8th IEEE International Symposium on Industrial Embedded Systems (SIES 2013)

Multi-ASIP Platform Synthesis for Real-Time Applications

Laura Micconi, Deepak Gangadharan, Paul Pop and Jan Madsen
Technical University of Denmark
Lyngby, Denmark
Email: {lmic, dega, paupo, jama}@dtu.dk

***Abstract*—In this paper we are interested in deriving a distributed platform, composed of heterogeneous processing elements, targeted to applications that have strict timing constraints. We consider that the platform may use multiple Application Specific Instruction Set Processors (ASIPs). An ASIP is synthesized and tuned for a specific set of tasks (i.e., a *task cluster*). During design space exploration (DSE), we evaluate each platform solution visited in terms of its cost and performance, i.e., its ability to execute the applications such that they meet their timing constraints. To determine if the applications are schedulable, we have to know the worst-case execution time (WCET) of each task. However, we can determine the WCETs only after the ASIPs are synthesized, which is time consuming and therefore cannot be done during DSE. To address this circular dependency (the ASIPs depend on the task clustering, and the WCETs of tasks, used to determine schedulability, depend on how ASIPs are synthesized), we propose an uncertainty model for the WCETs, which captures the range of possible ASIP implementations. Based on this model, we synthesize a multi-ASIP platform, such that the applications have a high chance of being schedulable and the cost constraints imposed on the platform are fulfilled. We propose an Evolutionary Algorithm-based approach, which uses a novel stochastic schedulability analysis to solve this optimization problem. The proposed approach has been evaluated using several benchmarks.**

I. Introduction

Current embedded platforms are increasingly executing a wide variety of applications from the automotive, multimedia and networking domains. Flexibility and performance are the key design constraints for these platforms. General Purpose Processors (GPPs) are flexible platforms and run applications from various domains, but they fall behind on performance in comparison to Application Specific Integrated Circuits (ASICs). On the other hand, ASICs are designed to run specific applications and therefore lack flexibility. ASIPs combine the best of both worlds by incorporating application specific custom instructions, thereby giving more flexibility than ASICs and better performance than GPPs. ASIPs are designed such that they are optimized to run a specific set of functions. Increasingly, multiple ASIPs are used in distributed embedded platforms together with heterogeneous processing elements (PEs) for the implementation of real-time systems (especially image/video processing systems) [1]–[4].

Platform synthesis is challenging for real-time applications when we consider multiple ASIPs in the platform solution (Fig. 1a). During DSE, we can evaluate the schedulability of the candidate platform solutions only if the WCETs of the tasks are known. However, it is possible to know the WCETs only after all the ASIPs are synthesized. The synthesis of an ASIP (Fig. 1c) starts with the decision on which tasks have to be implemented by the ASIP also called *task clustering*. Depending on the number of tasks and ASIPs included in the platform, a very large number of task clusters have to be evaluated during platform DSE. The microarchitecture synthesis of a *single* ASIP [5] involves a number of steps as shown in Fig 1c. Further, the design space of ASIP microarchitecture is very large (see Fig. 1b) depending on the number and data widths of registers (RF) and memory blocks (MEM) and number of functional units (FU). Hence, platform synthesis with multiple ASIPs is non-trivial as it needs to take the design space of ASIP microarchitecture into consideration when exploring various platform solutions.

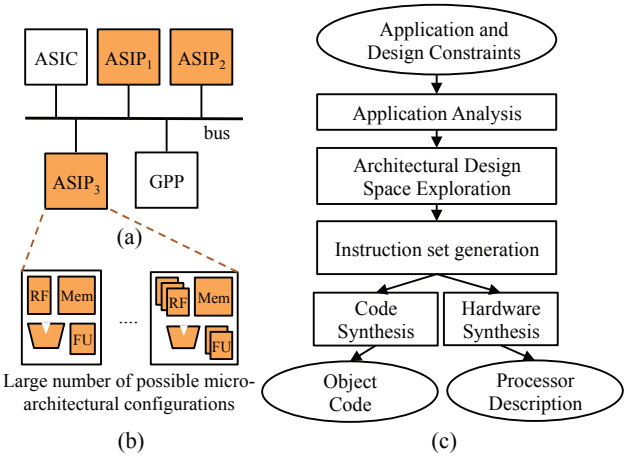


Fig. 1. (a) Multi-ASIP Platform Example, (b) ASIP Microarchitectural Design Space, and (c) Example of ASIP Microarchitecture Synthesis Flow [5]

There has been a significant effort in the development of platform synthesis methods.

1) Firstly, there are platform synthesis approaches that do not consider ASIPs. There is a large body of work in this category [6]–[8] and the assumption is that the details of each component are known.
2) Secondly, there are platform synthesis approaches, which consider multiple ASIPs. Most of these approaches [9]–[11] assume that the ASIPs have been synthesized, whereas in [3], [12], a small set of microarchitectural configurations is considered. Hence, these approaches severely limit the design space, disregarding very good solutions because they do not

978-1-4799-0658-1/13/$31.00

take into account the ASIP microarchitecture design space during platform synthesis.

3) To the best of our knowledge, there is no work on platform synthesis with multiple ASIPs, where the ASIPs are not synthesized beforehand.

All prior works address the circular dependency between the ASIP microarchitecture and WCET values by considering that the ASIPs or a limited set of microarchitectural configurations of the ASIP are given. However, this discards potentially very good solutions. In this paper, we address this circular dependency using a WCET uncertainty model that captures a wide range of ASIP microarchitectural configurations. We propose a platform synthesis approach that uses the WCET uncertainty model to derive a platform with heterogeneous processing elements (PEs) including multiple ASIPs, such that the applications have a high probability of meeting their timing constraints under given cost requirements. We use an Evolutionary Algorithm (EA) based approach to solve this optimization problem. Our platform synthesis decides the clustering of tasks into ASIPs and, after DSE, when this clustering is decided, we use existing ASIP synthesis tool flows [5] (see Fig. 1c) to derive the microarchitecture of each ASIP.

The paper is organized as follows: Section II describes the system model and the WCET uncertainty model (*UM*), Section III defines the platform synthesis problem and highlights the main challenges using a motivational example. The EA implemented for performing the DSE and its evaluation, using several benchmarks, are respectively presented in Sections IV and V, while the conclusions are in Section VI.

