Run-time Mapping Algorithm for Dynamic Workloads using Process Merging Transformations

Sima Sinaei
Electrical and Computer Department
University of Tehran
Tehran, Iran
simasinaei@ut.ac.ir

Omid Fatemi
Electrical and Computer Department
University of Tehran
Tehran, Iran
omid@fatemi.net

Andy D. Pimentel
Informatics Institute
University of Amsterdam
The Netherlands
a.d.pimentel@uva.nl

***Abstract*— Exploration of task mappings has an important role to achieve high performance in heterogeneous multi-processor system-on-chip (MPSoC) platforms. The application workloads in modern MPSoC-based embedded systems are becoming increasingly dynamic. Different applications concurrently execute and contend for resources in such systems. In this paper, a run-time algorithm is proposed to analytically evaluate the system throughput of to-be-executed applications (modelled as Kahn Process Networks, KPNs) in order to quickly determine a proper resource binding for these applications. Merging transformations on the KPNs are applied to capture the cases in which the number of processes in the KPN is larger than the number of available processing resources, thereby modeling the effects of binding multiple processes to a single processor. We evaluated our algorithm using a heterogeneous MPSoC system with several applications. Our experimental results revealed that during runtime, the performance of selected mapping with regard to available resources is close to the optimal performance obtained by exhaustive search and simulation. Therefore, the results clearly confirm that our algorithm is effective.**

Keywords—embedded systems, multiprocessor-systems-on-chip (MPSoCs), run-time mapping, design space exploration of heterogeneous MPSoCs, process merging, performance evaluation

I. INTRODUCTION

Application mapping on Multiprocessor-Systems-on-Chip (MPSoCs) has been identified as one of the most important problems in embedded systems design [1][2][3]. Before starting the mapping of applications on MPSoC platforms, first the applications need to be parallelized and synchronization and inter-task communication between the parallelized tasks need to be added. This job can be furnished by state-of-the-art application parallelization tools [4][5]. The parallel tasks can be executed on different platform resources concurrently in order to accelerate the application execution. Then, the parallelized code should be optimized for the given target platform.

For heterogeneous MPSoC systems, the mapping of tasks from applications involves the assignment and ordering of application tasks to processors and binding communications between tasks to memories in the system in view of some optimization criteria like reducing energy consumption, improving compute performance etc. The optimization is necessary to satisfy performance constraints of the applications.

Mapping applications' tasks on MPSoC platform resources can be accomplished at either design-time (static) or run-time (dynamic). Design-time mapping techniques consider a predefined set of applications with known computation and communication behavior and a static platform. Therefore, they are not suitable for dynamic workloads in which, e.g., new applications may appear (i.e., start their execution) on the platform at run-time. Dynamic (run-time) mapping techniques are required for scenarios where application tasks need to be loaded into the platform at run-time. After task mapping, if the user requirement is changed or a new application has entered the system, task migration can be used to revise placement of some of the already executing tasks.

In addition to the capability of run-time (dynamic) mapping techniques to handle dynamic workload scenarios, they also offer a number of additional advantages [6]:

- Adaptability to the available resources.
- Ability to enable unforeseeable upgrades.
- Ability to avoid defective parts of a SoC.

When the mapped applications start execution, the mapping of one or more running applications needs to be reconsidered in case of following events [6]:

- When a new application enters the system and it needs resources from the already executing applications.
- When a running application completes and releases some the occupied resources.
- When the performance requirements of a running application are changed. For example, it might need extra resources for performing some extra functionality.

- When the current task mapping is not sufficiently optimal, it requires (re-)mapping.

The aforementioned issues can be handled only by run-time mapping techniques as the issues are dynamic and need to be handled at run-time. In this paper, we consider the problem that a new application enters the system and needs to be mapped onto the available platform resources, where the number of available resources may be less than the number of processes in the application, i.e., multiple processes need to be mapped onto a single processor. We focus on mapping design space exploration (DSE), where mapping involves two aspects: allocation and binding. In the allocation step, decisions about the number and types of processors that should be deployed in the MPSoC platform are made. In the binding step, it is determined how these allocated resources should be used for running application tasks, i.e. determining which task is bound to which processor. We present a novel run-time binding approach, where identification of an efficient mapping for a use-case is done by the online execution trace analysis of the active applications for multimedia MPSoC-based embedded systems. The objective of our proposed technique is to minimize the execution time (performance) of running applications.

