DRuiD: Designing Reconfigurable Architectures with Decision-making Support

Giovanni Mariani†‡, Gianluca Palermo†, Roel Meeuws‡, Vlad-Mihai Sima‡, Cristina Silvano†, Koen Bertels‡

* Università della Svizzera Italiana - ALaRI. Email: marianig@alar.ch
† Politecnico di Milano. Email: {gpalermo, silvano}@elet.polimi.it
‡ Delft University of Technology. Email: {v.m.sima, k.l.m.bertels}@tudelft.nl

Abstract—Application development for heterogeneous platforms requires to code and map functionalities on a set of different computing elements. As a consequence, the development process needs a clear understanding of both, application requirements and heterogeneous computing technologies. To support the development process, we propose a framework called DRuiD capable of learning application characteristics that make them suitable for certain computing elements. The framework is composed of an expert system that supports the designer in the mapping decision and gives hints on possible code modifications to be applied to make the functionality more suitable for a computing element. The experimental results are tailored for a heterogeneous and reconfigurable platform (the Xilinx-ml510) including two computational elements, i.e. a Virtex5 FPGA and a PowerPC. The expert system identifies 88.9% of the times what are the functionalities that are accelerated efficiently by using the FPGA, without requiring the kernel porting. Additionally, we present two case studies demonstrating the potentialities of the framework to give hints on high level code modifications for an efficient kernel mapping on the FPGA.

I. INTRODUCTION

Heterogeneous computing architectures are becoming more and more widespread solutions in both general-purpose and application-specific domains. Generally heterogeneous architectures are coupling programmer-friendly general-purpose processing elements with specialized high-performance computing elements, such as FPGAs, GPUs and DSPs, increasing the degree of freedom for application development. To effectively utilize those architectures, it is important that the developers understand how to exploit the heterogeneous processing elements. Currently, existing tools are lacking of supporting the high level decision making process related to the design of heterogeneous systems [1]. Today solutions mainly rely on engineers’ experiences, however this problem becomes even more evident once automatic tools are used as intermediate translator to port the code on specialized computing elements, such as High Level Synthesis (HLS) tools [2], [3] or parallelizing compilers for GPUs [4]. Given an application description, it is very hard to say whether or what application functionalities can be accelerated by using heterogeneous processing elements (e.g. FPGAs or GPUs). Moreover, trying different application implementations to pick up the best one increases significantly the development costs.

To provide support during the development of heterogeneous systems, we propose to integrate the knowledge gathered by the development work of different expert designers into an expert system. More specifically, we propose a framework called DRuiD based on machine learning approach combining random forests [5] and genetic algorithms. The underlying idea is to let the DRuiD framework to extract and learn what characteristics make application functionalities (or kernels) be more suitable for a certain computing technology.

In this work, we focus on FPGA-based heterogeneous architectures. The methodology leverages a high-level synthesis tool to translate C code into a synthesizable hardware description language (such as VHDL). The availability of this tool allows us to describe both HW and SW by using a unique language, i.e. the ANSI C. Starting from the C implementation of the whole application, DRuiD selects the best computational element to be used for each application kernel and, if not accelerated by the computational elements, gives hints on code transformations to be applied for an efficient porting of the kernel. To demonstrate the effectiveness of the approach, we used a Xilinx-ml510 heterogeneous platform including the PowerPC and a Virtex-5 FPGA as computational elements.

The remainder of this work is organized as follows. Section II reports the background work in the specific field. Section III presents the proposed DRuiD framework, the training methodology and the support it can provide at design-time. Section IV provides insight about the prediction accuracy and presents two case studies to encompass the potentialities of the proposed approach, while Section V concludes the paper.

II. BACKGROUND

Several data mining and machine learning techniques have been proposed to model performance of computer architectures at different abstraction level [6], [7], [8]. Those works mainly differ in terms of the proposed performance model, target computing technology and/or performance metrics under study. While these models are precise enough to drive the design space exploration process of homogeneous computing architectures, they are not suitable for analyzing performance for heterogeneous systems.

Today solutions for the design of heterogeneous architectures either assume that costs related to mapping application kernels over different computational elements are available [9], [10], [11] or they rely on virtual platform simulators (or models) where designer can plug-in different computational elements and evaluate the proposed solution [12], [13]. In both approaches, first the kernels of interest should be ported for execution on the target computational elements and then evaluated either by real execution or via simulation, making the application mapping a relatively simple problem.

