Front. Comput. Sci.
DOI 10.1007/s11704-013-2196-1

RESEARCH ARTICLE

CADSE: communication aware design space exploration for efficient run-time MPSoC management

Amit Kumar SINGH(✉)[1,3], Akash KUMAR[1], Jigang WU[2], Thambipillai SRIKANTHAN[3]

1 Department of Electrical and Computer Engineering, National University of Singapore, Singapore 119077, Singapore
2 School of Computer Science and Software Engineering, Tianjin Polytechnic University, Tianjin 300160, China
3 School of Computer Engineering, Nanyang Technological University, Singapore 639798, Singapore

© Higher Education Press and Springer-Verlag Berlin Heidelberg 2013

Abstract Real-time multi-media applications are increasingly mapped on modern embedded systems based on multiprocessor systems-on-chip (MPSoC). Tasks of the applications need to be mapped on the MPSoC resources efficiently in order to satisfy their performance constraints. Exploring all the possible mappings, i.e., tasks to resources combinations exhaustively may take days or weeks. Additionally, the exploration is performed at design-time, which 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 (CADSE) 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.

Keywords multiprocessor systems-on-chip, design space exploration, run-time mapping, synchronous dataflow graphs, throughput

Received June 5, 2012; accepted December 5, 2012

E-mail: eleaks@nus.edu.sg
Some part of this research was presented in Proceedings of International Symposium on Parallel Architectures, Algorithms and Programming (PAAP). 2011, 72–76

1 Introduction

Advanced multimedia embedded systems (e.g., smart phones, tablets, PDAs) need to support multiple applications concurrently. For example, a smart phone might be used to view an image using a JPEG decoder and at the same time to listen to music using an MP3 decoder. The increasing performance demands of concurrently running applications are satisfied by relying on multiprocessor systems-on-ship (MPSoC), for example, IBM Cell [1] and NXP Nexperia [2]. The MPSoCs may contain different type of processing elements (PEs) connected by a communication network, where, distinct features of the different type of PEs can be exploited to achieve high performance.

The system users expect that timing (throughput) constraints of all applications running in the system are satisfied. This calls for a predictable timing nature for each of the running application. For an application, the timing property depends on its system resource uses, i.e., tasks to PEs mapping. Time-constrained multimedia applications are modeled using synchronous dataflow graphs (SDFGs) that provide predictability [3,4]. Additionally, techniques to find throughput of an SDFG already exist [5].

For a given set of applications and the underlying MPSoC platform, there is an enormous number of possibilities for mapping the individual application tasks onto the platform PEs. The mapping is accomplished either by design-time DSE [6,7] or run-time mapping strategies [8–10]. 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 to accomplish the job of efficient applications to platform mapping. There has already been few works that use design-time analysis results for run-time management but their analysis results are not optimized from throughput point of view and are applicable only to the analyzed platform [11,12].

The design-time DSE strategies need to find a number of mappings by taking the application and 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, which might take several days. Additionally, existing DSE strategies do not scale well with the number of tasks/PEs and do not always provide the largest throughput mapping as they perform DSE in view of optimizing for the performance metrics such as energy and resource optimization. Further, most of the existing run-time mapping strategies perform all the computations at run-time and the strategies using DSE results are not able to get the optimal mapping.

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 to as a tile that essentially contains a processing engine along with other elements such as memory. The processing engine type defines the tile type. The platform contains different types of tiles such as processor and reconfigurable hardware (RH). 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 design points and produces almost the same best trade-off points as that of the exhaustive DSE. To further accelerate the DSE, we incorporated pruning in the CADSE (PCADSE) where evaluation of the number of trade-off points is further decreased based on a pruning criteria. The quality of the best mappings generated by the CADSE and PCADSE strategies do not differ significantly, while the exploration process is speeded up. The trade-off points are used by a light-weight run-time manager to select the best point depending upon the available tiles in the platform and desired throughput. Some parts of this research are published in [13] and the same is extended for this paper.

Our DSE strategies consider a generic multiprocessor platform that contains tiles depending upon the tasks and their implementation alternatives (e.g., a task can be supported on a number of PE types) provided in the applications. The strategies provide mappings where tasks are distributed on different types of PEs. The considered platform contains tiles separated by a fixed distance from each other, referred to as hop_distance. A real-life 2×2 grid of tiles platform contains a few tiles separated by a hop_distance of 1 and others by 2. The DSE is performed by considering maximum separation between the tiles in the expected target platform. The DSE results are applicable to any platform on which the maximum distance between two tiles is less than or equal to the fixed considered distance and the platform tile types are subset of the tile types considered during DSE. Thus, no additional design-time analysis is needed in case of such different target platforms and the approach becomes analogous to analyze once & run everywhere, which is similar to Java's write-once-run-everywhere capability.

2 Related work

Several DSE strategies providing single mapping for an application have been reported in literature [14–19]. The strategy in [19] can be used iteratively to compute multiple mappings providing memory-performance trade-offs. 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 objective 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 run-time dynamism and allows them to be mapped on any architecture without the need of repetitive analysis.

DSE strategies providing multiple mappings for the application and a given platform have been recently presented [7,12,20]. In [7], DSE is performed in view of optimizing for the resource usage, whereas in [12] and [20], 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 do not scale 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. [21,22] propose DSE strategies that perform exploration in view of optimizing throughput by considering a generic platform, and the exploration follows a pruning strategy.

Some research has been focused on scenario-based DSE. A scenario contains a set of simultaneously active applications and is also referred to as a use-case [23,24]. In order to handle dynamism in number of active applications at run-time, multiple applications mapping scenarios are explored at design-time [25,26]. Such exploration is not scalable as the number of scenarios becomes intractable with the number of applications. Instead, the applications can be mapped one after another to avoid the exploration for a large number of scenarios and our mapping strategy follows the same approach.

At run-time, the applications mapping can be started with or without previously analyzed results. Most of the works reported in literature start mapping without any previous analysis and thus cannot guarantee for strict timing deadlines due to limited available computational resources at run-time [8,27–29]. Some works preprocess the applications at run-time before actual mapping is done in order to facilitate efficient mapping [30,31]. A few strategies use design-time analysis results [11,12,21,32]. In [32], 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 [12] and [11], analysis results do not include mappings satisfying the constraints in case of limited resources, which might force the application to wait until required resources are available. In [21], analysis results include mappings optimized from throughput point of view and for the limited resources case but they are applicable only to homogeneous platforms. Our strategy considers heterogeneous platforms and uses the analyzed results efficiently in order to provide timing guarantees.

3 Problem statement

This section defines the problem that we are heading to solve. First, we describe the hardware MPSoC architecture model and the application model that are used in this work.

3.1 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 and t_4, which are connected by end-to-end connections 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 (e.g., processor P, RH, Accelerator), 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 [33] and Eclipse [34] fit nicely into this platform model.