The rest of this paper is organized as follows: Section II discusses related research. Section III describes prerequisites and the definition of the problem. Details of the proposed algorithms are presented in Section IV. The experimental results are presented in Section V, followed by the conclusions of the paper in Section VI.

II. RELATED RESEARCH

In recent years, much research has been performed in the area of task mapping for embedded systems. As mentioned before, it can be accomplished at either design-time (static) or run-time (dynamic).

Design-time techniques: All the design-time techniques find placement of tasks at design-time. Therefore, these techniques are not suitable for run-time varying workloads in systems, which require re-mapping/run-time mapping of applications (e.g. networking and multimedia applications). Even if these mapping techniques are inadequate for the dynamic workload scenarios, such techniques might still be useful to find the initial task placement, or be optimized to be working at run-time.

Run-time Mapping: In contrast to the design-time mapping, run-time mapping needs to consider the time taken to map each task as this contributes to overall application execution time. Furthermore, the tasks are mapped one by one, unlike the static case where all the tasks are mapped at once by globally looking at the system. Therefore, greedy algorithms are typically used for efficient mapping to optimize performance metrics such as energy consumption, communication latency, execution time etc.

Dynamic task mapping on heterogeneous MPSoC platforms are investigated in [7][8][9][10][11][12][13][14][15]. An iterative hierarchical strategy is present in [7] to map an application to a parallel heterogeneous MPSoC architecture at run-time. Applications are modeled as a set of communicating PEs. The optimization objective is to minimize the energy consumption of the MPSoC while providing a certain level of Quality of Service.

In [8] a run-time spatial mapping technique is investigated with real-time requirements, considering streaming applications mapped onto heterogeneous MPSoCs. In the proposed work, the application remapping is determined according latency/throughput that is collected at design time, aiming to satisfy the QoS requirements, minimize resources usage and also the energy consumption. In [9], a distributed agent-based mapping scheme is proposed. The scheme divides the system into virtual clusters. A cluster agent is responsible for all mapping operations within a cluster. Global agents store information about all the clusters of the NoC and use a negotiating policy with cluster agents in order to define to which cluster an application will be mapped. In [10], authors investigate the performance of mapping algorithms in MPSoCs considering dynamic workloads. The heuristics targets NoC congestion minimization as a key function to optimize the NoC performance. The proposed Path Load mapping heuristic reduces the total execution time. This work is extended in Singh et al. [11], where several communication-aware run-time multitask mapping heuristics are proposed.

In [12], an ant colony optimization heuristic is introduced. Starting from a model of the target architecture and the application, the heuristic efficiently executes both scheduling and mapping in order to optimize the application performance. In [13], a novel technique is presented that is able to minimize the energy consumption of the entire multi-mode system while satisfying a given lifetime reliability constraint. In [14], a method is proposed that explores how to synthesize a heterogeneous multiprocessor platform with the partitioning of real-time tasks so that the energy consumption is minimized. By considering both static (leakage) power consumption and dynamic power consumption, they propose a method that uses a polynomial time approximation algorithm to minimize the energy consumption. In [15], a co-synthesis framework is introduced for design space exploration that considers heterogeneous scheduling while mapping multimedia applications onto such MPSoCs. The optimization key is energy.

In this paper, we propose a run-time mapping algorithm for heterogeneous MPSoCs that uses task merging transformations to accommodate systems with less available processing resources than the number of tasks to execute. While many works focus on clustered or grouped tasks in the domain of Synchronous Data Flow (SDF) graphs [16], we analyze and model applications using the Kahn Process Network (KPN) model of computation [21]. More specifically, we analytically model the throughput behavior of different mappings of these KPNs, where we apply process merging (creating compound processes) to capture the effects of mapping multiple tasks to a single processor . There are other works on throughput computation, but they are mainly

developed for SDF and CSDF models [17], [18]. A compile-time approach is presented in [19] to evaluate the system throughput of Polyhedral Process Networks (PPNs) in order to select a merging candidate which gives a system throughput as close as possible to the original PPN. Our work is inspired by this work but is different as their architecture platform was homogenous and they do no target run-time mapping. The work in [20] proposed a throughput model for mapping evaluation during design time. In that work, there are no constraints in the available resources in the system and the design space is explored by a NSGA-II genetic algorithm. The throughput model is used as fitness function interleaved with simulation. That approach significantly reduces the number of simulations that are needed during the process of DSE. In our problem, a resource binding decision has to be made at run-time and there is not enough time to perform simulation for this, reducing the search space that can be covered. Therefore, the analytical throughput model is used to make a fast decision at run-time about the optimal resource binding.

