Run-time resource allocation for embedded Multiprocessor System-on-Chip using tree-based design space exploration

Sinaei, S.; Pimentel, A.D.; Fatemi, O.

DOI
10.1109/DTIS.2017.7929873

Publication date
2017

Document Version
Final published version

Published in
2017 12th IEEE International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)

2017 12th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)

Run-time Resource Allocation for Embedded Multiprocessor System-on-Chip using Tree-based Design Space Exploration

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

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

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

Abstract— **The dynamic nature of application workloads in modern MPSoC-based embedded systems is growing. To cope with the dynamism of application workloads at run time and to improve the efficiency of the underlying system architecture, this paper presents a novel run-time resource allocation algorithm for multimedia applications with the objective of minimizing energy consumption for predefined deadlines. This algorithm is based on a novel tree-based design space exploration (DSE) method, which is performed in two phases: design-time and run-time. During design time, application clustering is combined with the tree-based DSE, and after that, feature extraction and application classification is performed during run-time based on well-known machine learning techniques. We evaluated our algorithm using a heterogeneous MPSoC system with several applications that have different communication and computation behaviors. Our experimental results revealed that during runtime, more than 91% of the applications were classified correctly by our proposed algorithm to select the best resources for allocation. Therefore the results clearly confirm that our algorithm is effective.**

Keywords-Run-time Mapping; Multi-processor System on Chip; Embedded Systems; Dynamic workloads; Clustering; Classification; Design Space Exploration;

I. INTRODUCTION

Modern embedded systems, which are based more and more on Multiprocessor System on Chip (MPSoC) architectures, often require supporting an increasing number of applications and standards. In these systems, multiple applications can run concurrently and are thus simultaneously contending for system resources. For each single application, there are often also different execution modes with different requirements [19]. As a consequence, the behavior of application workloads imposed on the embedded system can change dramatically over time. Typically, the target MPSoC architecture platforms are heterogeneous in nature, as such systems are capable of providing better performance and energy tradeoffs than their homogeneous counterparts [7]. Here, the process of application task mapping plays a crucial role in exploiting the system properties in such a way that applications can meet their often diverse demands for performance and energy efficiency. To satisfy the system demands, run-time mapping is required to optimize performance and energy consumption under dynamic application workloads [1][2][3].

The topic of run-time, or on-the-fly, mapping has received substantial research attention in recent years [4][5][6][7]. In these methods, the assignment of newly arriving tasks to the system resources is done by means of heuristics. The fact that these approaches must be light-weight (as they are performed at run-time), and therefore do not fully exploit the task mapping choices of the target platform, leads to lower quality mappings and schedulability problems.

Traditional design-time mapping solutions, on the other hand, usually yield mappings of higher quality compared to those derived from run-time algorithms as the former allow for exploring a larger design space for the underlying architecture. But as these algorithms typically involve slow computational methods [7] such as Integer Linear Programming (ILP), they cannot be used during run time. Another drawback of static mapping techniques is that they cannot cope with dynamic application behavior in which different combinations of applications that are contending for system resources could be executing commands concurrently over time.

To alleviate the problems of pure run-time decision making, new research directions are being investigated that propose a mixed design-time and run-time approach. That is, the compute intensive analysis is transferred to the design phase in these approaches [8][9][10][11][12][13]. The analysis results that have been obtained during design time can then be used during run-time to accelerate and improve run-time mapping. Such mapping approaches also facilitate the use of a light-weight run-time platform manager, which is required in modern embedded systems (e.g., smart phones and tablets).

A mapping problem can be divided into two important steps: 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. In this paper, a mixed method, which includes design-time and run-time phases, is proposed to solve the run-time allocation problem for multimedia MPSoC-

978-1-5090-6377-2/17/$31.00 ©2017 IEEE

based embedded systems. The objective of our proposed technique is to minimize the workload energy consumption while considering a deadline for execution time. This technique allows the system to support the handling of new incoming applications that were not known during design time.

