



**HAL**  
open science

# Empirical Model-Based Performance Prediction for Application Mapping on Multicore Architectures

Abdoulaye Gamatié, Xin An, Ying Zhang, An Kang, Gilles Sassatelli

► **To cite this version:**

Abdoulaye Gamatié, Xin An, Ying Zhang, An Kang, Gilles Sassatelli. Empirical Model-Based Performance Prediction for Application Mapping on Multicore Architectures. *Journal of Systems Architecture*, 2019, 98, pp.1-16. 10.1016/j.sysarc.2019.06.001 . lirmm-02151502

**HAL Id: lirmm-02151502**

**<https://hal-lirmm.ccsd.cnrs.fr/lirmm-02151502>**

Submitted on 8 Jun 2019

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Empirical Model-Based Performance Prediction for Application Mapping on Multicore Architectures

Abdoulaye Gamatié<sup>a</sup>, Xin An<sup>b</sup>, Ying Zhang<sup>b</sup>, An Kang<sup>b</sup>, Gilles Sassatelli<sup>a</sup>

<sup>a</sup>*LIRMM, CNRS - University of Montpellier, 161 rue Ada, 34095 Montpellier, France*

<sup>b</sup>*Anhui Province Key Laboratory of Affective Computing and Advanced Intelligent Machine, Hefei University of Technology, No.193 Tunxi Road, 230009 Hefei, China*

---

## Abstract

Application mapping in multicore embedded systems plays a central role in their energy-efficiency. The present paper deals with this issue by focusing on the prediction of performance and energy consumption, induced by task and data allocation on computing resources. It proposes a solution by answering three fundamental questions as follows: i) how to encode mappings for training performance prediction models? ii) how to define an adequate criterion for assessing the quality of mapping performance predictors? and iii) which technique among regression and classification enables the best predictions? Here, the prediction models are obtained by applying carefully selected supervised machine learning techniques on raw data, generated off-line from system executions. These techniques are Support Vector Machines, Adaptive Boosting (AdaBoost) and Artificial Neural Networks (ANNs). Our study is validated on an automotive application case study. The experimental results show that with a limited set of training information, AdaBoost and ANNs can provide very good outcomes (up to 84.8% and 89.05% correct prediction score in some cases, respectively), making them attractive enough for the addressed problem.

*Keywords:* Resource allocation, Application mapping, Model-based performance prediction, Machine learning

---

\*Corresponding authors: A. Gamatié and X. An

*Email addresses:* [abdoulaye.gamatie@lirmm.fr](mailto:abdoulaye.gamatie@lirmm.fr) (Abdoulaye Gamatié),  
[xin.an@hfut.edu.cn](mailto:xin.an@hfut.edu.cn) (Xin An), [yingzhang9403@163.com](mailto:yingzhang9403@163.com) (Ying Zhang),  
[kangan03@mail.hfut.edu.cn](mailto:kangan03@mail.hfut.edu.cn) (An Kang), [gilles.sassatelli@lirmm.fr](mailto:gilles.sassatelli@lirmm.fr) (Gilles Sassatelli)

---

## 1. Introduction

Multicore and manycore architectures have become the *de facto* solutions to meet the energy-efficiency requirement in modern computer systems. The aim is to provide the systems with higher performance levels at the cost of minimal power consumption. Typically, for high-performance and embedded computing systems, this amounts to maximize the number of floating-point operations per second (FLOPS) and the millions of instructions per second (MIPS) respectively, per consumed Watt. Nevertheless, the advantage of multicore architectures comes with a non-trivial resource allocation challenge on which depend the energy-efficiency gains. As a matter of fact, the mapping and scheduling of both tasks and data on available processing cores and memory have a strong impact on performance and power consumption.

Existing mapping methodologies [1] adopt either design-time or runtime optimization approaches to improve the behavior of both homogeneous and heterogeneous multicore systems. At runtime, the mapping management may incur data/tasks migrations onto the available computation resources. This is orchestrated in various ways: either centralized or distributed. Generally speaking, the problem of finding optimal mapping and scheduling solutions is known to be NP-hard. Some pragmatic approaches that address this problem exploit heuristics combined with domain-specific knowledge to explore nearly optimal solutions [1]. Having the relevant information on system behavior according to variable runtime situations is one major challenge in such adaptive system management [2]. Collecting these information (e.g., CPU usage, memory and communication interconnect usage) is often tedious and intrusive to the system, especially when targeting fine-grained data.

Given the important progress made recently in machine learning techniques, particularly in deep-learning [3], we envision opportunities to apply them when dealing with application mapping in multicore systems. Machine learning has gained an increasing attention in system design, including computer architec-

30 tures [4] or compilers [5]. To predict the performance of mappings, *supervised*  
31 machine learning techniques are considered in this work. They enable to build  
32 class or value prediction models while minimizing a *loss* function denoting the  
33 prediction error percentage on the training data set. On the other hand, *un-*  
34 *supervised* machine learning techniques enable to identify clusters of similar  
35 behavior or to determine insightful feature representations from raw data sets.  
36 Beyond these techniques, which are usually applied off-line, other approaches  
37 such as *reinforcement learning* and *evolutionary algorithms* enable online learn-  
38 ing.

### 39 1.1. Context of this Study

40 We consider the dynamic resource allocation question in multicore systems,  
41 as illustrated in Figure 1. Application workloads are described by hierarchical  
42 *task graphs*, where each task consists of a *runnable* graph [6]. Runnables are  
43 basic entities defining task behaviors in terms of runtime and communication. A  
44 mapping *performance predictor* is coupled loop-wise with a *mapping heuristics*  
45 *module*, which implements typical mapping selection techniques (e.g., evolution-  
46 ary algorithms) on a given multicore *execution platform*. A component, called  
47 *workload mapper*, is in charge of applying the selected mapping decisions at  
48 runtime. It acts as a centralized processing element that realizes every mapping  
49 suggested by the *mapping heuristics module*.

50 The dynamic resource allocation question has been thoroughly covered in  
51 a recent book [7], considering application domains such as high-performance  
52 computing, cloud computing and embedded computing. Several approaches  
53 have been discussed: allocation and optimization inspired by control automation  
54 theory, search-based allocation heuristics such as genetic algorithms, distributed  
55 allocation based on swarm intelligence, and value-based allocation. These ap-  
56 proaches are typical candidates for implementing the above *mapping heuristics*  
57 *module*.

58 The performance predictors, investigated in the current work, are the ideal  
59 complements of the above *mapping heuristics module*. Indeed, the predicted

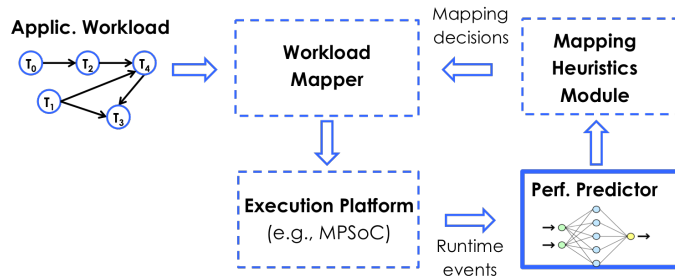


Figure 1: Dynamic resource allocation in multicore systems

60 performances, e.g., execution time, speedup or energy-efficiency, help in taking  
 61 efficient mapping decisions at runtime. Note that in place of performance pre-  
 62 dictors, alternative candidates are performance evaluation tools, such as multi-  
 63 core system simulators, analytic methods or worst-case performance estimation  
 64 methods. However, these solutions may come with an overhead in the global  
 65 execution time due to their inherent simulation time; or to their pessimistic  
 66 over-approximations. To avoid this issue, here, we rather investigate an empir-  
 67 ical approach that leverages prediction models trained on raw data generated  
 68 off-line from different system execution scenarios. The models are built with  
 69 machine learning techniques capable of extracting useful insights from system  
 70 behavior. When invoked, they are expected to predict estimates of mapping  
 71 performances in little-to-no time (e.g., for usage in fitness functions of genetic  
 72 algorithm-based heuristics). These estimates must be relevant enough to enable  
 73 the *mapping heuristics module* to take efficient decisions. While the current work  
 74 does not aim at any new mapping heuristics, its main purpose is to speedup  
 75 the decision loop shown in Figure 1, by reducing the computation complexity  
 76 associated with the *performance predictor* leveraged by the *mapping heuristics*  
 77 *module*.

## 78 1.2. Problem Formulation

79 The problem dealt with in this paper is defined as follows:

80 **Definition 1 (Mapping Performance Prediction Problem).** *Given an ap-*  
81 *plication to execute on a multicore platform, we are interested in its mapping*  
82 *issue onto the available cores. Here, the mapping is addressed at the granularity*  
83 *of the runnables. We consider machine learning techniques to predict the per-*  
84 *formance induced by the possible mapping choices, while meeting the following*  
85 *requirements:*

- 86 1. **accuracy:** *the successful prediction percentage reaches at least 80%;*
- 87 2. **feasibility:** *data used for learning are obtained at minimal and costless*  
88 *intrusion in systems;*
- 89 3. **responsiveness:** *predictions are performed in short delays.*

90 Intuitively, the above prediction issue is a regression problem, i.e., given a  
91 mapping scenario, we would like to predict its induced performance numbers.  
92 However, if we partition the domain of all possible values into sub-domains and  
93 predict the sub-domain to which the performance numbers of a given mapping  
94 scenario belong to, the above problem can be formulated then as a classification  
95 problem. Each sub-domain is seen as a class (or a label). For example, one  
96 may want to map an application according to three target performance ranges  
97 or classes: high, medium and low. A classification technique would be therefore  
98 preferred. Accordingly, if we refine the number of classes into more classes,  
99 fine-grained and more accurate predictions could be obtained.

### 100 1.3. Our Contribution

101 We address the above mapping problem by considering two off-line super-  
102 vised machine learning approaches: on the one hand, classification through  
103 *Support Vector Machine* (SVM) [8] and *Adaptive Boosting* (AdaBoost) [9] tech-  
104 niques, and on the other hand, *regression* by using *Artificial Neural Networks*  
105 (ANNs) [10]. These approaches have been widely applied with a great success  
106 in machine learning problems [11]. SVM has been very popular in machine  
107 learning thanks to its ability to apply in both classification or regression prob-  
108 lems, even though it is often used in the former. AdaBoost provides an original

109 vision combining different learners that enable accurate classifications while act-  
110 ing together. On the other hand, ANNs have been proved powerful enough to  
111 solve various regression problems. Compared to classification techniques, find-  
112 ing a good compromise between accuracy and training cost is however more  
113 challenging with ANNs due to their tedious parameterization.

114 To solve the mapping problem, three fundamental questions are identified  
115 and answered throughout this paper: *i) how to encode mappings for training*  
116 *performance prediction models? ii) how to define an adequate criterion for as-*  
117 *sessing the quality of mapping performance predictors? and iii) which technique*  
118 *among regression and classification enables the best prediction rates?* In this  
119 paper, we mainly consider execution time and energy consumption as target  
120 performance metrics to predict.

121 Based on these questions, the main contributions of the current paper are  
122 summarized as follows:

- 123 • different representations trade-offs are analyzed regarding mapping en-  
124 codings for prediction model training. The aim is to identify a simple  
125 representation, which is compact and informative enough to be tractable  
126 with the selected machine learning techniques. Three mapping encoding  
127 variants are compared. They all capture the positions of execution enti-  
128 ties and data in a given multicore system, under the form of vectors or  
129 matrices of topological coordinates.
  
- 130 • a custom metric for assessing the prediction accuracy is proposed, which  
131 fits well the mapping problem formulated above. The usual accuracy mea-  
132 sure relies on the difference, i.e., error percentage, between predicted val-  
133 ues and actual values: the lower this difference the better the prediction. It  
134 is not necessarily well-adapted for the mapping problem, especially when  
135 considering the potential imprecision affecting the values predicted by re-  
136 gression. The proposed metric relies on a relative comparison: it checks  
137 whether the performances induced by a pair of mappings are relatively  
138 comparable in the same way w.r.t. to their actual and predicted values.

139 For instance, if the actual performance of a mapping (computed here with  
140 a multicore system simulator) is actually better (or worse) than that of  
141 another mapping, then this also holds for their respective predicted per-  
142 formances. We refer to this metric as the *percentage of successful tendency*  
143 *prediction* (or PSTP for short).

- 144 • a comparative study of the considered supervised machine learning ap-  
145 proaches is carried out on an automotive application case study, composed  
146 of several tens of execution entities. A suitable mapping encoding is se-  
147 lected from the above analysis and the PSTP metric is applied to evaluate  
148 the considered classification and regression based machine learning tech-  
149 niques. The training process is done off-line and the resulting prediction  
150 models are usable for online prediction. Our results show that, under some  
151 conditions, AdaBoost and ANNs can enable respectively up to 84.8% and  
152 89.05% prediction accuracy w.r.t. PSTP, which is relevant enough for  
153 steering efficient resource allocation decisions.

154 The above contributions rely on our preliminary work, published in a confer-  
155 ence [12], now extended with the following new results: a formalization of used  
156 design concepts (Sections 4.1, 4.2 and 5.2); the application of two additional  
157 supervised machine learning techniques (Sections 6.2 and 6.3); the improve-  
158 ment of the ANN-based evaluation (Section 6.4); and a comparison of all three  
159 techniques w.r.t. an application case study (Section 7).

160 **Organization of the paper.** The rest of the paper is organized as follows:  
161 Section 2 discusses some related work; Section 3 introduces the machine learning  
162 techniques selected in this study; Section 4 describes our system design frame-  
163 work; Section 5 addresses how to effectively use the selected machine learning  
164 techniques for solving the mapping performance prediction problem; Section 6  
165 evaluates the machine learning techniques on an application case study; Section  
166 7 discusses some important outcomes resulting from these evaluations; finally,  
167 Section 8 gives concluding remarks and perspectives.



