2011 Fourth International Symposium on Parallel Architectures, Algorithms and Programming

Communication-aware Design Space Exploration for Efficient Run-time MPSoC Management

[1]Amit Kumar Singh, [2]Akash Kumar, [3]Wu Jigang and [1]Thambipillai Srikanthan
[1] School of Computer Engineering, Nanyang Technological University, Singapore
[2] Department of Electrical and Computer Engineering, National University of Singapore, Singapore
[3] School of Computer Science and Software, Tianjin Polytechnic University, Tianjin, China
Email: [1]{amit0011, astsrikan}@ntu.edu.sg, [2]akash@nus.edu.sg, [3]asjgwu@gmail.com

***Abstract*—Real-time multi-media applications are increasingly mapped on modern embedded systems based on Multiprocessor Systems-on-Chip (MPSoCs). Tasks of the applications need to be mapped on the MPSoC resources efficiently in order to satisfy their performance constraints. Exploring all the mappings, i.e. tasks to resources combinations exhaustively may take days or weeks. Additionally, the exploration is performed at design-time that cannot handle dynamism in applications and resources' status. A run-time mapping technique can cater for the dynamism but cannot guarantee for strict timing deadlines due to large computations involved at run-time. Thus, an approach performing feasible compute intensive exploration at design-time and using the explored results at run-time is required. This paper presents a solution in the same direction. Communication-aware design space exploration techniques have been proposed to explore different mapping options to be selected at run-time subject to desired performance and available MPSoC resources. Experiments show that the proposed techniques for exploration are faster over an exhaustive exploration and provides almost the same quality of results.**

I. Introduction

Advanced multimedia embedded systems (e.g., smart phones, tablets, PDAs) need to support multiple applications concurrently. The increasing performance demands of concurrently running applications are satisfied by relying the systems on multiprocessor systems-on-ship (MPSoCs), for example, IBM Cell [1] and NXP Nexperia [2]. The MPSoCs may contain different type of processing elements (PEs) connected by a communication network in order to achieve high performance by exploiting their distinct features.

The system users expect that throughput constraints of all applications running in the system are satisfied which heavily depends upon how efficiently application tasks are mapped onto system resources (PEs). There is an enormous number of possibilities for mapping the tasks onto the PEs. The mapping is accomplished either by design-time DSE [3][4] or run-time mapping strategies [5][6]. The design-time DSE strategies are incapable of handling dynamism such as adding a new application into the platform at run-time. On the other hand, the run-time mapping strategies cannot provide timing guarantees due to lack of any previous analysis and limited computational resources at run-time. Thus, an approach performing compute intensive analysis (DSE) at design-time and using the analysis results at run-time is required.

The design-time DSE strategies need to find a number of mappings by taking an application and a platform as input. An exhaustive DSE to find all the possible mappings is not scalable when the number of tasks/PEs is large as we need to explore for lot of tasks to PEs combinations. Existing DSE strategies are applicable only to fixed MPSoC platform, don't scale well with the number of PEs in the platform and don't always provide the largest throughput mapping as they perform DSE in view of optimizing for the performance metrics such as energy and resource optimization.

Contribution: This paper presents design-time DSE strategies that perform analysis on a *generic* MPSoC platform in view of optimizing throughput and produce resource-throughput trade-off points, i.e. tasks to PEs mappings with their throughput. A resource has been referred as a tile that essentially contains a processing engine along with other elements such as memory. The platform contains different type of tiles such as a processor or a reconfigurable hardware (RH) block as the processing engine, i.e. the platform is heterogeneous. The generated points can be used by a light-weight run-time manager to select the best point depending upon the available tiles in the platform and desired throughput. First, an exhaustive DSE strategy is presented that produces all the possible tasks to PEs mappings, which is not scalable with the number of tasks in the application. To overcome the large exploration overhead (may be a couple of weeks for large application size), we present a communication-aware DSE (CADSE) strategy that discards the evaluation of inefficient points and produces almost the same best trade-off points. To further accelerate the DSE, we incorporated pruning in the CADSE (PCADSE) where evaluation of the number of trade-off points is further reduced based on a pruning idea. The quality of the best mappings generated by the CADSE and PCADSE strategies do not differ significantly, while the exploration process is speeded up.