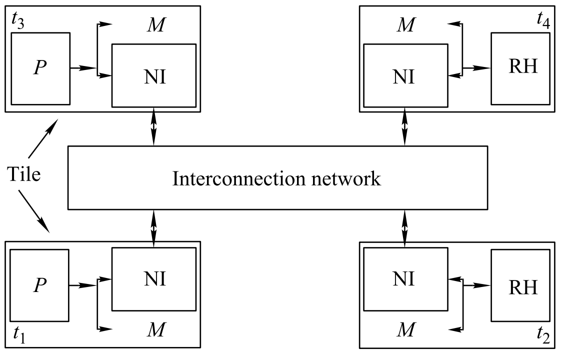

Fig. 1 Multiprocessor platform example

The communication network used in the example platform of Fig. 1 is arranged in a 2-D mesh topology. The *manhattan 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.

3.2 Application model

The application model considers throughput-constrained multimedia applications consisting of multiple tasks. Synchronous dataflow graphs (SDFGs) [3] 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* and *mc* and four edges d_1, d_2, d_3 and d_4. An actor has following attributes: its implementation alternatives (e.g., processor, Accelerator 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 as *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.

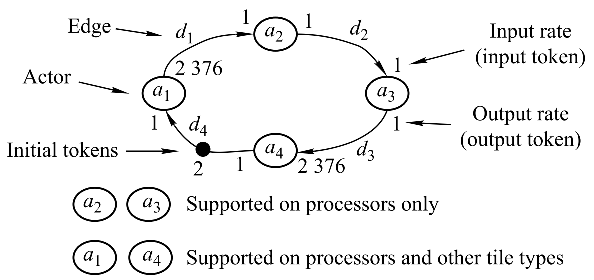
Edge
Actor
Initial tokens
Input rate (input token)
Output rate (output token)
d_1
d_2
d_3
d_4
a_1
a_2
a_3
a_4
1
2 376
2
Supported on processors only
Supported on processors and other tile types

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

3.3 Problem formulation

Existing DSE strategies find mappings while performing optimization for power, resource usage, etc. This might lead to mapping of parallel executing actors on the same tile and thus forcing their execution sequentially, resulting in reduced throughput. A number of mappings are evaluated for each multimedia application to be supported on a hardware platform. The evaluation considers finding different mappings and their throughput. For each mapping, actors are bound to tiles and edges to memory inside tiles or to connections in the platform. The binding is considered valid if memory imposed, allocated input/output connections and allocated incoming/outgoing bandwidth are less than or equal to the maximum available on each tile. 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. A list-scheduler is used to construct the static-order schedules for all the tiles at once by following the approach in [35]. 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 [5].

For each application, exhaustive design space exploration (EDSE) flow (e.g., [11]) evaluates all the possible actors to tiles combinations, i.e., mappings. The flow needs to consider a common platform graph that can evaluate all possible mappings for each application. Here, the applications are modeled such that the implementation alternatives of actors could be a number of tile types. The considered platform contains N tiles of each implementation alternative, where N 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. Thus, it explores all potential mappings providing maximum throughput. After considering a suitable platform, first, all the possible mappings using processor tiles can be evaluated, and then all the possible mappings using Heterogeneous tiles.

3.3.1 Exhaustive exploration of mappings using processor tiles

The exploration of all the possible actors to processor (Proc) tiles mappings 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 Eq. (1). An application with two actors (a_1, a_2) has two unique mappings that is computed from Eq. (2). One mapping contains actors on separate tiles (${}^1C_0$ 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 (${}^1C_1$ 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 Eq. (3). First, actor a_3 is mapped separately, i.e., not combined with others (from the remaining two actors a_1 and a_2, none is chosen to combine with a_3, indicated as ${}^2C_0$) and remaining two actors are mapped by using Eq. (2) ($f_{\text{EDSE}}(2, a_1, a_2)$), providing two unique mappings. Then, from the remaining two actors one actor is chosen to combine with actor a_3 (${}^2C_1$) 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. In the same manner, for an application with four actors (a_1, a_2, a_3, a_4), all the unique mappings are computed from Eq. (4) and we get a total of 15 unique mappings.

The equations can be extended in the similar manner to evaluate all the unique mappings to cater for the applications with larger number of actors. For an application with n actors $(a_1, a_2, \ldots, a_n)$, the mappings can be computed from Eq. (5). 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 $f_{\mathrm{EDSE}}(n-1, a_1, a_2, \ldots, a_{n-1})$ in $f_{\mathrm{EDSE}}(n, a_1, a_2, \ldots, a_n)$.

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

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

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

$$\begin{aligned} &f_{\mathrm{EDSE}}(4, a_1, a_2, a_3, a_4) \\ &\quad = {}^{3}C_0 \times f_{\mathrm{EDSE}}(3, a_1, a_2, a_3) \\ &\qquad + {}^{3}C_1 \times f_{\mathrm{EDSE}}(2, \text{remain\_actors}) \\ &\qquad + {}^{3}C_2 \times f_{\mathrm{EDSE}}(1, \text{remain\_actor}) + {}^{3}C_3. \end{aligned} \tag{4}$$

$$\vdots$$

$$\begin{aligned} &f_{\mathrm{EDSE}}(n, a_1, a_2, \ldots, a_n) \\ &\quad = {}^{(n-1)}C_0 \times f_{\mathrm{EDSE}}(n-1, a_1, a_2, \ldots, a_{n-1}) \\ &\qquad + {}^{(n-1)}C_1 \times f_{\mathrm{EDSE}}(n-2, \text{remain\_actors}) \\ &\qquad + {}^{(n-1)}C_2 \times f_{\mathrm{EDSE}}(n-3, \text{remain\_actors}) \\ &\qquad + {}^{(n-1)}C_{n-2} \times f_{\mathrm{EDSE}}(1, \text{remain\_actor}) + {}^{(n-1)}C_{n-1}. \end{aligned} \tag{5}$$

Mappings using processor tiles computed by EDSE: example application.

The computation process has been applied onto the example application H.263 decoder (see Fig. 2) to demonstrate how the Proc tiles mappings are computed. The application contains four actors *vld*, *iq*, *idct* and *mc*, so we need to compute mappings by using Eq. (4), i.e., $f_{\mathrm{EDSE}}(4, a_1, a_2, a_3, a_4)$. For the demonstration, actors *vld*, *iq*, *idct* and *mc* are considered as a_1, a_2, a_3 and a_4, respectively. Function $f_{\mathrm{EDSE}}(4, a_1, a_2, a_3, a_4)$ needs $f_{\mathrm{EDSE}}(3, a_1, a_2, a_3)$ as pre-computed and $f_{\mathrm{EDSE}}(3, a_1, a_2, a_3)$ needs $f_{\mathrm{EDSE}}(2, a_1, a_2)$ as pre-computed and so on. Thus, we need to proceed with $f_{\mathrm{EDSE}}(1, a_1)$ that maps actor a_1 on a Proc tile. The mappings are computed as follows:

$$f_{\mathrm{EDSE}}(1, a_1) = \begin{bmatrix} a_1 \end{bmatrix}.$$

$$f_{\mathrm{EDSE}}(2, a_1, a_2) = \begin{bmatrix} a_2 & a_1 \\ a_2a_1 & \end{bmatrix}.$$

