows different performance when utilizing different types of PEs. In order to support applications on a heterogeneous MPSoC, efficient mapping strategies need to be developed that can exploit the distinct advantages of heterogeneous PEs towards providing increased energy savings and ensuring the throughput requirement of all the applications.

There are mainly two kinds of mapping approaches: design-time and run-time. The design-time strategies [1, 10, 12, 17] consider static workloads (predefined applications) and thus cannot handle dynamic workload scenarios such as insertion of a new application into the system at run-time. On the other hand, run-time mapping [4, 18, 19, 30] may not provide a mapping solution that can guarantee the throughput requirement of applications due to limited time and available computation power at run-time. To address shortcomings of design-time and run-time approaches, hybrid mapping strategies that use design-time analysis results to support run-time decisions have been reported [21, 27, 28]. The increased heterogeneity in the architecture introduces new challenges for these mapping strategies. For example, the number of mappings that need to be evaluated increases exponentially with the number of PE types, i.e, the design space becomes multi-dimensional, whereas it is linear for the homogeneous case [23]. To overcome these issues, some heuristic DSE approaches have been proposed to prune the design space and thus reduce evaluation effort [23]. While pruning the design space, the existing approaches discard evaluation of mappings for a significant number of resource combinations. Consequently, the run-time mapping process needs to find a mapping solution dynamically in case of missing resource combinations during DSE. For such situations, the run-time mapping process may take a long time to find a mapping, which may violate the strict timing deadlines imposed on the mapping time. Moreover, existing strategies have focused mainly on improvement of design time analysis (DSE), whereas little attention has been paid on innovation of run-time techniques; although such an improvement can significantly reduce the energy consumption of the system. This paper proposes a mapping strategy that addresses shortcomings of existing strategies by providing following contributions:

- A design-time DSE technique that provide energy-throughput trade-off points for all the possible heterogeneous resource combinations.
- A run-time mapping technique that chooses best trade-off points from the design-time analysis results and considers different mapping options on the fly to optimize the energy consumption.

Most existing works usually consider only one performance metric like energy or throughput when performing optimization in DSE process [21, 23]. Hence, the best mapping for each resource combination (generated by DSE) may excel for one performance metric and show very bad

result for the other. In our strategy, both throughput and energy have been used in optimization process to achieve a balanced mapping solution for the system.

Recently, a large body of research works in mapping strategies has focused on energy optimization. However, most of them perform energy optimization either during design-time DSE [20, 23] or at run-time [8, 24]. In contrast, our approach perform energy optimization both at design-time and run-time to achieving maximum energy savings.

Overview. Section II reviews history and trend of mapping strategies for MPSoCs. Section III introduces preliminaries of multiprocessor and application model used in this work Our methodology is highlighted in Section IV and experimental results are presented in Section V to show the efficiency of our method. Finally, Section VI concludes the paper and provides directions for future work.

II. Related Work

Earliest DSE strategies that generate multiple mapping have been reported in [7, 14, 15, 26]. By generating various mapping solutions at design-time, they can provide supporting information to handle the dynamism in application throughput requirement and resource availability at run-time. However, they suffer from several shortcomings. They target only fixed architecture platform, do not scale well with the number of tiles, and perform duplicate (similar tasks to tiles allocations at different locations in the platform (mapping)) evaluations for large-size platforms. The duplications increase the number of evaluated mappings significantly and thus the overall evaluation time. In order to overcome the aforementioned limitations, our DSE strategy considers a *generic* heterogeneous platform to provide mapping solutions that are applicable to a variety of target platforms, which is not possible while considering a fixed platform. The *generic platform* contains *tiles* depending upon the number of tasks in the application. A *tile* includes a processing unit and other elements like memory or network interface (NI). Processing Unit may have different hardware realization such as general purpose processor (GPP), Graphics Processing Unit (GPU), digital signal processor (DSP), reconfigurable hardware (RH) , etc. and it determines the *tile type*. The results of our DSE analysis for a platform can be reused for multiple target platforms as long as the tile types and the maximum distance between tiles of the target platform are subset of those considered during DSE analysis. Therefore, the analysis results are applicable to variety of target platforms and repeated evaluations can be avoided. Furthermore, duplications during the analysis are avoided by not considering a bigger platform than required.

For run-time mapping, a large body of literature exists [4, 18, 19, 30]. These early studies generate the mapping solution on-line at the arrival of applications without any prior analysis. Therefore, the result is usually not optimal due to the limited computation resources at run-time. Recently, mapping strategies have changed their focus to hybrid approaches, which use the prior evaluation (done at design-time) to support the mapping decision at run-time [16, 21, 28, 29]. Most of these works perform the optimization for only one performance metric like energy consumption, throughput or resource usage. The method in [21] provides the optimal mapping in term of average power consumption only; therefore, it cannot guarantee the throughput constraint of applications. In [16] and [29], the DSE strategies take into account multiple quality parameters at design-time, but leave the resource constraint problem for a controller at run-time. On the other hand, our strategy produces mappings for all the possible resource combinations, where the mappings at each resource combination represent trade-off between energy and throughput. Therefore, it provides better mapping solutions for several performance metrics. The strategies in [16, 21, 28, 29] target a fixed platform, whereas our method is applicable to generic platform. In [22], a general approach is considered but it is applicable only to homogeneous platforms. In [23], the authors target a generic heterogeneous platform, but the DSE is conducted in terms of throughput optimization only. In contrast, our design-time analysis takes both throughput and energy consumption into account. Further, DSE in [23] reduces the number of mappings significantly while focusing on the high-quality (throughput) mapping solutions. However, evaluation of mappings at a number of resource combinations is discarded during the DSE process. Therefore, the analysis results might not contain mapping solutions for all the different resource combinations available at run-time. Our proposed strategy addresses this problem and reduces the energy consumption by mapping the highly communicating tasks onto the available closest tiles.