The problem is that, porting different kernels on different components is a time consuming and error prone procedure.
When considering FPGA based technology, application kernels can be ported to this component by using high-level synthesis tools such as *dvarg* or *Vivado* [2], [3]. Those tools generally accept a subset of the input programming language. To be compliant with tool specific constraints one might need to rewrite significant parts of the kernels’ code. Ideally, one would like to know the kernels to be mapped on the different computing elements before actually porting them, eventually employing a cross-component prediction method. Literature on this topic is very limited [14], [15], [16], [17], [18], [19]. The idea in these works is to predict performance for a target component given performance indices collected for a source component. The source component is either a host machine [14], [15], an intermediate code representation [16], [17], or simply a different configuration of the target architecture [18], [19]. In all cases, the computational element destination of the prediction belongs to a specific architecture family.

Performance prediction models suitable for truly heterogeneous architectures are today not available in literature. When dealing with the problem at hand, analytic performance prediction techniques are too much inaccurate. To tackle this issue, we propose a machine learning approach which goal is not to accurately predict kernel performance over heterogeneous elements but simply suggesting the optimal kernel mapping.

III. THE PROPOSED DRuiD METHODOLOGY

In this section we present the proposed DRuiD methodology. The implemented framework is based on an expert system that, given a parametric characterization of a specific kernel, classifies the kernel in terms of the processing element providing the best performance. Decision trees have been used in the past in the same area [15] as a flexible data-mining tool used to classify application performance depending on the code and machine characteristics. Additionally, it is rather easy to inspect such models to understand why a certain kernel is associated to a given class. Taking decision trees as baseline solution, we propose a methodology based on one of their extension known as random forests [5]. The random forest can be seen as a collection of trees where the classification is done by using a majority vote among the trees. This choice is driven by the evidence of a better classification accuracy (as it will be demonstrated in the experimental results section).

The basic assumption for the application of our methodology is that the translation process between the original software code and the code for the computing elements is done by means of an automatic tool. This requirement is needed since the correlation between the original code and the generated code is higher with respect to a manual translation, making the results more predictable. In our work, we focused on the design of reconfigurable architectures (FPGA-based) and we employed as automatic translator for the kernels an high level synthesis tool starting from ANSI C code.

The proposed methodology requires to characterize the original version of the kernel by means of a vector $\mathbf{v}_k$ of SW parameters (or metrics). These metrics are generated by using two different technologies: (i) a Static Code Analysis tool (SCA) [20] to extract information related to the source code itself (such as average/maximum loop nesting, control flow complexity), and (ii) a Microarchitectural Independent Characterization (MICA) [14] to extract dynamic profiling information on a host machine (such as memory usage, branch predictability, instruction types). The expert system takes as input the SW metrics characterizing the kernel and returns as output the decision on the computing element to employ, that for reconfigurable architecture is the accelerator implementation on the FPGA or the SW execution on a core.

Before being used, the expert system needs to be trained (Figure 1a). During the training process, kernels belonging to the training set $K$ are executed and profiled on the target computing element set $C$, and each kernel $k \in K$ is classified with a class $\eta(k) \in C$ that represents its best computational element. Formally:

$$\eta(k) = \arg \min_{c \in C} \delta(k, c)$$

where $\delta(k, c)$ is the profiled execution time of kernel $k$ on the computational element $c$.

Once the expert system is trained, it can be used to decide the best computational element $\hat{\eta}(v_k) \in C$ to be used for kernels coming from new, previously unobserved, applications (Figure 1b). Additionally, the expert system is provided with a query interface that generates a report about each of its decision. A report includes the relevance that each SW metric had during the selection of $\hat{\eta}(v_k)$, and suggests possible kernel code modifications to make it more suitable for the desired computational element.

Hereafter, we describe more in detail for the proposed DRuiD methodology the (i) kernel characterization, (ii) the training methodology and the (iii) expert system usage.

(i) Kernel Characterization. The proposed methodology requires that each kernel $k$ is characterized with a vector $v_k$ composed of 92 SCA metrics and 97 MICA metrics. For conciseness, we do not list here all 189 SW metrics, the reader can refer to [20], [14] for a more detailed analysis. Table I groups SW metrics by source (SCA or MICA) and highlights the number of metrics in each group.
TABLE I: Types of software metrics.