Overview. Section II introduces state-of-the-art multiprocessor DSE strategies. Our proposed DSE methodologies along with multiprocessor/application model used in this work are introduced in Section III. Section IV presents a set of experimental results on the efficiency of the proposed approach. Section V concludes the paper and provides directions for future work.

II. Related Work

Several DSE strategies providing single mapping for an application have been reported in literature [7][8][9]. These strategies are applicable only to fixed MPSoC platforms and mappings are not optimized from throughput point of view as throughput optimization is not their target but to satisfy some constraint. Further, they cannot handle dynamism in resource availability and throughput (QoS) requirement at run-time. However, our DSE strategies are applied to a generic MPSoC platform and generate a number of mappings with different resource requirement and throughput, which helps to handle

978-0-7695-4575-2/11 $26.00 © 2011 IEEE
DOI 10.1109/PAAP.2011.18

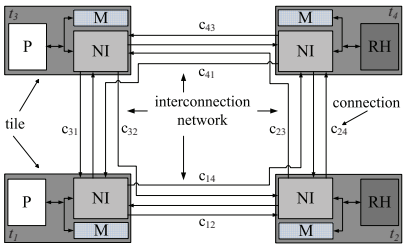

Fig. 1. Multiprocessor platform example.

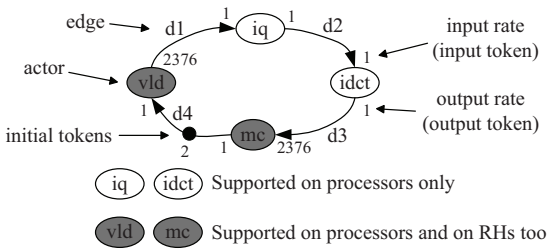

Fig. 2. SDF Graph model of an H.263 decoder.

run-time dynamism and allows them to be mapped on any architecture without requiring analysis to be repeated.

DSE strategies providing multiple mappings for homogeneous MPSoC platforms have been recently presented in [4], [10] and [11]. In [4], DSE is performed in view of optimizing for the resource usage, whereas in [10] and [11], for optimizing power. These strategies have several drawbacks, e.g., applicable only to fixed homogeneous platforms, generated mappings are not optimized from throughput point of view, generate duplicate mappings for larger platforms and not scalable with the platform size. The duplicate mappings have the same throughput but they differ in placement of tasks on different tiles with the same tasks to tiles binding. Singh et al. [12] propose a DSE strategy that performs exploration in view of optimizing throughput by considering a generic platform but the considered platform is homogeneous, i.e., contains only one type of tile, and the exploration follows a pruning strategy.

Our strategy considers a generic platform containing different types of tiles depending upon the application tasks' specifications and always provides mappings with maximum throughput by performing exploration in a communication-aware manner. The generated mappings are applicable to any platform containing tiles with maximum separation between two of them as the one considered during DSE without repeating the DSE. The generation of duplicate mappings are avoided by not considering a bigger platform than required that can exploit all the parallelism present in the application.

III. Proposed Design Space Exploration Methodologies

This section introduces our proposed DSE methodologies. First, we describe the hardware MPSoC architecture and application model used in our DSE methodologies.

Multiprocessor Architecture Model. The hardware architecture model describes platform processing units and the interconnection network between them. The platform model uses tile-based architecture that uses an interconnection network to connect the tiles as shown in the example platform of Fig. 1. The platform contains tiles t_1, t_2, t_3 & t_4, which are connected by point-to-point connections (c) with fixed latencies. Latency of connections through any Network-on-Chip (NoC) can be modeled so long as the latencies between tiles are provided. Each tile contains a processing engine (processor P or RH), 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 [13] and Eclipse [14] fit nicely into this platform model.

The communication network used in the example platform of Fig. 1 is arranged in a 2-D mesh topology. The distance between two tiles is referred to as *hop_distance*. Adjacent tiles t_1 & t_2 are at hop_distance of 1 and t_1 & t_4 at hop_distance of 2 (1 hop in X-direction to reach t_2 and 1-hop in Y-direction to reach t_4). The latency of connections between the tiles is directly proportional to hop_distance. We increase the latency of connections between the tiles to account for the higher hop distances. This facilitates for finding mappings even when the tiles are further apart in the actual platform.