II. System Model

In this paper we are targeting heterogeneous platforms, in which several hardware components are interconnected through a bus[1] (Fig. 1a). The platform may contain PEs such as GPPs, ASICs, digital signal processors (DSPs) or ASIPs. Some of these might be legacy components. We consider the specific case in which the platform includes multiple ASIPs. The *k-th* PE is denoted by PE_k. We denote the frequency of PE_k as f_{PE_k}. We explore different types of buses during the platform synthesis. A bus is defined as $b_w^{f_b}$, where w is the bus width and f_b is its frequency. We use a non-preemptive static scheduling policy for the execution of tasks on the platform.

A. ASIP Synthesis

Fig. 1c highlights the main steps for the synthesis of a single ASIP, according to [5]. Other ASIP synthesis approaches, e.g., using LISATek Toolkit are [13]–[15]. To be able to perform ASIP synthesis, we need to know which tasks are assigned to it. After the analysis of the code of these tasks, a microarchitectural DSE is performed in order to synthesize an ASIP compliant with the input constraints (e.g., performance, power, cost). The starting point for this exploration is an ASIP template (selected from a library) that most likely will satisfy the tasks' characteristics (according to the designer and an application profiling). The definition of a specific ASIP microarchitecture includes the identification of the appropriate number/type of functional units, memory, issue slot, etc., in order to satisfy the functionalities required by the tasks assigned to the ASIP. After an initial microarchitecture is defined, the instruction set is generated, which can include custom instructions. The next steps are the generation of the code and the HW synthesis of the ASIP. This design flow is not fully automated and it can take one or several days to complete.

[1]Network-on-Chip will be considered in our future work.

B. Application Model

We assume that we have multiple applications A_i, each of them modeled as a task graph $A_i(V_i, E_i)$ similar to [16] where each vertex in V_i represents a task τ_j (block of partitioned code) and the edges in E_i represent data dependencies. The data dependencies are modeled by *messages* $m_g \in E_i$. With static scheduling, a task can start only after all its input messages have arrived. Fig. 2a shows a task graph example. The task graph captures the task-level parallelism in the application. Each application has a deadline d_i and a period T_i.

C. Evaluation of task clustering solutions

Consider the example with three tasks τ_1, τ_2 and τ_3 for which a multi-ASIP bus-based platform has to be synthesized. We define the set of tasks, used at the input of the flow in Fig. 1c to synthesize an ASIP PE_k, as the *task cluster* S_k. Let us now consider two different ways of clustering the three tasks: case (a) and (b). In case (a), tasks are assigned to three different ASIPs, PE_1, PE_2 and PE_3, with the corresponding clusters $S_1 = \{\tau_1\}$, $S_2 = \{\tau_2\}$ and $S_3 = \{\tau_3\}$. The single ASIP synthesis flow (Fig. 1c) can be followed and each ASIP microarchitecture can be optimized to satisfy the particular task. Now that we have each ASIP synthesized for case (a), we can determine the WCET of each task. Let us denote with $C_j^{PE_k}$ the WCET of task τ_j on ASIP PE_k. Although an ASIP PE_k has been synthesized and tuned for a certain set of tasks S_k, it could also run a task $\tau_j \notin S_k$ (if binaries are compatible). Let us consider that we run τ_2 on PE_1, tuned for τ_1. The microarchitecture of PE_1 has not been specifically tuned for the functionality of τ_2, as a consequence we can expect that $C_2^{PE_2} \leq C_2^{PE_1}$ (note that the increase in $C_2^{PE_1}$ also depends on the similarity between τ_1 and τ_2).

In case (b), τ_1 is clustered with τ_2 such that $S_1' = \{\tau_1, \tau_2\}$ and $S_2' = \{\tau_3\}$. Since the task clustering has changed, we need to re-synthesize all the ASIPs from case (a), otherwise the WCET of τ_2 may increase too much. This will again impact the WCETs of the tasks. In our case, the WCET of τ_2 will decrease and that of τ_1 might increase, depending on how much the ASIP synthesis will satisfy the functionality required by one task compared to the other.

This example shows that every time a task clustering changes, the WCETs will change. For every WCET change, we need to evaluate again the schedulability of the applications. However, we cannot know the WCET of a task before we have synthesized the corresponding ASIP. Therefore, after each re-clustering decision we would have to run a complete ASIP synthesis flow, as described in Section II-A, for each affected ASIP. As mentioned, an ASIP synthesis takes days, so it cannot be done during the DSE at the platform level. Hence, to perform DSE during platform synthesis, we need the

WCETs to evaluate the schedulability, and WCETs can only be known after the platform has been fully synthesized. This circular dependency drastically limits the number of platform alternatives that can be considered during DSE. To address this dependency, and thus enable a fast evaluation of more platform alternatives, we propose an *Uncertainty Model* for the WCETs, presented in the next subsection.

D. Modeling WCET uncertainties

As mentioned in the previous section, the WCET value depends on the ASIP microarchitecture, which is synthesized depending on how tasks are clustered. In this paper we propose a model to capture the design space of possible ASIP microarchitecture implementations: the WCET of each τ_j is modeled as a stochastic variable C_j and the associated probability distribution function. Such uncertainty models are used in practice in the early design stages [17].

Note that the variability of the *worst-case* execution time C_j of a task τ_j is due to the variation among the possible ASIP implementations on which task τ_j will run, and does not reflect the variation in execution time, which is due to variation in the input data and modern architectural features, e.g., pipelines, branch prediction. The final implementation of the ASIP running τ_j will only be available after the time-consuming ASIP microarchitecture synthesis. We use the probability distribution of C_j during DSE in order to avoid synthesizing every ASIP microarchitecture resulting from a change in task clustering.

We assume that the designer captures the probability distribution function of the WCET C_j of a task τ_j using two bounds: the smallest WCET value C_j^l (lower bound) and the largest value C_j^u (upper bound). The designer can arrive at these two values based on his or her knowledge of the functionality of the task and the possible range of ASIP microarchitectures. These values can also be estimated; the lower WCET bound can correspond to the expected WCET when τ_j is executed on an ideal processor according to an as soon as possible (ASAP) scheduling without architectural constraints. Instead, the upper WCET bound can correspond to a sequential execution of τ_j on the slowest ASIP as possible. Within these two values, we use a normal distribution for C_j that models the WCETs of the task executing on an undefined ASIP that has not been synthesized yet. Section III-B shows an example of uncertainty model.