## 168 2. Related Work

169 Application mapping on multicore platforms has been studied for decades in  
170 literature [1]. To find out near-optimal mapping solutions, many mapping tech-  
171 niques adopt search-based approaches combined with some analyses to evaluate  
172 considered mappings w.r.t. the design requirements. The analyses typically rely  
173 on system-level simulations of application specification in C on FPGA platform  
174 [13], on analytical models [14, 15] for a fast evaluation of different mapping  
175 scenarios, or on UML-based model-driven design frameworks [16].

176 Some recent approaches distinguish themselves from others by advocating  
177 machine learning techniques to address the mapping problem. This trend is  
178 surveyed in [17]. The authors discuss the usual control methods employed to  
179 achieve the runtime management: mapping, dynamic voltage and frequency  
180 scaling (DVFS), and dynamic power management to optimize power/energy  
181 consumption. Then, cover a number of approaches relying on *reinforcement*  
182 learning and *supervised* learning. In [18], reinforcement learning is applied  
183 through a cross-layer system approach to predict the best energy-performance  
184 trade-off in multicore embedded systems. It relies on a biologically-inspired  
185 runtime power management framework implementing a Q-learning algorithm,  
186 which selects the voltage-frequency levels to minimize energy consumption. The  
187 Q-table is made up of state-action pairs, where a state represents the CPU cycle  
188 count and current performance, an action represents the appropriate voltage-  
189 frequency values to set up. Despite its attractive features, reinforcement learning  
190 is not easy to deploy in practice for various reasons (overhead of online learning,  
191 difficult setting of learning parameters, e.g., reward function – see [19]). For this  
192 reason, we rather consider supervised learning in this paper, as in the related  
193 work discussed next.

194 Generally speaking, when applying learning techniques to the mapping prob-  
195 lem w.r.t. a given optimization goal (e.g., performance metrics), one usually  
196 needs to investigate either key parameters, such as the number of threads to  
197 be partitioned, the task/thread-core binding choices, which influence the opti-

198 mization goal; or simply the performance metrics of interest. He or she could  
199 then formulate the target problem as a learning problem with the corresponding  
200 learning features in order to predict the values of the parameters. Most of learn-  
201 ing features found in existing works are: either *application-specific attributes*,  
202 such as number of loops and branch instructions; or *hardware resource-specific*  
203 *attributes*, such as cache and memory size and architecture; or system *runtime*  
204 *execution statistics*, such as cache miss and hit rates. Based on these criteria,  
205 we classify a selected related work as summarized in Table 1.

206 In [20], the authors propose a methodology named SMiTe to predict the  
207 performance interference on simultaneous multi-threading (SMT) processors.  
208 It employs a suite of software "stressors" to quantify applications' contention  
209 characteristics defined as sensitivity and contentiousness of shared resources,  
210 e.g., cache memories. A regression-based prediction model is then built by  
211 using measurements of such characteristics to predict the level of performance  
212 degradation that applications may suffer from co-locations. In [21], the authors  
213 develop statistical power models by using linear regression to estimate per-core  
214 power consumption. Only a small number of parameters such as the CPU cycles  
215 and L1 instruction/data cache access rates of each core are selected as the input  
216 features to train prediction models. The experimental results show that they  
217 could offer simple yet accurate enough power prediction models.

218 A machine learning based approach is proposed in [22] for the optimal map-  
219 ping of streaming applications described by the StreamIt formalism onto dy-  
220 namic multicore processors. To maximize the system performance, the authors  
221 employ a k-Nearest Neighbors (KNN) model to predict the best number of  
222 threads for streaming applications and a linear regression (LR) model to pre-  
223 dict optimal number of cores for threads allocation. Input features are extracted  
224 by using correlation analysis. Fine-grained features such as number of distinct  
225 multiplicities and number of unconditionally executed blocks for KNN, average  
226 number of conditional blocks and average size of all blocks for LR have been  
227 used. In [23], the authors apply machine learning to predict execution time,  
228 memory and disk consumption of two bioinformatics applications deployed on

Table 1: Summary of discussed learning techniques for application mapping. LR= linear regression, KNN = k-Nearest Neighbors, DT = Decision Tree, MTL= Multi-Task Learning, CPI = Cycles Per Instruction. The symbol \* denotes application-specific features, the symbol ◦ denotes hardware resource-specific features, and the symbol • indicates system runtime execution behavior attributes.

References	Optimization goals	Predicted parameters	Learning techniques	Typical input learning features
[20]	server utilization	performance	LR	• sensitivity and contentiouness of shared resources, e.g., L2 cache
[21]	power	per-core power	LR	• CPI, L1 cache access rates
[22]	performance	numbers of threads & cores	LR, KNN	* numbers of unconditional execution blocks, loops and vector operations, etc.
[23]	performance & resource usage	performance & resource usage	LR, KNN, DT, SVM, ANN	* nucleotide sequence length, taxa size ◦ CPU clock, amount of cache and memory
[24]	performance & resource usage	straggler task	MTL	• CPU, memory, network and disk utilizations
[25, 26]	performance & energy	processor type & frequency	SVM	* number of conditional branch instructions and number of successors to a basic block
[27]	energy	core type, voltage & frequency	LR, ANN, DT	• L1, L2 cache accesses and misses, branch mispredictions ◦ allocated cache space, off-chip bandwidth
[28]	resource allocation	performance	ANN	• recent cache access hits and misses
[29]	throughput	performance	ANN	• cache miss rates and instruction mix ratios

229 different hardware resources. Beyond KNN and LR, they address further tech-  
230 niques, e.g., SVM and Decision Trees (DT). The impact of application-specific  
231 attributes, such as the processed length of single nucleotide sequences and the  
232 taxa size of the input nucleotide datasets, as well as resource-specific attributes,  
233 e.g., as CPU speed, amount of memory, speed of memory, on the prediction  
234 accuracy is evaluated.

235 In [24], the authors propose multi-task learning (MTL) formulations to pre-  
236 dict and avoid slow running (or straggler) tasks. They formulate the straggler  
237 prediction problem as a binary classification problem, and consider system-  
238 level counters such as CPU and memory usages as learning features. Further  
239 studies on the mapping of OpenCL kernels onto CPUs and GPUs use SVM  
240 models [25, 26]. The authors formulate the mapping problem as a classification  
241 problem, and devise SVM-based prediction models. These models are trained by  
242 using fine-grained static code features (e.g., number and types of instructions)  
243 and some runtime parameters extracted from a compiler. These approaches  
244 focus on the analysis of each OpenCL kernel program, based on which the most  
245 suitable type of processor (CPU or GPU) can be predicted for kernel mapping,  
246 w.r.t. given optimization criteria.

247 In [27], the authors apply machine learning to find out energy-efficient con-  
248 figurations for running multi-threaded workloads on heterogeneous multicore ar-  
249 chitectures. Machine learning models including Multi-Layer Perceptron (MLP),  
250 regression and tree-based classifiers, are built while taking into account fine-  
251 grained hardware performance counters information, e.g., cache misses and ac-  
252 cesses, branch mispredictions at run-time from a multi-threaded application.  
253 These models aim at predicting parameter values such as core type, voltage  
254 and frequency for maximizing the energy-efficiency. While comparing the built  
255 machine learning models, the authors observed that complex predictors such as  
256 MLP achieve higher accuracy compared to simpler regression-based and tree-  
257 based classifiers, but they have higher overheads in hardware. In an earlier work  
258 [28], ANNs have been used for coordinating the dynamic allocation of shared  
259 multiprocessors-on-chip resources. The global resource allocation problem is for-

260 mulated based on monitored information about the execution of applications.  
261 Each ANN takes as input several fine-grain information related to the hard-  
262 ware resources, including L2 cache space, off-chip bandwidth, power budget,  
263 the number of read and write hits/misses in the L1 cache. Based on these in-  
264 formation the performance of the application is predicted for better allocation  
265 decisions. In [29], the authors apply ANN-based machine learning to predict  
266 the performance of multiple threads running on heterogeneous cores. The aim  
267 is to maximize the throughput. For this purpose, fine-grained system execution  
268 information such as L1, L2 and L3 cache miss rates, instruction mix ratios are  
269 collected to feed the ANN models.

270 In this paper, we mainly concentrate on the accurate performance prediction  
271 for application mapping onto multicore architectures by considering low-cost  
272 and coarse-grained input training information, i.e., mapping locations of tasks  
273 and data, combined with global performance numbers associated with each map-  
274 ping instance. To obtain high prediction accuracy, the aforementioned related  
275 work require fine-grained information as indicated via the input learning features  
276 in Table 1, and thus need to implement some non-trivial module to collect such  
277 data at runtime. On the other hand, these studies alleviate the performance  
278 prediction problem of mappings by either focusing on task/thread executions  
279 on some specific resources such as in [21, 25, 26, 24] without considering the  
280 communication aspects, or focusing on the prediction of threads and/or cores  
281 numbers or core configurations such as in [23, 22, 27] without investigating the  
282 explicit thread/task-core binding solutions. No microarchitecture-dependent in-  
283 formation is required in our approach contrarily to approaches such as [29] or  
284 [30]. By considering a minimal information, we show how selected machine  
285 learning techniques, i.e., SVM, AdaBoost and ANN, can be applied to build  
286 relevant performance prediction models useful for mapping decisions in the flow  
287 depicted by Figure 1.

288 **3. Selected Supervised Machine Learning**

289 We briefly recall in the next the main principles of the three supervised  
 290 machine learning techniques selected for our study. The tools used for applying  
 291 these techniques are briefly presented.

292 *3.1. Classification Techniques: SVM and AdaBoost*

293 The Support Vector Machines (SVM) [8] technique is usually considered a  
 294 must-try in machine learning approach [31]. Given a set of training examples,  
 295 each marked as belonging to a class among a number of classes, the aim of SVM  
 296 is to find the best classification<sup>1</sup> function to distinguish the class members.

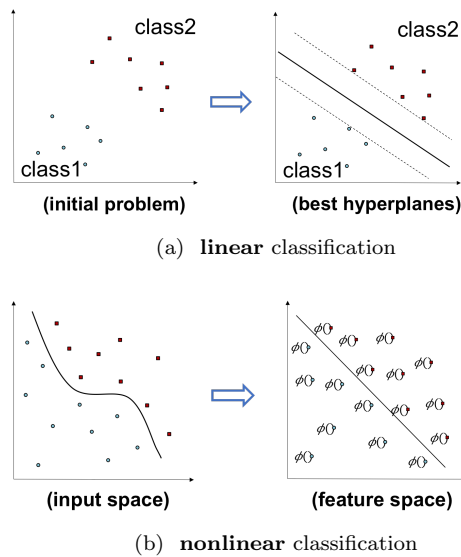


Figure 2: SVM applied to a 2-class learning problem: in case 2a the best classification function is denoted by the solid line; in case 2b the input space is transformed into a feature space with linearly separated dataset.

297 Figure 2a shows a two-class learning problem with a linearly separable  
 298 dataset, and a corresponding linear classification function consisting of a hy-

---

<sup>1</sup>SVM can be also applied in regression problem, even though it is only used for classification in our work.

299 perplane that separates the members of the two classes. As there are many  
 300 such linear hyperplanes, SVM enables to find the best function (e.g., the solid  
 301 line in Figure 2a, right-hand side) by maximizing the margin between the two  
 302 classes. Geometrically, this margin corresponds to the shortest distance between  
 303 the closest data points to a point on the hyperplane. In addition to linear clas-  
 304 sification, SVMs can also perform a nonlinear classification by using *kernel trick*  
 305 to deal with data sets that are not linearly separated. This is done by trans-  
 306 forming the input space into a high-dimensional feature space in which the data  
 307 set can be separated linearly as shown in Figure 2b. To perform such trans-  
 308 formation, a kernel function denoted by  $\phi$  is required. The most widely used  
 309 kernel functions are *Radial Basis Function* (RBF), *linear* and *polynomial*. Let  
 310  $x$  and  $y$  be two vectors in the input space, the simplest linear kernel is defined  
 311 by their inner product plus an optional constant, whereas RBF and degree- $d$   
 312 polynomial kernels are respectively defined as:

$$K(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right) \quad (1)$$

313 and

$$K(x, y) = (x^T y + c)^d \quad (2)$$

314 where  $\sigma$  and  $c$  are free parameters trading off the influence of higher-order versus  
 315 lower-order terms.

316 Since the mapping problem addressed in this paper is a non-linear classifica-  
 317 tion problem, choosing the suitable kernel function  $\phi$  is very important to find  
 318 the best SVM classification models.

319 The Adaptive Boosting (AdaBoost) algorithm [9] is one of the most im-  
 320 portant ensemble methods [32]. Its main idea is to construct a strong learner  
 321 by combining multiple weak or base learners. It is adaptive in the sense that  
 322 consequent weak learners are adjusted iteratively in favor of those instances  
 323 misclassified by previous classifiers.

324 Given a weak or base learning algorithm and a training set as shown in  
 325 Figure 3 (left-hand side), where the symbols  $+$  and  $-$  represent instances that

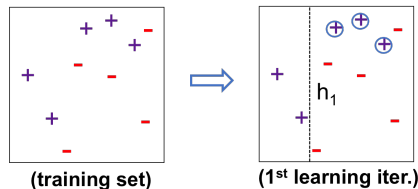


Figure 3: Constructing a strong learner by combining weak learners generated iteratively in AdaBoost.