<table>
<thead>
<tr>
<th>Source</th>
<th>Group name and acronym</th>
<th>Number of metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCA</td>
<td>Instruction type (IT)</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>Control flow (LF)</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>Data flow (DF)</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>Variables scope and location (VSL)</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>Instruction level parallelism (ILP)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Memory and register access pattern (MRAP)</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>Branch predictability (BP)</td>
<td>7</td>
</tr>
<tr>
<td>MICA</td>
<td>Instruction type (IT)</td>
<td>4</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>189</td>
</tr>
</tbody>
</table>

SCA metrics [20] report static code characteristics as follows. Instruction types present in the code, (e.g. float or integer addition/multiplication, bitwise and logical operations, etc.). Control flow information (e.g. number and length of basic blocks, cyclomatic complexity, etc.). Data flow information (e.g. number of operands and operators, the Elshof complexity metrics, etc.). variable scope and location (e.g. use of global/local variables, use of stack or heap memory, etc.).

MICA metrics [14] include information related to dynamic execution (on a host machine) as follows. Instruction types of executed code (e.g. arithmetic, control flow, floating point operations etc.). Instruction level parallelism information about potential code parallelism on an ideal machine with infinite computing resources. Memory and register access pattern information refers to read and write requests, access stride information, memory and register reuse distance.

SCA and MICA metrics represent complementary information (we will provide empirical evidence to validate this statement in the experimental results section). However, having 189 SW metrics generates several issues in the construction and comprehension of the model and many of the 189 metrics are correlated each other and some do not carry useful data for the tackled problem. The decision on what metrics to include in the model is not trivial and for this reason we included all the metrics for our analysis leaving to the training phase to select which ones to use.

(ii) Training the expert system. The training of the expert system is done through a combination of a genetic algorithm (GA) and random forests (RF) (Figure 2). The GA is responsible for selecting the most suitable SW metrics to build an accurate decision-making system based on RF.

The GA processes a population of individuals using the traditional crossover and mutation operators (GEN block) and tournament based fitness selection (SEL block). Each individual guesses a function that reduces the vector $v_k$ into a shorter vector $g(v_k)$. A bit vector of the same length of $v_k$ is used to encode GA individuals $g$. A ‘1’(’0’) in the encoding of $g$ means that the corresponding SW metric is reported/not-reported in $g(v_k)$. Figure 3 shows an example for 4 SW metrics. For each individual $g$, a random forest model $\hat{\eta}_g$ is trained to fit the kernel classes $\eta(k)$ for the kernels in the training library $k \in K$: $\hat{\eta}_g(v_k) \sim \eta(k)$. Each random forest model $\hat{\eta}_g$ differs in terms of the SW metrics it includes, i.e. $g(v_k)$. The GA fitness function is defined in terms of prediction accuracy of the trained RF and, in case of the same value, the configuration using the less number of SW metrics is preferred. The GA population $G$ is initialized by a random sampling technique.

While the GA is in charge of the metric selection, the RF is the entity that is in charge of the mapping decision. The random forest (RF) is trained by using $R$ [21]. In particular, each tree $\tau \in RF$ is trained separately by using a kernel set $K_\tau$ such that $K_\tau \subseteq K$ by using an out of the bag approach, i.e. removing randomly some of the kernels. The training of each tree $\tau$ is an iterative process. At each iteration, some variables in the vector $g(v_k)$ are randomly selected as next decision candidates. Among these ones, the one being the best discriminant (in terms of the Gini index [5]) for the identification of the class $\eta(k)$ is introduced in the tree. The introduction of randomness in the tree construction and the presence of multiple trees let RF be a more accurate and general model than a single decision tree. The output of the entire training process is the RF providing the best prediction accuracy and which classification model is referred as $\hat{\eta}(v_k)$.

(iii) Querying the expert system. Once the RF is trained, it can be used for predicting the best computing element where to map a new kernel $c$. Each tree $\tau \in RF$ contributes with a vote $\tau(v_k) \in C$ for the computing element allocation $\hat{\eta}(v_k)$. A majority voting scheme is considered:

$$\hat{\eta}(v_k) = \underset{c \in C}{\text{argmax}} \left| \left\{ \tau(v_k) \in RF : \tau(v_k) = c \right\} \right|$$

(2)