More formally, the cumulative distribution function (CDF) F_j of C_j is denoted as $F_j = P(C_j \leq x)$, which is the probability of how many ASIP configurations lead to the task WCET C_j smaller than a value x. The distribution is built such that $P(C_j \leq C_j^u) \approx 1$. This means that task τ_j will finish in C_j^u time units or less on *all* possible ASIP microarchitecture configurations. At the same time, we also assume that $P(C_j \leq C_j^l) \approx 0$[2]. This means that according to the designer's evaluation, *none* of the possible microarchitecture configurations will finish faster than C_j^l time units. Fig. 7(a) shows an example of CDFs for three different tasks. Regarding messages, we assume that we know the size (in bits) of each message m_g. In this paper, we consider bus-based systems, and our DSE can explore different types of buses (for frequency and data width). Hence, we know

[2]The CDF of the normal distribution does not reach the one and zero values, therefore we use values that approximate them.

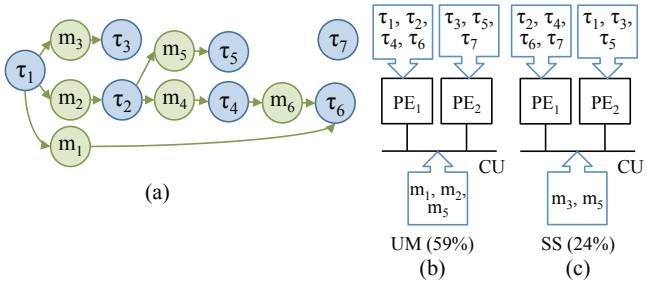


Fig. 2. Comparison between *UM* and *SS* approaches

the worst-case transmission time C_{m_g} of each message m_g. C_{m_g} is a single value and *not* a stochastic variable: for each type of bus that we want to explore, we have a different C_{m_g}.

III. Problem Formulation

Given a set of applications A_i (see Section II-B), a set of legacy components (ASICs, DSPs) and a platform cost constraint PC_{max}, the problem is to synthesize a system-level multi-ASIP platform, such that the probability of having a schedulable implementation is maximized under the specified cost constraint PC_{max}.

Synthesizing a system-level platform means performing DSE to decide the clustering of tasks and the interconnection. Our uncertainty model takes as input the task graphs of the applications, their deadlines and the cost constraint PC_{max} that is defined as the maximum number[3] of ASIPs that can be included in the platform. Moreover we can consider a library of buses with different speed and bandwidth, from where the DSE selects the appropriate bus. There can be a set of legacy components that have to be used in the architecture and it is also possible that some tasks might be clustered on some specific PEs by the designer. Our optimization takes these constraints into account. The designer, based on his/her knowledge or an analysis of the task code, provides the upper and lower bounds for the WCET as presented in Section II-D. Given the size in bytes of each message and the library of buses, it is possible to estimate the communication time for each message. The DSE evaluates different clustering solutions as presented in Section IV-A and selects the one which maximize the probability of having a schedulable implementation. After DSE, we use ASIP synthesis flow (Fig. 1c) to synthesize an ASIP for each task cluster. At the output of our multi-ASIP platform synthesis approach, we get a platform architecture, consisting of several ASIPs and possibly also legacy components, and their interconnection. For each ASIP, we have its microarchitecture, and the interconnection consists of a bus with a certain speed and bandwidth. This synthesis is performed under the platform cost constraint PC_{max}. Fig. 3 shows our flow for the synthesis of a multi-ASIP platform.

A. Motivational Example

Given an application task graph (as in Fig. 2a), we want to synthesize the platform architecture with multiple ASIPs such that the probability of having a schedulable implementation is maximized under the platform cost constraint. Two extreme

[3]We will consider the ASIP area cost in our future work.

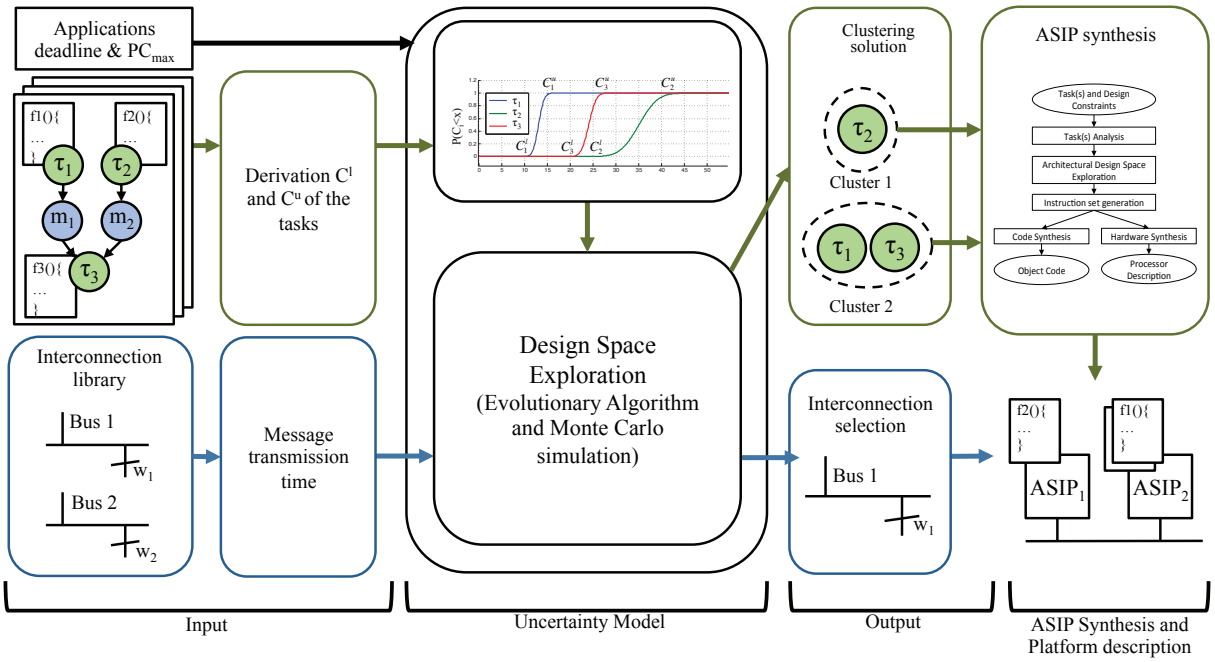


Fig. 3. Multi-ASIP platform synthesis flow

possibilities for clustering of tasks are having one ASIP for each task or having a single ASIP for all tasks. These two extremes are infeasible as the latter has a small likelihood to meet the performance requirements, whereas the former it may be a better solution performance wise (if the overall communication overhead does not exceed the task execution overhead), but it is too costly. Our problem is to determine that clustering solution, which maximizes the probability of having a schedulable implementation, under a given platform cost constraint.