326 belong to two different classes, AdaBoost works as follows. First, it assigns equal  
 327 weights to all the training examples. Let  $D_i$  denote the weights distribution at  
 328 the  $i^{th}$  learning round. From the training set and  $D_1$  the algorithm generates  
 329 a weak learner denoted by  $h_1$  as shown in Figure 3 (right-hand side) by calling  
 330 the base learning algorithm. Then, the weights of the incorrectly classified  
 331 instances denoted by circles are increased, and an updated weight distribution  
 332  $D_2$  is obtained. From the training set and  $D_2$ , AdaBoost generates again another  
 333 weak learner. This process is repeated for a fixed number of rounds, and the final  
 334 model is derived by combining the weighted outputs of the previously generated  
 335 weak learners. The weights of the weak learners are determined during this  
 336 training process. It has been proven that even when the base learners are weak,  
 337 as long as the performance of each one is slightly better than random guessing,  
 338 the final model can converge to a strong learner [33].

### 339 3.2. Artificial Neural Networks (ANNs)

340 We consider the *feed-forward neural networks*, also known as Multi-Layer  
 341 Perceptron (MLP) [10], consisting of: one input layer of neurons, one output  
 342 layer of neurons, and one or several hidden layers of neurons. An example of  
 343 such a network is illustrated in Figure 4. The connections between the neurons  
 344 of different layers are weighted. The weights of the connections, denoted by  
 345  $w_k$ , are adapted during the training process. Given an input mapping  $M_i$ , the  
 346 output of the network  $o = pred(M_i)$  should match as much as possible the  
 347 expected value  $eval(M_i)$ . Once the network is trained enough, it is used as a  
 348 predictor for unseen mappings.



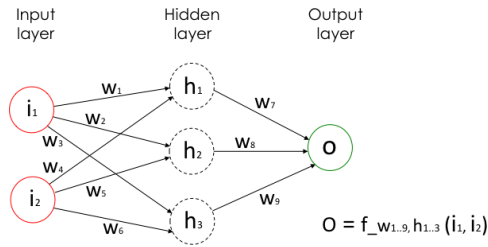


Figure 4: Multi-layer Perceptron with one hidden layer.

349 The MLP network features interesting approximation properties: any con-  
 350 tinuous function can be approximated closely by an MLP [34] with a single  
 351 hidden layer. However, the number of neurons in the hidden layer may be large  
 352 and cannot be determined algorithmically. To learn a function, an input vector  
 353 of values is fed to the network through the input layer. The algorithm used to  
 354 adapt the weights during the training phase is *back-propagation*. The weights  
 355 are adapted in order to minimize the error between the output value calculated  
 356 by the network and the actual value of the function computed at that input  
 357 vector. This learning process is repeated for every input vector. Its outcome,  
 358 i.e., whether or not the network approximates well the function, is dependent on  
 359 the initial values of the weights and on the number of the neurons in the hidden  
 360 layer. To obtain a suitable network, the process needs to be performed multi-  
 361 ple times by varying the weights and/or the number of hidden layers and their  
 362 included neurons until suitable parameter values are found, w.r.t. the expected  
 363 accuracy of the approximated function.

### 364 3.3. Considered Machine Learning Tools

365 There are several machine learning tools nowadays. Two of them are con-  
 366 sidered in this work: the `scikit-learn` v0.9.1<sup>2</sup> package and the `Weka` v3.8.0  
 367 toolset [35]. The former is used to train classification models with SVM and  
 368 AdaBoost, while the latter is applied for training regression-based prediction  
 369 models with ANNs.

<sup>2</sup><http://scikit-learn.org>

370 For SVM-based classification with the `scikit-learn` package, the main pa-  
371 rameters one needs to tune are the following:

- 372 • *kernel function*: one can choose among `linear`, `poly`, `rbf` and `sigmoid`;
- 373 • *gamma*: kernel coefficient for `poly`, `rbf` and `sigmoid` functions;
- 374 • *C*: penalty parameter of error term.

375 For AdaBoost-based classification, the tuning parameters in the `scikit-learn`  
376 package are as follows:

- 377 • *base\_estimator*: the base estimator from which the boosted ensemble is  
378 built;
- 379 • *n\_estimators* : the maximum number of estimators;
- 380 • *learning rate*: it is used to shrink the contribution of each classifier;
- 381 • *algorithm*: either `SAMME.R(default)` or `SAMME`. The former uses the prob-  
382 ability estimates to update the additive model, while the latter uses the  
383 classifications only. The `SAMME.R` algorithm enables a faster training.

384 For regression-based prediction with the `Weka v3.8.0` toolset, we consider  
385 its associated `MLPRegressor` package: a multilayer perceptron with a single hid-  
386 den layer. This package exploits the optimization capability provided in `Weka`,  
387 by minimizing the given loss function plus a quadratic penalty with the *Broyden-*  
388 *Fletcher-Goldfarb-Shanno* (BFGS) method. The ANN tuning parameters of the  
389 `MLPRegressor`-based prediction are described as follows:

- 390 • *number of hidden neurons* (large numbers induce long learning durations);
- 391 • *ridge parameter*: used to determine the penalty on the size of the weights;
- 392 • *seed value* for initializing the weight values of the networks;
- 393 • *activation functions*: `Sigmoid` or `Softplus`;
- 394 • *loss function*: `squared error` or `approximated absolute error`;

- 395 • a *tolerance* parameter for the delta values;
- 396 • *conjugate gradient descent* (rather than BFGS) for accelerating the train-  
397 ing process;
- 398 • *parallel calculation* of loss function and gradient when training on multiple  
399 CPU cores.

400 The application of the above machine learning techniques to the case study  
401 addressed in Section 6 will consist in finding the parameter values that provide  
402 precise-enough performance predictions.

#### 403 **4. Multicore System Design**

404 We present the design concepts used in this study for the description and  
405 simulation of multicore systems. These concepts enable to specify applications  
406 through a task graph oriented representation (see Section 4.1). Existing appli-  
407 cation parallelization tools [36] [37], combined with designers' analysis, help to  
408 derive such task graphs. Network-on-Chip based multicore system models are  
409 used for application mapping and execution with a simulator (see Section 4.2).  
410 Finally, the encoding of the resulting mappings is addressed (see Sections 5.1  
411 and 5.2) for performance prediction.

##### 412 *4.1. Application Design Concepts*

413 We define the modeling concepts dedicated to application description. These  
414 concepts are inspired by the Amalthea formalism [6], which has been introduced  
415 for automotive software design.

416 **Definition 2 (Runnable and labels).** *We consider the following notions:*

- 417 • a *runnable*  $r$  is a function representing the smallest unit of code schedulable  
418 by an operating system, and associated with non functional attributes, e.g.,  
419 execution time;

420 • a label  $l$  is a symbolic concept representing a memory location, associated  
 421 a size attribute.

422 The value of a non functional attribute of a runnable  $r$  can be either a  
 423 point-wise value  $v \in \mathbb{R}$  or an interval of values  $(lwb, upb)$ ,  $lwb, upb \in \mathbb{R}$  or a  
 424 probabilistic distribution. This enables to specify various value approximations.  
 425 For instance, considering the execution time of a runnable, a point-wise value  
 426 can be used to capture an average/worst-case/best-case execution time. An  
 427 interval captures a variation of execution time between worst-case and best-  
 428 case scenarios, while a probabilistic distribution will describe a probabilistic  
 429 law characterizing the execution time behavior. The unit of label size is *byte*.

430 In the sequel, we respectively denote by  $\mathcal{R}$  and  $\mathcal{L}$  the sets of all runnables and  
 431 labels. Runnables and labels are combined to build a *task*, which corresponds  
 432 to an aggregate execution entity.

433 **Definition 3 (Tasks).** A task  $t = (R, L, dep, release)$  is a labeled directed  
 434 graph of runnables such that the set of runnables  $R \subseteq \mathcal{R}$  represents the graph  
 435 vertices;  $L \subseteq \mathcal{L}$  is a non-empty set of labels associated with the edges connecting  
 436 the runnables  $r \in R$ ;  $dep \subseteq R \times (L \cup \emptyset) \times R$  defines the edges of the graph; and  
 437 *release* is an attribute specifying whether the release mode of task  $t$  is either  
 438 periodic or sporadic or aperiodic, together with the corresponding periodicity  
 439 value.

440 From the above definition of *dep*, the edge connecting two different runnables  
 441 within a task can be either associated with a label or not: a labeled edge ex-  
 442 presses a data communication between connected runnables, while non-labeled  
 443 edges model precedence between connected runnables.

444 **Example 1.** The task  $t = (R, L, dep, release)$  where  $R = \{r_0, r_1, r_2, r_3, r_4\}$ ,  
 445  $L = \{l_1, l_2\}$ ,  $dep = \{(r_0, l_1, r_2), (r_1, r_4), (r_1, l_2, r_3), (r_2, r_4), (r_4, r_3)\}$  and *release* =  
 446  $\langle \text{aperiodic}, -- \rangle$  represents an aperiodic task, composed of five connected runnables.  
 447 Here, only two runnable connections correspond to data communications achieved

448 through labels  $l_1$  and  $l_2$ . The specification of task  $t$  is the same as for the task  
 449  $T_4$  shown graphically in Figure 5.

450 Upon the release of a task, all its associated runnables are scheduled for  
 451 execution. Let us denote by  $\mathcal{T}$  the set of all tasks. Tasks are combined together  
 452 to build applications as described in the next.

453 **Definition 4 (Application).** An application  $a = (T, dep)$  is a directed graph  
 454 of tasks such that  $T \subseteq \mathcal{T}$  and  $dep \subseteq T \times T$ .

455 Concretely, applications are graphically described by using Amalthea nota-  
 456 tions [6], which capture the design concepts defined above.

457 **Example 2.** An application model composed of five tasks with various release  
 458 modes is illustrated in Figure 5. The periodic task  $T_0$  has a period of 5ms. It  
 459 interacts with the periodic tasks  $T_1$  and  $T_2$ . Task  $T_3$  denotes a sporadic task with  
 460 a minimum inter-release interval specified as  $(lwb, upb)$ . Task  $T_4$  is an aperiodic  
 461 task with a release mode defined according to a given distribution law. A zoom  
 462 in this task shows a sub-graph of five runnables  $R_{i,i \in 0..4}$ . Runnables  $R_0$  and  $R_2$   
 463 communicate via the label  $L_1$ :  $R_0$  writes  $L_1$  while  $R_2$  reads  $L_1$ . The size of  $L_1$   
 464 represents the exchanged data volume.

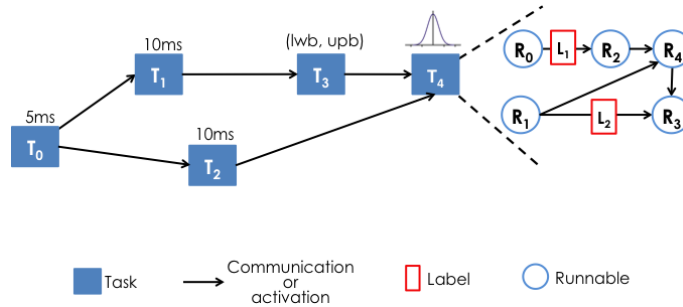


Figure 5: A simple application model in Amalthea

465 In the remainder of the paper, for the sake of simplicity we will use the  
 466 notation  $X.a$  in order to refer to an attribute  $a$  of a concept  $X$ . For instance,  
 467 given a task  $t$ , the runnable  $r_i$  in task  $t$  is denoted by  $t.r_i$ .

468 *4.2. Application Mapping on Execution Platforms*

469 We consider execution platforms composed of multiple cores that exchange  
 470 data via a communication interconnect, e.g., a crossbar. Each individual core  
 471 is composed of a CPU and a local memory. Let  $\mathcal{C}$  denote the set of all cores.

472 **Definition 5 (Execution Platform).** *An execution platform  $p = (C, I)$  is*  
 473 *defined as a subset  $C \subseteq \mathcal{C}$  of cores, interconnected by an interconnect  $I$  as*  
 474 *communication infrastructure.*

475 With the high number of cores in target execution platforms, the chosen  
 476 communication interconnect is Network-on-Chip (NoC), as it scales better com-  
 477 pared to bus and crossbar.

478 When applications are mapped on a given execution platform, each task (or  
 479 runnable) is assigned to a core CPU in charge of processing the corresponding  
 480 functions. Label variables are assigned to memory locations in the cores. When  
 481 a runnable and its accessing labels are mapped onto different cores, the corre-  
 482 sponding communications become remote and require transactions via the NoC.  
 483 Otherwise, the memory accesses are local and do not incur any NoC transaction.

484 **Definition 6 (Application mapping on execution platform).** *Given an ap-*  
 485 *plication  $a$  and an execution platform  $p$ , a mapping  $m$  of  $a$  on  $p$  is defined as:*

$$(a.T \times p.C) \equiv_{def} (a.T.R \times p.C) \cup (a.T.L \times p.C) \quad (3)$$

486 *i.e., the runnables  $a.T.R$  and labels  $a.T.L$  associated with each task  $T$  of the*  
 487 *application  $a$  are mapped onto the cores  $p.C$  of the platform  $p$ .*

488 Figure 6 depicts a typical scenario where runnables are mapped onto the  
 489 CPU part of the cores in an execution platform. The labels are mapped onto  
 490 memory locations within cores. The bottom part of Figure 6 illustrates a mul-  
 491 ticore platform models where cores communicate with each other via a network  
 492 interface (NI), connecting them to the NoC. Each core model includes a CPU  
 493 (dark blue box) and a local memory (red dashed box).