III. Preliminaries

A. Application Model

The applications are modeled as Synchronous Dataflow Graphs (SDFGs) [13]. SDFGs facilitate for easier modeling of streaming multimedia applications with timing constraints. A SDFG model of H.263 decoder is shown in Fig. 1. The nodes (*VLD, IQ, IDCT*, & *MC*) and edges (e_1, e_2, e_3, & e_4) model tasks and dependencies, respectively. The nodes have been referred to as *actors* that communicate with *tokens* sent from one actor to another through the edges. Each actor is associated with its attributes: execution time and memory requirement when mapped on a tile. If the actor has many implementation alternatives (e.g., GPP, DSP, RH) then it's attributes are listed for each implementation alternative. Implementation alternatives of actor refer to different types of processing tiles on which the actor can be implemented. Each edge has following attributes: size of a token, memory needed on the tile when connected actors are allocated to the same tile, memory needed in source and destination

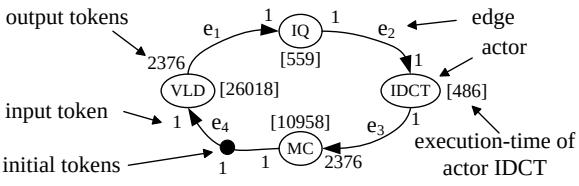

Figure 1. SDFG model of an H.263 Decoder

tiles when connected actors are allocated to different tiles and respective bandwidth requirements between the tiles. 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 connections. In each firing, the actor consumes a fixed amount of tokens from the input edges (*input tokens*) and produces a fixed amount of tokens on the output edges (*output tokens*). These token amounts are referred to as *rates*. An edge may contain *initial tokens*.

Throughput of an application is determined as the inverse of the long term period, which is calculated as 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 acquired. 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. It should be noted that actors *IQ* and *IDCT* have to execute 2376 times in one iteration and the number of executions is referred to as *repetition vector* of the actor. The calculated period does not include network and memory access delays. An SDFG with a throughput of 1000 Hz takes 1 millisecond (ms) to complete one iteration, i.e., its period is 1 ms.

B. Multiprocessor Platform Model

The multiprocessor platform used in this work is a tile-based architecture as shown in an example platform of Fig. 2. The platform contains three tiles, which are connected by an interconnection network in order to facilitate communication amongst the tiles. Each tile contains a processor (e.g., general purpose processor (GPP), digital signal processor (DSP) or reconfigurable hardware (RH) as shown in Fig. 2), a local memory (M) and a network interface (NI) containing set of communication buffers that are accessed both by the interconnect and the local processor. The interconnection network provides end-to-end connections between the tiles. However, the latencies of connections can be modeled for different network-on-chips (NoCs).

IV. Proposed Mapping Strategy

This section describes our mapping strategy. In contrast to conventional existing mapping strategies, our strategy differs in following aspects: *1)* performs both energy and throughput aware design-time DSE, *2)* the DSE results contain mapping solutions for all the possible resource combinations to cater for different run-time resource availability aspects, and *3)* performs throughput and energy optimization during the run-time process as well.

An overview of our mapping flow is presented in Fig. 3. The overall flow has two main steps: *1)* DSE phase at design-time (*Design-time DSE*) to analyze the applications, and *2)* run-time mapping of required applications by utilizing the DSE results (*Optimal Mapping Database*) with the help of a platform manager (*Run-time Platform Controller (RTPC)*). In the DSE phase, multiple mapping solutions are generated for each application to be supported onto a hardware platform. The run-time phase takes the required

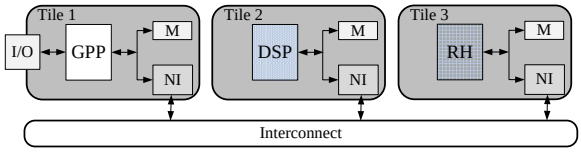

Figure 2. An example multiprocessor platform

applications, their throughput requirements, DSE results and the current platform status (*available resources*) as input and provides an *energy optimized mapping*.

A. Design-time DSE

The design-time DSE step takes the applications one after another and evaluates a number of mapping solutions for each of them. The evaluation finds different mappings along with their throughput & energy consumption. For each mapping, the platform resources are allocated to the application: actors are bound to tiles while edges are bound to connections between tiles or local memory of tiles. Based on the resource allocations, the throughput and energy consumption of the mapping are then computed.

Throughput Computation: For the mapping, first, static-order schedule that orders the execution of bound actors on each tile is constructed. Then, all the binding and scheduling decisions are modeled in a graph called binding-aware SDFG. Thereafter, throughput is computed by self-timed state-space exploration of the binding-aware SDFG [6].

Energy Consumption Computation: The total energy consumption for a mapping is computed as the sum of communication and computation energy for one iteration of the application. *Communication energy* is required to transfer data (tokens) from source tile to destination tile and *computation energy* is required to process the transferred token on the destination tile. The *communication energy for each edge (e)* mapped to a connection (c) is estimated as product of the number of tokens (in bits) to be transferred through c, delay (D) and power consumption (P_{bit}) for transferring one bit through c. Total communication energy for all the edges is estimated from (2). The number of tokens for an edge is computed as the product of repetition vector (*repV*) of source (or destination) actor and source (or destination) port rate (1). The power required to transfer one bit is denoted as P_{bit} [9]. *Computation energy for each actor (a)* mapped to tile (t) is estimated as product of the number of executions of a ($repV[a]$), execution time ($ET[a]$) and power consumption (pow) on t. ET and pow could be different for different types of tiles. Total computation energy for all actors is estimated from (3). Power consumption on a tile is estimated as $C \times v^2 \times f$, where C, v and f denote average load capacitance, supply voltage and operating frequency, respectively. In our approach, we focus on mapping of applications on the architecture after it is designed. Therefore, we cannot optimize static energy consumption and restrict our focus on optimizing only dynamic energy consumption ($E_{comm} + E_{comp}$).

$$nrTokens[e] = repV[e \rightarrow srcActor] \times (e \rightarrow srcPortRate) \quad (1)$$

$$E_{comm} = \sum[\{nrTokens[e] \times tokenSize[e]\} \times D \times P_{bit}] \quad (2)$$