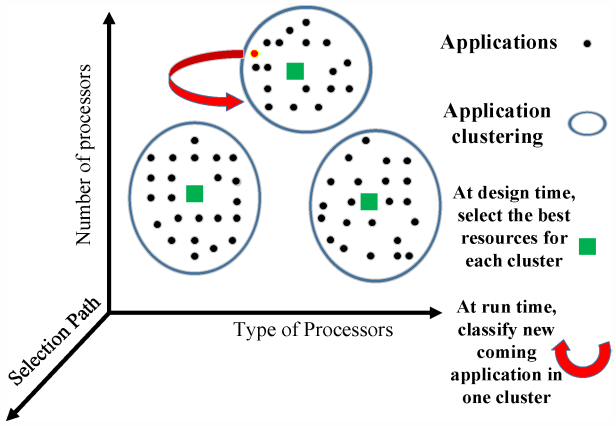

Figure 1. Clustering at design time and classification at run time.

During design time, a novel tree-based design space exploration (DSE) algorithm is proposed to select the optimal resource allocation. It indicates a path in the tree in addition to specifying the number and types of processors to be allocated. As will be explained, this path is used to cluster applications. In Figure 1, the dots indicate several applications that have been explored during design time and their optimal resource allocation is determined by the proposed tree-based DSE method. Application clusters are shown by the circles and the resource allocations of each cluster are indicated by the squares in Figure 1. During run time, resource allocations can then quickly be determined for new incoming applications by means of classification methods based on well-known machine learning techniques.

The rest of this paper is organized as follows: Section II discusses related works, Section III describes our approach for the modeling of multimedia embedded systems 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

The process of application mapping can be done during design-time or during run time. Techniques that are used for design-time mapping ([1][2][3][4][5]) are more appropriate for scenarios in which the workload behavior is static (e.g., a predefined set of applications to be mapped on a static platform) as they cannot handle any application dynamism that occurs over time.

Run-time mapping techniques should be able to efficiently map the tasks of applications on the platform resources, and in the meantime, have an accurate knowledge of resource occupancy. The literature on run-time mapping can be categorized into two main groups: on-the-fly mapping and mapping using design-time DSE results.

In the on-the-fly mapping techniques, all the required computations and decisions are performed when a new application enters the system or the system condition changes. Several works on on-the-fly mapping [6][7][8][9][10][11][12] have been previously done with the aim of optimizing one or more performance metrics (such as overall execution time and energy consumption). These works employ different kinds of heuristics and allocate tasks of incoming applications to the system resources. The downside of these methods is their limited on-line processing capability, which implies that on-line decisions have to be made quickly and thus the process of assigning tasks to resources may be performed poorly and so may lead to low-quality mapping.

By using the results of design-time mapping exploration (and storing these results such that they can be used for making run-time mapping decisions), approaches such as [13][14][15][16][17][18][19] obtain better results than on-the-fly mapping. However, most of these approaches still suffer from shortcomings regarding their adaptivity to cope with application dynamism. For example, many approaches do not support the handling of newly incoming applications that were not known during design time. In this paper, we propose a novel algorithm for run-time resource allocation - which is an essential ingredient of the mapping process - based on application clustering and classification that can handle newly incoming applications.

III. Prerequisites and Problem Definition

In this section, we explain the necessary prerequisites for this work 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[20] 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. Figure 2. illustrates a KPN for a Motion-JPEG (MJPEG) encoder application.

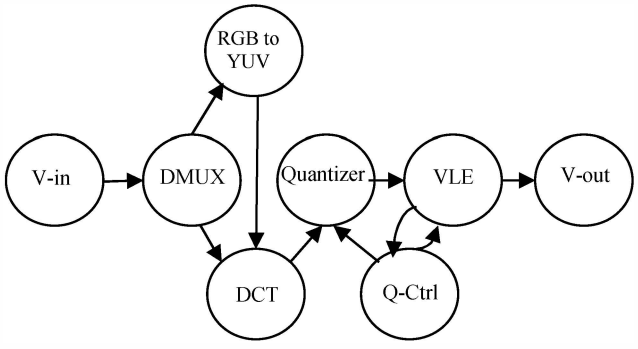

Figure 2. KPN Model for an MJPEG encoder.