494 The McSim-TLM-NoC (Manycore platform Simulation tool for NoC-based  
495 systems at Transactional Level Modeling) [38] [39] is an Amalthea-based sim-  
496 ulator that is used to evaluate mapping scenarios. The multicore architecture  
497 considered in this simulator relies on an abstract cache-less core model [40]  
498 [41], which supports priority-preemptive runnable execution (and *Round-Robin*  
499 *scheduling* for runnables with the same priority level). The runnables mapping  
500 decisions are defined in the *mapping heuristics module* (see Figure 1). An ex-  
501 ample of mapping consists in allocating tasks that strongly communicate with  
502 each other on the same (or closest) cores, in order to reduce the overall com-  
503 munication traffic [40]. Each core in McSim-TLM-NoC is composed of two  
504 main units: an execution unit and a communication unit, which deal with their  
505 corresponding instructions within the executed runnables. The different cores  
506 communicate through either a bus, a crossbar or a mesh-oriented packet-based  
507 Network-on-Chip (NoC). In the current work, we use a NoC, where each node  
508 in the network includes a core and a local memory. An XY routing algorithm is  
509 applied for packet exchanges between nodes. The runtime and energy consump-  
510 tion information computed by McSim-TLM-NoC are estimated on the basis of  
511 instruction costs relying on ARM Cortex-A7 and Cortex-A15 CPUs. Further  
512 details on the simulator implementation can be found in [41].

513 McSim-TLM-NoC provides a clean and simple interface allowing to map  
514 runnables and labels onto platform resources, through custom mapping algo-  
515 rithms. Once the mapping is defined, the different runnables are scheduled  
516 and executed [40]. Contrarily to cycle-accurate simulators such as gem5 [42],  
517 McSim-TLM-NoC is fast enough to enable the evaluation of thousands of appli-  
518 cation mappings in a quite reasonable time. This enables to produce mapping  
519 examples usable as training data for performance prediction.

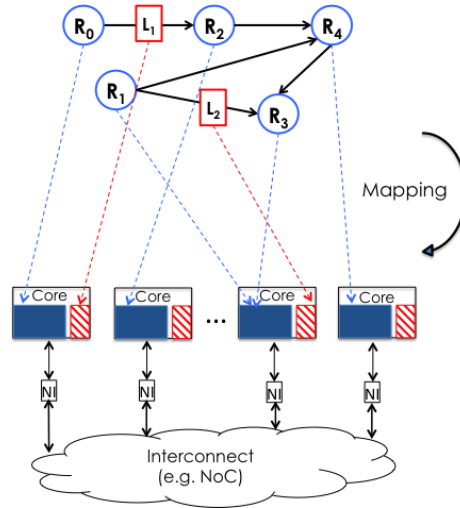


Figure 6: Application mapping on a multicore platform.

520 **5. Application of Selected Machine Learning Techniques to Mapping**  
 521 **Performance Prediction**

522 The effective use of the selected machine learning techniques (see Section 3)  
 523 to address the mapping performance prediction problem, requires some answers  
 524 to two crucial questions: i) how to define a relevant mapping encoding for model  
 525 training? ii) how to adequately assess the quality of the generated prediction  
 526 models? These questions are addressed in the sequel.

527 *5.1. Mapping Encoding for Training*

528 We discuss three candidate mapping encodings, as illustrated in Figure 7:

529 • **Encoding 1** (Figure 7a). In this scenario, the vector describing a mapping  
 530 has as many entries as there are runnables and labels in the model of an  
 531 application. To build such a vector, the runnable and label identifiers are  
 532 sorted in an arbitrary order once and for all. The cores of the platform  
 533 are indexed using integers. Then:

- 534 – each mapping vector component, corresponding to a runnable identifier,  
 535 is initialized with the index value of the core on which this



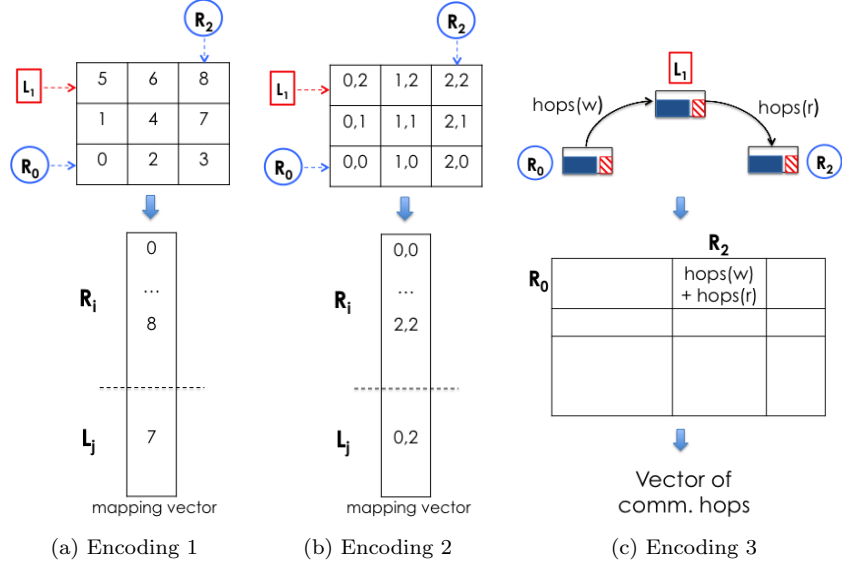


Figure 7: Three application mapping encodings.

536 runnable is mapped. For instance, if the  $i^{th}$  component of such a  
 537 vector  $V$  corresponds to a runnable  $R_i$ , then the value of  $V[i]$  is  
 538 equal to the index of the core on which  $R_i$  is mapped;

539 – in a similar way, for each label, the corresponding mapping vector  
 540 component is initialized with the index of the core containing the  
 541 memory on which the label is mapped.

542 • **Encoding 2** (Figure 7b). This scenario is similar to the previous one ex-  
 543 cept that now core indexes are not single integers but two integers, corre-  
 544 sponding to the Cartesian coordinates of cores within the two-dimensional  
 545 space inherited from the mesh topology of the considered NoC intercon-  
 546 nect. Here, the size of the vector representing the mapping is twice as  
 547 large as in the first encoding approach.

548 • **Encoding 3** (Figure 7c). In this scenario, we encode a mapping through  
 549 a square matrix. The number of columns and the number of rows of the

550 matrix are equal to the number of runnables in an application. Each row  
551 (and column) entry is associated with a runnable identifier.

552 Let us assume that a runnable  $R_i$  writes data to a label  $L$  and another  
553 runnable  $R_j$  reads data from  $L$ . The flits (i.e., the elements composing a  
554 packet exchanged in a NoC) sent by  $R_i$  to  $L$  have to perform  $h(w)$  hops  
555 in the NoC depending on the mapping locations of  $R_i$  and  $L$ . When  $R_j$   
556 reads data from  $L$ , the flits traveling from  $L$  to  $R_j$  perform  $h(r)$  hops.  
557 Finally, the value at entry  $(i, j)$  of the encoding matrix is defined as:

$$(h(w) * n_w) + (h(r) * n_r) \quad (4)$$

558 where  $n_w$  is the number of flits written by  $R_i$  on  $L$  and  $n_r$  is the number  
559 of flits read by  $R_j$  from  $L$ . Finally, the matrix resulting from the encoding  
560 is transformed into a vector by putting its columns on top of each other  
561 or by aligning its rows next to each other.

562 **Which mapping encoding to select?** The first encoding may not render  
563 well the similarity or dissimilarity between different mappings. Typically given  
564 the scenario shown in Figure 7a, let us consider a first pair of mappings M1 and  
565 M2 such that M1 and M2 only differ by the location of one specific runnable.  
566 In M1 this runnable is mapped on core 5 (in the matrix shown on top of Figure  
567 7a) while in M2 the runnable is mapped on core 3. The Manhattan distance  
568 between the vectors representing M1 and M2 is 2. Now, let us consider mappings  
569 M3 and M4 such that in M3 the same runnable is mapped on core 4 and in  
570 M4 this runnable is mapped on core 8. The Manhattan distance between the  
571 vectors encoding M3 and M4 is 4. By comparing with the Manhattan distance,  
572 mappings M1 and M2 appear to be more similar than mappings M3 and M4.  
573 However, M3 and M4 are topologically more similar since the locations of the  
574 runnable of interest are closer in that case than in the case of M1 and M2:  
575 cores 4 and 8 are closer to each other compared to cores 5 and 3. From this  
576 observation, the first mapping encoding scenario does not appear appropriate  
577 enough. So, we will consider the two other encodings.

578 The size of vectors in the second encoding is linearly proportional to the  
579 number of runnables and labels. In the third encoding, the size of the vector  
580 depends quadratically on the number of runnables. This can make the training  
581 of learning models more complex. Indeed, real-life applications can feature  
582 huge numbers of runnables and labels. Thus, the data needed to successfully  
583 train learning models can grow exponentially in the dimension of the input  
584 vector. Reducing the size of this vector is necessary to speed-up the training  
585 by accelerating the training algorithms and by reducing the size of the required  
586 training set of data.

587 From this remark, we finally select the second encoding scenario for our  
588 experiments in this paper, since it provides the best compromise in terms of rel-  
589 evance and tractability in size. Note that this encoding induces some constraint  
590 on the reusability of obtained prediction models for different applications. In-  
591 deed, the applications must have similar task graph structures, but the attribute  
592 values of the task, runnables and labels can vary. This restriction can be lifted  
593 however by building the prediction models at runtime, e.g., through an initial  
594 training phase during application execution where mappings are encoded and  
595 evaluated. Of course, this online learning process can have some cost, especially  
596 when achieved on the same execution platform as the application itself.

## 597 5.2. Mapping Prediction Model Assessment

598 The natural way to assess learned predictive models for both classification  
599 and regression problems is to calculate the *prediction accuracy*<sup>3</sup>, i.e., ratio of  
600 correct predictions over total predictions, obtained with trained models on previ-  
601 ously unseen test data instances. The higher the accuracy the better the model.  
602 **F-measure [43] is another widely used metric to evaluate classification models,**

---

<sup>3</sup>The prediction accuracy is different from the *loss* metric (generally a percentage), which is rather computed on training and validation data instances. The validation data set enables to tune the parameters of the prediction model under training phase. The loss can be seen then as a summation of the approximation errors made for predicted versus actual values/classes in the training or validation sets.

603 especially for imbalanced data. It is the harmonic average of the precision and  
 604 recall metrics. A high F1-score indicates that the model has low false positives  
 605 and low false negatives, and is thus able to correctly identify real threats and  
 606 not disturbed by false alarms.

607 Another way to assess the relevance of mapping performance prediction  
 608 could rely on a ranking of considered mappings according to their predicted  
 609 classes or performance metrics. Let  $M_i$  and  $M_j$  denote two different mappings;  
 610 let  $eval(M_i)$  and  $eval(M_j)$  be respectively their actual metric values; and let  
 611  $pred(M_i)$  and  $pred(M_j)$  denote their respective predicted classes or metric val-  
 612 ues. No matter the difference between the predicted and the actual classes or  
 613 metric values of  $M_i$  and  $M_j$ , if  $eval(M_i)$  and  $eval(M_j)$  strictly compare simi-  
 614 larly as  $pred(M_i)$  and  $pred(M_j)$ , then the predictions become relevant enough  
 615 to be exploited in the *mapping heuristics module* (see Figure 1). For instance,  
 616 if  $eval(M_i) > eval(M_j)$ , then one should have  $pred(M_i) > pred(M_j)$ . We refer  
 617 to this relative comparison as mapping metrics *tendency prediction*, i.e., how  
 618 the predicted classes or performances of mappings "tend" to behave relatively  
 619 to each other, w.r.t. actual metric values.

620 **Definition 7 (Consistent tendency prediction).** *Let  $M_i$  and  $M_j$  denote two*  
 621 *mappings; let  $eval(M_i)$  and  $eval(M_j)$  be respectively their actual metric values,*  
 622 *and let  $pred(M_i)$  and  $pred(M_j)$  denote their respective predicted classes or met-*  
 623 *ric values. A tendency prediction is said to be consistent if the values  $pred(M_i)$*   
 624 *and  $pred(M_j)$  are comparable in the same way as  $eval(M_i)$  and  $eval(M_j)$ , i.e.:*

$$eval(M_i) \sim eval(M_j) \leftrightarrow pred(M_i) \sim pred(M_j) \quad (5)$$

625 where the operator  $\sim$  belongs to  $\{<, =, >\}$ .

626 In general, when the prediction accuracy of a trained model is high, the  
 627 tendency will be very consistently predicted. However, the inverse is not true.  
 628 Thus, prediction accuracy is not necessary the most suitable assessment criterion  
 629 for our learning problem. Instead, we introduce a simpler yet adequate measure  
 630 relying on tendency prediction.

631 **Definition 8 (Percentage of successful tendency prediction – PSTP).**

632 *Given a reference set  $T$  of testing mapping pairs, we define the percentage*  
633 *of successful tendency prediction (PSTP) as the percentage of mapping pairs*  
634  *$\langle M_i, M_j \rangle \in T$  that satisfies the formula (5).*

635 Accurate prediction models are expected to provide very high PSTP values.  
636 In practice, it is difficult to reach a maximum prediction accuracy, especially  
637 with regression techniques, because of the approximations applied for value pre-  
638 diction. For instance, given two different application mappings  $M_i$  and  $M_j$ , let  
639 us consider  $eval(M_i)$  and  $eval(M_j)$  are close values when executed on an actual  
640 platform. The comparison of their predicted values,  $pred(M_i)$  and  $pred(M_j)$ , ac-  
641 cording to PSTP will be consistent only if the prediction accuracy is high enough  
642 to distinguish how they compare. However, when  $eval(M_i)$  and  $eval(M_j)$  are  
643 quite different, the comparison of  $pred(M_i)$  and  $pred(M_j)$  according to PSTP  
644 has higher chance to be consistent, even without a moderate prediction accuracy.