Application Model. The application model considers throughput-constrained multimedia applications consisting of multiple tasks. Synchronous Dataflow Graphs (SDFGs) [15] are used to model such applications. Throughput is an important constraint and determines how often tasks of the application finish their execution, which is determined by the cycles in the SDFG. An SDFG model of H.263 decoder application is shown in Fig. 2. Nodes modeling tasks are called *actors* that communicate with *tokens* sent from one actor to another through *edges* modeling dependencies. The application is modeled with four actors *vld, iq, idct* & *mc* and four edges *d1, d2, d3* & *d4*. An actor has following attributes: its implementation alternatives (e.g., processor and RH tile), execution time (in time-units) and memory needed (in bits) on the implementation alternatives. An edge has following attributes: size of a token (in bits), memory (in tokens) needed when connected actors are allocated to the same tile, memory (in tokens) needed in source and destination tiles, bandwidth (in bits/time-unit) needed when connected actors are allocated to different tiles. An actor *fires* (executes) when there are sufficient tokens on all of its input edges and sufficient buffer space on all of its output channels. At each firing, a fixed number of tokens from the input edges are consumed and a fixed number of tokens on the output edges are produced. These numbers are referred to *rates* that define how often actors have to fire with respect to each other. The edges may have *initial tokens* to start the actor firing, indicated by a bullet in the Fig. 2. The application model also specifies a throughput-constraint.

Existing DSE strategies find mappings while performing optimization for power and resource usage. This might lead to mapping of parallel executing actors on the same tile and thus forcing their execution sequentially, resulting in reduced throughput. However, our strategies optimize throughput so produce mappings with maximum throughput. Our strategies map the connected and sequentially executing actors on the same tile, resulting in reduced communication overhead that might maximize throughput. A number of mappings are evaluated for each multimedia application to be supported on a hardware platform. The evaluation considers finding different

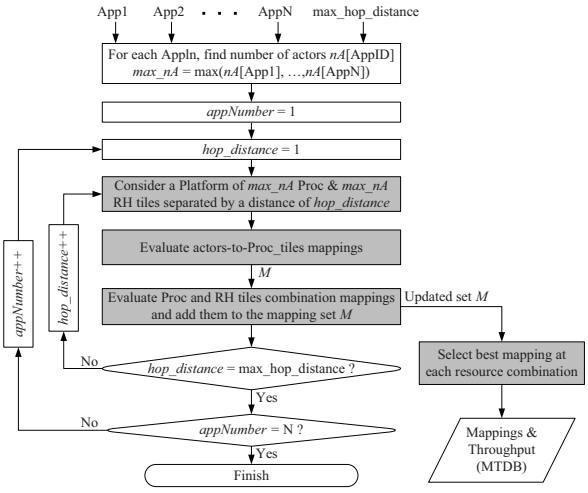

Fig. 3. Exhaustive Design Space Exploration Flow.

mappings and their throughput. For each mapping, actors are bound to tiles and edges to memory inside tiles or to connections in the platform. The binding is considered valid if memory imposed, allocated input/output connections and allocated incoming/outgoing bandwidth are less than or equal to the maximum available on each tile. Only the valid bindings are considered and throughput for the same is computed. For computing throughput, first, a static-order schedule for each tile is constructed, which orders the execution of bound actors. Then, all the binding and scheduling decisions are modeled in a graph called binding-aware SDFG. Finally, throughput is computed by self-timed state-space exploration of the binding-aware SDFG [16]. We now introduce our DSE strategies.

A. Exhaustive Design Space Exploration

The exhaustive design space exploration (EDSE) flow evaluates all the possible actors to tiles combinations, i.e., mappings. The flow is presented in Fig. 3. The flow takes application models as input and stores the best mapping (MTDB) at each possible resource combination. The applications are evaluated one after another by incrementing the application number (*appNumber*++). The main steps of the flow are highlighted and described subsequently.