$$f_{\mathrm{EDSE}}(3, a_1, a_2, a_3) = \begin{bmatrix} a_3 & a_2 & a_1 \\ a_3 & a_2a_1 & \\ a_3a_2 & a_1 & \\ a_3a_1 & a_2 & \\ a_3a_2a_1 & & \end{bmatrix}.$$

$$f_{\mathrm{EDSE}}(4, a_1, a_2, a_3, a_4) = \begin{bmatrix} a_4 & a_3 & a_2 & a_1 \\ a_4 & a_3 & a_2a_1 & \\ a_4 & a_3a_2 & a_1 & \\ a_4 & a_3a_1 & a_2 & \\ a_4 & a_3a_2a_1 & & \\ a_4a_3 & a_2 & a_1 & \\ a_4a_3 & a_2a_1 & & \\ a_4a_2 & a_3 & a_1 & \\ a_4a_2 & a_3a_1 & & \\ a_4a_1 & a_3 & a_2 & \\ a_4a_1 & a_3a_2 & & \\ a_4a_3a_2 & a_1 & & \\ a_4a_3a_1 & a_2 & & \\ a_4a_2a_1 & a_3 & & \\ a_4a_3a_2a_1 & & & \end{bmatrix}.$$

For the four actors of H.263 decoder, we get a total of 15 Proc tiles mappings represented by the above matrix through $f_{\mathrm{EDSE}}(4, a_1, a_2, a_3, a_4)$. Each row denotes a mapping with distribution of actors on Proc tiles in each of the column. 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 that depends upon the placement of actors. Similarly, we get a total of 115 975 Proc tiles mappings for an application with 10 actors. Thus, exhaustive exploration may take days or weeks for applications with large number of actors. The exploration time will increase further when additional mappings that use heterogeneous tiles need to be evaluated, which is described subsequently.

3.3.2 Exhaustive exploration of mappings using heterogeneous tiles

The heterogeneous tiles combination mappings can be evaluated by using the Proc tiles mappings obtained as described earlier. We use Algorithm 1 for evaluating all such mappings.

The algorithm evaluates a number of mappings corresponding to each Proc tile mapping in set *Maps* and adds the evaluated mappings in the same set. For each Proc tile mapping, the actors on each Proc tile are moved to another tile type (implementation alternative) in order to generate a new mapping provided all the actors on the Proc tile can be supported on the other tile type. The actors moving condition avoids the evaluation of mappings using non-supported tile-combinations. The generated mapping with its throughput is added to set Maps. Thus, the final mapping set Maps contains all the mappings evaluated in the EDSE.

Algorithm 1 Heterogeneous tiles Comb. Mappings Evaluation

```
Input: Proc tiles mappings (set Maps)
Output: Heterogeneous tile-combinations combination mappings to be
        added to set Maps
for each Proc tiles mapping $\alpha$ ($\in$ Maps) do
    findHeterogeneousTilesCombMappings($\alpha$, $t_1$);
end
function findHeterogeneousTilesCombMappings(Mapping $\beta$, Tile
startProcTile)
if startProcTile == lastProcTile+1 then
    return;
end
for Tile $i$ = firstProcTile to lastProcTile (in current mapping) do
    for each implementation alternative $\kappa$ (e.g., DSP, ACC, RH tiles) do
        if tile $i$ contains actor(s) and all have their implementation
          alternatives as $\kappa$ then
            Move actor($s$) of tile $i$ to a $\kappa$ having no previous actor to
            generate a new mapping $\alpha$ provided $\kappa$ has required
            resources (memory/area);
            Compute throughput of $\alpha$;
            Add $\alpha$ with its throughput to set Maps;
            findHeterogeneousTilesCombMappings ($\alpha$, $i$+1);
        end
    end
end
```

The number of mappings evaluated by EDSE strategy increases exponentially with the number of actors. Further, the number of mappings increases even more when the implementation alternatives of actors (number of tile types on which the actors can be supported) get increased. The total number of mappings ($\text{nrMaps}_{\text{EDSE}}$) follows Eq. (6), which uses the mappings using different Proc tiles and the number of implementation alternatives (nrTileTypes).

$$\begin{aligned}
&\text{nrMaps}_{\text{EDSE}}\\
&\quad= \text{nrMapsUsing\_1\_ProcTile} \times (\text{nrTileTypes})^1\\
&\qquad+\text{nrMapsUsing\_2\_ProcTiles} \times (\text{nrTileTypes})^2\\
&\qquad+\text{nrMapsUsing\_3\_ProcTiles} \times (\text{nrTileTypes})^3\\
&\qquad+\cdots\\
&\qquad+\text{nrMapsUsing\_}n\text{\_ProcTiles} \times (\text{nrTileTypes})^n \qquad (6)
\end{aligned}$$

For the example H.263 decoder application, we get a total of 94 mappings (including 15 Proc tiles mappings), which contain tasks distributed on Proc or Proc/RH or RH tiles when Proc and RH tiles are considered as the implementation alternatives. We have not shown tasks to tiles distribution for different mappings as the number of mappings is large, which requires larger space to show them. Similarly, we get a total of 4 412 798 mappings for an application with 10 actors when each actor can be supported on a Proc and RH tile. The number of mappings increases further with the number of supported tile types. Evaluation of such a large number of mappings (for large size applications) is not feasible within a reasonable time. Therefore, DSE strategies that should discard evaluation of inefficient mappings (providing less throughput) need to be developed in order to accelerate the exploration process. Next, we discuss our proposed DSE strategy for faster and efficient exploration.

4 Proposed design space exploration methodologies

This section introduces our proposed DSE methodologies for exploring multiple mappings.

4.1 Communication-aware design space exploration

The CADSE strategy performs exploration in communication-aware manner, i.e., by looking at the directly communicating (connected) actors. The exploration 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 flow first considers a common platform graph that can evaluate all possible mappings for each application. The considered platform contains the same number of tiles for each implementation alternative and this number is the maximum value of number of actors in an application amongst all the applications, i.e., max_nA. This platform is capable of exploiting all actors to tiles combinations, i.e., mappings, for each application.