In our approach, we use the uncertainty model (from Section II-D) to break the circular dependency between ASIP microarchitecture and the WCET of a task. We denote our proposed synthesis approach, which uses this Uncertainty Model as *UM*.

Without such an uncertainty model, the only option for a designer would be to characterize the WCET of each task τ_j with a reference value denoted with C_j^{ref} in Table I. This value can be obtained executing the task on a template processor. We denote this straightforward solution as *SS*. We use the *SS* approach as a reference to compare the results obtained with our *UM* approach. We have used the following approach to derive the values in Table I for the example in Fig. 2a. We have considered a simple functionality, consisting of a loop and operations such as multiplication and addition, for the tasks in Fig. 2a. We have used the Silicon Hive ASIP synthesis tool flow [18] (now part of Intel Corp.) to synthesize the ASIPs. We have synthesized a three-issue slot Very Long Instruction Word (VLIW) ASIP, and we have analyzed the tasks to obtain their WCETs. We have considered this WCET value as the reference WCET C_j^{ref}. Furthermore, we have varied the microarchitecture of this ASIP to obtain two extremes. The WCETs on the slowest ASIP thus obtained were considered the upper bound C_j^u, whereas the WCETs on the fastest ASIP synthesized were considered the lower bound C_j^l. The obtained values are presented in Table I. We consider the platform to use at most two ASIPs (platform cost $PC_{max} = 2$). Each platform solution (consisting of a certain clustering of tasks) is evaluated using the schedulability analysis from Section IV, which gives the probability p_i of a solution to be schedulable, once it is implemented. We perform an exhaustive DSE and the best clustering solution obtained with *UM* is shown in Fig. 2b, having a $p_i = 59\%$. Then, we perform an exhaustive DSE of all possible clustering solutions with the *SS* approach, aiming at minimizing the schedule length, considering the given C_j^{ref}. The clustering obtained with *SS* is shown in Fig. 2c. In order to compare the solutions obtained with the *UM* and *SS* approaches, we calculate the probability p_i of the SS solution to be schedulable using the WCET uncertainty model, as in the *UM* approach. Thus p_i for the solution with *SS* approach is 24%.

To validate the conclusions of the comparison between *UM* and *SS*, we have synthesized the platform solutions in Fig. 2b and 2c, produced by *UM* and *SS*, respectively. Next, we have determined the WCETs C_j of each task τ_j on their respective ASIP. Then, we have obtained the optimal schedule lengths for the two cases. The schedule length in the case of the *UM* platform solution is $1,955\ \mu s$, whereas for *SS* is $2,275\ \mu s$. For a deadline of $2,000\ \mu s$, *UM* solution is schedulable, while *SS* solution is not. This confirms that if a *UM* solution has higher chances to be schedulable compared to a *SS* solution according to our evaluation (Section IV-A), this is also true in the final implementation, as our synthesis using the Silicon Hive tools has shown. The comparison of the two approaches shows that with our *UM* approach, we are able to identify a solution that has a higher probability to be schedulable, once it is implemented.

TABLE I. C VALUES FOR THE MOTIVATION EXAMPLE (IN μs)

C	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7
C_j^u	702	3437	8801	702	702	702	3437
C_j^l	450	180	300	450	450	450	180
C_j^{ref}	602	3237	400	602	602	602	3237

TABLE II. MICRO-ARCHITECTURE FEATURES EXPLORED

Task	*Issue width*	*num. ALU*	*num. MUL*	*RF size*	*Data Cache (KB)*	*Data Cache Line (bytes)*	*Load slot*	*Store slot*
mp3 decoder	1,2,3, 4,5,6, 7,8	4,5,6, 7,8	2,3,4, 5,6,7, 8	32, 64	4, 8, 16	16, 32, 64, 128	4	2
jpeg decoder	2,3,4, 5,6,7,8	4, 5,6,7, 8	1,2,3, 4,5,6, 7,8	16, 32, 64	4, 8, 16	16, 32, 64	4	2

B. WCET Uncertainty Example

We have validated the proposed WCET uncertainty model through several experiments. Here, we show the results for two tasks of different size and complexity: a mp3 decoder, part of the MAD library [19] and a jpeg decoder [20] task. We are interested to determine how the WCET of these tasks varies depending on the micro-architecture features, and if our WCET uncertainty model proposed in Section II-D is able to capture this variation.

Hence, we have run these tasks on a VLIW architecture similar to the ASIP architectures considered in this paper. We have used the VLIW Example (VEX) [21], which is a VLIW compiler and simulator developed at HP Laboratories. VEX is highly configurable; we have used a set of configurations which captures the variability of a microarchitecture design, considering the features of VLIW processors available on the market and the characteristics of the tasks considered. Table II presents the microarchitecture design space used for the experiments. Thus, we varied the number of arithmetic and logic units (ALU), multipliers (MUL), registers in the register file (RF), the issue, load and store slots, the data cache size and the data cache line size. Within the parameters in Table II we considered a large number of micro-architecture configurations: 1,632 for the mp3 decoder task and 1,068 for the jpeg decoder task.