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 $F_{ij} = (c_i, c_j) \in C \times C$ represents a communication channel (like Bus, NOC, etc.) between processors c_i and c_j.

C. Resource Allocation

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 energy consumption required to execute the application is minimized with regards to a pre-defined deadline for finishing the tasks. As mentioned before, the mapping problem can be divided into allocation and binding steps. In this paper, a mixed method is proposed to solve the run-time *allocation* problem for multimedia MPSoC-based embedded systems. That is, the method makes decisions about the number and types of processors that should be deployed in the MPSoC platform.

IV. PROPOSED RUN-TIME ALLOCATION METHOD

As shown in Figure 3, the entire workflow of our approach can be divided into two phases: design time and run time. In the design-time phase, application clustering is combined with a tree-based DSE. The proposed tree-based DSE, which will be explained in subsection A, is used for exploring the design space of pre-known applications. In addition to exploring the design space, this method clusters applications by specifying a path within the resource selection tree, as will be explained in subsection B. This clustering is based on the selected resource types, number of each resource type and the resource selection priority (i.e., the level of affinity of applications to a certain type of resource). Applications that are placed in the same cluster need similar resources to run optimally in terms of energy consumption within the predefined deadline. After the DSE step, features of each cluster are determined that will be used as classification parameters. Information about the application clustering and their allocation of resources is saved in a system database, which will be used during the run-time phase.

During run-time, when a new and possibly unknown application enters the system, a fast decision about the resource allocation for that application needs to be made and there is not sufficient time to explore all or even a small fraction of the allocation possibilities. So with a fast profiling of the new application, determining its features, and comparing them to those in the system database, this new application can be classified as belonging to one of the existing clusters. Subsequently, the resource allocation specified for that cluster will then be used to run the new application.

The following subsections will explain the application clustering and classification methods in more detail. After the resource allocation step, the application binding must be performed. However, this binding step is not within the scope of this paper.

A. Tree-based DSE for Clustering

Our proposed tree-based DSE method prunes and explores the design space by gradually assessing the deployment of an increasing number of processors in the heterogeneous MPSoC platform. To this end, it tries to efficiently and effectively find a resource allocation for the target application(s) on the MPSoC system with the objective of minimizing energy consumption within a predefined deadline for completing the application tasks. By using this method, it is also possible to identify the priority for the selection of certain types of resources (i.e. resource affinity) which is not feasible in, for example, widely-used mapping methods based on metaheuristics such as genetic algorithms.

Algorithm 1 shows the pseudo code for the tree-based DSE.