Additionally, the application developer willing to port a kernel $c$ to a target component $c \in C$ can query the expert system to generate a report about the SW metrics that influence the decision of mapping $k$ on $c$, with the goal of receiving hints on possible code modifications to apply. In fact, the presence of multiple trees $\tau \in RF$ lets the expert system be fuzzy, i.e. it is generally the case that some trees have a vote $\tau(v_k)$ that differs from the majority $\hat{\eta}(v_k)$. Let us call $\rho_{RF}(c, v_k)$ the number of trees $\tau \in RF$ such that $\tau(v_k) = c$. Let us define the SW metric vectors $v_k^+\rho$ and $v_k^-\rho$ such as the SW metric vector $v_k$ where the SW metric $m$ has been set respectively to its maximum and minimum values. We define the influence $\iota^+] \rho$ and $\iota^\rho$ as the variation in the votes with respect to a positive or negative variation of the metric $m$:

$$\rho_{RF}(c, v_k) = \left| \{ \tau(v_k) \in RF : \tau(v_k) = c \} \right|$$

(3)

$$\iota^+] \rho(c, v_k) = \rho_{RF}(c, v_k^+\rho) - \rho_{RF}(c, v_k)$$

(4)

$$\iota^-\rho(c, v_k) = \rho_{RF}(c, v_k^-\rho) - \rho_{RF}(c, v_k)$$

(5)
Given the target computational element $c$ and the SW metric vector $v_k$, the expert system reports as output the following metrics: (i) Number of trees voting for the suggested component $\hat{v}(v_k)$ (the target component $c$ might differs from $\hat{v}(v_k)$); (ii) Number of trees voting for the target component $c$. (iii) Influences $\delta^{+}(c, v_k)$ and $\delta^{-}(c, v_k)$ for each SW metric $m$ considered in the RF model.

Note that usually, porting a kernel to a given component (e.g. GPU or FPGA) requires to edit the kernel code to be compliant with a subset of the C standard and/or to insert pragmas to specify where computations and variables shall be mapped to. This work might be long and tedious, depending on the target computational element and the tool-chain in use. The usage of DRuiD does not require to actually port the kernel on the target processing element, thus saving significant development time.

### IV. Experimental Results

To validate and assess the proposed methodology, we applied it to a computing element set composed of the architectural elements available on an m510 machine [22]: a Virtex5 FPGA and a PowerPC (PPC). The experimental campaign has been conducted by using a collection of 97 kernels/applications, some of them taken from MediaBench [23] and MiBench [24], originally written in C and coming from different application domains such as multimedia, cryptography, digital signal processing, automotive, mathematics, physics. The C to VHDL translation of the set of benchmarks is done by using dwarv [2] while the static code analysis (SCA) is implemented by using CoSy [25]. The dynamic profiling (MICA-based) and the subsequent evaluation of the kernels/applications set on the computing element set, i.e. PPC and FPGA, has been done by using at least 5 different datasets from real scenarios. The SW metric values $v_k$ and the execution time $\delta(k, c)$ used in this work refer to the median values over the different datasets. Finally, while the PPC operating frequency is 400 MHz and the maximum frequency supported for the FPGA is 325 MHz, the actual operating frequency of the the kernels, once map on the FPGA, can be lower than 325 MHz, depending on the synthesis process.

Three sets of experimental results will be shown hereafter: (i) the prediction capability of the proposed approach; (ii) the analysis on SW metrics used to build the predictive model; (iii) two case studies where the predictive model has been used to give hints about how to modify the original C code to have benefits of using the FPGA instead of the PPC.

#### (i) Model Accuracy

To assess the prediction capability of the proposed random forest (RF) technique, we compared it with respect to a decision-tree-based methodology (Tree) that implements the state of the art technique proposed in [15] for heterogeneous systems. Additionally, to demonstrate that SCA and MICA metrics provide complementary information, the proposed expert system has been evaluated in three versions: RF-SCA, RF-MICA and RF-all where the random forest has been trained respectively by using SCA only, MICA only or, as in DRuiD, both set of metrics. Finally, to complete the validation we add a random selection process (Random), where the best computational element $\eta(v_k)$ is randomly selected among the computing elements.

To evaluate the model accuracy in terms of prediction capability, we use a 30-fold cross-validation method. This method consists of first partitioning the set $K$ of 97 kernels in 30 subsets and then, for each one of the 30 subsets, removing that subset, training the models with the remaining subsets, and perform the evaluation of the predictions $\hat{\eta}(v_k) \sim \eta(k)$ for the kernels $k$ within the removed subset (not involved in the training phase).