1) Considering a Suitable Platform Graph: This step of the analysis flow considers a platform graph that can evaluate all possible mappings for each application. The application actors implementation alternatives could be a number of tile types. In particular, we have considered processor (Proc) and reconfigurable hardware (RH) tiles. The considered platform contains *max_nA* Proc & *max_nA* RH tiles, where *max_nA* is the maximum value of number of actors in an application amongst all the applications. This platform is capable of exploiting all the parallelism present in each of the application and considering any bigger platform wouldn't provide better performance.

The initial considered platform contains tiles separated by a distance of one hop_distance (*hop_distance = 1*), which caters for a minimum latency for all the connections between the tiles. The DSE flow is repeated by considering a similar platform containing tiles separated by one higher hop_distance (*hop_distance*++), i.e., with increased latency for connections, till *hop_distance* reaches to *max_hop_distance* (input to the DSE flow). The designers can opt for a suitable value of *max_hop_distance* depending upon the expected hardware platform at run-time, where, maximum hop_distance between two tiles can be up to *max_hop_distance*.

Varying hop_distance consideration provides mappings where application edges are mapped to connections at hop_distance of one (to cater for minimum latency) to *max_hop_distance* (to cater for maximum latency). This caters for the run-time aspects when the available tiles are at different hop_distances.

2) Evaluating Processor Tiles Mappings: This step evaluates all the possible actors to processor (Proc) tiles mappings. The evaluation follows a set of steps described subsequently. An application with one actor (a_1) to be mapped on Proc tiles has only one unique actor to tile mapping, which is computed from equation 1. An application with two actors (a_1,a_2) has two unique mappings that is computed from equation 2. One mapping contains actors on separate tiles ($^1\mathrm{C}_0$ implies that from the remaining one actor a_1, it is not chosen to combine it with actor a_2) and another on the same tile ($^1\mathrm{C}_1$ implies that actor a_1 is chosen to combine it with actor a_2). Similarly, for an application with three actors (a_1,a_2,a_3), the unique mappings are computed from equation 3. First, actor a_3 is mapped separately, i.e., not combined with others (from the remaining two actors a_1 & a_2, none is chosen to combine with a_3, indicated as $^2\mathrm{C}_0$) and remaining two actors are mapped by using equation 2 ($f_{EDSE}(2,a_1,a_2)$), providing two unique mappings. Then, from the remaining two actors one actor is chosen to combine with actor a_3 ($^2\mathrm{C}_1$) and the remaining actor is mapped separately, providing two unique mappings. Next, from the remaining two actors, both are chosen to combine with actor a_3, providing one unique mapping. Thus, for an application with three actors, a total of five unique actors to tiles mappings are evaluated.

The equations are extended in the similar manner for larger number of actors. For n actors (a_1,a_2,...,a_n), the mappings can be computed from equation 4. It can be observed that when computing mappings for larger number of actors, the mappings computed at lower number of actors are used, such as $f_{EDSE}(n-1,a_1,a_2,...,a_{n-1})$ in $f_{EDSE}(n,a_1,a_2,...,a_n)$.

$$f_{EDSE}(1, a_1) = 1 \tag{1}$$

$$f_{EDSE}(2, a_1, a_2) =^1 C_0 \times f(1, a_1) +^1 C_1 \tag{2}$$

$$\begin{aligned} f_{EDSE}(3, a_1, a_2, a_3) =^2 C_0 \times f(2, a_1, a_2) \\ +^2 C_1 \times f(1, remain\_actor) +^2 C_2 \end{aligned} \tag{3}$$

$$\begin{aligned} f_{EDSE}(n, a_1, a_2, ..., a_n) =^{(n-1)} C_0 \times f(n-1, a_1, a_2, ..., a_{n-1}) \\ +^{(n-1)} C_1 \times f(n-2, remain\_actors) \\ +^{(n-1)} C_2 \times f(n-3, remain\_actors) \\ . \\ . \\ +^{(n-1)} C_{n-2} \times f(1, remain\_actor) +^{(n-1)} C_{n-1} \end{aligned} \tag{4}$$