III. Prerequisites and Problem Definition

Application task mapping on an MPSoC platform involves assignment and ordering of the tasks and their communications onto the platform resources in view of some optimization criteria like reducing energy consumption, improving compute performance etc. The optimization is necessary to satisfy performance constraints of the applications. Therefore, efficient mapping techniques are required in order to optimize the performance. The mapping techniques need the following models and parameters:

- An application model (e.g., Task Graph, Data Flow Graph, KPN model, etc.).
- An architecture model of the MPSoC platform (e.g., topology, number of processing elements (PEs) and their type, interconnection scheme, etc.).
- The constraints of the application (e.g., compute performance and/or power requirements, etc.).
- An estimate of the worst-case execution time of the task/process implementations on different PEs.

In this section, we explain the necessary prerequisites for this paper and provide a detailed problem definition.

A. Application Model

In this paper, we target the multimedia application domain. For this reason, we use the Kahn Process Network (KPN) model of computation [21] to specify application behavior since this model of computation fits well with the streaming behavior of multimedia applications. In a KPN, an application is described as a network of concurrent processes that are interconnected via FIFO channels. This means that an application can be represented as a directed graph $KPN = (P, E)$, where P is a set of processes (tasks) p_i in the application and $f_{ij} \in E$ represents the FIFO channel between two processes: p_i and p_j. Fig. 1 illustrates a KPN for a Motion-JPEG (MJPEG) encoder application.

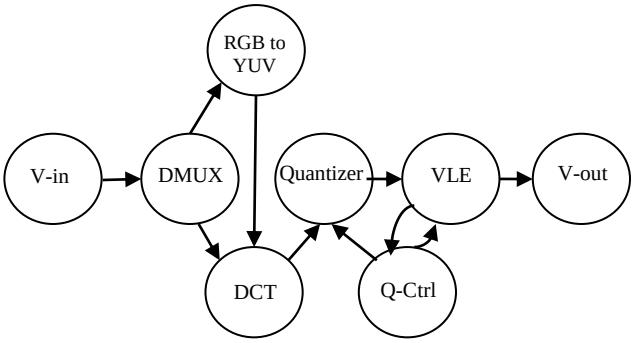

Fig. 1. KPN Model for an MJPEG encoder.

B. Architecture Model

The MPSoC hardware platform is also modelled as a graph $H = (C, F)$, where C is the set of processing elements used in the architecture, F is a multiset of pairs $F_{ij} = (c_i,c_j) \in C \times C$ representing a communication channel (like Bus, NOC, etc.) between processors c_i and c_j.

C. Mapping applications to platform architecture

Given the application and architecture models, the mapping problem can be defined as assigning application processes to architecture components in such a way that the overall performance of the application is optimized. In this paper, we consider the problem of finding the best process merging in a KPN such that we have an optimal 1-to-1 mapping of (possibly merged) processes to processors. That is, the proposed algorithm reduces the number of processes in a KPN by sequentializing a selected number of processes in a single compound process. Thus, less processes need to be mapped on the platform's processing elements, at the cost of possibly having less processes running in parallel. This process merging transformation needs to be applied in case the number of processes is larger than the number of processing elements, i.e., multiple processes need to mapped onto a single processing resource. The problem is that many different options exist to merge two or more processes, where the challenge is to efficiently find the best solution from all these options at run-time. Therefore, the main contribution of this paper is: finding the best task binding for new incoming applications at run-time when the type and number of available processors are specified. This goal is achieved by using an analytical throughput estimation model for KPNs to evaluate the throughput of different process mergings in order to select the best option which gives a system throughput as close as possible to the original KPN.

The performance of an application can be measured by considering the execution time or throughput of that application. It is important to state that our goal is not to compare different KPNs, but to compare transformed KPNs derived from a single KPN. Therefore, in the throughput modeling, we do not take into account the latency of a token, i.e., the time that elapses between injecting a token in the KPN and the time when that token leaves the KPN. Thus, we do not calculate the total execution time of KPNs, but only want to capture the throughput trend instead.