In this work, we are interested in verifying if the model is capable to predict when the FPGA efficiently accelerate a kernel $k$ with respect to its software execution. Let’s define as accelerated set $Acc$ the set of kernels that are successfully accelerated by the FPGA, $Acc = \{k \in K : \eta(k) = FPGA\}$, and non-accelerated set the set of kernels which FPGA implementation leads to a slowdown, $NAcc = \{k \in K : \eta(k) = PPC\}$. Similarly, we define the accelerated and non-accelerated kernel sets referring to model predictions; respectively $\hat{Acc} = \{k \in K : \hat{\eta}(v_k) = FPGA\}$ and $\hat{NAcc} = \{k \in K : \hat{\eta}(v_k) = PPC\}$.

Let us then define the truly accelerated probability $TAcc$ as the probability that a kernel $k$ classified by the model as accelerated is actually accelerated, $TAcc = \{k \in \hat{Acc} : k \in Acc\}/\hat{Acc}$, and the truly non-accelerated probability $TNAcc$ is the probability for a kernel classified as non-accelerated to actually be non-accelerated: $TNAcc = \{k \in \hat{NAcc} : k \in NAcc\}/\hat{NAcc}$. Those probabilities represent the capability of the models to discriminate between accelerated and non-accelerated kernels and can be used as accuracy metrics for our goal. In particular, higher are the values better is the prediction accuracy.

Figure 4 shows the comparison results in term of accuracy of all the methods involved in the validation of the proposed approach. Due to its nature the Random method has the $TAcc (TNAcc)$ probability equal to the ratio of kernels that are accelerated(non-accelerated) in the entire set (horizontal lines in the figure). Regarding the other methods, both probabilities are always higher than those for the Random method, but while for the $TAcc$ it is evident, only small improvements occur for the $TNAcc$. The phenomenon on the $TNAcc$ is due to the large fraction of kernels that do not present advantages in being mapped on FPGA (79%) that makes less evident the learning capability of both Tree and RF methods. On the other hand, the $TAcc$ of the RF-based methods (regardless the type of used SW metrics) is higher (around 80% on average) with

<table>
<thead>
<tr>
<th>Probability [%]</th>
<th>Accelerated</th>
<th>Non-accelerated</th>
</tr>
</thead>
<tbody>
<tr>
<td>$TAcc$</td>
<td>Random</td>
<td>Random</td>
</tr>
<tr>
<td>$TNAcc$</td>
<td>Random</td>
<td>Random</td>
</tr>
</tbody>
</table>

![Fig. 4: Model accuracy comparison in terms of truly accelerated (TAcc) and truly non-accelerated (TNAcc) probabilities.](image-url)
TABLE II: Most relevant SW metrics used by RF-all.

<table>
<thead>
<tr>
<th>Name</th>
<th>Source</th>
<th>Group</th>
<th>SW metric description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stride32K</td>
<td>MICA</td>
<td>MICA</td>
<td>% of memory accesses with stride lower than 32K</td>
</tr>
<tr>
<td>Operands</td>
<td>SCA</td>
<td>DF</td>
<td>Number of operands (variables or constants)</td>
</tr>
<tr>
<td>Arithmetic</td>
<td>MICA</td>
<td>II</td>
<td>% of arithmetic operations executed</td>
</tr>
<tr>
<td>Heap</td>
<td>SCA</td>
<td>VSL</td>
<td>% of code statements accessing heap memory</td>
</tr>
<tr>
<td>RegAge</td>
<td>MICA</td>
<td>MICA</td>
<td>Probability that register age is lower than 16</td>
</tr>
<tr>
<td>FloatPre</td>
<td>SCA</td>
<td>IT</td>
<td>Number of floating point operators weighted by their precision</td>
</tr>
<tr>
<td>FloatExc</td>
<td>SCA</td>
<td>IT</td>
<td>% of floating point operations executed</td>
</tr>
<tr>
<td>MemReuse</td>
<td>MICA</td>
<td>MICA</td>
<td>Memory reuse distance 32-64 [14]</td>
</tr>
<tr>
<td>BranchPre</td>
<td>MICA</td>
<td>BP</td>
<td>% of branch misprediction (using a Partial Match with global history table of length 12)</td>
</tr>
<tr>
<td>FloatAUs</td>
<td>SCA</td>
<td>IT</td>
<td>Required floating point ALUs</td>
</tr>
</tbody>
</table>

![Fig. 5: Percentage of SCA and MICA metrics used.](image)

respect to the Tree method (50%). The proposed method (RF-all) outperforms all the others showing a $T_{Acc}$ value of 89%.