For each micro-architecture configuration, we compiled and ran the respective task. VEX returns a number of cycles, and considering the microarchitecture explored, we assumed a frequency of 100 MHz in order to calculate the execution time in *ms*. For a particular microarchitecture, we have considered as WCET the largest value of the execution time, after extensive simulations with multiple input files. We know that such a value does not represent the WCET, which is a theoretical upper bound determined through analysis, but we believe this value is a good approximation for our experiments. The results for the mp3 decoder are presented in Fig. 4a and 4b and those for jpeg in Fig. 4c and 4d. Fig. 4a and 4c present with bars the probability distribution function of the WCET of mp3 and jpeg tasks, respectively, obtained after our experiments. We used a fitting function of MATLAB to determine the probability distribution type that better approximates these WCET values: it is possible to observe that a Normal distribution is a reasonable approximation (represented with a continuous red line in Fig. 4a and 4c). The corresponding cumulative distribution functions (CDF) are plotted in Fig. 4b and 4d. Each figure shows two CDF curves: the CDF resulted after experiments (depicted with a continuous blue line) and the CDF obtained by using our model (the green dotted line). Our WCET model (the green dotted CDF) was obtained as explained in Section II-D, considering a Normal distribution between a lower bound C^l and a upper bound C^u of the WCET. As mentioned, we assume that these bounds are provided by the designer by evaluating two extreme microarchitectures from the range considered for the mp3 and jpeg tasks. The microarchitecture corresponding to the upper and lower bounds of the WCET for the two tasks are summarized in Table III. Our WCET model validation experiments have shown that the WCET has a Normal distribution and that our proposed uncertainty model is a good approximation. Note that the CDF of our model leads to more pessimistic (larger) WCETs compared to experimental measurements. However, the WCETs produced by our experiments might be optimistic (smaller), since they are not a theoretical upper bound obtained through analysis. It is important to mention that the proposed WCET uncertainty model is used only for design space exploration, and not for providing timing guarantees.

TABLE III. MICROARCHITECTURES ASSOCIATED TO THE WCET UPPER AND LOWER BOUNDS

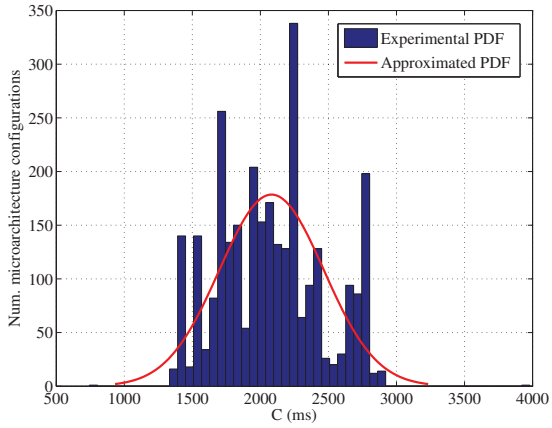
Task	*WCET*	*Issue width*	*num. ALU*	*num. MUL*	*RF size*	*Data Cache (KB)*	*Data Cache Line (bytes)*	*Load slot*	*Store slot*
mp3 decoder	C_l	8	8	8	64	16	128	4	2
	C_u	1	4	2	32	4	16	4	2
jpeg decoder	C_l	8	8	8	64	16	64	4	2
	C_u	2	4	1	32	4	16	4	2

IV. PLATFORM SYNTHESIS USING AN EVOLUTIONARY APPROACH

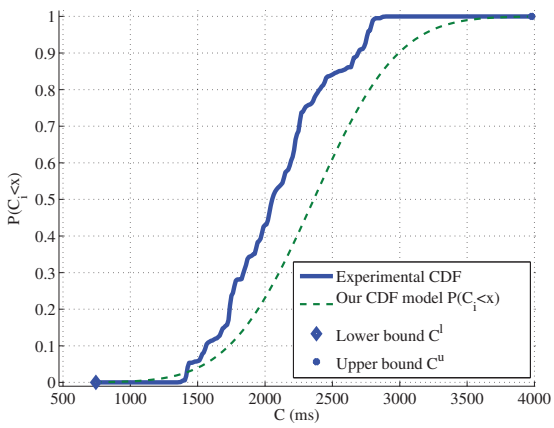
The design space of all possible clustering solutions can be huge, so an Evolutionary Algorithm (EA) has been implemented. The objective function guiding the exploration is the maximization of the probability p_i for a certain clustering solution to meet the deadline, under a platform cost constraint PC_{max}. Section IV-A presents how a single clustering solution is evaluated, while Section IV-B presents the EA used for the DSE.

A. Schedulability Analysis

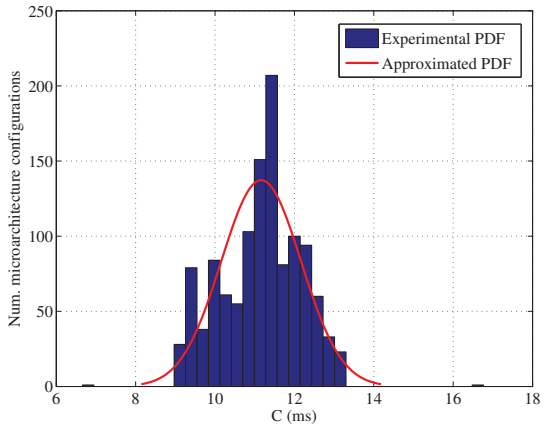
We use a non-preemptive static scheduling policy. If the WCET of each task is known, it is possible to determine the schedule table and thus perform a schedulability check. This cannot be done in our case, as the WCET is expressed by a stochastic variable. Instead, we perform a schedulability analysis of each clustering solution and use this as the basis for calculating the objective function during DSE. Note that the analysis presented in this section is only used to guide the search, not to provide schedulability guarantees. We assume that a detailed schedule table will be built during the later design and development stages (when more accurate information about WCETs and the ASIP microarchitecture is available) to check the schedulability of an implementation.



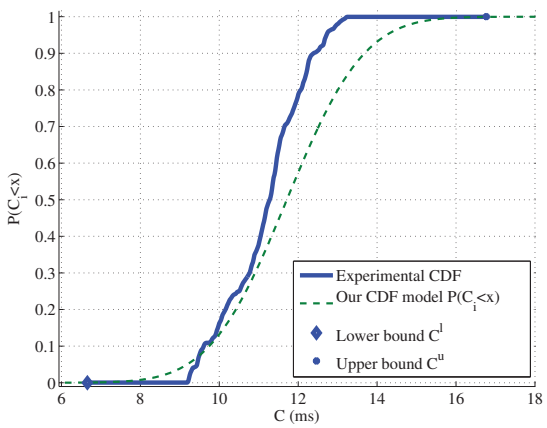
(a) Histogram of the probability density function of the C_i obtained with VEX for the mp3 decoder task



(b) Comparison of our proposed CDF model ($P(C_i < x)$) with the simulation results for mp3 decoder task