IV. Solution Approach

In this paper, the KPN model of computation is considered for application modeling in which parallel processes communicate with each other via unbounded FIFO channels. In the Kahn paradigm, reading from channels is done in a blocking manner, while writing is non-blocking. To evaluate each possible mapping, one need to perform throughput analysis for the KPN and a given mapping, like in [20]. As mentioned before, at run-time, when a new application arrives to the system, the number and type of available processors can be less than the number of processes. In that case, multiple processes have to map onto a single processor. To model this, we use merging transformations on the KPN to reflect the mapping of the different processes onto a single resource. If two processes are mapped onto the same architectural component, they are merged into a single process in the KPN, as is illustrated in Fig. 2. Mapping multiple KPN tasks onto one processor allows for MPSoC implementations with less processing and communication components, i.e. with reduced implementation cost, but at the cost of potentially additional execution overhead. Subsequently, to evaluate the performance of a mapping decision, we perform throughput analysis on the transformed KPN. With 3 motivating examples we show that selecting the best merging option is not a straightforward task as it depends on the inter-play of many factors.

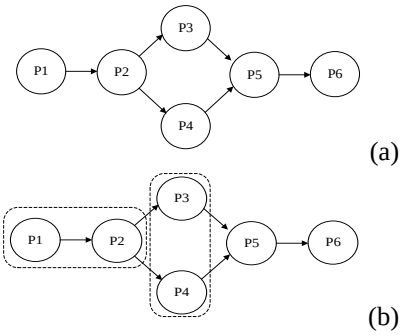

Fig. 2. Process merging in an example (a) Initial Kahn Process Network (b) KPN processes (1,2) and (3,4) are mapped onto a single processor.

A. Motivating Examples

The first factor to be considered is the workload of a process. The workload W_{P_i} of a process P_i denotes the number of time units that are required to execute a function, i.e., the pure computational workload, excluding the communication [19]. Fig. 3 shows a KPN consisting of 6 processes. The network has two data paths DP1 = (P1, P2, P3, P6) and DP2 =(P1,P4,P5,P6) that transfer an equal number of tokens. The system throughput is determined by Process P3. The system throughput of the original KPN is 11 time units (1+2+7+1) needed to produce the first token for Process P6. Then, it produces a new token each 7 cycles which is dictated by the slowest process P3. If the process merging transformation is applied to processes P2 and P3, then compound process P23 becomes the most computationally intensive process of the network. Processes P2 (2 time units) and P3 (7 time unit) are put into a sequence and thus it will take 7+2=9 time units instead of 7 time units for a new token to be produced by process P6. We can see that the throughput of this network is lower than the original KPN. After merging processes P4 and P5, the system throughput is not impacted, i.e., it is identical to the original KPN, because the two merged and sequentialized processes do not determine the system throughput. Thus, these processes can be safely merged together and achieve the same system throughput as the original KPN.

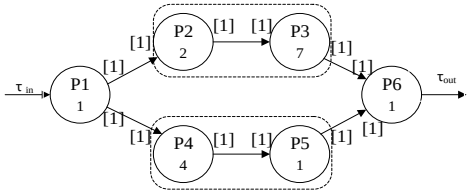

Fig. 3. Process Workload Influencing the System Throughput

The second factor that needs to be taken into account is the rate of token production [19]. Consider the KPN in Fig. 4. In this figure, both data-paths transfer a different number of tokens. This is indicated by the patterns [100] and [011] at which process P1 writes to its outgoing FIFO channels. A ”1” in these patterns indicates that data is read/written and a ”0” implies that no data is read/written. So, the FIFO channel connecting P1 and P2, for example, is written on the first firing of P1, but not in the remaining two firings. As a consequence of these patterns, the second data path DP2 becomes the throughput-limiting path for this particular network and token firings. So, despite the fact that the largest workload of 7 time units can be attributed to process P3, process P4 with a workload of 4 is more dominant. Therefore, processes P2 and P3 can be safely merged as opposed to P4 and P5 to achieve a system throughput equal to the original KPN.

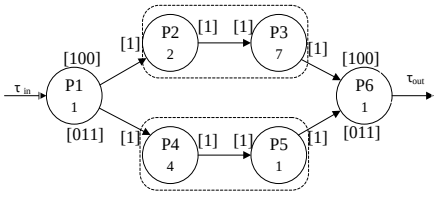

Fig. 4. Production Rate Influencing the System Throughput

The third factor that needs to be also taken into account is the sequentialization of FIFO communication [19]. In Fig. 5, process P1 is the computationally most intensive process with a workload of 25 time units. A logical choice would be to combine P2 and P3 and not to consider the heavy process P1. For this reason, we expect performance results that are equally good as the original KPN. However, when the performance results of both the original KPN and the transformed KPN are measured [22], the performance results of the transformed KPN are depreciated. Despite the workload of compound process P23 being lower than that of P1, the compound process reads in a sequential order from two input channels and writes in a sequential order to two output channels. This means it is the heaviest process in the network. So, besides sequential execution of the process workloads, the sequential FIFO reading/writing is another aspect that needs to be taken into account.