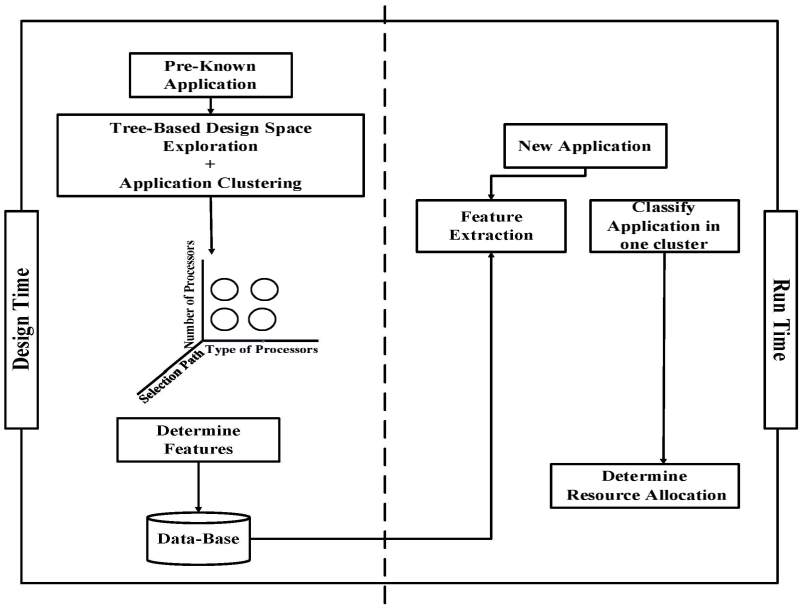

Figure 3. Workflow of proposed run-time mapping approach

The *ProcessorsVector* indicates the selected resources to run the application and is formed by traversing the different levels of the tree. The design space exploration is performed for different tree levels (line 2) and also at each level for different processors (line 3). The maximum number of levels in the tree is equal to the number of tasks in the application, and the number of nodes at each level is equal to the number of processors in the architecture. The *Resources* variable (line 4) is used to explore the architecture processing elements. In each tree node, it is the summation of the selected resources of the previous levels and the processor related to that node. If the application size is small, DSE can be done exhaustively for the selected *Resources*, considering the mapping of all application tasks to these resources. This results in optimal resource allocation at a specific tree level (lines 5 and 6). If the application size is large, however, then the exploration at a certain tree level will be done using a genetic algorithm (lines 7 and 8). If the energy consumption of the selected resource allocation is better than the previous values, then the selected processor will be added to the *ProcessorsVector* and its energy consumption will be considered as the best one (lines 9, 10 and 11). At the end of the DSE at each level, the best resource allocation and configuration is selected and *ProcessorsVector* will be updated.

Algorithm 1: *Proposed Tree-based Design Space Exploration*

Input: *Application Graph, Architecture Graph*
Output: *Best resource allocation, best energy consumption*

```
1:  ProcessorsVector {}
2:  For S=1 to Levels_of_Tree  // i.e., # of application tasks
3:      For K=1 to Num_of_Processors
4:          Resources = ProcessorsVector + K
5:          If (application size is small)
6:              Search exhaustively all mappings with determined
                resources to find mapping with best energy
                consumption within predefined deadline
7:          If (application size is large)
8:              Search design space to find the mapping with
                best energy consumption within predefined deadline
                using a genetic algorithm
9:          If (energy consumption of best mapping is improved)
             && the deadline is not missed
10:             ProcessorsVector [S] = K;
11:             Set its energy as BestEnergyConsumption;
12:  Return "BestEnergyConsumption"
13:  Return "ProcessorsVector"
```

B. *Application Clustering and Classification*

In the proposed methodology, application clustering is combined with design space exploration during design time. Using our tree-based DSE method, it is possible to identify the priority for the selection of resources. Considering such priorities for each application, a label can be assigned to it and the applications can be clustered based on these labels (resource demands). Actually, the best resource allocation for each application will be determined and applications with the same resource demand (i.e. same label) will be assigned to the same group/cluster. The number of clusters is equal to the number of nodes in the tree *(n^m-1, n: Number of processor types and m: Number of tasks)*. The label of each cluster reflects the resource demands for all applications placed in that cluster and is represented by a vector.

During run time, when a new unknown application is started in the system, a fast decision about resource allocation must be taken so that the application can be executed. There is not enough time for a full or even limited design space exploration during run time. Therefore, we use a classification by means of feature extraction of the new application in order to fit it to one of the pre-determined application clusters. By knowing to which application cluster the new application belongs, the resource allocation belonging to that cluster (as indicated by the cluster label) is selected for executing the new application.

In this work, we deploy well-known methods for classifying applications such as naive Bayes [27], support vector machines[25], logistic regression [26], and decision trees [24].

C. *Feature Extraction*

To classify an application, a clear learning goal must be specified. It is often necessary to select a subset of relevant features (variables, predictors) for use in model construction (i.e., feature extraction and selection). Application requirements can be divided into two categories: computational requirements and communication requirements. TABLE I. shows selected features of applications that are used for classification.