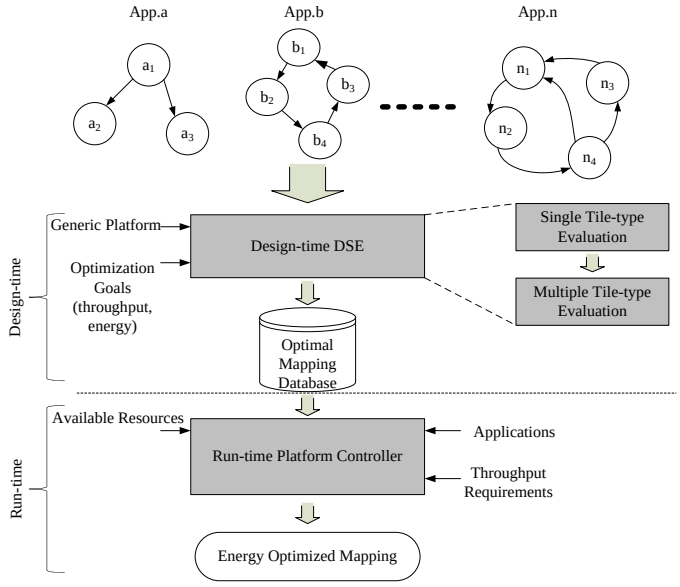

Figure 3. Overall flow of proposed mapping strategy

$$E_{comp} = \sum[repV[a] \times (ET[a] \rightarrow t) \times (pow \rightarrow t)] \quad (3)$$

The proposed DSE flow takes an application & a generic platform model as input and performs exploration to evaluate mappings while optimizing for both throughput and energy consumption (Fig. 3). A heterogeneous platform that contains tiles depending on the number of actors (n) and their implementation alternatives provided in the application is considered. To cover all potential mappings for different possible resource combinations, a platform with n tiles of each implementation alternative is considered. Since the chosen platform can exploit all the parallelism present in the application, considering a bigger platform would not be necessary. On the other hand, a smaller platform might not exploit all the parallelism as concurrent executing tasks may get mapped on the same tile.

The considered platform contains tiles that are separated by a fixed distance from each other, referred to as hop_distance in this work. Initially, the hop_distance is considered as one. However, at run-time, a real-life platform might have available tiles at varying hop_distances, for example, a 2×2 grid (mesh) of tiles platform may have few available tiles separated by a hop_distance of 1 while others at a hop_distance of 2. To cope with the distance variation between available tiles at run-time, our DSE flow is repeated for all possible hop_distances in the expected target hardware platform at run-time. For example, two available tiles of a 4×4 mesh platform may have the hop distance varying from 1 to 6, so our DSE is repeated 6 times (while considering hop_distance 1 to 6) to account for varying resource availability scenarios that might incur at run-time. By performing the DSE with higher hop_distances, the applicability of the DSE results increases even for bigger platforms, but the evaluation time increases. For example, DSE results (evaluated mappings) with hop_distance value of 8 are applicable to any platform where maximum separation between the tiles is less than or equal to 8 hops such as mesh of 2×2, 3×3, 4×4, and 5×5 tiles platforms. The main steps of the DSE flow (projected in Fig. 3) are described subsequently.

1) Single tile-type evaluation: The mappings using the single tile type (homogeneous tiles) are generated by using the DSE strategy proposed in [22] as it discards evaluation of inefficient mappings (providing less throughput) and performs faster evaluation without missing the efficient mappings. First, 1_actor-to-1_tile mapping is evaluated, where *n* actors of the application are mapped onto *n* homogeneous tiles so that each tile contains exactly one actor and the edges are mapped onto connections. Then, mappings using reduced number of tiles ($p = n - 1$) are evaluated by taking the best mapping using $(p + 1)$ tiles as input. For each pair of $(p + 1)$ tiles, all the actors from one tile are moved to another to generate a new mapping. A total of $(p + 1)$-choose-2, i.e., ${}^{(p+1)}\mathrm{C}_2$ unique pairs are found for $p + 1$ tiles and the same number of mapping using p tiles are evaluated. Out of all the evaluated mappings using p tiles, the best mapping is chosen to evaluate mappings at further reduced tile count, i.e. mappings using $p - 1$ tiles by following the similar steps. The same process is repeated until the mapping using one tile gets evaluated. Thus, all the mappings using different number of tiles are evaluated. The strategy in [22] chooses the maximum throughput mapping as the best one as their optimization goal is only throughput. In contrast, we choose the best mapping as the one having maximum throughput/energy in order to perform throughput and energy aware exploration. Similar exploration process is applied by considering different types of tiles one after another to get the homogeneous tiles mappings for each tile type.

2) Multiple tile-type evaluation: Our strategy finds the most efficient mappings for all heterogeneous resource combinations by using the homogeneous tiles mappings calculated in the earlier step. In a general platform architecture (A) with *m* tile-types, all the resource combinations are represented by *m-dimensional* array $A(t_1, t_2, \cdots, t_m)$, where t_i is the number of used tiles of tile-type i^{th}. If we call *p(k,n)* as the number of ways to partition *n* balls into *k* slots; then the number of resource combination in our generic platform with *m* tile-types that can cover n-actors applications is $\sum_{k=1}^{n} P(k, n)$. For example, Table I presents all the possible resource combinations when a 5-actors application and 3 tile-types (GPP, DSP,RH) are considered.

Table I
EXAMPLE OF 5 ACTORS AND 3 TILE-TYPES

GPP	DSP	RH	GPP+DSP	DSP+RH	GPP+RH	GPP+DSP+RH
A(5,0,0)	A(0,5,0)	A(0,0,5)	A(4,1,0)	A(0,4,1)	A(1,0,4)	A(3,1,1)
			A(3,2,0)	A(0,3,2)	A(2,0,3)	A(2,2,1)
			A(2,3,0)	A(0,2,3)	A(3,0,2)	A(2,1,2)
			A(1,4,0)	A(0,1,4)	A(4,0,1)	A(1,1,3)
						A(1,2,2)
						A(1,3,1)
A(4,0,0)	A(0,4,0)	A(0,0,4)	A(3,1,0)	A(0,3,1)	A(1,0,3)	A(2,1,1)
			A(2,2,0)	A(0,2,2)	A(2,0,2)	A(1,1,2)
			A(1,3,0)	A(0,1,3)	A(3,0,1)	A(1,2,1)
A(3,0,0)	A(0,3,0)	A(0,0,3)	A(2,1,0)	A(0,2,1)	A(1,0,2)	A(1,1,1)
			A(1,2,0)	A(0,1,2)	A(2,0,1)	
A(2,0,0)	A(0,2,0)	A(0,0,2)	A(1,1,0)	A(0,1,1)	A(1,0,1)	
A(1,0,0)	A(0,1,0)	A(0,0,1)				