(c) Histogram of the probability density function of the C_i obtained with VEX for the jpeg decoder task



(d) Comparison of our proposed CDF model ($P(C_i < x)$) with the simulation results for jpeg decoder task

Fig. 4. Probability density function and cumulative distribution function for mp3 and jpeg decoder tasks

With static cyclic scheduling, δ_{A_i} is the length of the schedule table for A_i. As previously mentioned, we cannot calculate the schedule table of a clustering solution, but we can perform a schedulability analysis to determine the probability of having a schedule table length that meet the deadline. Thus, the probability of A_i to meet the deadline d_i is defined as $P(\delta_{A_i} \leq d_i)$. The work done in [22] presents how to determine δ_{A_i} in case of stochastic execution times (WCETs in our case), using an analytical approach that relies on the assumption of independence between the starting and finishing time of each task/message. This assumption in our case does not hold, so we substitute the analytical method with Monte Carlo simulation (*MCS*), which, in addition, is able to take into account the task dependencies and provide more precise results.

The calculation of δ_{A_i} is done considering certain fixed values for the WCETs C_j. However, a WCET is a stochastic variable modeled by the CDF F_j. Our approach is to use the MCS to decide randomly, in each iteration, a new value for C_j based on its CDF F_j. We collect all the values of δ_{A_i} thus determined, and we calculate the probability $p_i = P(\delta_{A_i} \leq d_i)$ of an application A_i to be schedulable. As a *MCS* is known for being time consuming, a reduced number of samples has been used to speed up the execution. The results obtained with 5,000 iterations have then been compared to results obtained with 50,000 iterations, and the difference in the value of p_i between them was less than 3%. We have also sped up the *MCS* by moving the random generation of WCET values outside of the DSE.

We use an approach similar to As Soon As Possible (ASAP) [23] scheduling to calculate δ_{A_i}. Let us illustrate how δ_{A_i} is obtained using the application from Fig. 5a and the platform implementation solution from Fig. 5b. The clustering solution in Fig. 5b uses two ASIPs, PE_1 and PE_2. Messages are assigned to the bus (CU). We start by identifying a layer subdivision [22] of the task graph, see Fig. 5c. If communicating tasks are on the same PE, the communication cost is ignored. Each task/message has a priority assigned (tasks on the critical path get assigned a higher priority [24]). A layer identifies the tasks/messages of the applications A_i that can be executed in parallel, i.e. that have no data dependency

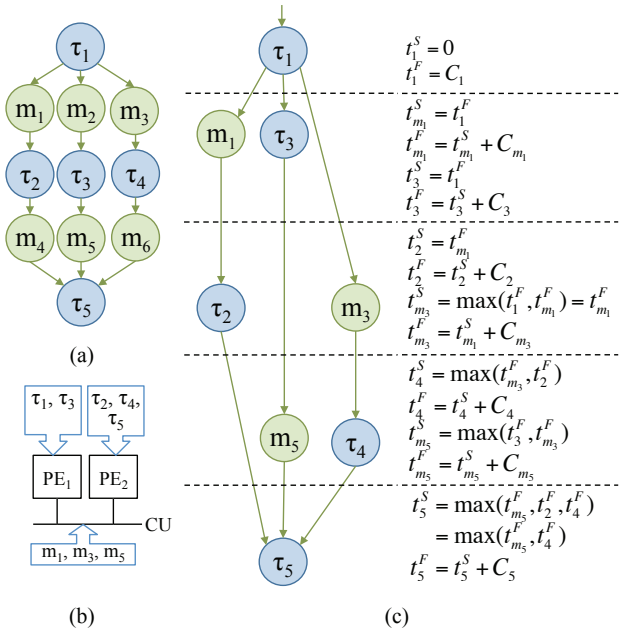


Fig. 5. Example of δ_{A_i} calculation

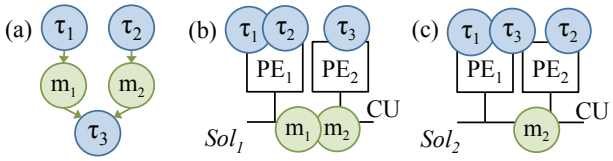


Fig. 6. Example of evaluation of clustering solutions

and are clustered on different HW resources (*PE*s and *CU*s). When tasks/messages are clustered on the same HW resource and there is a contention of the same layer, their execution order is set according to their priority. Once the layers are identified (indicated by the dotted lines in Fig. 5c), starting from the first layer, the starting time t_j^s ($t_{m_g}^s$) and finishing time t_j^f ($t_{m_g}^f$) of each task τ_j (and message m_g) are calculated. They are obtained combining the C_j of the tasks and C_{m_g} of the messages, as follows. The starting time t_j^s of a task τ_j is given by the maximum of the finishing times of all the tasks that τ_j depends on. If τ_j has no data dependencies or HW resource contention, its starting time is zero. The finishing time t_j^f of τ_j is given by the sum of the estimated starting time and WCET of task τ_j, i.e., $t_j^s + C_j$. Finally, the maximum finishing time among the terminal tasks (tasks without successors) in an application A_i corresponds to δ_{A_i}, the end-to-end response time. In Fig. 5c the computations performed at each layer are described.

For example, let us consider the task graph in Fig. 6a with the WCET CDFs from Fig. 7a. Two different clustering solutions Sol_1 (Fig. 6b) and Sol_2 (Fig. 6c) are evaluated in term of p_i, as presented in this section. The results are shown in Fig. 7b: Sol_1 has a probability p_i of 2% while Sol_2 of 93%. This indicates that the second clustering is much more likely to meet the application deadline when the platform is synthesized.