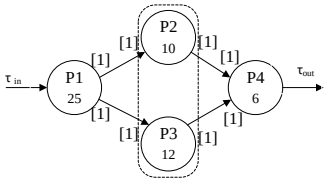

Fig. 5. Sequentialized FIFO Accesses Influencing the System Throughput

B. Throughput propagation to estimate overal throughput

In this paper, the throughput analysis method is based on the work presented in [19][20], in which the solution approach for the overall KPN throughput modeling relies on calculating the throughput τ_{P_i} of a process (i.e., node) P_i for all KPN processes and propagation of the lowest process throughput to the sink process. A depth first search is used to determine the order of the processes for propagating throughputs. For a process P_i, the propagation consists of selecting either the aggregated incoming FIFO throughput τ_{Faggr,P_i} or the isolated process throughput $\tau_{P_i}^{iso}$.

The isolated throughput $\tau_{P_i}^{iso}$ is the throughput of a process P_i when it is considered to be completely isolated from its environment. This means that the isolated process throughput is determined only by the workload W_{P_i} of a process and the number of FIFO reads/writes per process execution provided that no blocking occurs:

$$\tau_{P_i}^{iso} = \frac{1}{W_{P_i} + x.C^{Rd} + y.C^{Wr}} \quad (1)$$

where x and y denote how many FIFOs are read and written per process execution and C^{Rd} and C^{Wr} the performance costs for reading/writing a token from/to a FIFO channel. The throughput of a FIFO-channel f is determined by the throughput of the processes accessing it:

$$\tau_f = \min(\tau_f^{Wr}, \tau_f^{Rd}) \quad (2)$$

Subsequently, the throughput τ_{P_i} of a process Pi is determined by either the throughput of the FIFOs from which process P_i receives its data or by the computational workload of the process itself, i.e., $\tau_{P_i}^{iso}$. For merged KPN processes, the incoming FIFO throughput is the aggregated throughput of the merged channels and the isolated throughput is calculated using the aggregated computational workloads. Consequently, the throughput associated to each process in a KPN graph is computed as:

$$\tau_{P_i} = \min(\tau_{Faggr,P_i}, \tau_{P_i}^{iso}) \quad (3)$$

C. Handling cycles

It is possible that the aforementioned merging transformations to account for mapping decisions might introduce cycles in the transformed KPN. As shown in Fig. 6, processes 1, 3 and 5 are mapped to the same processor, resulting in a KPN with two cycles. Cycles in a KPN are responsible for sequential execution of some of the processes involved in the cycle. The sequential execution can vary from a single initial delay to a delay at each execution of some of the processes. For accurate throughput modeling, these cycles must be taken into account.

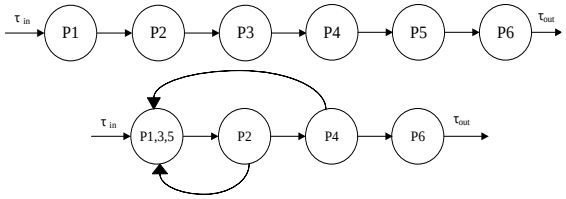

Fig. 6. Transformation into a cyclic KPN

We therefore *approximate* the isolated throughput of a process P_i that is member of a cycle by:

$$\tau_{Cycl_{P_i}}^{iso} = \frac{1}{\sum_{P_j \in Cycle} \tau_{P_j}^{iso}} \quad (7)$$

From equation 7, it is clear that the isolated throughput of a cycle is lower than the regular isolated throughput ($\tau_{P_i}^{iso}$) of any of the processes involved in the cycle. It also implies that the isolated throughput of a cycle can be lower than the isolated throughput of the bottleneck process. This is an important observation because, in such a case, the throughput of the cycle

will determine the overall KPN performance. To conclude, the throughput associated to each process P_i will be computed as:

$$\tau_{P_i} = \min(\tau^{iso}_{Cycl_{P_i}}, \tau_{Faggr,P_i}, \tau^{iso}_{P_i}) \quad (5)$$

For example, in Fig. 6, two cycles are generated due to the KPN transformation. In this case, we assume that the resulting $\tau^{iso}_{Cycl_{P_i}}$ for a process P_i would be

$$\tau^{iso}_{Cycl_{P_i}} = \min(\tau^{iso}_{Cycl_{P_i}}(1),...,\tau^{iso}_{Cycl_{P_i}}(n)) \quad (6)$$

Where $\tau^{iso}_{Cycl_{P_i}}(1),...,\tau^{iso}_{Cycl_{P_i}}(n)$ are all the throughputs of the cycles involving process P_i .