V. EXPERIMENTAL RESULTS

For our experiments, we have used more than 500 KPN models of synthetic applications with different numbers of processes (tasks) and different computation and communication requirements as well as operation types. The target architecture that has been considered in our experiments consists of a MPSoC platform with five heterogeneous processors. These processors are connected via a bus to a shared memory. The open-source Sesame system-level MPSoC simulator has been used to evaluate the mappings. For all experiments, a PC with a 2.9GHz Intel Core i7 CPU has been used.

TABLE I. FEATURE TYPES

No	Features
1	**Number of tasks in application**
2	**Communication/Computation ratio (in terms of energy)**
3	**Communication/Computation ratio (in terms of performance)**
4	**Computation Ratio with neighbor processes (in terms of energy)**
5	**Computation Ratio with neighbor processes (in terms of performance)**

In our work, the open-source Sesame system-level MPSoC simulator [21] 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.

A. Application Clustering by proposed Tree-based design space exploration algorithm (design-time)

Our proposed tree-based exploration algorithm has been used to explore the vast solution space and to find the (near) optimal mapping with the objective of minimizing the energy consumption with regard to the deadline for application tasks. In the first experiment, we considered the KPN models of several applications. For small applications, an exhaustive search can be done at each level of the tree and the comparison is done with non-tree-based exhaustive DSE. These experiments were done for applications with 3 to 6 tasks and the results are shown in TABLE II. For applications with more tasks, the exploration of the design space cannot be performed exhaustively anymore as the design space grows exponentially with the number of tasks. In these cases, we used a genetic search at each level of the tree and the comparison was made with a traditional, non-tree-based DSE method using a NSGA-II genetic algorithm (GA) [19]. The results are shown in TABLE III.

TABLE II. COMPARISION OF TREE-BASED DSE WITH EXHAUSTIVE DSE (SMALL APPLICATIONS)

#Application tasks	#Simulations (Exhaustive DSE)	#Simulations (Tree-based DSE)	Average Mapping Accuracy
3	**125**	**32**	**95%**
4	**625**	**134**	**97%**
5	**3125**	**642**	**94%**
6	**15625**	**3178**	**92%**

TABLE III. COMPARISION OF TREE-BASED DSE WITH GA-BASED DSE (LARGE APPLICATIOS)

#Application tasks	NSGA-II DSE Average Time	Tree-based DSE Average Time	Average Mapping Accuracy
10	**135 min**	**65 min**	**94%**
11	**700 min**	**390 min**	**93%**
12	**3200 min**	**2000 min**	**91%**

As shown in TABLE II. and TABLE III. , simulation results show that our algorithm, on average, yields an accuracy level of 91% or higher. Furthermore, significantly fewer simulations are needed to achieve the best solution with this level of accuracy. Actually, the design space was pruned by our method and at each level, several unessential simulations were eliminated.

For clustering applications, the design space was explored using our tree-based method for applications with a varying number of tasks, different communication and computation ratios as well as diverse operations. Clustering was done at the same time when the design space was searched and suitable resources were determined to execute the applications; applications with the same resource allocations were placed in the same cluster.

TABLE IV. APPLICATION CLASSIFICATIN AT RUN TIME

No	Classifier Method	Correctly Classified Instances
1	**Decision Tree**	**496(96%)**
2	**Support vector machine**	**470 (91%)**
3	**Naive Bayes**	**463(90%)**
4	**Logistic Regression**	**491(95%)**

B. Application Classification and feature extraction (Run-time)

During run-time, when a new unknown application enters the system, a fast decision about resource allocation must be taken to execute this application. With fast profiling, this new application is classified to fit it to one of existing clusters. To this end, the fast profiling extracts and uses of the application features as listed in TABLE I.