Algorithm 1 Procedure $Generate[A(..., t_i, ..., t_j, ...) - > A(..., t_i - 1, ..., t_j + 1, ...)]$

Input: Best mapping for $A(..., t_i, ..., t_j, ...)$
Output: Best mapping for $A(..., t_i - 1, ..., t_j + 1, ...)$
$bestMapping = 0$, $max = 0$;
Find free tile $p \in j^{th}$ tile-type
for $u = 1$ to t_i **do**
  Move all actors from tile u to tile p to generate new mapping b
  Compute throughput and energy for b
  Compute metric $\mu = \frac{throughput}{energy}$
  if $\mu > max$ **then**
    $max = \mu$
    $bestMapping = b$
  end if
end for
Store $bestMapping$ as optimal solution for $A(..., t_i - 1, ..., t_j + 1, ...)$

Algorithm 2 Algorithm for multiple tile-type combination

Input: Best GPP tile mapping from database
Output: Most efficient mapping for multiple tile-type
for $m' = 2$ to m **do**
  for $tile\_count = n$ downto m' **do**
    for all combination of m' used tile-type from m tile-types **do**
      if $m' = 2$ **then**
        for $t_{i_1} = tile\_count$ downto 2 **do**
          $t_{i_{m'}} = tile\_count - t_{i_1}$
          $Generate[A(..., t_{i_1}, ..., t_{i_{m'}}, ...) \longrightarrow A(..., t_{i_1} - 1, ..., t_{i_{m'}} + 1, ...)]$
        end for
      else
        for $q = tile\_count - m' + 2$ downto 2 **do**
          for all partition ways of $(tile\_count - q)$ tiles into $(m - 2)$ tile-types **do**
            for $t_{i_1} = q$ downto 2 **do**
              $t_{i_{m'}} = q - t_{i_1}$
              $Generate[A(..., t_{i_1}, ..., t_{i_{m'}}, ...) \longrightarrow A(..., t_{i_1} - 1, ..., t_{i_{m'}} + 1, ...)]$
            end for
          end for
        end for
      end if
    end for
  end for
end for

To analyze the heterogeneous tiles mappings, we introduce a heuristic approach to find the most efficient mapping for all resource combinations while evaluating a manageable number of mappings. The essence of our heuristic is the generation step denoted by procedure $Generate[A(..., t_i, ..., t_j, ...) - > A(..., t_i - 1, ..., t_j + 1, ...)]$, which is presented in Algorithm 1. This procedure takes the best mapping of the previous resource combination $A(..., t_i, ..., t_j, ...)$ as input and construct the best mapping for the later resource combination $A(..., t_i - 1, ..., t_j + 1, ...)]$. In each execution of generation procedure, there will be a tile-type with incremented number of used tiles (*destination tile-type* j^{th}) while another tile-type have its used tile-number decremented (*source tile-type* i^{th}).

Given the best mapping for a resource combination, the algorithm will find the first empty tile *p* of destination tile-type. Thereafter, mappings for new resource combination are generated by moving all actors from each tile of source tile-type to the destination tile *p*. The throughput, energy and metric μ for each mapping is computed, stored into our mapping database and compared with the current best mapping solution (*bestMapping*). If the current mapping has better result than *bestMapping*, it will become the *bestMapping* and is used to compare with subsequent mapping options. At the end of the generation procedure, the most efficient mapping for new resource combination will be *bestMapping* and is stored in the *Optimal Mapping Database*. By selecting the mapping having maximum μ ($\frac{throughput}{energy}$) at different stages, our heuristic can avoid evaluating a large number of inefficient mappings. Hence, the evaluation time is reduced significantly.

Our strategy to evaluate mappings using different type of tiles is presented in Algorithm 2. First, the heuristic iterate through all resource combinations with different number of tile-types m' and total number of used tiles, referred to as *tile_count*. *tile_count* varies from number of actors in application n down to number of used tile-type m'. Then we consider all the resource combinations that use m' tile-types from given *m* tile-types. The amount of such combinations will be ${}^{m}C_{m'}$. Thereafter, the algorithm will conduct the generation procedure for tile-type i_1 and tile-type $i_{m'}$. We define *q* as the total number of tiles used in i_1 tile-type and $i_{m'}$ tile-type; hence $tile\_count - q$ is the number of tiles available for the rest $(m' - 2)$ tile-types. To cover all the resource combinations, the main generation procedure (explained previously) $Generate[A(..., t_{i_1}, ..., t_{i_{m'}}, ...) - > A(..., t_{i_1} - 1, ..., t_{i_{m'}} + 1, ...)]$ should be repeated for all partitions of $(tile\_count - q)$ tiles into $(m' - 2)$ tile-types that do not participate into the generation step. In case of $m' = 2$, partition $P(m' - 2, tile\_count - q)$ is not available, so the generation step is done outside the loop (if $m' = 2$). Algorithm 2 ensures that all the input mappings for generation steps are available in the optimal mapping database before used.

3) DSE complexity: The complexity of our algorithm depends on the number of actors *n*, the number of tile-types *m*, and maximum hop distance *h* considered for the DSE. Table II introduces the notations to be used for complexity calculation. The complexity has been calculated in terms of the number of evaluated mappings during the DSE. The number of homogeneous tiles mappings is calculated by (4). In heterogeneous case, the number of mappings is computed based on the observation that in each generation step, the number of evaluated mappings is the same as the number of used tiles of source tile-type. The number of heterogeneous mappings for 2 tile-types combination is calculated by (5). Generally, number of mappings is calculated by (6), where $P(m' - 2, tile\_count - q)$ is the number of ways to partition $(tile\_count - q)$ tiles into $(m' - 2)$ tile-types if $(m' > 3)$; otherwise, $P(m' - 2, tile\_count - q) = 1$. The total number of mapping is calculated as the sum of all homogeneous and heterogeneous tiles mappings by (7).