645 To assess the quality of built prediction models, it is worth evaluating PSTP  
646 on pairs of mappings  $\langle M_i, M_j \rangle$  whose actual performance values differ by  $\Delta\%$   
647 (where  $\Delta \in \mathbb{R}^+$ ). The idea behind this filtering of mapping pairs is to elimi-  
648 nate test cases for which the performance comparison is highly sensitive to the  
649 prediction accuracy. We thus define such  $\Delta$ -filter PSTP measure as follows:

650 **Definition 9 ( $\Delta$ -filter PSTP).** *Given a reference test set  $T$  of mapping pairs*  
651 *whose actual values differ by  $\Delta\%$  (where  $\Delta \in \mathbb{R}^+$ ), we define  $\Delta$ -filter PSTP over*  
652  *$T$  as the percentage of mapping pairs  $\langle M_i, M_j \rangle \in T$  that satisfies the formula*  
653 *(5).*

654 Note that for classification techniques, given a mapping pair  $\langle M_i, M_j \rangle$ , their  
655 respective predicted classes  $pred(M_i)$  and  $pred(M_j)$  are, instead of real numbers  
656 for regression techniques, class labels representing sub-domains of performance  
657 values. To make them directly comparable as real numbers, we encode class  
658 labels as natural numbers  $\lambda \in \mathbb{N}$  in a way that reflects the greater than/less  
659 than/equal to relationships for the sub-domains derived from the domain of  
660 performance values.

661 Given a mapping pair  $\langle M_i, M_j \rangle$  whose actual performance values  $M_i$  and  $M_j$   
662 differ by  $\Delta\%$ , let us assume that the number of target classes enables to assign  
663  $M_i$  and  $M_j$  into different classes. If both mappings are, however, classified into  
664 the same class, representing the same sub-domain of performance values, it then  
665 indicates that the classifier is not accurate enough to distinguish them. We refer  
666 to such predictions as **unknown** tendency predictions, characterized as follows:

667 **Definition 10 (Percentage of unknown tendency prediction – PUTP).**  
668 *Given a reference test set  $T$  of mapping pairs to be classified, we define the per-*  
669 *centage of unknown tendency prediction (PUTP) as the percentage of mapping*  
670 *pairs  $\{M_i, M_j\} \in T$  that satisfy:*

$$eval(M_i) \sim eval(M_j) \rightarrow pred(M_i) = pred(M_j) \quad (6)$$

671 where the operator  $\sim$  belongs to  $\{<, >\}$ .

672 Similarly to  $\Delta$ -filter PSTP, the  $\Delta$ -filter PUTP for classification is defined as  
673 follows:

674 **Definition 11 ( $\Delta$ -filter PUTP.).** *Given a reference test set  $T$  of mapping*  
675 *pairs whose actual values differ by  $\Delta\%$  (where  $\Delta \in \mathbb{R}^+$ ), we define the  $\Delta$ -filter*  
676 *PUTP over  $P$  as the percentage of mapping pairs  $\langle M_i, M_j \rangle \in T$  that satisfies*  
677 *the formula (6).*

678 To summarize, the outcome of the classification of two different mappings  
679  $M_i$  and  $M_j$  falls within one of the following cases:

- 680 • *correct* prediction: when the predicted classes are ranked consistently  
681 w.r.t. the actual mapping performances values  $eval(M_i)$  and  $eval(M_j)$ ;
- 682 • *wrong* prediction: when the predicted classes  $pred(M_i)$  and  $pred(M_j)$  are  
683 ranked in an opposite way w.r.t. the actual mapping performances values  
684  $eval(M_i)$  and  $eval(M_j)$ ;
- 685 • *unknown* prediction (only for classification): when the predicted classes  
686  $pred(M_i)$  and  $pred(M_j)$  are identical while they should be distinct w.r.t.  
687 the actual mapping performances values  $eval(M_i)$  and  $eval(M_j)$ ;

688 In our experiments, we will mainly use PSTP as accuracy assessment metric  
689 for defined prediction models. The coverage of this assessment on the testing  
690 mapping set will be evaluated with PUTP in applied classification techniques.

691 To formulate the mapping performance prediction problem as a classification  
692 problem, we partition the generated simulation data into a number of classes  
693 according to the metric value ranges. For execution time, it is done by taking  
694 the minimal and maximal execution times (denoted by  $minExec$  and  $maxExec$ )  
695 as the possible range of execution times  $[minExec, maxExec]$ , and by dividing  
696 this range into sub-ranges of same length. The length is computed as follows:

$$length = (maxExec - minExec)/N \quad (7)$$

697 where  $N$  denotes a selected number classes. As a result, we obtain  $N$  intervals,  
698 as follows:

$$[minExec, minExec + length], \dots, [maxExec - length, MaxExec] \quad (8)$$

699 denoted by  $I_1, \dots, I_N$ . The data samples can thus be classified into  $N$  classes ac-  
700 cordingly. An instance is classified in class  $C_i$ , if its execution time for instance,  
701 falls into the interval  $I_i$ . In this way, instead of predicting the execution time,  
702 we predict the class or interval a given mapping falls into. The larger the  $N$   
703 gets, the more informative the prediction result gets.

704 Finally, the model training, we partition the *working mapping set* as follows:  
705 65% of the mappings are used for training and the remaining 35% are used as  
706 unseen data for testing the quality of the prediction models. This partition-  
707 ing is compatible with common practices in machine learning – e.g., see the  
708 partitioning suggested in Weka [35].

## 709 6. Comparison of Machine Learning Techniques on a Case Study

710 We consider an automotive application case study [12] in order to evaluate  
711 the quality of the prediction models derived using the selected machine learning  
712 techniques: SVM, AdaBoost and ANN. The application, referred to as Demo-  
713 Car, corresponds to an engine control system, provided by Robert Bosch GmbH,

714 within the DreamCloud European FP7 project. As briefly mentioned in the in-  
715 troductory section, comparing the quality of our results w.r.t. existing mapping  
716 heuristics [1] is beyond the scope of this paper. Instead, we focus on the quality  
717 of performance value prediction, which is used by the *mapping heuristics module*  
718 to assess candidate mappings (see Figure 1).

719 The inputs of DemoCar application are typical in automobiles, e.g., engine  
720 speed, temperature, battery voltage. Its outputs are the triggered cylinder  
721 number, the ignition time and the desired throttle position. In total, there are  
722 10 input message sources and 4 output message sinks. The considered Amalthea  
723 model of DemoCar is composed of 43 runnables and 71 labels. Out of these  
724 runnables, 22 runnables operate at high activation rate, 4 runnables operate  
725 at low activation rate, and 17 runnables get activated aperiodically upon some  
726 event occurrences.

727 In the following, we discuss the generation of DemoCar mapping instances for  
728 training and testing the target prediction models. Classification techniques are  
729 first presented. Then, ANNs are applied. Finally, we discuss the effectiveness  
730 and efficiency of the three techniques.

### 731 6.1. Experimental Setup

732 **Generation of the DemoCar Application Mapping Instances.** The  
733 mappings of DemoCar feature a multicore execution platform composed of 6  
734 cores with a 2x3-mesh NoC architecture for communication. Here, each core  
735 model in McSim-TLM-NoC features an ARM Cortex-A15 CPU running at  
736 1GHz. Current automotive on-chip multicore systems do not exceed this core  
737 count. **Note that even though a homogeneous multicore execution platform is**  
738 **considered here, our proposal can also deal with heterogeneity by associating**  
739 **tasks/runnables with instruction costs pertaining to different target computing**  
740 **elements, in McSim-TLM-NoC. This would probably result in different per-**  
741 **formance/energy outcomes in the resulting mapping vectors. Then, the exact**  
742 **same training and prediction methods remain applicable, as illustrated in the**  
743 **homogeneous design considered here.**



744 The mapping of labels is fixed and identical in all mappings generated in this  
745 study. Only the mapping of runnables on core CPUs is variable. This choice  
746 has been made for the sake of simplicity as we can straightforwardly evaluate  
747 the impact of changes in runnable mappings. Even though relevant, taking  
748 into account possible changes in label mappings would make the exploration  
749 space much larger. Given a number  $R$  of runnables to be mapped on a number  
750  $C$  of cores, there are  $C^R$  possible mappings of the runnables on these cores.  
751 For DemoCar, this corresponds to  $6^{43}$ , which is a very large exploration space.  
752 Within this space, we decided to compute with McSim-TLM-NoC simulator  
753 a maximum set of 30K mappings generated<sup>4</sup> randomly according to a uniform  
754 distribution (for a relevant coverage of the possible mapping space). We checked  
755 there is redundant and no outlier mapping instance within this set of mappings.  
756 Each mapping instance is associated with its corresponding execution time and  
757 energy consumption computed with the simulator.

758 Four different working mapping sets are considered for the training with all  
759 three supervised learning techniques: 3K, 5K, 10K and 30K mapping instances.  
760 This enables to explore how the quality of the prediction evolves with the size  
761 of working mapping sets.

762 **Prediction Model Evaluation Scenarios.** The PSPT measure introduced  
763 previously is used for assessing the generated prediction models. For this pur-  
764 pose, we consider the set  $P_{(M_i, M_j)}$  of all possible pairs of mappings without  
765 redundancy resulting from the testing subset mappings. Then, we evaluate the  
766 following cases:

- 767 • **case-0:** PSTP over the set  $P_{(M_i, M_j)}$ . Given  $n$  the number of mappings in  
768 the test subset, the number of all possible pairs of mappings in  $P_{(M_i, M_j)}$   
769 is defined as:  $(n * (n - 1))/2$ ;
- 770 • **case-1:**  $\Delta$ -filter PSTP over the set  $P_{(M_i, M_j)}$ , where  $\Delta = 5$  (i.e., only pairs

---

<sup>4</sup>All mapping data sets used in the current study are available at <https://seafire.lirmm.fr/d/ca11a19a75c44013988f>.

771 of mappings such that the actual metric values of one mapping are 5%  
772 different from those of the other mapping);

773 • **case-2:**  $\Delta$ -filter PSTP over the set  $P_{(M_i, M_j)}$ , where  $\Delta = 10$ ;

774 • **case-3:**  $\Delta$ -filter PSTP over the set  $P_{(M_i, M_j)}$ , where  $\Delta = 20$ .

775 For classification-based approaches, we also use the PUTP measure (see  
776 Definition 10 in Section 5) to evaluate the coverage of PSTP on the testing  
777 set. In the reported experiments, we consider different numbers of classes  $N \in$   
778  $\{3, 9, 81, 512\}$ . Figure 8 shows the distributions of the 30K mapping instances  
779 used later on, according to their induced execution times, and w.r.t. different  
780 numbers of classes. We notice that the resulting distributions are moderately  
781 unequal and may lead to *imbalanced data* problem [44], sometimes faced in  
782 classification problems. While the data set could be transformed to have more  
783 balanced distribution (e.g., by collecting more data or by applying sampling  
784 methods), we decide to keep the set unchanged. Advanced machine learning  
785 algorithms can deal with imbalanced data [44]. This is typically the case of  
786 Decision Trees.

787 Our experiments are performed on an Intel Core i5-6600 host operating at  
788 3.4GHz. Most of the prediction model evaluations shown in the next sections  
789 concern execution time. Even though not reported, similar results are observed  
790 when focusing on energy. The main reason is that the evaluations obtained with  
791 McSim-TLM-NoC are often proportional for both metrics.

## 792 6.2. SVM-based Prediction Modeling

793 Figure 9 reports the performance of SVM models w.r.t. different numbers  
794 of classes. These results rely on the most favorable values selected for the  
795 *kernel function* function, *gamma* and *C* values: `rbf`, 1 and 1000 respectively.  
796 In particular, the kernel function value domain has been exhaustively explored.  
797 For each function, in average 30 combinations of *gamma* and *C* values are  
798 explored. The duration required to train every SVM model varies between less  
799 than 1 second to 13 minutes.

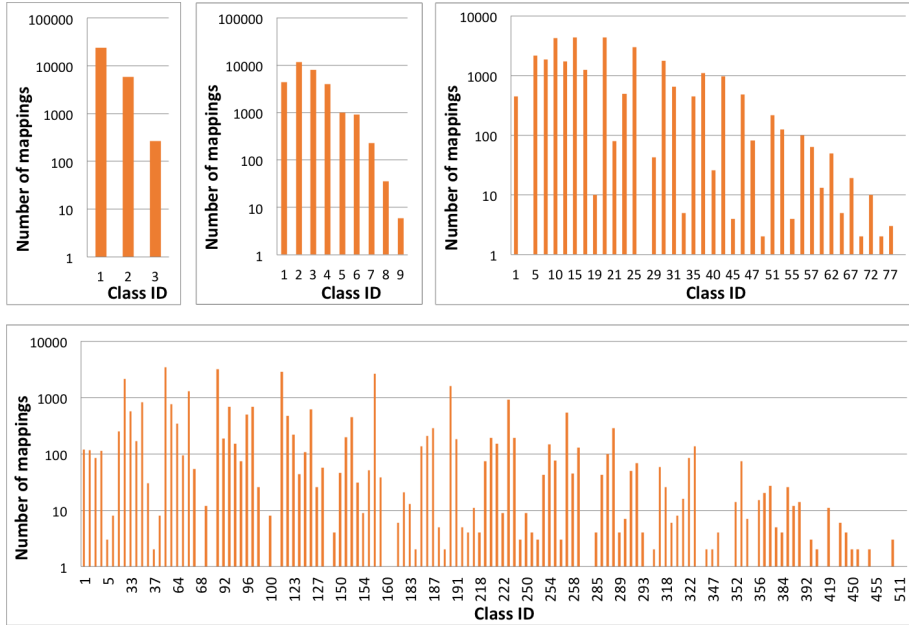


Figure 8: Distributions of 30K mapping instances w.r.t. execution time ranges. The Y-axis denotes the number of mapping instances (in log scale) corresponding to each class ID shown on the X-axis.