Algorithm 1*: Run-time mapping algorithm using KPN Throughput Estimation Pseudo-code*

Input: *Application Graph (KPN), Architecture Graph, Available Resources*
Output: *Best resource binding and mapping, best throughput*

Requires: W_{Pi}*, the computational workload of all processes.*

```
for l in itertools.product(Allocated_Resources, repeat=#Processes):
    h = list(l)
    if len(set.intersection(set(h),set(Allocated_Resources))) ==
      len(Allocated_Resources):
        gmapping = h;
        Task_List ← Create topological ordering for KPN
        for all processes P_i ∈ Task_List do
            τ^iso_Pi = set isolated throughput(P_i , W_Pi)
            Set τ^Rd_fj for all incoming FIFOs f_j
            τ_Pi = min (τ_Faggr,Pi , τ^iso_Pi )
            Set τ^Wr_fj for all outgoing FIFOs f_j of P_i.
        end for
        return Throughput = τ_sink

        if Throughput < best_Throughput:
            best_Throughput = Throughput
            best_map = gmapping
```

The pseudo code of the run-time algorithm to select the best binding using the described analytical throughput calculation and propagation method is shown in Algorithm 1. The KPN model of application and available resources in the system must be specified as algorithm input. The best mapping and resource binding will be determined by the algorithm. Here, the space has been explored exhaustively using *itertools.product()* function of Python. Using analytical throughput estimation as fitness function during the search for best mapping can yield significant efficiency improvements.

V. EXPERIMENTS AND RESULTS

For our experiments, three KPN models of synthetic applications with different number of processes (tasks) and different workload of processes and different rates of token production are used. Furthermore, a Motion JPEG (MJPEG) encoder is used as a real multi-media application. The target architecture that has been considered in our experiments consists of a MPSoC platform with five heterogeneous processors, i.e., five different processor types. These processors are connected via a bus to a shared memory. For all experiments, a PC with a 2.9GHz Intel Core i7 CPU has been used.

The open-source Sesame system-level MPSoC simulator [23] is deployed to evaluate mappings. The Sesame modeling and simulation environment facilitates efficient performance analysis of embedded (media) systems architectures. It recognizes separate definitions of application and architecture models in which an application model describes the functional behavior of an application and the architecture model defines architecture resources and captures their performance constraints. After explicitly mapping an application model onto an architecture model, they are co-simulated via trace-driven simulation. This allows for evaluation of the system performance of a particular application, its underlying architecture, and mapping.

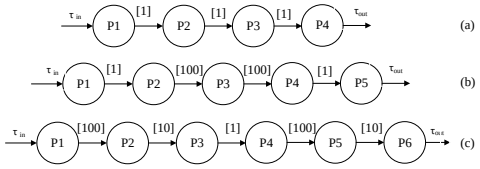

Fig. 7. KPN model of Applications used for experiments

In the first experiment, it is supposed that an application as modeled in Fig. 7(a) arrives to the system at run-time and only three processors, identified by the IDs '1', '2' and '4', are available to execute this incoming application. Here, 36 different configurations exist to merge processes and form a mapping. Exhaustively evaluating these mappings with the Sesame framework takes about 32 minutes to select the best mapping. The simulation time grows exponentially with the number of application tasks and available resources. Therefore, it is not feasible to use such simulations to make decisions at run-time. By evaluation with throughput model, it takes 0.42 second to select the best binding solution. TABLE I. shows the number of points which exist to evaluate, time needed for evaluation by proposed run-time algorithm based on throughput estimation model and Sesame simulation.