Algorithm 1: Proc/RH tiles Comb. Mappings Evaluation

```
Input: Proc tiles mappings M
Output: Proc and RH tiles combination mappings to be added to set M
for each Proc tiles mapping α (∈ M) do
    findProc&RHTilesCombMappings(α, t1);
end

function findProc&RHTilesCombMappings(Mapping β, Tile
startProcTile)

if startProcTile == lastProcTile+1 then
    return;
end
for Tile i = firstProcTile to lastProcTile (in current mapping) do
    if tile i contains actor(s) and all of them can be mapped on RH
    tile then
        Move actor(s) of tile i on a free RH tile to generate a new
        mapping α;
        Compute throughput of α;
        Add α with its throughput to set M;
        findProc&RHTilesCombMappings(α, i+1);
    end
end

end function
```

3) Evaluating Processor and Reconfigurable Hardware Tiles Combination Mappings: The Proc and RH tiles combination mappings are evaluated by Algorithm 1, which takes Proc tiles mappings (*M*) obtained in the previous step (Fig. 3) as input. The algorithm evaluates a number of mappings and adds them in the same set. Thus, the final mapping set *M* contains all the evaluated mappings using different Proc/RH tiles combinations. For the example H.263 decoder application when each actor has two implementation alternatives, we get a total of 94 mappings at each hop_distance value.

4) Selecting Best Mapping at Each Resource Combination: At each Proc/RH tiles combination, we get a number of mappings. For example, at 2Proc & 2RH tiles resource combination, we get a total of 6 mappings for the H.263 decoder. This step selects the maximum throughput mapping at each resource combination and stores it into the mappings & throughput database (MTDB) (Fig. 3). These stored mappings provide options for mapping the application at run-time. The best mapping can be selected based on the available platform resources and desired throughput. The selected mapping is then used to configure the platform.

B. Communication-Aware Design Space Exploration

The communication-aware design space exploration (CADSE) strategy incorporates communication awareness in the EDSE (Fig. 3). The step to explore Proc tiles mappings (*Evaluate actors-to-Proc_tiles mappings*) is modified to perform the exploration in communication-aware manner. For an application with n actors (a_1,a_2,...,a_n), the Proc tiles mappings are evaluated from equation 5, which requires *n-1*, *n-2*, *n-3*, ..., *2*, *1* actors mappings in advance as in the EDSE. These mappings can be calculated by putting different values of n in the equation 5. This equation differs from equation 4 while choosing actors to be combined with actor a_n on the same tile. The chosen actors and actor a_n are checked if they are connected (conn) before they are mapped on the same tile. For example, ${}^{(n-1)}\mathrm{C}_{2-conn}$ in equation 5 specifies that the chosen (C) two actors and actor a_n are connected.

Algorithm 2: Distinct Channels Calculation

```
Input: Application Graph
Output: Number of distinct channels
distinctChannelCount = 0;
for actor a_i = FirstActor (a_1) to LastActor (a_n) do
    for actor a_j = actor next to a_i (i.e. a_i+1) to LastActor (a_n) do
        if a channel exists between a_i and a_j then
            distinctChannelCount++;
        end
    end
end
```

$$
\begin{aligned}
f_{CADSE}(n, a_1, a_2, ..., a_n) =& {}^{(n-1)}\ C_{0-conn} \times f(n-1, a_1, a_2, ..., a_{n-1}) \\
&+ {}^{(n-1)}C_{1-conn} \times f(n-2, remain\_actors) \\
&+ {}^{(n-1)}C_{2-conn} \times f(n-3, remain\_actors) \\
&\ \ . \\
&\ \ . \\
&+ {}^{(n-1)}C_{(n-2)-conn} \times f(1, remain\_actor) + {}^{(n-1)}\ C_{(n-1)-conn}
\end{aligned} \tag{5}
$$

To find whether a set of actors are connected, we find number of distinct channels between them. If there are more than one channel between two actors then only one channel is counted as the distinct channel. The actors are connected if the number of distinct channels between the actors is $\geq$ (the number of actors – 1). For n actors (a_1,a_2,...,a_n), the total number of distinct channels are calculated from Algorithm 2.