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

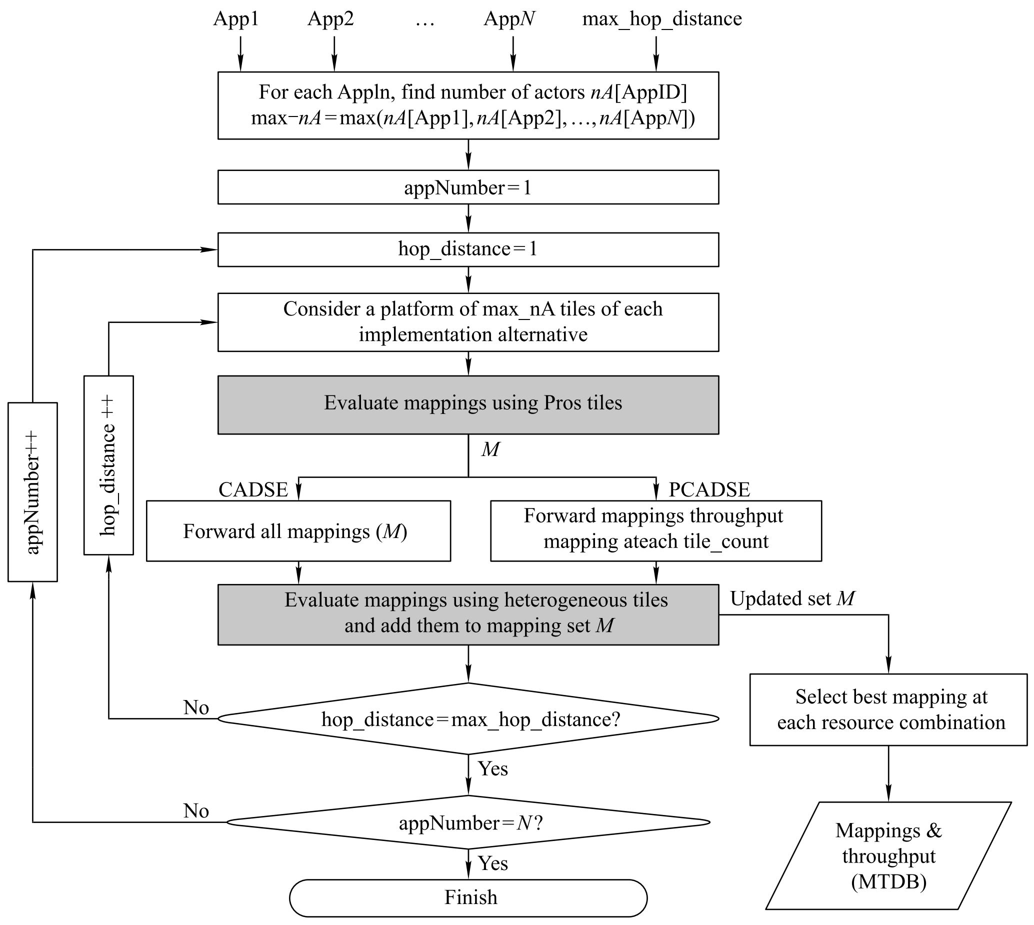
App1
App2
…
AppN
max_hop_distance
For each Appln, find number of actors nA[AppID]
max−nA = max(nA[App1], nA[App2], …, nA[AppN])
appNumber = 1
hop_distance = 1
Consider a platform of max_nA tiles of each implementation alternative
Evaluate mappings using Pros tiles
M
CADSE
PCADSE
Forward all mappings (M)
Forward mappings throughput mapping ateach tile_count
Evaluate mappings using heterogeneous tiles and add them to mapping set M
Updated set M
Select best mapping at each resource combination
Mappings & throughput (MTDB)
hop_distance = max_hop_distance?
No
Yes
appNumber = N?
No
Yes
Finish
hop_distance ++
appNumber++

Fig. 3 Communication-aware design space exploration flow

for connections, till hop_distance reaches to max_hop_ distance (one 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. By opting a higher value of max_hop_distance, the DSE flow evaluates larger number of mappings, which needs more evaluation time but the mappings can then be applied to larger platforms. For example, evaluated mappings with max_hop_distance value of 8 are applicable to any platform where maximum separation between two tiles is less than or equal to 8 hops.

Varying hop_distance consideration provides mappings where each edge of the application is mapped to a connection 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. After considering the platform, first, the mappings using Proc tiles and then mappings using Heterogeneous tiles (highlighted in Fig. 3) are evaluated as described subsequently.

4.1.1 Communication-aware exploration of mappings using processor tiles

In CADSE, for an application with n actors $(a_1, a_2, \ldots, a_n)$, the Proc tiles mappings are evaluated from Eq. (7), which requires $n-1, n-2, \ldots, 2, 1$ actors mappings in advance as in the EDSE. These mappings can be calculated by putting different values of n in the Eq. (7). This equation differs from Eq. (5) while choosing actors to be combined with actor a_n on the same tile. The chosen actors and actor a_n are checked whether they are connected (conn). If they are found to be connected then the chosen actors are mapped with actor a_n on the same tile. For example, ${}^{(n-1)}C_{2-\mathrm{conn}}$ in Eq. (7) specifies that the chosen (C) two actors and actor a_n are connected that qualifies them to be mapped on the same tile.

$$\begin{aligned}
&f_{\mathrm{CADSE}}(n, a_1, a_2, \ldots, a_n)\\
&\quad = {}^{(n-1)}\,C_{0-\mathrm{conn}} \times f_{\mathrm{CADSE}}(n-1, a_1, a_2, \ldots, a_{n-1})\\
&\qquad + {}^{(n-1)}C_{1-\mathrm{conn}} \times f_{\mathrm{CADSE}}(n-2, \mathrm{remain\_actors})\\
&\qquad + {}^{(n-1)}C_{2-\mathrm{conn}} \times f_{\mathrm{CADSE}}(n-3, \mathrm{remain\_actors})\\
&\qquad + \cdots
\end{aligned}$$

$$+^{(n-1)}C_{(n-2)-\text{conn}} \times f_{\text{CADSE}}(1, \text{remain\_actor})$$
$$+^{(n-1)}C_{(n-1)-\text{conn}}. \qquad (7)$$