To evaluate the application classification, we used Weka [22] using cross validation with 10 folds. In *k*-fold cross-validation, the original sample is randomly partitioned into *k* subsamples. a single subsample is retained as the validation data for testing the model, and the remaining *k−1* subsamples are used as training data. The cross-validation process is then repeated *k* times (the folds). The *k* results from the folds then can be averaged (or otherwise combined) to produce a single estimation. As explained before, for the actual classification, we used well-known machine learning algorithms like decision trees, support vector machines, Naive Bayes and logistic regression. The results for the different classifiers are shown in TABLE IV. The decision tree classifier has the most correctly classified instances. The table shows that during run-time, more than 96% of the applications were classified correctly by our proposed algorithm to select the best resources for allocation using this classifier.

VI. CONCLUSION

In this paper, we have proposed a run-time resource allocation methodology for MPSoC-based embedded systems in order to improve their energy consumption for a predefined deadline. This is done by capturing the dynamism of the application workloads, executing on the system and supporting new, unknown applications. This algorithm is performed in two phases: design time and run time. In the design-time phase, using a novel tree-based algorithm, the design space is explored and Selected Processors Vectors are extracted. Subsequently, applications can be clustered based on this vector as their resource demands. In the run-time phase, when a new application enters the system, application features are extracted and based on these features, the application can be classified in order to fit it to one of the pre-determined application clusters. Using the Selected Processors Vector of the identified cluster, a suitable resource allocation for executing the new application can then be selected.

REFERENCES

[1] M. Ruggiero, A. Guerri, D. Bertozzi, F. Poletti, and M. Milano, "Communication-aware allocation and scheduling framework for streamoriented multi-processor systems-on-chip," in *Proceedings of IEEE Conference on Design, Automation and Test in Europe (DATE)*, 2006, pp. 3–8.

[2] S. Murali, M. Coenen, A. Radulescu, K. Goossens, and G. De Micheli, "A methodology for mapping multiple use-cases onto networks on chips," in *Proceedings of IEEE Conference on Design, Automation and Test in Europe (DATE)*, pp. 118–123, 2006.

[3] G. Chen, F. Li, S. Son, and M. Kandemir, "Application mapping for chip multiprocessors," in *Proceedings of ACM Design Automation Conference (DAC)*, pp. 620–625, 2008.

[4] C. Marcon, E. Moreno, N. Calazans, and F. Moraes, "Comparison of network-on-chip mapping algorithms targeting low energy consumption," *IET Computers Digital Techniques*, pp. 471 –482, 2008.

[5] H. Javaid and S. Parameswaran, "A design flow for application specific heterogeneous pipelined multiprocessor systems," in *Proceedings of ACM Design Automation Conference (DAC)*, pp. 250–253, 2009.

[6] E. L. d. S. Carvalho, N. L. V. Calazans, and F. G. Moraes, "Dynamic task mapping for MPSoCs," *IEEE Des. Test of Comp.*, vol. 27, no. 5 , pp. 26–35, 2010.

[7] A. K. Singh, T. Srikanthan, A. Kumar, and W. Jigang, "Communicationaware heuristics for run-time task mapping on NoC-based MPSoC platforms," *Elsevier Journal of Systems Architecture (JSA)*, vol. 56, pp. 242–255, 2010.

[8] V. Nollet, P. Avasare, H. Eeckhaut, D. Verkest, and H. Corporaal, "Runtime management of a MPSoC containing FPGA fabric tiles," *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 16 , pp. 24–33, 2008.

[9] O. Moreira, J. J.-D. Mol, and M. Bekooij, "Online resource management in a multiprocessor with a network-on-chip," in *Proceedings of ACM Symposium on Applied Computing (SAC)*, pp. 1557–1564, 2007.

[10] L. Chen, T. Marconi, and T. Mitra, "Online scheduling for multi-core shared reconfigurable fabric," in *Proceedings of IEEE Conference on Design, Automation and Test in Europe (DATE)*, pp. 582 –585, 2012.