800 Let us consider Figure 9c. We obtain for **case-0** a maximum PSTP value  
 801 of 53.6% while the minimum is 43.9%. It means that with a 30K working  
 802 mapping set, the best SVM prediction model would enable correct mapping  
 803 pair comparisons only on 53.6% of evaluated pairs. For **case-1**, **case-2** and  
 804 **case-3**, after filtering the tested mapping pairs based on the difference in their  
 805 execution times, we observe better results (i.e.,  $\Delta$ -filter PSTP where  $\Delta = 5\%$ ,  
 806  $10\%$ ,  $20\%$ ). This respectively leads to 59.76%, 63.9% and 68.97% of correct  
 807 mapping comparison. More generally, for each target number of classes shown  
 808 in Figure 9, we observe the same trend: given a working mapping data set (i.e.,  
 809 3K, 5K, 10K and 30K mapping sets), the PSTP gets better as 1) the number of  
 810 training samples grows, and 2) the filtering ratio of the testing set of mapping  
 811 pairs increases.

812 In addition, the PSTP values get better when the number of classes is refined.

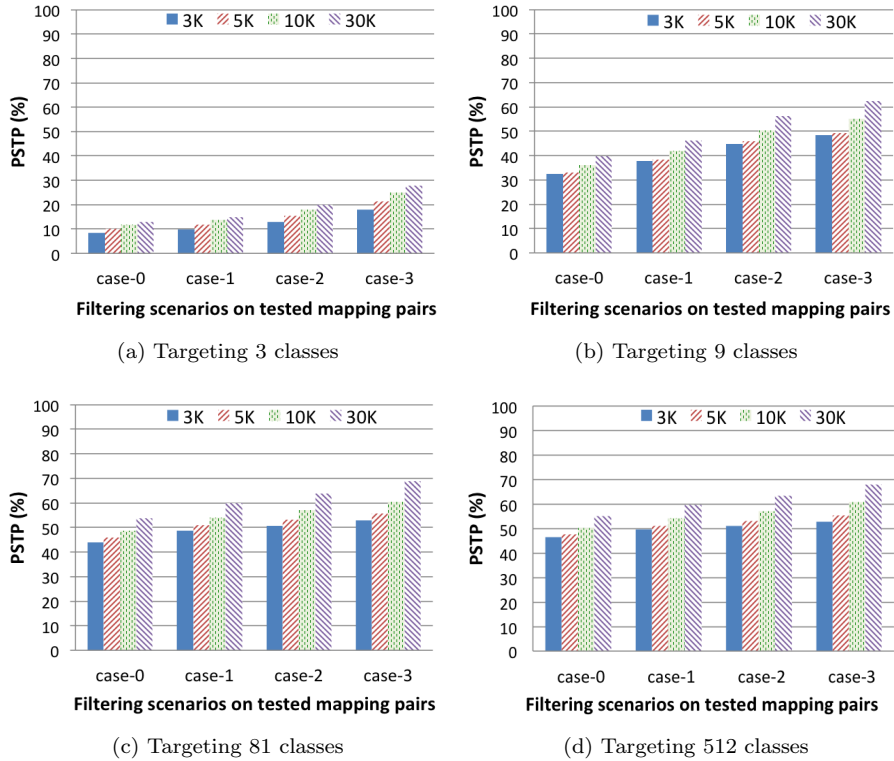


Figure 9: Performance of SVM models according to PSTP criterion w.r.t. different target classes.

813 The reason behind is that with a low number of classes, more mapping pairs tend  
 814 to be classified into the same class. Then, the prediction model would not be  
 815 able to predict their related tendency by comparing their execution time. This  
 816 is characterized through the PUTP measures reported in Figure 10 accordingly.

817 For instance, the PUTP value obtained in Figure 10a explains why the  
 818 PSTP's shown in Figure 9a are quite low. Let us take the best PSTP in Figure  
 819 9a, which is 27.9% and its corresponding PUTP in Figure 10a, which is 63.0%.  
 820 It means that only 37.0% of tested mapping pairs have been classified into dif-  
 821 ferent classes, and among such pairs whose tendency can be compared, 75.4%  
 822 were predicted correctly. However, a predictor that cannot compare mappings  
 823 in most of the cases, even when providing good predictions whenever possible,

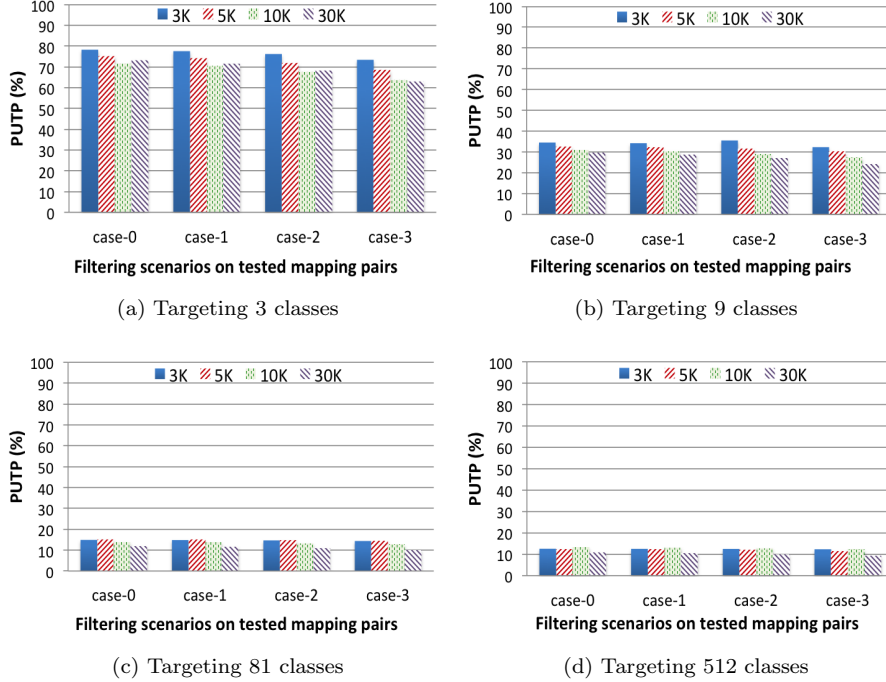


Figure 10: The percentage of unknown tendency predictions for SVM w.r.t. Figure 9.

824 is not preferable. Better predictions can be actually obtained by increasing the  
 825 number of classes from 3 to 81, as observed in Figures 9 and 10: the PUTP  
 826 decreases fast while the PSTP becomes better. When further refining the num-  
 827 ber of classes from 81 to 512, the PUTP decreases very slowly, resulting in a  
 828 poor evolution of the PSTP values (decrease from 69.0% to 68.0%). This in-  
 829 dicates that the benefit from class refinement holds up to certain partitioning  
 830 granularity.

831 On the other hand, one possible reason behind the modest correct prediction  
 832 scores obtained above with SVM technique can relate to the aforementioned  
 833 possible data imbalance of the mapping instance sets (see Figure 8). In the  
 834 next, experiments, we apply the AdaBoost algorithm, which includes Decision  
 835 Trees as a weak learner, to check whether the correct prediction scores can be  
 836 improved.

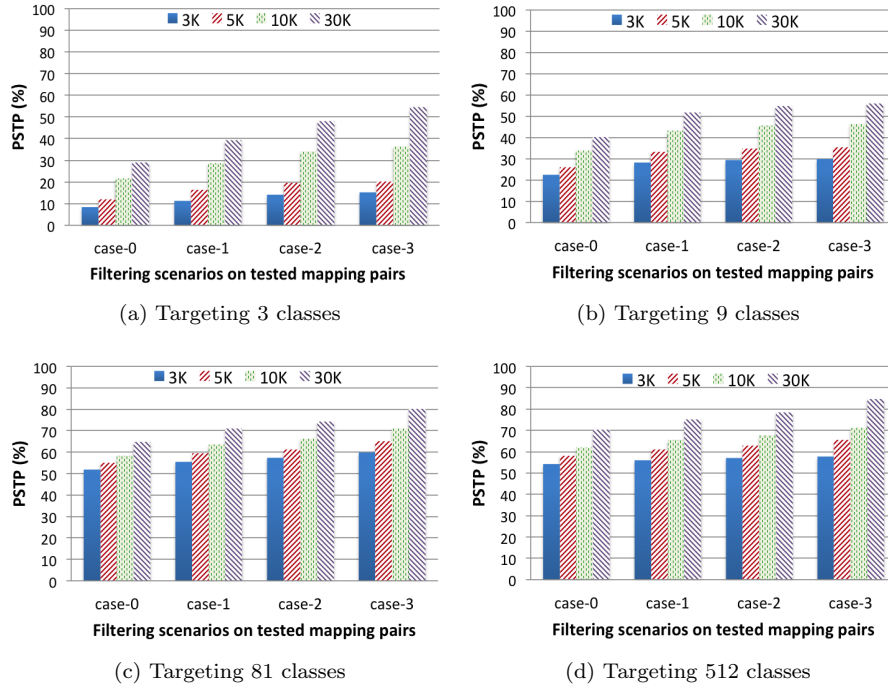


Figure 11: Performance of AdaBoost models according to PSTP criterion w.r.t. different target classes.

837 *6.3. AdaBoost-based Prediction Modeling*

838 To anticipate any imbalanced data issue as discussed in Section 6.1, we  
 839 have selected Decision Trees as *base\_estimator*. Different combinations of the  
 840 other parameter values are then explored, and the best values are selected from  
 841 around 100 explored combinations. Figure 11 reports the obtained prediction  
 842 performance scores w.r.t. different numbers of classes, i.e., 3, 9, 81 and 512.  
 843 These results are obtained by selecting 600 for *n\_estimators*, 0.4 for *learning*  
 844 *rate*, and SAMME as *algorithm*. The duration required to train the AdaBoost  
 845 models on the different training sets varies between 6 to 93 seconds.

846 Comparing the prediction performance values of SVM and AdaBoost, we  
 847 observe that the former outperforms the latter only when targeting 9 classes  
 848 with a maximum PSTP of 62.6% versus 56.1%. However, when targeting 81

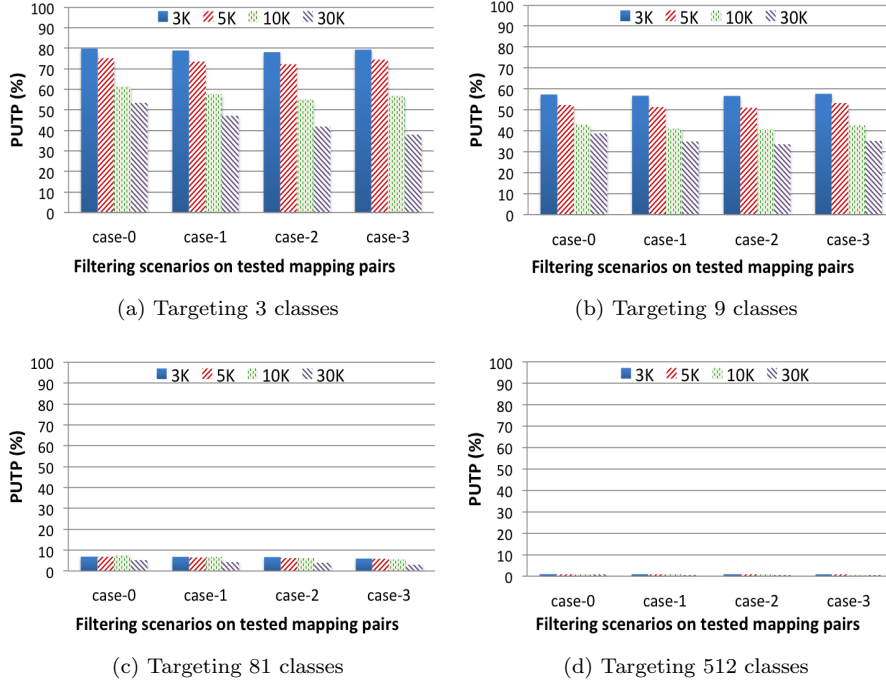


Figure 12: The percentage of unknown tendency predictions w.r.t. Figure 11.

849 and 512 classes, AdaBoost models achieves 80.1% and 84.8% PSTP scores re-  
 850 spectively. In the same time, with AdaBoost a prediction performance always  
 851 keeps improving as the number of classes increases, in contrast to SVM beyond  
 852 81 classes (see Figure 9). To some extent, this indicates that the AdaBoost  
 853 models distinguish more accurately different mapping instances.

854 This is confirmed in Figure 12, which reports the PUTP scores for AdaBoost  
 855 experiments. Here, the PUTP keeps decreasing as the number of classes grows,  
 856 and drops to less than 1% with 512 classes. This result is quite promising,  
 857 in particular when considering that the training time is less than 2 minutes!  
 858 Finally, another interesting observation here is that the PUTP also remains  
 859 stable w.r.t. different working mapping sets.

860 *6.4. ANN-based Prediction Modeling by Regression*

861 We apply now a regression technique combined with MLP instead of classi-  
 862 fication, to the same mapping performance prediction problem.