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

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
```

An example application of mappings using processor tiles computed by CADSE is giren below.

The CADSE strategy has been applied to the example application H.263 decoder (see Fig. 2). The mappings using Proc tiles are computed by Eq. (7) by putting n equal to four. Application actors *vld*, *iq*, *idct*, and *mc* are considered as a_1, a_2, a_3, and a_4, respectively. The computed mappings are represented by function $f_{\text{CADSE}}(4, a_1, a_2, a_3, a_4)$ as follows:

$$f_{\text{CADSE}}(4, a_1, a_2, a_3, a_4) = \begin{bmatrix} a_4 & a_3 & a_2 & a_1 \\ a_4 & a_3 & a_2a_1 & \\ a_4 & a_3a_2 & a_1 & \\ a_4 & a_3a_2a_1 & & \\ a_4a_3 & a_2 & a_1 & \\ a_4a_3 & a_2a_1 & & \\ a_4a_1 & a_3 & a_2 & \\ a_4a_1 & a_3a_2 & & \\ a_4a_3a_2 & a_1 & & \\ a_4a_3a_1 & a_2 & & \\ a_4a_2a_1 & a_3 & & \\ a_4a_3a_2a_1 & & & \end{bmatrix}.$$

The CADSE strategy discards the evaluation of mappings where a tile contains non-connected actors and thus evaluates less number of mappings as compared to the EDSE. For the four actors of H.263 decoder, we get a total of 12 Proc tiles mappings represented by the matrix $f_{\text{CADSE}}(4, a_1, a_2, a_3, a_4)$. This strategy takes advantage of the fact that mapping only connected actors on the same tile will lead to reduced communication overhead between the actors. However, if the connected actors are executing in parallel then we will be forcing their execution sequentially. So, we will be having gain by reducing communication overhead and loss by reducing the potential parallelism in execution. If the gain is greater than the loss, then we always get the same quality of best mappings by CADSE and EDSE.

4.1.2 Communication-aware exploration of mappings using heterogeneous tiles

The heterogeneous tiles combination mappings are evaluated by using the Proc tiles mappings obtained by the communication-aware exploration as described earlier. The same algorithm as of EDSE, i.e., Algorithm 1 is used for evaluating all such mappings by providing the Proc tiles mappings as input. For the example H.263 decoder application, the CADSE explores a total of 54 mappings (including 12 Proc tiles mappings), whereas EDSE explores a total of 94 mappings with two types of implementation alternatives as described earlier. The evaluated best mapping at each resource combination by CADSE has almost the same quality (throughput) as that of the EDSE. For applications with larger number of actors, the difference in the number of evaluated mappings by EDSE and CADSE increases and thus the difference in the exploration time. The reduced number of explored mappings by CADSE facilitates for faster exploration and reduces the exploration time significantly when compared to EDSE.

4.1.3 Selecting best mapping at each resource combination

At each possible processing tile-combination, we get a number of mappings. This step of the exploration flow selects the maximum throughput mapping at each resource combination and stores it into the mappings & throughput database (MTDB) (see Fig. 3). For the example H.263 decoder application, at 2Proc & 2RH tiles resource combination, we get a total of six mappings at each hop_distance value. This step of the flow filters out the maximum throughput mapping out of the six mappings and stores it in the MTDB. Similar process is carried out for each resource combination.

These stored mappings are kept to be used at run-time. For mapping an application at run-time, out of all the stored mappings for the application, 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.

4.2 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 best (maximum throughput) mapping at each Proc tile count is passed to evaluate heterogeneous tiles combination mappings, whereas, all the Proc tiles mappings are passed in CADSE. Proc tile count for a mapping is defined as the number of used Proc tiles in the mapping. For the H.263 decoder (see Fig. 2), the CADSE evaluates three mappings using three Proc tiles, and only the best mapping out of the three mappings is passed to evaluate heterogeneous tiles combination mappings. A total of only 34 mappings are explored by PCADSE when considering two implementation alternatives of each application.

The pruning consideration facilitates for speeded exploration over the CADSE and provides almost the same quality (throughput) of best mapping at each resource combination. This strategy assumes that by starting with the best Proc tile mapping to evaluate heterogeneous tiles combination mappings, we should get the best mapping at heterogeneous tiles combinations as well. The quality of the best mapping by the PCADSE might be a bit lower as compared to CADSE.

4.2.1 Run-time mapping

The DSE flow stores the maximum throughput mapping at each resource combination for the applications which are expected to be mapped on a platform at run-time. The stored mappings are used at run-time in order to accelerate the run-time mapping process. A run-time platform manager (RTPM) handles the mapping process by assigning the platform resources to the required applications one after another, i.e., after accomplishing mapping process for one application it goes on to map the next application. In order to map an application, the RTPM takes the application, its desired throughput, platform with updated resources status and the mapping storage MTDB as input and selects a throughput satisfying mapping from the MTDB. The selected mapping uses minimum possible resources (number of tiles) and the platform is configured based on the actors to tiles allocations of the selected mapping provided the platform has sufficient available resources.

4.2.2 Resource sharing by applications

The applications for which mappings are stored in the database use 100% of the available time wheel at each used tile. Thus, completely free tiles are chosen at run-time. For generalizing the approach when each tile is shared by multiple applications, the same design-time analysis flow can be applied for analysis at different reserved time slices and then the design points can be used at run-time depending upon the available time slices at available tiles, and required throughput. For each application, we can perform design-time analysis to store mappings and their throughput when reserved time slice is 25%, 50%, 75%, and 100% of the available time wheel at each tile. This provides us mappings with their throughput where all the used tiles are occupied 25%, 50%, 75%, or 100% of the time. To map a throughput-constrained application at run-time, the RTPM need to provide available tile slices on the available tiles. The tile having minimum available time slice determines the scanning into MTDB. First the MTDB at reserved time slice of 25% is scanned for the available tiles, then storage at reserved time slice of 50% and so on. For example, if minimum available time slice is 75%, then storage at reserved time slice of 25% will be scanned first, followed by the storage at reserved time slice of 50% and then of 75%. The storage at reserved time slice of more than 75%, i.e., 100% will not be scanned as this exceeds the availability at some selected tiles. The scanning stops as soon as a mapping satisfying the throughput-constraint is found. The complexity of the run-time strategy to find a mapping becomes higher in this case.

5 Performance evaluation