C. Pruning-based Communication-Aware Design Space Exploration

The pruning-based communication-aware design space exploration (PCADSE) strategy incorporates pruning in the CADSE strategy. In Fig. 3, after evaluating Proc tiles mapping by CADSE, only the maximum throughput mapping at each Proc tile count is passed to evaluate Proc/RH tiles combination mappings as earlier. Proc tile count for a mapping is defined as the number of used Proc tiles. This strategy assumes that by starting with the best Proc tile mapping we should get the best mappings at Proc/RH tiles combinations. The pruning consideration facilitates for speeded exploration over the CADSE and provides almost the same quality (throughput) mappings.

IV. Experiments

The proposed DSE methodologies have been implemented as an extension to the publicly available tool set SDF^3 [17]. To evaluate run-time and quality of the methodologies, 100 random applications modeled as SDFGs with 4, 5, 6 and 7 actors having one of their implementation alternatives as Proc tile and other RH tile if present have been considered. The same generic platform graph is considered to evaluate the different DSE methodologies. We have adopted a tile-based architecture but any type of architecture can be modeled so long latencies between the tiles are known. The experiments have been performed on a Core 2 Duo processor at 3.16 GHz.

The number of mappings evaluated by EDSE increases exponentially with the number of actors. Further, the number of mappings increases even more when the implementation alternatives of actors get increased. Thus, the exploration may take a couple of days. This makes the EDSE non-scalable although it always provides the best quality of mapping at each resource combination. The CADSE has been employed

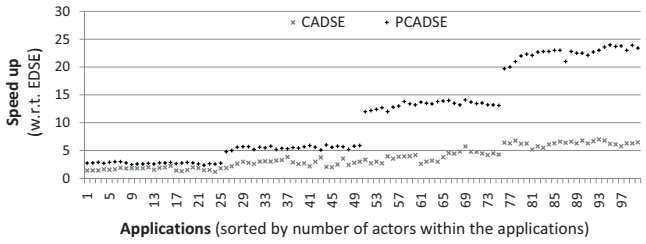

Fig. 4. Speed up obtained by CADSE and PCADSE over EDSE

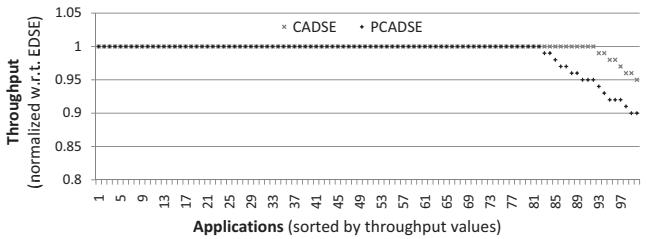

Fig. 5. Quality of mappings by CADSE and PCADSE over EDSE

to speed up the exploration process while providing almost the same quality (throughput) of mappings. The PCADSE speeds up the exploration process further while providing a bit of degraded quality of mappings.

Fig. 4 shows the speed up obtained by CADSE and PCADSE over the EDSE for all the 100 applications. The speed up by CADSE and PCADSE is calculated by dividing execution time of EDSE to the execution time of CADSE and PCADSE, respectively. The applications are sorted by the number of actors within them. It can be observed that CADSE is faster over the EDSE, and the PCADSE is faster even over the CADSE for all the applications. It is also clear that as the number of actors increases in the applications, the speed up obtained by the CADSE increases as the strategy discards evaluation of more number of mappings by incorporating communication-aware exploration, whereas speed up obtained by the PCADSE increases further as the strategy has to prune from a larger number of Proc tiles mappings. On an average, the CADSE and PCADSE is faster by 3.7× and 11×, respectively when compared to EDSE.

Fig. 5 shows the quality (throughput) of the best mapping at 2 Proc and and 1 RH tiles resource combination for all the applications when tiles are assumed to be separated by a fixed hop_distance. The best mapping throughput obtained by CADSE and PCADSE are normalized with respect to (w.r.t.) EDSE. The normalized throughput values are plotted after sorting them in descending order. It can be observed that the CADSE and PCADSE provide the same best mappings as of EDSE for more than 90% and 80% of the applications respectively. Similar behavior is obtained at other resource combinations. Thus, we can say that for most of the applications, we get the same quality of mappings by all the DSE strategies.