863 From the explored parameter values, only the most promising ones are re-  
 864 ported here. Four neurons are considered within the single hidden layer in  
 865 selected ANN models. Depending on the size of the considered working map-  
 866 ping sets, the other parameters of the ANN models vary as follows: the ridge  
 867 parameter, the seed and the tolerance parameter are respectively 13.1, 34 and  
 868  $10^{-3}$  for 3K mappings; 13.1 (and 10 for energy prediction), 34 and  $10^{-4}$  for 5K  
 869 mappings; 0.21 (and 10 for energy prediction), 185 and  $10^{-7}$  for 10K mappings,  
 870 and 0.03, 185 and  $10^{-7}$  for 30K mappings. The obtained prediction perfor-  
 871 mance scores are depicted in Figure 13. Here, in addition to execution time,  
 872 we also report the prediction of energy consumption. The training durations  
 873 required for building the corresponding prediction models varies from 5 seconds  
 874 to 4 minutes (note that during the initial ANN parameter exploration, some  
 875 settings took even more than a day to complete, without giving better scores).

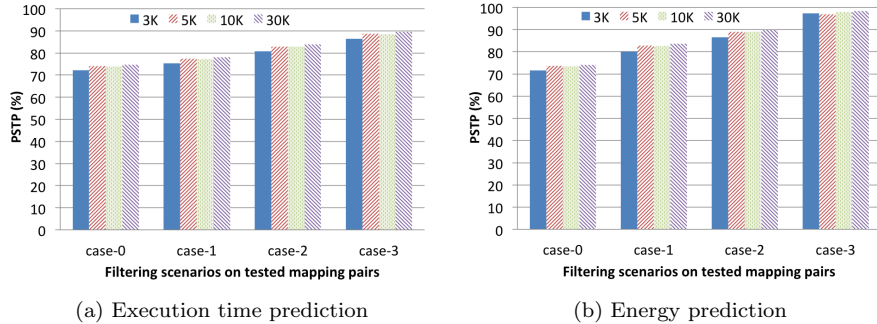


Figure 13: Performance of ANNs according to PSTP criterion, w.r.t. different metrics.

876 In Figure 13a, we obtain for **case-0** a maximum PSTP of about 74.6% while  
 877 the minimum PSTP value is 72.23%. In other words, it means that in the  
 878 best case (i.e., 30K mappings) the obtained prediction model enabled a correct  
 879 comparison for 74.6% of evaluated mapping pairs. In **case-1**, after filtering  
 880 the set of mapping pairs, there remain around 85.87% of this set. The PSTP



881 on this reduced set of mapping pairs yields a better score that reaches up to  
 882 77.98%, which is slightly better than previously. By increasing the filtering of  
 883 the set of mapping pairs, respectively in **case-2** and **case-3**, we observe better  
 884 results, leading respectively to 83.88% and 89.47% of correct mapping tendency  
 885 prediction.

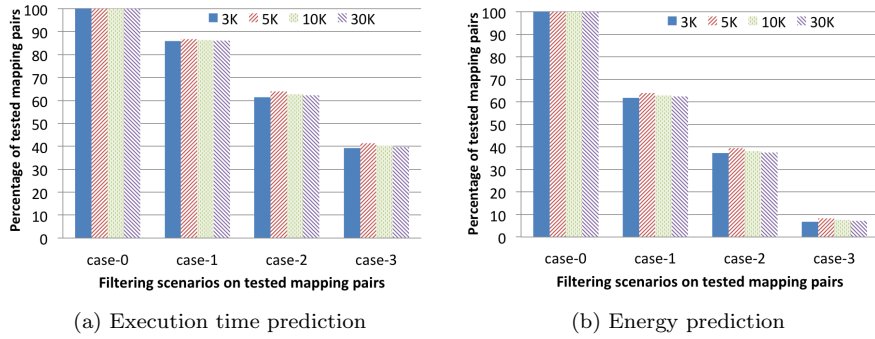


Figure 14: Percentage of tested mapping pairs with ANN-based prediction for **case-0**, **case-1**, **case-2** and **case-3**

886 The prediction performance scores for energy consumption (see Figure 13b)  
 887 generally follow similar trends compared to execution time. In particular, the  
 888 very high score observed in **case-3** results from the low number of tested map-  
 889 ping pairs after the  $\Delta$ -filtering. As a matter of fact, the variation in the en-  
 890 ergy values of the generated mappings is not as large as for the corresponding  
 891 execution time. Figure 14 shows the percentage of tested mapping pairs with  
 892 ANN-based prediction for **case-0**, **case-1**, **case-2** and **case-3**, as a consequence  
 893 of the  $\Delta$ -filter PSTP assessment.

## 894 7. Gained Insights and Discussion

895 The different experiments presented above show how classification and re-  
 896 gression can be used to deal with the prediction of performances multi-task  
 897 application mapping on multicore architectures. First of all, despite the poten-  
 898 tial complexity of the addressed problem, the results obtained especially with  
 899 the AdaBoost classification and ANN regression models are promising.

900 **On the accuracy of evaluated techniques.** Among the two evaluated clas-  
901 sification techniques, AdaBoost provides better results than SVM. Despite the  
902 high success of the latter in literature, it seems that the diversity of learners  
903 combined by the former is beneficial when facing typical situations such as data  
904 imbalance, which is more tractable with Decision Trees supported in AdaBoost.  
905 On the other hand, the ANN-based regression technique provides the most ac-  
906 curate prediction models in terms of PSTP score.

907 In order to confirm the above observations about the three evaluated ma-  
908 chine learning techniques, we carried out similar experiments with a different  
909 application, executed on a 2x3-mesh multicore architecture. This application,  
910 referred to as *light-weight DemoCar*, is composed of 18 periodic runnables and 61  
911 labels [12]. We obtained similar trends as for the case study detailed in Section  
912 6. While all these results are obtained on a 2x3-mesh multicore architecture,  
913 we still expect similar trends when comparing the three techniques for architec-  
914 tures comprising more cores. Nevertheless, their corresponding training costs  
915 may increase as there would be more possible mapping vector configurations to  
916 be taken into account.

917 Now, when focusing on the prediction errors about both execution time and  
918 energy values with ANNs, we obtain the distributions depicted in Figures 15a  
919 and 15b. Their respective mean values are 1.46% and 0.3%, while the standard  
920 deviations are 12.35 and 5.72. The number of mappings with an error less than  
921 20% accounts for 90.0% and 99.8% of tested mapping sets w.r.t. execution  
922 time and energy consumption respectively. This makes the built performance  
923 predictors relevant enough for a meaningful mapping comparison.

924 **On the implications about models integration in dynamic resource**  
925 **allocation.** Our study on mapping performance prediction is motivated by  
926 the dynamic resource allocation flow illustrated in Figure 1. Here, the *mapping*  
927 *heuristics module* is responsible of taking efficient resource allocation decisions  
928 at runtime for enhanced energy-efficiency. For this purpose, it exploits mapping  
929 performance estimations or prediction to select the best resource allocation de-

930 cisions.

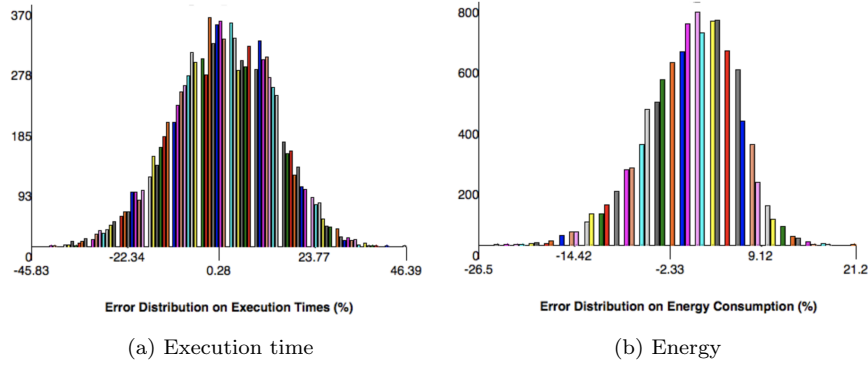


Figure 15: Prediction errors distribution for ANN-based generated models. The Y-axis denotes the number of mapping instances within the different error ranges reported on the X-axis.

931 In embedded real-time systems, the computing and memory resources are  
932 generally limited compared to high-performance or cloud computing systems.  
933 Therefore, it is difficult to envision a mapping design space exploration at run-  
934 time in embedded real-time systems as it will induce an overhead on the actually  
935 executed workload. An alternative pragmatic approach would pre-evaluate dif-  
936 ferent mapping options off-line, which could be leveraged afterwards at runtime.  
937 For more effectiveness, one should make sure to cover *a priori* all relevant design  
938 options. This is not easy to guarantee. An alternative solution, as promoted in  
939 the DreamCloud European project, is to consider fast performance estimation  
940 tools such as the McSim-TLM-NoC [40] or the Interval Algebra simulator [45].  
941 The current work opens an opportunity for an aggressive mitigation of the over-  
942 head related to the on-demand evaluation of mappings with these tools. For  
943 instance, the size and response time (for estimating the performance of a map-  
944 ping) of the ANN-based prediction model defined for the DemoCar application  
945 are respectively about a few tens of kilobytes and microseconds (See Table 2).  
946 The size and response time of the Interval Algebra simulator are respectively  
947 about a few megabytes and milliseconds, while they are about a few seconds  
948 and megabytes respectively for McSim-TLM-NoC. AdaBoost is more costly than

949 Interval Algebra in both response time and model size. For applications with  
 950 high reactivity constraints, the ANN-based prediction model appears then as  
 951 the most preferable.

Table 2: Prediction models versus simulator for mapping performance estimation.

	Response time ( $\mu s$ )	Implement. size (KB)
Interval Algebra simulator [45]	$2 \times 10^3$	$4.13 \times 10^3$
McSim-TLM-NoC simulator [40]	$1.2 \times 10^6$	440
<b>ANN prediction model</b>	<b>63</b>	<b>37</b>
AdaBoost prediction model	$2.4 \times 10^3$	$1.0 \times 10^4$

952 **On the requirements about the solution to the problem addressed in**  
 953 **this paper (see Definition 1).** The AdaBoost and ANN prediction models  
 954 can meet the *accuracy* requirement specified earlier in the problem definition,  
 955 with their respective  $\Delta$ -filter PSTP scores of 84.8% and 89.05%, when  $\Delta =$   
 956 20. Concretely, these scores make the associated prediction models capable of  
 957 identifying, when they exist, candidate mappings that can improve by 20%,  
 958 e.g., the execution time, w.r.t. a reference mapping. While the above PSTP  
 959 scores can be considered as reasonable enough for soft real-time automotive  
 960 applications, higher scores would be however necessary for hard real-time tasks  
 961 in order to make sure they meet their timing requirements.

962 On the other hand, the mapping instances used to train the built prediction  
 963 models are simple enough to be extractable in a costless manner from system  
 964 executions. Only information about task/data allocation on target cores and  
 965 memories, together with the induced global performance numbers, are required.  
 966 This is easily captured via the proposed mapping encodings for fast learning,  
 967 confirming that our approach favors the *feasibility* requirement.

968 Finally, the *responsiveness* requirement is met by the selected prediction  
 969 models. For instance, the average performance prediction time for a mapping  
 970 is 63  $\mu s$  on the desktop machine used to carry out the previous experiments,  
 971 which is quite reasonable.

## 972 **8. Conclusions and Perspectives**

973 In this paper, we applied machine learning to deal with the performance  
974 and energy consumption prediction of applications mapped onto multicore plat-  
975 forms. Our solution relies on simple coarse-grained information, i.e., the map-  
976 ping coordinates of application tasks, and thus avoids intrusion into a system  
977 to obtain training parameters. Two supervised machine learning approaches  
978 are investigated: classification based on SVM and AdaBoost, and regression  
979 based on ANNs. They have been experimented on an automotive application  
980 case study to evaluate their efficiency and effectiveness. The results show that,  
981 under some conditions, AdaBoost and ANNs can achieve very promising pre-  
982 diction accuracy with up to 84.8% and 89.05% respectively, which confirms the  
983 effectiveness of these two models for learning the multicore system behaviors.

984 In the future, we would like to deepen our current proposal with methods  
985 enabling to overcome the possible learning scalability issue while enhancing the  
986 current prediction scores. One possible idea is to enrich the mapping encoding  
987 with more information about system characteristics. This could help the ma-  
988 chine learning models to better learn the system behavior. For instance, making  
989 explicit the data dependency information between runnables or the number of  
990 NoC traversal hops may contribute to a better performance prediction. This  
991 enhancement may come at the cost of large size input data for networks as there  
992 will be additional information to encode. and huge mapping encoding vectors  
993 could be difficultly tractable. Complementary techniques such as unsupervised  
994 machine learning (e.g., feature or attribute selection, which enables to keep only  
995 the most relevant features w.r.t. the learning problem) could be considered to  
996 mitigate this possible risk.