Table II
NOTATIONS TO BE USED

Notation	Meaning
m	total number of tile-types in platform
n	total number of tile-types in platform
m'	number of used tile-types
tc	number of used tiles in platform
t_{i_1}	number of used tiles of $i_1 - th$ tile-type
q	number of used tiles of i_1-th tile-type and i_m-th tile-type

Table III
COMPLEXITY

m	$P(m-2,n)$ **Ref. [2]**	$\Theta(P(m-2,n))$	$M(m,n)$	$\Theta(M(m,n))$
1	na	0	$M(1,n) = \frac{n^3-n+6}{6}$	n^3
2	na	0	$M(2,n) = \frac{n^3+n^2-2n+4}{2}$	n^3
3	1	c	$M(3,n) = \frac{n^4+26n^3+23n^2-50n+72}{24}$	n^4
4	$\lfloor \frac{n}{2} \rfloor + 1$	n	$M(4,n)$	n^5
5	$\{\frac{(n+3)^2}{12}\}$	n^2	$M(5,n)$	n^6
6	$\{(n+5)(n^2+n+22+18\lfloor \frac{n}{2} \rfloor)/144\}$	n^3	$M(6,n)$	n^7
7	$\{(n+8)(n^3+22n^2+44n+248+180\lfloor \frac{n}{2} \rfloor)/2880\}$	n^4	$M(7,n)$	n^8

$$C(1,m,n) = m * [1 + \sum_{p=1}^{n-1} ({}^{p+1}C_2)] = m * [1 + \frac{n^3-n}{6}] \quad (4)$$

$$C(2,m,n) = \binom{m}{2} \sum_{tc=2}^{n} \sum_{t_{i_1}=2}^{tc} t_{i_1} = \frac{m^2-m}{2} * \frac{n^3+3n^2-4n}{6} \quad (5)$$

$$C(m',m,n) = \binom{m}{m'} \sum_{tc=2}^{n} \sum_{q=2}^{tc-m'+2} P(m'-2, tc-q) * \frac{q^2+q-2}{2} \quad (6)$$

$$M(m,n) = \sum_{m'=1}^{m} C(m',m,n) \quad (7)$$

It can be seen from (6) that the total number of mappings is related to the partition problem solution. Therefore, the general expression for $M(m,n)$ can be derived if the analytical formula of $P(k,n)$ is available. Based on formulas of $P(k,n)$ reported in [2], Table III presents the complexity of our algorithm for $m = 1$ to 7, where $\Theta(M(m,n))$ and $\Theta(P(m-2,n))$ presents the complexity of the whole algorithm and the partition problem respectively.

B. Run-time mapping

Run-time mapping of applications onto a platform is handled by the *Run-time Platform Controller (RTPC)* (Fig. 3). In the platform, one processor is used as the RTPC (manager) that is responsible for actor mapping, actor scheduling, platform resource control and configuration control. The resources' status is updated at run-time when an actor is loaded in the platform. The RTPC maps the applications on the platform one after another till all the applications are mapped. The sequential mapping is scalable as it avoids the overhead for considering large number of scenarios containing different simultaneously active applications. For each application, the RTPC takes its desired throughput, platform with updated resources' status and the optimal mapping database (OMDb) as input (Fig. 3) and selects the best mapping satisfying the desired throughput by following Algorithm 3.

The algorithm selects a $mapping$ having minimum $energyConsumption$ from the *OMDb* by iterating from tile count one to Max_Used_Tiles. The provided mapping by this kind of iteration uses minimum possible number of tiles, resulting in improved resource utilization. Max_Used_Tiles is considered as **min**(number of actors in the application, number of available tiles) to restrict unnecessary search in *OMDb*. The existing approaches allocate actors to tiles based on a selected $mapping$ but do not consider relative position of actors, which might require a large amount of communication energy to facilitate communication amongst them through the edges. In our approach, we allocate highly communicating actors in close proximity by following the Algorithm 3 in order to save the communication energy. If a throughput satisfying mapping is not found then the application cannot be supported with available platform resources. In general, throughput computation for a mapping is a time consuming process. Our approach just selects the best mapping without involving throughput computation at run-time and thus accelerates the overall run-time mapping process. Further, our approach uses minimum possible number of tiles and performs energy aware allocation towards facilitating efficient mapping.

Fig. 4 demonstrates an example of run-time mapping of

Algorithm 3 Run-time Mapping

```
for tile_count = 1 to Max_Used_Tiles do
    for each mapping μ using tile_count tiles in OMDb do
        Select closest available tile_count tiles used by μ in the platform;
        max_hop = findMaximumHop(selected tiles);
        Mapping_list = Find all throughput satisfying mappings that use
        tile_count tiles separated by max_hop and have the same resource
        combination as μ;
        if Mapping_list! = NULL then
            Select the mapping having minimum energyConsumption;
            Edge_list = Find edges mapped to connections in mapping;
            Sort Edge_list in descending order of number of transferred bits;
            for each edge e in Edge_list do
                Allocate connected actors of e on tiles in close proximity based on
                the allocations in mapping;
            end for
            Terminate algorithm;
        end if
    end for
end for
```