As discussed in the previous section, decision making regarding process binding to find the best mapping can efficiently be realized at run-time using the discussed analytic throughput model. For example, application (a) has four

application tasks and for each task we assign the identifier of the processor to which the task is mapped. For instance, the mapping {1,1,1,2} indicates that tasks 1,2 and 3 are mapped to processor '1', while tasks 4 is mapped to processor '2'. For this application, our run-time algorithm based on the throughput model has found the {1,2,2,4} mapping as the optimal binding solution. By exhaustively searching and simulating all mappings with Sesame the same {1,2,2,4} mapping has also been selected. We have also performed the throughput model evaluation and Sesame simulation validation for applications (b) and (c) in Fig. 7, where we assumed that the available resources in the system are <1,3> and <1,2,3,4>, respectively. For these cases, the mappings obtained from Sesame simulation and throughput estimation are very close to each other in term of performance. The performance accuracy for these applications are reported in TABLE II.

TABLE I. TIME NEEDED TO FIND BEST MAPPING

Application	# points to evaluate	Time (Throughput estimation model)	Time (Sesame Simulation)
App1	36	0.42 sec	32 min
App2	30	0.40 sec	25 min
App3	1560	1.92 sec	8 h 54 min
MJPEG	540	0.91 sec	3 h 12 min

TABLE II. COMPARISION OF MAPPING OBTAINED BY SIMULATION AND THROUGHPUT MODEL EVALUATION

Application	Allocated Resources	Best Mapping (Sesame)	Best Mapping (Throughput model)	Performance Accuracy
App1	<1,2,4>	{1,2,2,4}	{1,2,2,4}	95%
App2	<1,3>	{1,1,3,3,1}	{1,1,3,3,3}	97%
App3	<1,2,3,4>	{1,3,2,4,4,2}	{1,3,2,4,4,4}	94%
MJPEG	<1,2,4>	{1,1,4,2,4,4}	{1,1,2,2,4,4}	87%

In addition, the KPN of the MJPEG application is used as shown in Figure 1. To compare the performance accuracy of this application, consider Fig. 8. In this experiment, we assume the allocated resources available at the system are <1,2,4>. This figure shows the normalized performance ranking of ten random mappings when evaluating the mappings using Sesame simulations (in red) or using the throughput model (in blue). Evidently, the normalized ranking of the mappings for the MJPEG application is correct most of the times.

VI. CONCLUSION

In this paper, we have proposed a run-time resource binding for Heterogeneous MPSoC-based embedded systems in order to improve their performance. This is done by throughput modeling of Kahn process networks to evaluate process merging transformations and supporting new arriving applications. Our approach can be used at run-time to quickly evaluate different resource bindings, taking into account all factors that influence the throughput. Therefore, we can accurately capture the throughput trend and select the best possible mapping as illustrated with the experiments.

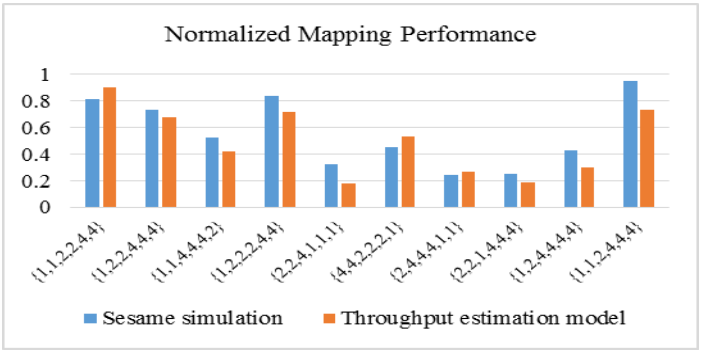

Fig. 8. MJPEG application normalized mapping using Sesame simulation and throughput estimation model

REFERENCES

[1] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on Multi/Many-core Systems: Survey of Current and Emerging Trends," *in Proceedings of ACM Design Automation Conference (DAC)*, pp. 1:1–1:10, 2013.

[2] P. Marwedel, J. Teich, G. Kouveli, I. Bacivarov, L. Thiele, S. Ha, C. Lee, Q. Xu, and L. Huang, "Mapping of applications to MPSoCs," *in Proceedings of IEEE/ACM/IFIP Conference on Hardware/Software Codesign and System Synthesis (ISSS+CODES*), pp. 109–118, 2011.

[3] R. Marculescu, U. Ogras, L. Peh, N. Jerger, and Y. Hoskote, "Outstanding Research Problems in NoC Design: System, Microarchitecture, and Circuit Perspectives," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 28, no. 1, pp. 3–21, 2009.

[4] J. Ceng, J. Castrillon, W. Sheng, H. Scharwachter, R. Leupers, G. Ascheid, H. Meyr, T. Isshiki, and H. Kunieda, "MAPS: an integrated framework for MPSoC application parallelization," *in Proceedings of the Design Automation Conference*, pp. 754–759, 2008.