## 997 **Acknowledgements**

998 *This work has been supported by the French ANR agency under the grant*  
999 *ANR-15-CE25-0007-01, within the framework of the CONTINUUM project, and*

1000 *by the Chinese NSFC under grant 61502140. It has been initiated within the Eu-*  
1001 *ropean Community's Seventh Framework Programme (FP7/2007-2013) under*  
1002 *the DreamCloud project, grant agreement no. 611411. The authors would like*  
1003 *to thank Roman Ursu and Manuel Selva who participated in earlier discussions*  
1004 *about the current work.*

## 1005 **References**

- 1006 [1] A. K. Singh, M. Shafique, A. Kumar, J. Henkel, Mapping on multi/many-  
1007 core systems: Survey of current and emerging trends, in: Design Automa-  
1008 tion Conference, 2013, pp. 1:1–1:10.
- 1009 [2] A. Jantsch, N. D. Dutt, A. M. Rahmani, Self-awareness in systems on chip  
1010 - A survey, IEEE Design & Test 34 (6) (2017) 8–26.
- 1011 [3] Y. LeCun, Y. Bengio, G. E. Hinton, Deep learning, Nature 521 (7553)  
1012 (2015) 436–444.
- 1013 [4] D. Nemirovsky, T. Arkose, N. Markovic, M. Nemirovsky, O. Unsal,  
1014 A. Cristal, M. Valero, A general guide to applying machine learning to  
1015 computer architecture, Supercomputing Frontiers and Innovations 5 (1).
- 1016 [5] Z. Wang, M. O'Boyle, Machine learning in compiler optimisation, CoRR  
1017 abs/1805.03441.
- 1018 [6] AMALTHEA Project Consortium, Amalthea - An Open Platform  
1019 Project for Embedded Multicore Systems, [http://amalthea-project.](http://amalthea-project.org/)  
1020 [org/](http://amalthea-project.org/) (2015).
- 1021 [7] L. S. Indrusiak, P. Dziurzanski, A. K. Singh, Dynamic Resource Allocation  
1022 in Embedded, High-Performance and Cloud Computing, River Publishers  
1023 Series in Information Science and Technology, River Publishers, 2016.
- 1024 [8] C. Cortes, V. Vapnik, Support-vector networks, Machine Learning 20 (3)  
1025 (1995) 273–297.

- 1026 [9] Y. Freund, R. E. Schapire, A decision-theoretic generalization of on-line  
1027 learning and an application to boosting, *J. Comput. Syst. Sci.* 55 (1) (1997)  
1028 119–139.
- 1029 [10] C. Fyfe, Artificial neural networks, in: B. Gabrys, K. Leiviskä, J. Strack-  
1030 eljan (Eds.), *Do Smart Adaptive Systems Exist?*, Vol. 173 of *Studies in*  
1031 *Fuzziness and Soft Computing*, Springer Berlin Heidelberg, 2005, pp. 57–  
1032 79.
- 1033 [11] S. B. Kotsiantis, Supervised machine learning: A review of classification  
1034 techniques, *Informatica (Slovenia)* 31 (3) (2007) 249–268.
- 1035 [12] A. Gamatié, R. Ursu, M. Selva, G. Sassatelli, Performance prediction of  
1036 application mapping in manycore systems with artificial neural networks,  
1037 in: *MCSoc*, IEEE Computer Society, 2016, pp. 185–192.
- 1038 [13] T. Stefanov, A. Pimentel, H. Nikolov, Daedalus: System-level design  
1039 methodology for streaming multiprocessor embedded systems on chips,  
1040 *Handbook of Hardware/Software Codesign* (2017) 1–36.
- 1041 [14] X. An, A. Gamatié, É. Rutten, High-level design space exploration for  
1042 adaptive applications on multiprocessor systems-on-chip, *Journal of Sys-*  
1043 *tems Architecture - Embedded Systems Design* 61 (3-4) (2015) 172–184.
- 1044 [15] X. Y. Zhu, M. Geilen, T. Basten, S. Stuijk, Multiconstraint static schedul-  
1045 ing of synchronous dataflow graphs via retiming and unfolding, *IEEE*  
1046 *Transactions on Computer-Aided Design of Integrated Circuits and Sys-*  
1047 *tems* 35 (6) (2016) 905–918.
- 1048 [16] A. Gamatié, S. Le Beux, E. Piel, R. Ben Atitallah, A. Etien, P. Marquet,  
1049 J.-L. Dekeyser, A model-driven design framework for massively parallel  
1050 embedded systems, *ACM Trans. Embed. Comput. Syst.* 10 (4) (2011) 39:1–  
1051 39:36. doi:10.1145/2043662.2043663.  
1052 URL <http://doi.acm.org/10.1145/2043662.2043663>

- 1053 [17] A. K. Singh, C. Leech, B. K. Reddy, B. M. Al-Hashimi, G. V. Merrett,  
1054 Learning-based run-time power and energy management of multi/many-  
1055 core systems: Current and future trends, *J. Low Power Electronics* 13 (3)  
1056 (2017) 310–325.
- 1057 [18] D. Biswas, V. Balagopal, R. Shafik, B. M. Al-Hashimi, G. V. Merrett,  
1058 Machine learning for run-time energy optimisation in many-core systems,  
1059 in: *Proceedings of the Conference on Design, Automation & Test in Europe,*  
1060 *DATE '17*, European Design and Automation Association, 3001 Leuven,  
1061 Belgium, Belgium, 2017, pp. 1592–1596.  
1062 URL <http://dl.acm.org/citation.cfm?id=3130379.3130749>
- 1063 [19] A. Irpan, Deep reinforcement learning doesn't work yet, [https://www.  
1064 alexirpan.com/2018/02/14/r1-hard.html](https://www.alexirpan.com/2018/02/14/r1-hard.html) (2018).
- 1065 [20] Y. Zhang, M. A. Laurenzano, J. Mars, L. Tang, Smite: Precise qos predic-  
1066 tion on real-system smt processors to improve utilization in warehouse scale  
1067 computers, in: *Proceedings of the 47th Annual IEEE/ACM International*  
1068 *Symposium on Microarchitecture, MICRO-47*, IEEE Computer Society,  
1069 Washington, DC, USA, 2014, pp. 406–418. doi:10.1109/MICRO.2014.53.  
1070 URL <http://dx.doi.org/10.1109/MICRO.2014.53>
- 1071 [21] S. Sankaran, Predictive modeling based power estimation for embedded  
1072 multicore systems, in: *Proceedings of the ACM International Conference*  
1073 *on Computing Frontiers, CF '16*, ACM, New York, NY, USA, 2016, pp.  
1074 370–375. doi:10.1145/2903150.2911714.  
1075 URL <http://doi.acm.org/10.1145/2903150.2911714>
- 1076 [22] P.-J. Micolet, A. Smith, C. Dubach, A machine learning approach to map-  
1077 ping streaming workloads to dynamic multicore processors, *SIGPLAN Not.*  
1078 51 (5) (2016) 113–122. doi:10.1145/2980930.2907951.  
1079 URL <http://doi.acm.org/10.1145/2980930.2907951>
- 1080 [23] A. Matsunaga, J. A. B. Fortes, On the use of machine learning to predict  
1081 the time and resources consumed by applications, in: *Proceedings of the*



- 1082 2010 10th IEEE/ACM International Conference on Cluster, Cloud and  
1083 Grid Computing, CCGRID '10, IEEE Computer Society, Washington, DC,  
1084 USA, 2010, pp. 495–504. doi:10.1109/CCGRID.2010.98.  
1085 URL <https://doi.org/10.1109/CCGRID.2010.98>
- 1086 [24] N. J. Yadwadkar, B. Hariharan, J. E. Gonzalez, R. Katz, Multi-task learn-  
1087 ing for straggler avoiding predictive job scheduling, *J. Mach. Learn. Res.*  
1088 *17* (1) (2016) 3692–3728.  
1089 URL <http://dl.acm.org/citation.cfm?id=2946645.3007059>
- 1090 [25] Y. Wen, Z. Wang, M. F. P. O’Boyle, Smart multi-task scheduling for opencl  
1091 programs on cpu/gpu heterogeneous platforms, in: 21st International Con-  
1092 ference on High Performance Computing (HiPC), 2014, pp. 1–10.
- 1093 [26] B. Taylor, V. S. Marco, Z. Wang, Adaptive optimization for opencl pro-  
1094 grams on embedded heterogeneous systems, in: Proceedings of the 18th  
1095 ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools  
1096 for Embedded Systems, 2017, pp. 11–20.
- 1097 [27] H. Sayadi, N. Patel, A. Sasan, H. Homayoun, Machine learning-based ap-  
1098 proaches for energy-efficiency prediction and scheduling in composite cores  
1099 architectures, in: 2017 IEEE International Conference on Computer De-  
1100 sign, ICCD 2017, Boston, MA, USA, November 5-8, 2017, IEEE Computer  
1101 Society, 2017, pp. 129–136. doi:10.1109/ICCD.2017.28.  
1102 URL <https://doi.org/10.1109/ICCD.2017.28>
- 1103 [28] R. Bitirgen, E. Ipek, J. F. Martinez, Coordinated management of multiple  
1104 interacting resources in chip multiprocessors: A machine learning approach,  
1105 in: Proceedings of the 41st Annual IEEE/ACM International Symposium  
1106 on Microarchitecture, MICRO 41, IEEE Computer Society, Washington,  
1107 DC, USA, 2008, pp. 318–329.
- 1108 [29] D. Nemirovsky, T. Arkose, N. Markovic, M. Nemirovsky, O. Unsal,  
1109 A. Cristal, A machine learning approach for performance prediction and

- 1110 scheduling on heterogeneous cpus, in: 29th International Symposium on  
1111 Computer Architecture and High Performance Computing (SBAC-PAD),  
1112 2017, pp. 121–128.
- 1113 [30] C. V. Li, V. Petrucci, D. Mossé, Exploring machine learning for thread  
1114 characterization on heterogeneous multiprocessors, *Operating Systems Re-*  
1115 *view* 51 (1) (2017) 113–123.
- 1116 [31] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J.  
1117 McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J.  
1118 Hand, D. Steinberg, Top 10 algorithms in data mining, *Knowledge and*  
1119 *Information Systems* 14 (1) (2008) 1–37.
- 1120 [32] T. G. Dietterich, Machine-learning research – four current directions, *AI*  
1121 *MAGAZINE* 18 (1997) 97–136.
- 1122 [33] R. E. Schapire, The strength of weak learnability, *Machine Learning* 5 (2)  
1123 (1990) 197–227. doi:10.1023/A:1022648800760.  
1124 URL <https://doi.org/10.1023/A:1022648800760>
- 1125 [34] Wikipedia, Artificial Neural Networks: approximation theorem, [https:](https://en.wikipedia.org/wiki/Universal_approximation_theorem)  
1126 [//en.wikipedia.org/wiki/Universal\\_approximation\\_theorem](https://en.wikipedia.org/wiki/Universal_approximation_theorem).
- 1127 [35] Machine Learning Group at the University of Waikato, *Weka - Data Mining*  
1128 *Software in Java*, <http://www.cs.waikato.ac.nz/~ml/weka> (2017).
- 1129 [36] J. Ceng, J. Castrillón, W. Sheng, H. Scharwächter, R. Leupers, G. Ascheid,  
1130 H. Meyr, T. Isshiki, H. Kunieda, Maps: An integrated framework for mpso  
1131 application parallelization, 2008, pp. 754–759.
- 1132 [37] A. Mallik, S. Mamagkakis, C. Baloukas, L. Papadopoulos, D. Soudris,  
1133 S. Stuijk, O. Jovanovic, F. Schmoll, D. Cordes, R. Pyka, P. Marwedel,  
1134 F. Capman, S. Collet, N. Mitas, D. Kritharidis, Mnemee – an automated  
1135 toolflow for parallelization and memory management in mpso  
1136 in: 48th ACM/IEEE Design Automation Conf. (DAC’11), San Diego, CA  
1137 - USA, 2011.

- 1138 [38] K. Latif, C. Effiong, A. Gamatié, G. Sassatelli, L. Zordan, L. Ost, P. Dzi-  
1139 urzanski, L. Soares Indrusiak, An Integrated Framework for Model-Based  
1140 Design and Analysis of Automotive Multi-Core Systems, in: FDL: Forum  
1141 on specification & Design Languages, Work-in-Progress Session, Barcelona,  
1142 Spain, 2015.  
1143 URL <https://hal-lirmm.ccsd.cnrs.fr/lirmm-01418748>
- 1144 [39] DreamCloud Project Consortium, McSim - Manycore platform Simula-  
1145 tion tool for NoC-based platform at a Transactional Level Modeling level,  
1146 <https://github.com/DreamCloud-Project> (2016).
- 1147 [40] K. Latif, M. Selva, C. Effiong, R. Ursu, A. Gamatié, G. Sassatelli, L. Zor-  
1148 dan, L. Ost, P. Dziurzanski, L. Indrusiak, Design space exploration for  
1149 complex automotive applications: An engine control system case study, in:  
1150 8th Workshop on Rapid Simulation and Performance Evaluation: Methods  
1151 and Tools, Prague, Czech Republic, 2016.
- 1152 [41] CNRS, Deliverable D5.2 – Abstract Simulation Platform, European Dream-  
1153 Cloud FP7 Project - <http://www.dreamcloud-project.org/results>  
1154 (2015).
- 1155 [42] A. Butko, F. Bruguier, A. Gamatié, G. Sassatelli, D. Novo, L. Torres,  
1156 M. Robert, Full-system simulation of big.little multicore architecture for  
1157 performance and energy exploration, in: 2016 IEEE 10th International  
1158 Symposium on Embedded Multicore/Many-core Systems-on-Chip (MC-  
1159 SOC), 2016, pp. 201–208. doi:10.1109/MCSoc.2016.20.
- 1160 [43] C. J. van Rijsbergen, Information Retrieval, Butterworth, 1979.
- 1161 [44] H. He, E. A. Garcia, Learning from imbalanced data, *IEEE Trans. on*  
1162 *Knowl. and Data Eng.* 21 (9) (2009) 1263–1284.
- 1163 [45] L. S. Indrusiak, P. Dziurzanski, An interval algebra for multiprocessor re-  
1164 source allocation, in: SAMOS, IEEE, 2015, pp. 165–172.