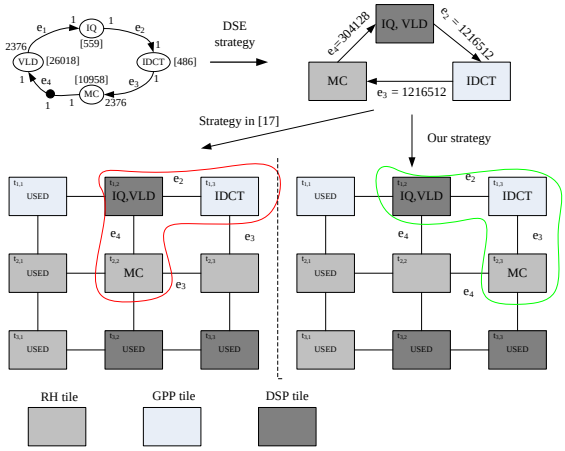

Figure 4. Example of Run-time Mapping

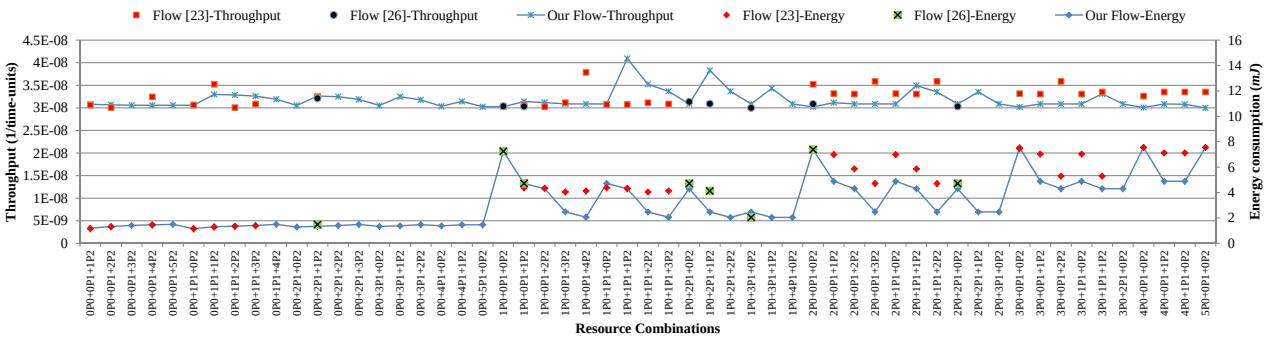

Figure 5. Throughput and energy of MPEG at all resource combinations for different flows

H.263 decoder on a platform when employing existing and our approach. Although the communication overhead (in bits) of e_3 (between RH (containing MC) and GPP (containing IDCT)) is greater than the communication overhead (in bits) of e_4 (between DSP (containing IQ, VLD and RH (containing MC)), the *existing work* assigns the connected actors onto two tiles with *hop_distance=2*. In contrast, our strategy considers communication overhead between actors and tries to map highly communicating actors (on RH and GPP) close to each other, so that energy consumption can be further reduced.

V. Performance Evaluation

Our strategy has been implemented as an extension of the tool set SDF3, which is publicly available [25]. The experiments are conducted on a Core i5 processor running at 2.4 GHz. As a benchmark, models of real-life multimedia applications H.263 decoder (4 actors), H.263 encoder (5 actors), MPEG-4 decoder (5 actors), JPEG decoder (6 actors) and sample rate converter (6 actors) have been considered to examine the efficiency of proposed strategy. MPEG-4 decoder, JPEG decoder, and sample rate converter will also be referred to as MPEG, JPEG, and samplerate respectively. All the applications are considered to be mapped onto a generic platform with 3 tile-types: GPP, DSP and RH. Larger number of tile-types can also be considered as explained earlier. We assume that all actors of applications can be implemented in these tile-types and their execution times on different tile-types are known a priori. Since we consider a *generic platform* as mentioned in Section IV, the maximum number of processing elements in the platform depends on the number of actors in evaluated applications. In the experiments, we compare our approach with the flows reported in [26] and in [23]. Since the strategy in [26] considers mapping for scenario, we applied it to a single scenario, i.e., a single version of the application that has always the same behavior. The approach in [23] performs optimization similar to that of ours, thus has been considered for the comparison. Therefore, we have fair comparison for all approaches. Several experiments have been performed to evaluate these strategies in term of throughput, energy consumption and execution time.

The throughput and energy of mappings produced by different DSE flows are calculated by the SDF3 tool set [25], which is modified according to evaluated mapping algorithms. The results for MPEG-decoder at different possible resource combinations are illustrated in Fig. 5. In this experiment and later in Fig. 7, *P0, P1, P2* represent 3 types of Processing Elements (PEs): GPP, DSP and RH; while each resource combination is referred as "*iP0+jP1+kP2*", where i, j, k are the number of PEs of each type. Our strategy provides throughput and energy values at all the resource combinations, which has been shown by two continuous lines. In contrast, other flows cannot cover all the resource combinations so they provide discrete points of throughput and energy values, and there are no values at uncovered resource combinations. It can be seen that mappings from our strategy have lower energy consumption while maintaining the throughput almost at the same level as that of other flows. Moreover, we have computed the energy saving of our DSE over the DSE strategy in [23] to illustrate the improvements. The results have shown that our DSE strategy reduces the energy consumption of H263 Decoder, H263 Encoder, JPEG, MPEG, and Samplerate by 11.32%, 12.63%, 8.26%, 24.93%, and 14.45%, respectively.

For different multimedia applications, Table IV shows the number of resource combinations covered by different DSE flows. The number of resource combinations depends on the number of actors in applications. The strategy in [26]] missed a large number of resource combinations since they look only load balanced mappings and there are a lot of duplications generated by their flow. The strategy in [23] has better result but still missing about 40% and 50% resource combinations in case of 5 actors (H263 Encoder, MPEG) and 6 actors (JPEG, Samplerate) respectively. In contrast, our approach is designed to cover all the resource combinations for all the applications. The number of covered resource combinations is important for hybrid mapping strategy since it decides the flexibility for Platform Manager at run-time under the resource constraint. Since our flow provides more mapping options for run-time, the RTPC can be better supported.

Table IV
Covered resource combinations

Applications	Our Flow	Flow in [23]	Flow in [26]
H263 Decoder	34	24	10
H263 Encoder	55	34	10
MPEG	55	34	11
JPEG	83	44	11
Samplerate	83	42	10

One of the most important features that define the efficiency of a DSE strategy is the number of evaluations performed by the strategy. A DSE with exhaustive search analyzes all the possible mappings for each resource combination. Therefore, it cannot scale well with the number of actors in application or the number of tile types in platform. Moreover, large number of evaluations require more computation power, evaluation time at design time, and more storage memory, more searching time in the memory at run-time. On the other hand, heuristic DSE approaches significantly reduce the number of evaluated mappings but might not provide an optimal mapping for run-time [26] or might discard mappings at several resource combinations [23]. Table V shows the number of mappings evaluated by different DSE strategies when three types of tile are considered.