The DSE methodologies store the best mappings at each resource combination at varying hop_distance values (referred as hops). The best mapping at different hops remains the same with a bit of different quality of the mapping as the delays of connections between the tiles get changed. At run-time, the best mapping can be selected depending upon the available resources and hop_distance between them.

V. Conclusion

This paper presents design space exploration (DSE) strategies for supporting efficient run-time MPSoC management. The strategies store the best mapping at each resource combination, which can be directly used at run-time depending upon the available resources and desired throughput. Three DSE strategies have been presented. One strategy performs DSE exhaustively (EDSE) and produces the best quality of mapping at each resource combination, whereas, this strategy has worst run-time. Next, a communication-aware DSE (CADSE) strategy is presented to perform the exploration in communication-aware manner. This reduces the total number of mappings to be evaluated and thus the run-time for exploration. The quality of mappings remains almost the same. To further reduce the exploration run-time, a pruning criteria has been incorporated in the CADSE, which provides a bit of degraded quality of mappings. In future, we plan to develop more ways of faster DSE in order to further speed up the exploration process while providing almost the same quality of mappings as of the EDSE.

References

[1] M. Kistler et al., "Cell multiprocessor communication network: Built for speed," *IEEE Micro*, vol. 26, pp. 10–23, 2006.
[2] M. Kim et al., "Energy-aware cosynthesis of real-time multimedia applications on mpsocs using heterogeneous scheduling policies," *ACM Trans. Embed. Comput. Syst.*, vol. 7, pp. 9:1–9:19, 2008.
[3] 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.*, vol. 53, pp. 733–750, 2007.
[4] S. Stuijk et al., "A predictable multiprocessor design flow for streaming applications with dynamic behaviour," in *Proceedings of the 13th Euromicro Conference on Digital System Design*, 2010, pp. 548–555.
[5] V. Nollet et al., "Run-time management of a mpsoc containing fpga fabric tiles," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 16, pp. 24–33, 2008.
[6] A. K. Singh et al., "Communication-aware heuristics for run-time task mapping on noc-based mpsoc platforms," *J. Syst. Archit.*, vol. 56, pp. 242–255, 2010.
[7] O. Moreira et al., "Scheduling multiple independent hard-real-time jobs on a heterogeneous multiprocessor," in *Proc. EMSOFT'07*, 2007, pp. 57–66.
[8] A. Bonfietti et al., "Throughput constraint for synchronous data flow graphs," in *Proc. CPAIOR '09*. Springer-Verlag, 2009, pp. 26–40.
[9] A. Schranzhofer et al., "Dynamic power-aware mapping of applications onto heterogeneous mpsoc platforms," *Industrial Informatics, IEEE Transactions on*, vol. 6, no. 4, pp. 692 –707, 2010.
[10] C. Ykman-Couvreur at al., "Linking run-time resource management of embedded multi-core platforms with automated design-time exploration," *Computers Digital Techniques, IET*, vol. 5, no. 2, pp. 123 –135, 2011.
[11] G. Mariani et al., "An industrial design space exploration framework for supporting run-time resource management on multi-core systems," in *Proceedings of DATE*, 2010, pp. 196–201.
[12] A. K. Singh et al., "A hybrid strategy for mapping multiple throughput-constrained applications on mpsocs," in *accepted for publication in Proc. of CASES*, 2011.
[13] P. G. Paulin et al., "Application of a multi-processor soc platform to high-speed packet forwarding," in *Proc. DATE*, 2004, pp. 58–63.
[14] 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*, vol. 19, pp. 39–50, 2002.
[15] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Trans. Comput.*, vol. 36, pp. 24–35, 1987.
[16] A. H. Ghamarian et al., "Throughput analysis of synchronous data flow graphs," in *Proc. ACSD*, 2006, pp. 25–36.
[17] S. Stuijk, M. Geilen, and T. Basten, "SDF^3: SDF For Free," in *Proc. ACSD*, 2006, pp. 276–278.