This section evaluates our DSE and run-time mapping methodologies. The methodologies have been implemented as an extension to the publicly available tool set SDF^3 [4]. As a benchmark to evaluate run-time and quality of the methodologies, we have considered two scenarios of applications: (i) 100 random applications modeled as SDFGs with 4, 5, 6, and 7 actors having different implementation alternatives, and (ii) models of multimedia applications H.263 decoder (4 actors), H.263 encoder (5 actors), JPEG decoder (6 actors) and JPEG encoder (4 actors) to perform a case study for real-life applications, where the actors implementation alternatives are specified in the application models. The experiments have been performed on a Core 2 Duo processor at 3.16 GHz.

The same platform graph is considered to evaluate the different DSE methodologies in order to have a fair comparison amongst the methodologies. The platform tiles are considered based on the application containing maximum number of actors as described in the earlier section. We have adopted a tile-based architecture but any type of architecture can be modeled based on the known latencies between the tiles as discussed earlier.

5.1 Design space exploration

Table 1 shows the number of mappings evaluated by the EDSE (Eq. (6)) as the number of actors (nrActors) increases at different number of available implementation alternatives (nrTileTypes) for each of the actor. For *n* actors having nrTileTypes implementation alternatives, the total number of mappings follows Bell numbers: ways of placing *n* labeled balls into *n* unlabeled (but nrTileTypes-colored) boxes [36]. The number of mappings in Table 1 is for a fixed hop_distance (e.g., 1) and the same number of mappings needs to be evaluated for each hop_distance.

Table 1 Number of mappings by EDSE

nrActors	nrTileTypes		
	1	2	3
1	1	2	3
2	2	6	12
3	5	22	57
4	15	94	309
5	52	454	1 866
6	203	2 430	12 351
7	877	14 214	88 563
8	4 140	89 918	681 870
9	21 147	610 162	5 597 643
10	115 975	4 412 758	48 718 569

It can be observed from Table 1 that the number of mappings are going really high and the number will increase further in case of more nrTileTypes. Thus, the exploration will take a very long time for larger value of nrActors and nrTileTypes; in some cases it 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 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. The three methodologies EDSE, CADSE and PCADSE are applied to the scenario (i) to capture the best mapping at each resource combination for all the 100 applications. Figure 4 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 provides the same best mappings for more than 90% of the applications and the PCADSE for more than 80% of the applications. 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. Additionally, for remaining applications where we do not get the same quality of mappings by CADSE and PCADSE, the quality varies only by 10% as compared to that of the EDSE. So, in case of 10% relaxed throughput constraint at run-time, the mappings generated for all the applications by CADSE and PCADSE will be acceptable.

Figure 5 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 and for the same number of

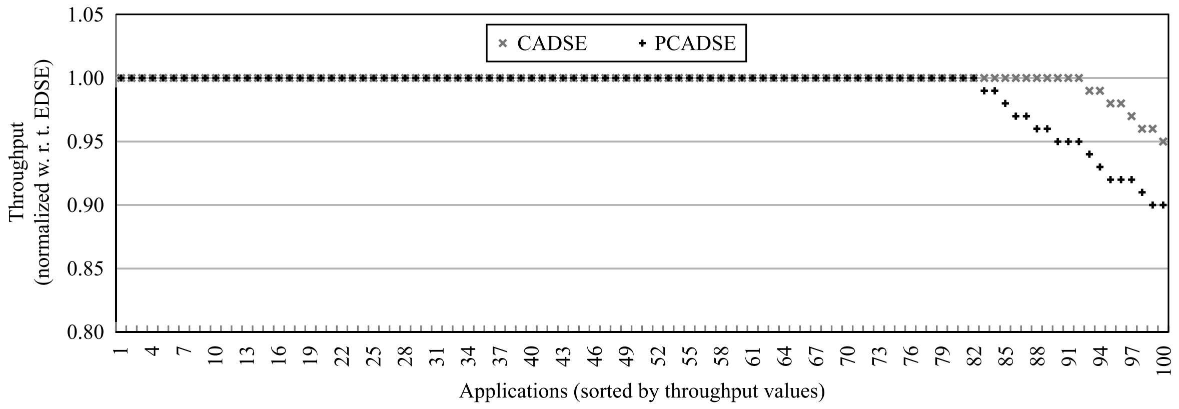

Fig. 4 Quality of mappings by CADSE and PCADSE over the EDSE in the first evaluated scenario

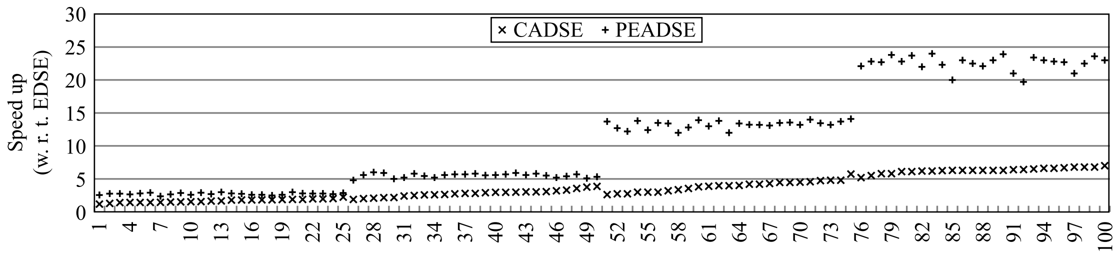

Fig. 5 Speed up obtained by CADSE and PCADSE over the EDSE in the first evaluated scenario

actors the speed up is sorted on CADSE. 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 for evaluating heterogeneous tiles combinations mappings. On an average, the CADSE and PCADSE is faster by 3.7× and 11×, respectively when compared to EDSE. Thus, we get speeded exploration providing almost the same quality of mappings.

5.2 Case study: multimedia applications DSE

We have performed a case study on models of real-life multimedia applications (scenario (ii)). All the DSE strategies have been applied to the applications in scenario (ii). Table 2 shows number of evaluated mappings at different resource combinations for H.263 encoder (5 actors) and H.263 decoder (4 actors) when different DSE methodologies are employed at a fixed value of hop_distance. The same number of mappings are evaluated at each hop_distance value. All the actors have been assumed to have their implementation alternatives as Proc and RH tiles, whereas higher number of implementation alternatives can be considered. The number of mappings for H.263 decoder at different resource combinations using total 5 tiles is zero as there would not be any such mapping where 4 actors will be distributed on 5 tiles. The last row shows total number of mappings by different methodologies. It can be observed that total number of mappings get reduced significantly when CADSE is employed. The number of mappings is further reduced when PCADSE is employed. Thus, we get speed up when CADSE and PCADSE strategies are employed. It has been observed that the best (maximum throughput) mapping at each resource combination by all the strategies is the same for H.263 encoder/decoder. Mappings of JPEG encoder (4 actors) shows similar behavior as of H.263 decoder.

Table 2 Number of mappings by different DSEs

Resource	H.263 encoder DSE			H.263 decoder DSE		
combinations	EDSE	CADSE	PCADSE	EDSE	CADSE	PCADSE
5Proc	1	1	1	0	0	0
4Proc & 1RH	5	5	5	0	0	0
3Proc & 2RH	10	10	10	0	0	0
2Proc & 3RH	10	10	10	0	0	0
1Proc & 4RH	5	5	5	0	0	0
5RH	1	1	1	0	0	0
4Proc	10	5	5	1	1	1
3Proc & 1RH	40	20	4	4	4	4
2Proc & 2RH	60	30	6	6	6	6
1Proc & 3RH	40	20	4	4	4	4
4RH	10	5	1	1	1	1
3Proc	25	10	10	6	3	3
2Proc & 1RH	75	30	3	18	9	3
1Proc & 2RH	75	30	3	18	9	3
3RH	25	10	1	1	3	1
2Proc	15	7	7	7	3	3
1Proc & 1RH	30	14	2	14	6	2
2RH	15	7	1	7	3	1
1Proc	1	1	1	1	1	1
1RH	1	1	1	1	1	1
Total mappings	454	222	81	94	54	34

For JPEG decoder (6 actors), the EDSE, CADSE and PCADSE strategies evaluate a total of 2 430, 718, and 178 mappings, respectively. We observed that the best mapping at each resource combination by EDSE and CADSE is the same. However, PCADSE provides mappings having lower quality (throughput) at resource combinations 3Proc & 1RH, 2Proc & 2RH, 2Proc & 1RH, and 1Proc & 1RH, i.e., the best mappings get missed.

5.3 Run-time mapping

The DSE methodologies store the best mappings at each resource combination at varying hop_distance values (referred to 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 RTPM selects the best mapping depending upon the available resources and hop_distance between them. The results obtained from the existing run-time strategies that start the application mapping without any previous analysis (perform required analysis at run-time) are compared with our run-time strategy (using the DSE results). The existing run-time strategies take time first in finding a mapping and then in computing throughput for the same, whereas our strategy just selects the best mapping from the mapping database MTDB. Throughput computation for a mapping takes much more time than in finding the mapping and thus strategies performing analysis at run-time are not efficient. In contrast, the total time in our strategy consists of only the selection and placement only. In placement, the actors are configured on the platform tiles based on the selected mapping. Figure 6 shows the time required (in milliseconds) to map throughput-constrained multimedia applications on a 4×4 MPSoC platform when nearest neighbor (NN) proposed in [28], communication-aware nearest neighbor (CNN) proposed in [10] and our run-time mapping strategy is employed. On average, our run-time strategy is faster by about 90% when compared to CNN.