It can be seen from Table V that the number of mappings evaluated by exhaustive DSE (EDSE) increases exponentially with the number of actors. Therefore, when number of actors is large (greater than 10), the exhaustive flow cannot be executed within a reasonable time. The flow in [26] significantly reduces the number of mappings when compared to EDSE. However, they still perform a large number of mappings in comparison with our strategy and strategy in [23]. Although strategy in [23] is better flow in term of number of mappings, it does not cover all the resource combinations. The number of mappings by our strategy is in between that of flow [26] and flow [23], but our flow provides mappings with better quality as demonstrated previously. The number of mappings is closely related to the execution time of the strategies. Fig. 6 shows execution time of different DSE strategies for different applications. Our strategy provides speed up over the strategy in [26], but spends more time to analyze mappings for all the resource combinations when compared with strategy in [23].

To show the improvement of our flow in term of energy consumption, we compare our results with existing hybrid approach of [23]. Our design-time DSE approach shows significant energy savings for all the considered applications when compared to existing approaches as mentioned earlier. Table VI presents the energy saving at run-time obtained by our flow for mapping different applications when compared with the flow in [23]. At run-time, the main goal of our technique is to reduce the communication energy by allocating highly-communicating actors close to each other. Our technique provides energy savings over existing techniques when at least 3 tiles are used in the mapping. If less than 3 tiles are used, there is no edge for which communication overhead can be reduced and our approach provides similar results as that of [23]. Especially, in applications (H263 Encoder, JPEG, Samplerate) where the communication overhead is high, our technique has better improvement on energy savings (up to 17.8%). Similar improvements are obtained for other applications (Table VI).

Table V
NUMBER OF MAPPINGS WITH THREE TILE-TYPES

Number of Actors	EDSE	Strategy in [26]	Strategy in [23]	Our strategy
1	3	3	3	3
2	12	42	10	12
3	57	180	25	38
4	309	372	51	90
5	1,866	615	91	178
6	12,351	918	148	313
7	88,563	1281	225	507
8	681,870	1704	325	773
9	5,597,643	2187	451	1125
10	48,718,569	2730	606	1578
14	461,101,962,108	5502	1576	4735

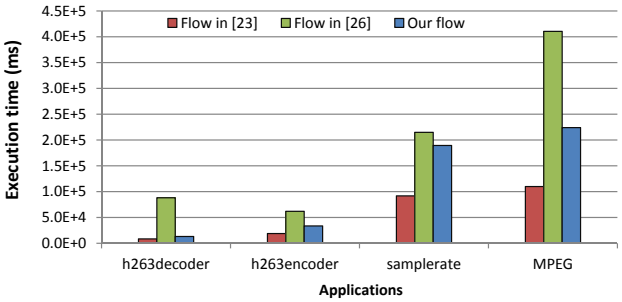

Figure 6. Execution time of different DSE strategies

We also have evaluated the efficiency of choosing parameter $\mu(\frac{throughput}{energy})$ in the optimization process. We evaluated our strategy with three different optimization criteria: *throughput*, *energy*, and μ. Fig. 7 shows throughput and energy for the best mappings at different resource combinations for H263 decoder when different parameters are chosen. The DSE optimized by energy always provides better results than DSE with Throughput Optimization in term of energy consumption. Similarly, reverse implication can be made when throughput is chosen as the optimization criteria. When we choose the optimization criteria μ, for energy, the results lie between the Throughput and Energy Optimization and almost overlap the result of Energy Optimization. If we consider throughput as the guideline of optimization process, the μ option sometimes obtain better results over the Throughput Optimization approach. Due to the heuristic behavior of our approach, the Throughput Optimization might drop several mapping options and miss some optimal points which can be found by the μ-Optimization. Therefore, using μ as the guideline for optimization process not only achieves better trade-off between throughput and energy, but also provides better throughput at several resource combinations.

Table VI
ENERGY SAVING USING OUR RUNTIME TECHNIQUE

Application	Number of tiles	Energy consumption *(mJ)*		Percent of improvement (%)
		Strategy in [23]	Our Strategy	
H263 Decoder	4 tiles	2.909	2.872	2.82
	3 tiles	2.827	2.786	1.45
H263 Encoder	5 tiles	6.072	5.038	17.03
	4 tiles	5.814	4.779	17.80
	3 tiles	5.038	4.521	10.26
JPEG	6 tiles	0.365	0.334	8.56
	5 tiles	0.360	0.328	8.89
	4 tiles	0.354	0.323	8.76
	3 tiles	0.344	0.318	7.56
MPEG	5 tiles	8.131	7.86	3.33
	4 tiles	8.053	7.821	2.88
	3 tiles	8.015	7.783	2.89
Samplerate	6 tiles	5.857	5.323	9.12
	5 tiles	5.768	5.234	9.26
	4 tiles	5.590	5.145	7.960
	3 tiles	5.501	5.056	8.09

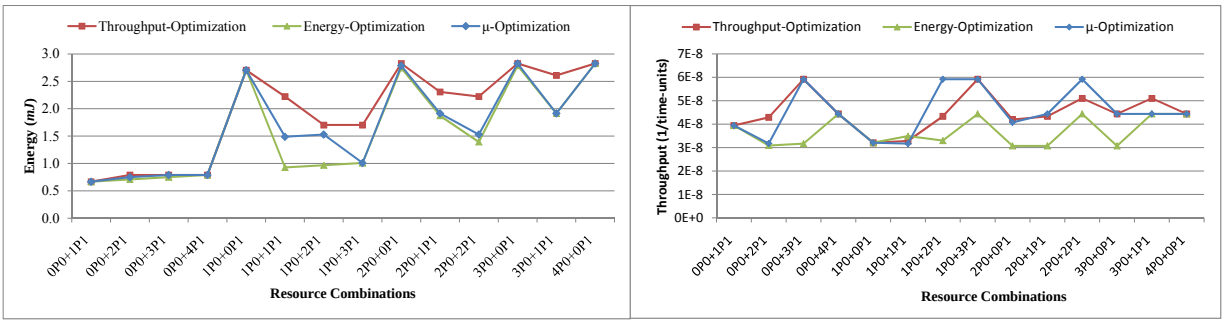

Figure 7. Throughput and energy consumption for H263 Decoder at different resource combinations for different optimization criteria

VI. Conclusion

This paper presents an efficient mapping strategy for heterogeneous MPSoC platform. Our DSE approach covers all the resource combinations at design-time within a small evaluation time. The qualities of the mappings in term of throughput and energy are proven by experiments on a series of real-life streaming applications. Especially, our DSE takes the trade-off between throughput and energy as the optimized criteria; so that the mapping results can achieve more balance performance. Moreover, our run-time mapping technique further improves the energy consumption of the system by consider communication overhead in real time. The experimental results show that our approach provides better energy savings and performance in comparison to existing approaches. In future, based on the DSE strategy, we plan to develop an analytical approach for evaluating the mappings at design-time to further reduce the exploration time. Another improvement that we would like to consider is to allow resource sharing between applications.