[5] "IMEC MPSoC Mapping Tools," 2008, http://www.imec.be/ScientificReport/SR2008/HTML/1225004.html (Last visited: 23 December, 2011).

[6] A. K. Singh, W. Jigang, A. Kumar and T. Srikanthan, "Run-time mapping of multiple communicating tasks on MPSoC platforms", *Procedia Computer Science*, vol. 1, no. 1, pp. 1019-1026, 2010.

[7] L.T. Smit, J.L. Hurink and G.J.M. Smit, "Run-time mapping of applications to a heterogeneous SoC". *In: International Symposium on System-on-Chip (SoC'05)*, 2005.

[8] P.K.F. Hölzenspies, L. Johann, J.K. Hurink and G.J.M. Smit, "Run-time spatial mapping of streaming applications to a heterogeneous multi-processor system-on-chip (MPSoC)." *In Proceedings of the conference on Design, automation and test in Europe*, pp. 212-217. ACM, 2008.

[9] M. A. Faruque, R. Krist and J. Henkel, "ADAM: run-time agent-based distributed application mapping for on-chip communication", *In 45th ACM/IEEE Design Automation Conference, (DAC'08)*, pp. 760-765, 2008.

[10] E. Carvalho, N. Calazans, and F. Moraes, "Heuristics for dynamic task mapping in NoC-based heterogeneous MPSoCs", *In 18th IEEE/IFIP*

International Workshop on Rapid System Prototyping (RSP 2007), pp. 34-40, 2007.

[11] A. K. Singh, T. Srikanthan, A. Kumar, and W. Jigang, "Communication-aware heuristics for run-time task mapping on NoC-based MPSoC platforms", *Journal of Systems Architecture*, vol. 56 no. 7, pp. 242-255, 2010.

[12] F. Ferrandi, P. L. Lanzi, C. Pilato, D. Sciuto and A. Tumeo, "Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 6, 2010.

[13] L. Huang, and Q. Xu, "Energy-efficient task allocation and scheduling for multi-mode MPSoCs under lifetime reliability constraint". *In: Design, Automation and Test in Europe (DATE'10)*, pp. 1584-1585, 2010.

[14] J. J. Chen and L. Thiele, "Platform synthesis and partitioning of real-time tasks for energy efficiency". *Journal of Systems Architecture*, vol. 57, no. 6, pp. 573-583, 2011.

[15] M. Kim, S. Banerjee, N. Dutt and N. Venkatasubramanian, "Energy-aware cosynthesis of real-time multimedia applications on MPSoCs using heterogeneous scheduling policies", *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 2, 2008.

[16] J. Falk, J. Keinert, C. Haubelt, J. Teich, and S. S. Bhattacharyya, "A generalized static data flow clustering algorithm for mpsoc scheduling of multimedia applications", *In Proceedings of the 8th ACM international conference on Embedded software*, pp. 189–198, , 2008.

[17] A. H. Ghamarian, M. C. Geilen, W. T. Basten and S. Stuijk, S, "Parametric throughput analysis of synchronous data flow graphs," *In Design, Automation and Test in Europe*, pp. 116–121, 2008.

[18] A. Moonen, M. Bekooij, R. van den Berg and J.A. van Meerbergen, "Practical and accurate throughput analysis with the cyclo static dataflow model," *In 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'07)*, pp. 238–245, 2007.

[19] S. Meijer, H. Nikolov, and T. Stefanov, "Throughput modeling to evaluate process merging transformations in polyhedral process networks", *In Design, Automation & Test in Europe Conference (DATE'10)*, pp. 747–752, 2010.

[20] R. Piscitelli and A. D. Pimentel. "Design space pruning through hybrid analysis in system-level design space exploration." *In Design, Automation & Test in Europe Conference (DATE'12)*, pp. 781-786, 2012.

[21] G. Kahn, "The semantics of a simple language for parallel programming", *I In Information Processing*, pp. 471-475, 1974.

[22] H. Nikolov, T. Stefanov, and E. Deprettere, "Systematic and automated multiprocessor system design, programming, and implementation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 27, no. 3, pp. 542–555, 2008.

[23] A.D. Pimentel, M. Thompson, S. Polstra, and C. Erbas, "Calibration of abstract performance models for system-level design space exploration," *Journal of Signal Processing Systems*, vol. 50, no. 2, pp. 99-114, 2008.