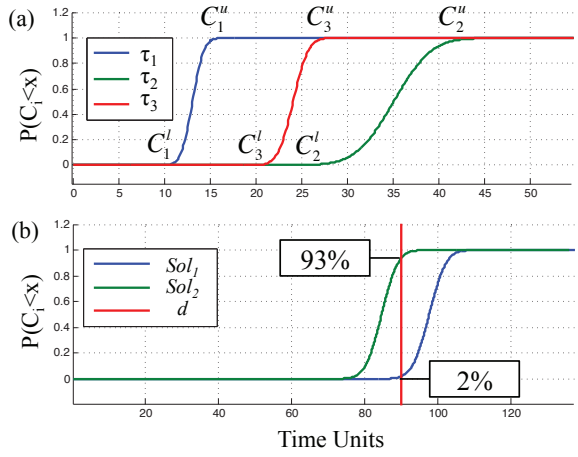


Fig. 7. (a) Input and (b) output CDF for the example in Fig. 6

B. Evolutionary Algorithm

We use a Steady State Evolutionary Algorithm (SSEA) [25] to decide the clustering of tasks. SSEA takes as input the applications (including the uncertainty model), the legacy components and task assignment constraints, and the maximum number of ASIPs allowed, PC_{max}. The algorithm returns that clustering solution, which maximizes the schedulability probability p for all applications, i.e., $p = (\sum p_i)/n$ where n is the number of applications, under the given cost constraint PC_{max}. SSEA is inspired from the process of natural evolution, where a set of solutions is called a *population* and each solution is encoded using a string called a *chromosome*. The population is evolved by performing recombination and mutation, and the population is replaced with the *offspring* population, which has better *fitness* according to the cost function. SSEA has been chosen because it is suitable for the case when the computation of the cost function is time-consuming (a small portion of the population is replaced at each new generation). The algorithm works by adding the offspring of the individuals selected from each generation to the pre-existing one, so individuals are retained between generations.

We defined the chromosome (a single clustering solution) as an array of tasks and messages; the value of each element (*gene*) represents the identifier of the *PE* or *CU* on which the tasks and messages are respectively clustered. We defined a custom crossover operator with the purpose of maximizing the task level parallelism while generating the new offspring individuals, i.e. we favor the transmission of the genetic material of the parent with a higher degree of parallelism. This is done partitioning the parents' chromosomes according to the layer subdivision presented in IV-A. The parent chromosome with the higher task level parallelism (number of tasks/messages in the same layer) is used as starting point and it is partitioned and distributed among the two offspring solutions. The offspring solutions are then filled with the other parent's genetic material. To complete the generation of the offspring chromosomes, we identify their layer subdivision and we perform a rotation of the *PE* assigned to each task within the same layer. The mutation operator is used to randomly vary the *PEs* and *CUs* on which the tasks and messages are clustered. The parameters used for the execution of the

SSEA are: crossover probability P_c, mutation probability P_m and the population size Pop. SSEA finishes when a given time-limit has been reached. The tuning of these parameters has been done running multiple executions of the algorithm with different synthetic applications.

V. Experimental Evaluation

For the experimental evaluation of our platform synthesis approach, we used both realistic and synthetic benchmarks. Thus, we used two subsets of applications from E3S benchmarks [26], taken respectively from the telecommunication (*telecom-cords*, *TLC*) and automotive/industrial (*auto-indust-cords*, *IND*) domains and two synthetic case studies (*Synth*), which represent a smaller and wider examples. The case study *Synth 1* is the same presented in III-A. The details of the case studies, in terms of number of applications and tasks are presented in Table IV, columns 2 and 3, respectively. All applications have a deadline d (column 4) and a platform constraint (column 5).

For each real benchmark, we considered the WCET value in the benchmark as the reference WCET value C_j^{ref}, and we have scaled this value to obtain the lower (C_j^l) and upper (C_j^u) bounds. For *Synth 1* we used the values in Table I and arbitrary values for *Synth 2*. We have run our proposed SSEA platform synthesis approach, which uses the WCET uncertainty model from Section II-D (denoted with *UM*) on each of the three benchmarks and have obtained a clustering solution. The probability p of each benchmark to be schedulable with the obtained clustering is presented in Table IV, column 6, which also presents the number of ASIPs used (column 7). The overall schedulability probability p is calculated as an average of the probability p_i of each application A_i in the benchmark.

Together with *UM*, Table IV also presents a Straightforward Solution (*SS*) (see Section III-A), which uses a reference value for WCET (C_j^{ref}) of each task τ_j. This value is then used inside the SSEA optimization, and instead of the MCS used for the evaluation of a platform alternative (Section IV-A), we use only the calculation of the end-to-end response time (δ_{A_i}), considering C_j^{ref} as the WCET. This is what a good designer would do if a WCET uncertainty model would not be available. The parameters used for the execution of the SSEA are: $P_c = 40\%$, $P_m = 20\%$ and $Pop = 100$. The execution time limit has been set to 1 hour.

As we can see from Table IV, *UM* is able to obtain much better results in terms of the probability of finding schedulable implementations compared to *SS*. This is because, using our proposed uncertainty model, we can better take into account the ASIP microarchitecture possibilities during the DSE for a multi-ASIP system-level platform. Moreover, for case study *Synth 1*, it was possible to synthesize the ASIPs and verify that the difference in the p returned by *UM* and *SS* is reflected later in the final schedule table. This proves that our DSE with *UM* is able to lead to good final implementations: a scheduling length of 1.955 *ms* for the clustering solution found with the *UM* versus the 2.275 *ms* of the one found with the *SS*. Due to time constraints it was not possible to perform the synthesis for larger benchmark and this is left for future work.

TABLE IV. Comparison of *UM* and *SS*

Case Study	No. of Apps.	No. of Tasks	d (ms)	Max. cost	*UM*		*SS*	
					p	ASIPs	p	ASIPs
TLC	4	10	5.35	3	71%	3	55%	3
IND	4	13	5.25	4	70%	4	59%	4
Synth 1	2	7	5	2	59%	2	24%	2
Synth 2	10	22	5.6	5	59%	5	49%	5

Our *UM* approach assumes that the designer provides the upper and lower bounds of the WCET of each task. Additionally, as we scale the values of the WCET to obtain the bounds in our experiments, it is necessary to observe the influence of variations of these bounds on the clusters obtained. Therefore, we performed a sensitivity analysis on C_j^l and C_j^u values of each task. For this, we used the case study *Synth 1*, which task graphs are described in Fig. 2. For each task, we considered variations from $\pm 1\%$ to $\pm 5\%$ of C_j^l and C_j^u values. In particular, we considered a total of 70 cases in which all tasks or a subset of them are suffering variations. These changes in the bounds will reflect in slightly different input CDFs for our DSE. We have run our DSE for each of these cases, and we have obtained the same clustering solution, which means that our DSE is not sensitive to small variations in the WCET estimation provided by the designer.