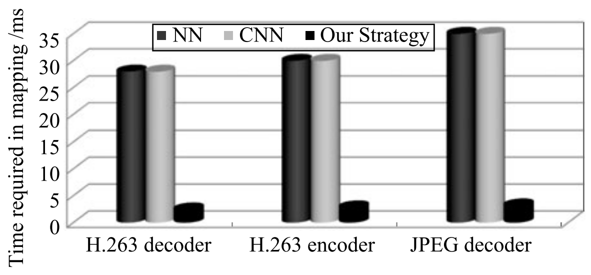

Fig. 6 Time required/ms to map the applications by different strategies

In case of limited memory to store MTDB database, all mappings for all the applications might not get stored. In such cases, we can store only a few mappings for each application in the limited memory. The mappings having maximum throughput need to be stored to satisfy varying throughput requirements at run-time. If the limited memory cannot store even a single mapping for an application, the mapping process cannot be accelerated for the application as the mapping and its throughput need to be computed at run-time.

5.4 Real-time through guarantee

Table 3 shows the DSE results for the H.263 decoder (4 actors) at max_hop_distance of 6. At each hop, the best mapping's throughput at different resource combinations (Proc/RH tiles combinations) using a total of four tiles is shown. At other resource combinations using different number of tiles, similar results are obtained. It can be observed that at each shown resource combination, the throughput (quality) of the best mappings from hop_1 to hop_6 does not vary much. At 4 Proc tiles, the best mappings throughput at hop_1 and hop_6 differ only by 0.6%. The throughput differs by almost the same percentage at all other resource combinations. Thus, the quality of mappings does not change much at higher hops and we can store the best mappings only for the maximum hop in order to reduce memory required to store the mappings. This also reduces overhead of the run-time manager as it has to select from a relatively smaller set of mappings.

Our DSE strategies evaluate 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. At run-time, for finding a throughput satisfying mapping from the explored mappings at design-time, one needs to look for a mapping containing tiles separated by a hop_distance of maximum possible hop between the available tiles. If the found mapping satisfies the throughput constraint then mapping the actors on the available tiles will satisfy the constraint for sure as latency of some connections will be smaller as compared to considered during DSE. So, after mapping, we never get worse throughput than the stored one, making the results suitable to use in real-time applications.

6 Conclusions

This paper presents DSE strategies for supporting efficient

Table 3 DSE results for H.263 decoder at varying hop_distance

Tile count	Resource combination	Best mappings throughput (10^{-10}/time-unit)					
		hop_1	hop_2	hop_3	hop_4	hop_5	hop_6
4	4Proc	28 655.3	28 616.8	28 578.4	28 540.0	28 501.8	28 463.7
	3Proc & 1RH	29 170.0	29 130.1	29 090.2	29 050.5	29 010.9	28 971.4
	2Proc & 2RH	29 612.5	29 571.4	29 530.3	29 489.4	29 448.6	29 407.9
	1Proc & 3RH	29 612.5	29 571.4	29 530.3	29 489.4	29 448.6	29 407.9
	4RH	29 612.5	29 571.4	29 530.3	29 489.4	29 448.6	29 407.9

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, facilitating for faster run-time mapping. Three DSE strategies have been explained. One strategy performs EDSE and produces the best quality of mapping at each resource combination, whereas, this strategy has worst run-time. Next, a CADSE strategy is presented to perform the exploration in communication-aware manner in order to reduce the total number of mappings to be evaluated. The communication-aware consideration reduces the exploration time and provides almost the same quality of mappings. To further reduce the exploration time, a pruning criteria has been incorporated in the CADSE, which provides a bit of degraded quality of mappings. The DSE strategies have been applied on models of real-life multimedia applications to show their real-time applicability. 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.

Acknowledgements The authors would like to thank the reviewers for their feedback and suggestions. We also wish to mention that this work is partly supported by Singapore Ministry of Education Academic Research Fund Tier 1 (R-263-000-655-133) and National Natural Science Foundation of China (NSFC) (Grant No. 61173032).

References

1. Kistler M, Perrone M, Petrini F. Cell multiprocessor communication network: built for speed. IEEE Micro, 2006, 26(3): 10–23
2. Kim M, Banerjee S, Dutt N, Venkatasubramanian N. Energy-aware cosynthesis of real-time multimedia applications on mpsocs using heterogeneous scheduling policies. ACM Transactions on Embedded Computing Systems, 2008, 7(2): 1–19
3. Lee E, Messerschmitt D. Static scheduling of synchronous data flow programs for digital signal processing. IEEE Transactions on Computers, 1987, 100(1): 24–35
4. Stuijk S, Geilen M, Basten T. SDF^3: SDF for free. In: Proceedings of the 6th International Conference on Application of Concurrency to System Design. 2006, 276–278
5. Ghamarian A H, Geilen M C W, Stuijk S, Basten T, Moonen A J M, Bekooij M J G, Theelen B D, Mousavi M R. Throughput analysis of synchronous data flow graphs. In: Proceedings of the 6th International Conference on Application of Concurrency to System Design. 2006, 25–36
6. Ascia G, Catania V, Di Nuovo A, Palesi M, Patti D. Efficient design space exploration for application specific systems-on-a-chip. Journal of Systems Architecture, 2007, 53(10): 733–750
7. Stuijk S, Geilen M, Basten T. A predictable multiprocessor design flow for streaming applications with dynamic behaviour. In: Proceedings of the 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools. 2010, 548–555
8. Nollet V, Avasare P, Eeckhaut H, Verkest D, Corporaal H. Run-time management of a MPSoC containing FPGA fabric tiles. IEEE Transactions on Very Large Scale Integration Systems, 2008, 16(1): 24–33
9. Singh A K, Jigang W, Kumar A, Srikanthan T. Run-time mapping of multiple communicating tasks on MPSoC platforms. Procedia Computer Science, 2010, 1(1): 1019–1026
10. Singh A K, Srikanthan T, Kumar A, Jigang W. Communication-aware heuristics for run-time task mapping on NoC-based MPSoC platforms. Journal of Systems Architecture, 2010, 56(7): 242–255
11. Yang P, Marchal P, Wong C, Himpe S, Catthoor F, David P, Vounckx J, Lauwereins R. Managing dynamic concurrent tasks in embedded real-time multimedia systems. In: Proceedings of the 15th International Symposium on System Synthesis. 2002, 112–119
12. Ykman-Couvreur C, Avasare P, Mariani G, Palermo G, Silvano C, Zaccaria V. Linking run-time resource management of embedded multicore platforms with automated design-time exploration. IET Computers Digital Techniques, 2011, 5(2): 123 –135
13. Singh A K, Kumar A, Jigang W, Srikanthan T. Communication-aware design space exploration for efficient run-time mpsoc management. In: Proceedings of the 4th International Symposium on Parallel Architectures, Algorithms and Programming. 2011, 72–76
14. Moreira O, Valente F, Bekooij M. Scheduling multiple independent hard-real-time jobs on a heterogeneous multiprocessor. In: Proceedings of the 7th ACM & IEEE International Conference on Embedded software. 2007, 57–66
15. Bonfietti A, Lombardi M, Milano M, Benini L. Throughput constraint for synchronous data flow graphs. In: Proceedings of the 6th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. 2009, 26–40
16. Ahn Y, Han K, Lee G, Song H, Yoo J, Choi K, Feng X. SoCDAL: system-on-chip design accelerator. ACM Transactions on Design Automation of Electronic Systems, 2008, 13(1): 1–38