References

[1] Y. Ahn et al. SoCDAL: System-on-chip design AcceLerator. *ACM TODAES*, 13:17:1–17:38, 2008.

[2] G. E. Andrews and K. Eriksson. *Integer partitions*. Cambridge University Press, 2004.

[3] L. Benini, E. Flamand, D. Fuin, and D. Melpignano. P2012: Building an ecosystem for a scalable, modular and high-efficiency embedded computing accelerator. In *DATE*, pages 983–987, 2012.

[4] E. Carvalho and F. Moraes. Congestion-aware task mapping in heterogeneous MPSoCs. In *SoC*, pages 1–4, 2008.

[5] P. Cumming. The ti omap platform approach to soc. *Winning the SOC Revolution*, 2003.

[6] A. H. Ghamarian et al. Throughput Analysis of Synchronous Data Flow Graphs. In *ACSD*, pages 25–36, 2006.

[7] B. Giovanni, L. Fossati, and D. Sciuto. Decision-theoretic design space exploration of multiprocessor platforms. *IEEE TCAD*, 29:1083–1095, 2010.

[8] P. Holzenspies, J. Hurink, J. Kuper, and G. J. M. Smit. Run-time spatial mapping of streaming applications to a heterogeneous multi-processor system-on-chip (mpsoc). In *DATE*, pages 212–217, 2008.

[9] J. Hu and R. Marculescu. Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints. In *DATE*, pages 10234–, 2004.

[10] J. Keinert et al. SystemCoDesigner an automatic ESL synthesis approach by design space exploration and behavioral synthesis for streaming applications. *ACM TODAES*, 14:1:1–1:23, 2009.

[11] R. Kumar, D. M. Tullsen, N. P. Jouppi, and P. Ranganathan. Heterogeneous chip multiprocessors. *Computer*, 38(11):32–38, 2005.

[12] C. Lee, S. Kim, and S. Ha. A systematic design space exploration of mpsoc based on synchronous data flow specification. *JSPS*, 58(2):193–213, 2010.

[13] 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.

[14] M. Lukasiewycz, M. Glaß, C. Haubelt, and J. Teich. Efficient symbolic multi-objective design space exploration. In *ASP-DAC*, pages 691–696, 2008.

[15] G. Mariani et al. An industrial design space exploration framework for supporting run-time resource management on multi-core systems. In *DATE*, pages 196–201, 2010.

[16] G. Mariani et al. Using multi-objective design space exploration to enable run-time resource management for reconfigurable architectures. In *DATE*, pages 1379–1384, 2012.

[17] O. Moreira, J. J.-D. Mol, and M. Bekooij. Online resource management in a multiprocessor with a network-on-chip. In *SAC*, pages 1557–1564, 2007.

[18] O. Moreira, F. Valente, and M. Bekooij. Scheduling multiple independent hard-real-time jobs on a heterogeneous multiprocessor. In *Proceedings of the International conference on Embedded software*, pages 57–66, 2007.

[19] V. Nollet, P. Avasare, H. Eeckhaut, D. Verkest, and H. Corporaal. Run-time management of a MPSoC containing FPGA fabric tiles. *IEEE TVLSI*, 16:24–33, 2008.

[20] A. Schranzhofer, J.-J. Chen, and L. Thiele. Power-aware mapping of probabilistic applications onto heterogeneous mpsoc platforms. In *RTAS*, pages 151–160, 2009.

[21] A. Schranzhofer, J.-J. Chen, and L. Thiele. Dynamic Power-Aware Mapping of Applications onto Heterogeneous MPSoC Platforms. *IEEE Transactions on Industrial Informatics*, 6(4):692 –707, 2010.

[22] A. K. Singh, A. Kumar, and T. Srikanthan. A Hybrid Strategy for Mapping Multiple Throughput-constrained Applications on MPSoCs. In *CASES*, pages 175–184, 2011.

[23] A. K. Singh, A. Kumar, and T. Srikanthan. Accelerating throughput-aware runtime mapping for heterogeneous mpsocs. *ACM TODAES*, 18(1):9:1–9:29, 2013.

[24] L. T. Smit, J. L. Hurink, and G. J. Smit. Run-time mapping of applications to a heterogeneous soc. In *SoC*, pages 78–81, 2005.

[25] S. Stuijk, M. Geilen, and T. Basten. SDF^3: SDF For Free. In *ACSD*, pages 276–278, 2006.

[26] S. Stuijk, M. Geilen, and T. Basten. A Predictable Multiprocessor Design Flow for Streaming Applications with Dynamic Behaviour. In *DSD*, pages 548 –555, 2010.

[27] P. Yang et al. Managing dynamic concurrent tasks in embedded real-time multimedia systems. In *ISSS*, pages 112–119, 2002.

[28] C. Ykman-Couvreur, P. Avasare, G. Mariani, G. Palermo, C. Silvano, and V. Zaccaria. Linking run-time resource management of embedded multi-core platforms with automated design-time exploration. *Computers Digital Techniques, IET*, 5(2):123 –135, 2011.

[29] C. Ykman-Couvreur, P. A. Hartmann, G. Palermo, F. Colas-Bigey, and L. San. Run-time resource management based on design space exploration. In *CODES*, pages 557–566, 2012.

[30] C. Ykman-Couvreur, V. Nollet, F. Catthoor, and H. Corporaal. Fast Multi-Dimension Multi-Choice Knapsack Heuristic for MP-SoC Run-Time Management. In *SoC*, pages 1 –4, 2006.