VI. Conclusion

In this paper we have proposed an approach for the synthesis of multi-ASIP platforms for real-time applications. The synthesis of an ASIP starts from a cluster of tasks. We have developed an evolutionary algorithm for deciding the clustering of tasks to ASIPs, such that the applications have a high chance of meeting their deadlines, and the imposed platform cost is satisfied. We have proposed an uncertainty model for the WCET values to capture the range of the possible ASIP microarchitectural implementations. Using this WCET model, we have outlined an architecture evaluation heuristic based on Monte Carlo Simulation, which calculates the probability of a platform solution to meet the deadlines. As the experimental results show, by considering the range of possible ASIP microarchitectural implementations during the design space exploration for multi-ASIP platform synthesis, we can obtain platform solutions that have a high chance of being schedulable. Moreover for a simple case study, we also concluded that our design space engine is not sensitive to slight variations of the upper and lower bounds used as input for our model.

Acknowledgements

The work on this paper has been performed in the scope of the ASAM project of the European ARTEMIS Research Program and has been partly supported by the ARTEMIS Joint Undertaking under grant no. 100265.

References

[1] H. C. Doan, H. Javaid, and S. Parameswaran, "Multi-asip based parallel and scalable implementation of motion estimation kernel for high definition videos," in *ESTImedia*, 2011, pp. 56–65.

[2] S. Saponara, L. Fanucci, S. Marsi, and G. Ramponi, "Algorithmic and architectural design for real-time and power-efficient retinex image/video processing," *J. Real-Time Image Processing*, vol. 1, no. 4, pp. 267–283, 2007.

[3] H. Javaid and S. Parameswaran, "Synthesis of heterogeneous pipelined multiprocessor systems using ilp: jpeg case study," in *Proceedings of the 6th IEEE/ACM/IFIP International conference on Hardware/Software codesign and system synthesis*. New York, NY, USA: ACM, 2008, pp. 1–6.

[4] L. Jozwiak and M. Lindwer, "Issues and challenges in development of massively-parallel heterogeneous mpsocs based on adaptable asips," in *Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on*, 2011, pp. 483–487.

[5] M. K. Jain, M. Balakrishnan, and A. Kumar, "ASIP design methodologies: Survey and issues," in *Proceedings of the IEEE / ACM International Conference on VLSI Design*, 2001, pp. 76–81.

[6] T. Kangas, P. Kukkala, H. Orsila, E. Salminen, M. Hännikäinen, T. D. Hämäläinen, J. Riihimäki, and K. Kuusilinna, "UML-based multiprocessor SoC design framework," *ACM Transanctions on Embedded Computing Systems*, vol. 5, pp. 281–320, 2006.

[7] 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*, vol. 27, no. 3, pp. 542–555, 2008.

[8] A. Kumar, S. Fernando, Y. Ha, B. Mesman, and H. Corporaal, "Multiprocessor systems synthesis for multiple use-cases of multiple applications on FPGA," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 13, no. 3, pp. 40:1–40:27, Jul. 2008.

[9] O. Muller, A. Baghdadi, and M. Jézéquel, "From parallelism levels to a multi-ASIP architecture for turbo decoding," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 17, no. 1, pp. 92–102, 2009.

[10] C. Brehm, T. Ilnseher, and N. Wehn, "A scalable multi-ASIP architecture for standard compliant trellis decoding," in *International SoC Design Conference*, 2011, pp. 349–352.

[11] F. Ieromnimon, D. Kritharidis, and N. S. Voros, "Application of the mosart flow on the wimax (802.16 e) phy layer," in *Scalable Multicore Architectures*. Springer, 2012, pp. 197–223.

[12] S. L. Shee and S. Parameswaran, "Design methodology for pipelined heterogeneous multiprocessor system," in *Proceedings 44th Design Automation Conference*, 2007, pp. 811–816.

[13] K. Karuri, R. Leupers, G. Ascheid, and H. Meyr, "A generic design flow for application specific processor customization through instruction-set extensions (ises)," in *Embedded Computer Systems: Architectures, Modeling, and Simulation*, ser. Lecture Notes in Computer Science, vol. 5657. Springer Berlin Heidelberg, 2009, pp. 204–214.

[14] R. Muhammad, L. Apvrille, and R. Pacalet, "Evaluation of ASIPs design with LISATek," in *SAMOS*, ser. Lecture Notes in Computer Science, M. Berekovic, N. J. Dimopoulos, and S. Wong, Eds. Springer, 2008, pp. 177–186.

[15] A. Nohl, F. Schirrmeister, and D. Taussig, "Application specific processor design architectures, design methods and tools," in *Proceedings of the International Conference on Computer-Aided Design*, 2010, pp. 349–352.

[16] T. Pop, P. Eles, and Z. Peng, "Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems," in *Proceedings 10th International Symp. on HW/SW Codesign*, 2002, pp. 187–192.

[17] J. Axelsson, "A method for evaluating uncertainties in the early development phases of embedded real-time systems," in *Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 72–75.

[18] A. Kumar, A. Hansson, J. Huisken, and H. Corporaal, "Interactive presentation: An fpga design flow for reconfigurable network-based multi-processor systems on chip," in *Proceedings of the conference on Design, automation and test in Europe*, 2007, pp. 117–122.

[19] "MAD, MPEG Audio Decoder," http://www.underbit.com/products/mad/.

[20] T. U. o. E. T. Electronic Systems, "Mamps project, partitioned jpeg decoder algorithm," http://www.es.ele.tue.nl/mamps/example.php.

[21] J. A. Fisher, P. Faraboschi, and C. Young, "VEX, a VLIW Example," http://www.hpl.hp.com/downloads/vex/.

[22] Y. A. Li and J. K. Antonio, "Estimating the execution time distribution for a task graph in a heterogeneous computing system," in *Proceedings of the 6th Heterogeneous Computing Workshop*, 1997, pp. 172–184.

[23] R. Walker and S. Chaudhuri, "Introduction to the scheduling problem," *Design Test of Computers, IEEE*, vol. 12, no. 2, pp. 60 –69, 1995.

[24] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Comput. Surv.*, vol. 31, pp. 406–471, 1999.

[25] E. K. Burke and G. Kendall, *Search methodologies : introductory tutorials in optimization and decision support techniques*. Springer, 2005.

[26] "E3S benchmark," http://ziyang.eecs.umich.edu/ dickrp/e3s/.