17. Keinert J, Schlichter T, Falk J, Gladigau J, Haubelt C, Teich J, Meredith M. Systemcodesigner — automatic ESL synthesis approach by design space exploration and behavioral synthesis for streaming applications. ACM Transactions on Design Automation of Electronic Systems, 2009, 14(1): 1–23
18. Stuijk S, Basten T, Geilen M, Corporaal H. Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs. In: Proceedings of the 44th annual Design Automation Conference. 2007, 777–782
19. Castrillon J, Tretter A, Leupers R, Ascheid G. Communication-aware mapping of KPN applications onto heterogeneous MPSoCs. In: Proceedings of the 49th Annual Design Automation Conference. 2012, 1266–1271
20. Mariani G, Avasare P, Vanmeerbeeck G, Ykman-Couvreur C, Palermo G, Silvano C, Zaccaria V. An industrial design space exploration framework for supporting run-time resource management on multi-core systems. In: Proceedings of the 2010 Conference on Design, Automation and Test in Europe. 2010, 196–201
21. Singh A, Kumar A, Srikanthan T. A hybrid strategy for mapping mul-

tiple throughput-constrained applications on MPSoCs. In: Proceedings of the 14th International Conference on Compilers, Architectures and Synthesis for Embedded Systems. 2011, 175–184

22. Singh A K, Kumar A, Srikanthan T. Accelerating throughput-aware runtime mapping for heterogeneous MPSoCs. ACM Transactions on Design Automation of Electronic Systems. To appear
23. Kumar A, Fernando S, Ha Y, Mesman B, Corporaal H. Multiprocessor systems synthesis for multiple use-cases of multiple applications on FPGA. ACM Transactions on Design Automation of Electronic Systems, 2008, 13(3): 40–66
24. Benini L, Bertozzi D, Milano M. Resource management policy handling multiple use-cases in MPSoC platforms using constraint programming. In: Proceedings of the 24th International Conference on Logic Programming. 2008, 470–484
25. Stralen v P, Pimentel A. Scenario-based design space exploration of MPSoCs. In: Proceedings of the 2010 IEEE International Conference on Computer Design. 2010, 305 –312
26. Palermo G, Silvano C, Zaccaria V. Robust optimization of SoC architectures: a multi-scenario approach. In: Proceedings of the 2008 IEEE/ACM/IFIP Workshop on Embedded Systems for Real-Time Multimedia. 2008, 7 –12
27. Singh A, Jigang W, Prakash A, Srikanthan T. Mapping algorithms for noc-based heterogeneous mpsoc platforms. In: Proceedings of the 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools. 2009, 133–140
28. Carvalho E, Moraes F. Congestion-aware task mapping in heterogeneous MPSoCs. In: Proceedings of the 2008 International Symposium on System-on-Chip. 2008, 1–4
29. Ykman-Couvreur C, Nollet V, Catthoor F, Corporaal H. Fast multidimension multi-choice knapsack heuristic for MP-SoC run-time management. In: Proceedings of the 2006 International Symposium on System-on-Chip. 2006, 1–4
30. Kaushik S, Singh A K, Srikanthan T. Computation and communication aware run-time mapping for NoC-based MPSoC platforms. In: Proceedings of the 2011 IEEE International SOC Conference. 2011, 185–190
31. Kaushik S, Singh A K, Jigang W, Srikanthan T. Run-time computation and communication aware mapping heuristic for noc-based heterogeneous mpsoc platforms. In: Proceedings of the 4th International Symposium on Parallel Architectures, Algorithms and Programming. 2011, 203 –207
32. Schranzhofer A, Chen J J, Thiele L. Dynamic power-aware mapping of applications onto heterogeneous MPSoC platforms. IEEE Transactions on Industrial Informatics, 2010, 6(4): 692 –707
33. Paulin P G, Pilkington C, Bensoudane E, Langevin M, Lyonnard D. Application of a multi-processor SoC platform to high-speed packet forwarding. In: Proceedings of the 2004 Conference on Design, Automation and Test in Europe. 2004, 58–63
34. Rutten M, Van Eijndhoven J, Jaspers E, Van Der Wolf P, Gangwal O, Timmer A, Pol E. A heterogeneous multiprocessor architecture for flexible media processing. Design & Test of Computers, IEEE, 2002, 19(4): 39–50
35. Murthy P K. Scheduling techniques for synchronous and multidimensional synchronous dataflow. PhD thesis, EECS Department, University of California, Berkeley, 1996
36. Encyclopedia of integer sequences. http://oeis.org/

Amit Kumar Singh received the BS. in Electronics Engineering from Indian School of Mines, Dhanbad, India, in 2006. Thereafter, he worked with HCL Technologies, India for a year and half. He joined Nanyang Technological University (NTU), Singapore, in 2008 and worked at Centre for High Performance Embedded Systems (CHiPES), School of Computer Engineering, NTU, Singapore as a research student towards the completion of his PhD till January 2012. Since February 2012, he has been working with the Department of Electrical and Computer Engineering, NUS as a post doctoral researcher. His research interests include network-on-chip (NoC) based multiprocessor systems-on-chip (MPSoC), design space exploration (DSE), and run-time mapping techniques for MPSoC. He has published over 15 papers in leading related international journals/conferences.

Akash Kumar received the BS in Computer Engineering from the National University of Singapore (NUS), Singapore, in 2002. He received the joint Master of Technological Design degree in embedded systems from NUS and the Eindhoven University of Technology (TUe), Eindhoven, The Netherlands, in 2004, and received the joint PhD in Electrical Engineering in the area of embedded systems from TUe and NUS, in 2009. Since 2009, he has been with the Department of Electrical and Computer Engineering, NUS. Currently, he is an assistant professor in the department. His research interests include analysis, architectures, design methodologies, and resource management of embedded multiprocessor systems. He has published over 40 papers in leading international electronic design automation journals and conferences.

Jigang Wu received the BS from Lanzhou University, China in 1983 and PhD from University of Science and Technology of China (USTC) in 2000. He was with the Department of Computer Science of Lanzhou University, China from 1983 to 1993, as an assistant professor, and with the Department of Computer Science and Engineering of Yantai University, China from 1993 to 2000, as an associate professor. He was with the Center for High Performance Embedded Systems, School of Computer Engineering, Nanyang Technological University, Singapore from 2000 to 2009, as a postdoctoral research fellow. He has published more than 100 technical papers including journals in IEEE Transactions, IEE Proceedings and other reputed international journals. His research interests include in reconfigurable VLSI design, hardware/software co-design, parallel computing, and combinatorial search.

Thambipillai Srikanthan joined Nanyang Technological University (NTU), Singapore in 1991. He holds a full professor and joint appointment as Director of a 100 strong Centre for High Performance Embedded Systems (CHiPES). He founded CHiPES in 1998 and elevated it to a university level research Centre in February 2000. His research interests include design methodologies for complex embedded systems, architectural translations of compute intensive algorithms, computer arithmetic and high-speed techniques for image processing and dynamic routing. He has published more than 250 technical papers including 60 journals in IEEE Transactions, IEE Proceedings and other reputed international journals. He was awarded the Public Administration Medal (Bronze) on 2006 National Day in recognition of his contributions to education in Singapore.