[11] J. Huang, A. Raabe, C. Buckl, and A. Knoll, "A workflow for runtime adaptive task allocation on heterogeneous MPSoCs," in *Proceedings of IEEE Conference on Design, Automation and Test in Europe (DATE)*, pp. 1 –6, 2011.

[12] F. Wang, Y. Chen, C. Nicopoulos, X. Wu, Y. Xie, and N. Vijaykrishnan, "Variation-aware task and communication mapping for mpsoc architecture," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, no. 2, pp. 295 –307, 2011.

[13] A. Schranzhofer, J.-J. Chen, and L. Thiele, "Dynamic Power-Aware Mapping of Applications onto Heterogeneous MPSoC Platforms," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 4, pp. 692 –707, 2010.

[14] C. Ykman-Couvreur, P. A. Hartmann, G. Palermo, F. Colas-Bigey, and L. San, "Run-time resource management based on design space exploration," in *Proceedings of IEEE/ACM/IFIP Conference on Hardware/Software Codesign and System Synthesis (ISSS+CODES)*, pp. 557–566, 2012.

[15] P. Yang, P. Marchal, C. Wong, S. Himpe, F. Catthoor, P. David, J. Vounckx, and R. Lauwereins, "Managing dynamic concurrent tasks in embedded real-time multimedia systems," in *Proceedings of IEEE/ACM/IFIP Conference on Hardware/Software Codesign and System Synthesis (ISSS+CODES)*, pp. 112–119, 2002.

[16] C.-L. Chou and R. Marculescu, "Designing heterogeneous embedded network-on-chip platforms with users in mind," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 29, no. 9, pp. 1301–1314, 2010.

[17] A. K. Singh, A. Kumar, and T. Srikanthan, "Accelerating throughputaware runtime mapping for heterogeneous MPSoCs," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 18, no. 1 , pp. 9:1–9:29, 2013.

[18] C. Lee, S. Kim, and S. Ha, "Efficient Run-time Resource Management of a Manycore Accelerator for Stream-based Applications," in *Proceedings of IEEE/ACM/IFIP Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia)*, pp. 51–60, 2013.

[19] W. Quan, , A. D. Pimentel. "A Hybrid Task Mapping Algorithm for Heterogeneous MPSoCs." *ACM Transactions on Embedded Computing Systems (TECS)* vol. 14, no. 1, 2015.

[20] H. Orsila, E. Salminen, and T. Hämäläinen, "Parameterizing simulated annealing for distributing Kahn Process Networks on multiprocessor SoCs," *In Proc. of International Symposium on System-on-Chip (SOC),* pp. 19-26, 2009.

[21] 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.

[22] Witten, I.H., Frank, E., Trigg, L., Hall, M., Holmes, G. & Cunningham, S.J. Weka: Practical machine learning tools and techniques with Java implementations. (Working paper 99/11), University of Waikato, Department of Computer Science, Hamilton, New Zealand, 1999.

[23] W. Quan, A. D. Pimentel, "Exploring Task Mappings on Heterogeneous MPSoCs using a Bias-Elitist Genetic Algorithm," *in Euromicro Conference on Parallel, Distributed and Network-Based Processing (DSD),* pp. 655-658, 2014.

[24] S.R. Safavian, D. Landgrebe, "A Survey of Decision Tree Classifier Methodology," *IEEE Transactions on Systems Man and Cybernetics*, vol. 21, no. 3, pp. 660- 674, 1991.

[25] A. K. Suykens, and J. Vandewalle. "Least squares support vector machine classifiers." *Neural processing letters* vol. 9, no. 3, pp. 293-300, 1999.

[26] D.W. Hosmer, and S. Lemeshow, "Introduction to the logistic regression model," *Applied Logistic Regression, Second Edition*, pp.1-30, 2000.

[27] I. Rish, "An empirical study of the naive Bayes classifier." *In IJCAI workshop on empirical methods in artificial intelligence*, vol. 3, no. 22, pp. 41-46, IBM New York, 2001.