(ii) SW metrics analysis. In this section, we analyze more in detail the usage of the SW metrics within the proposed methodology. Figure 5 shows, for each one of the methodologies considered in the validation, the percentage of SCA and MICA metrics used to build the models. The Random methodology has not been shown since it does not use any SW metric. It is possible to note that while the RF-SCA and RF-MICA use respectively only SCA and MICA metrics, both RF-all and Tree-all models have a good balance between the two sets of metrics, coming from both static code analysis (SCA) and dynamic profiling (MICA).

Analyzing more in detail the proposed expert system (RF-all), it includes a total of 24 metrics, 11 out of them coming from SCA and 13 from MICA. Additionally, considering the metrics’ relevance measured in terms of the mean Gini index [5] reduction obtained by introducing the metric within the trees of the random forest [21], Table II shows the 10 most relevant metrics out of the 24 used for the model. The table reports the name of each metric together with a brief description, the source set (MICA or SCA) and the metric group type (see Table I). Almost all the groups of Table I are represented (except for SCA-CF) with predominance of the instruction type (MICA/SCA-II) related metrics.

(iii) Querying the expert system. The proposed expert system can even be used to receive hints on how to modify the code to move the kernel from the non-accelerated set to the accelerated-one. In fact, the final user can query the expert system to generate a report about the SW metrics that influence the decision of mapping a kernel $k$ on the component $c$. The report consists of the number of trees $\rho_{RF}(c, v_k)$ in the random forest that voted in favor of the computational element $c$ and the possible variations in these votes that can be obtained by manipulating each metric $m$ (i.e. $\iota^+m$ and $\iota^-m$ from Equations 4 and 5). The produced reports are sorted by means of the maximum between $\iota^+m$ and $\iota^-m$, giving the possibility to the user to better understand how to modify the code to accelerate it. As examples, a pairs of metric-suggestions/modifications among the metrics of Table II are:

- **Operands** - Using different variables over different code statements reduces operation scheduling constraints improving parallelism opportunities. Additionally, some loop optimizations can implicitly increase the number of variables in use (e.g. unrolling).

- **FloatPre** - Despite of floating point operations are faster on FPGA since the PPC does not have a FP-unit, large floating point operators impact negatively on the FPGA operating frequency. Using float type rather than double type when possible mitigates this issue.

Here in the following we will show two case studies to better clarify this concept.

- **Susan edge case study.** The Susan edge application is an image processing algorithm for edge detection taken from the MiBench benchmark suite [24]. This application was excluded from the training kernel set. We focus on porting on the FPGA a kernel named susan_principle (Listing 6). The kernel takes three arrays that are: $a$ the input image in gray scale (in), $b$ the gray scale output image where to store edge shapes ($r$ and $c$) an array of 516 gray scale levels to use ($bp$). The expert system classifies the kernel as a non-accelerated kernels meaning that the preferred computational element to use is the PPC. However, by querying the expert system about more details on the decision, it comes out that the first concern is related to memory usage, suggesting to reduce the memory access stride and the use of heap memory (the arrays passed as parameters). In the inner loop of Listing 6, a pixel is compared to its eight surrounding pixels using 9 memory reads (the array $in$ is read through the pointer $p$). By caching the data used by the inner loop into small local arrays (Listing 7) implemented in HW as registers, we reduce the access to the shared memory and improve data locality. Summarizing, when mapped on the FPGA, while the original version of the kernel has a speedup of 0.38 (i.e. a slow-down) w.r.t. the PPC, the second version has a speedup of 3.96. The proposed optimization is clearly technology dependent since the impact of the same kernel modification on the PPC brings to a performance slow-down.

- **Bitreversal case study.** As second case study we synthesize a simple kernel that does not access memory arrays. We write a simple kernel (Listing 8) that takes a 32 bits unsigned integer and reverse its bits. Also in this case the expert system classifies the kernel as a non-accelerated kernel. Querying on the expert system, it mainly highlighted two metrics: the first one was related to the number of floating point operations (FloatExc), while the second one is related to the total number of Operands, that if (in both cases) are increased could bring to a better implementation. We modified the code to improve the second metric since the improvement of the first metric was impossible being the kernel without any floating point operation. Summarizing, the original HW and the optimized HW versions of the kernel present w.r.t. the original PPC implementation a speed-up of 0.7 and 4.4 respectively.

V. CONCLUSION

In this work we propose the DRuID framework for automatically identifying the most suitable computing element for
mapping the application functionalities in a context of heterogeneous platforms. The framework learns to discriminate between the heterogeneous elements on the base of a kernel characterization based on both static code analysis and dynamic profiling that can be performed before porting the kernel onto the different computing elements. The learning phase is carried out using data mining techniques such as random forests and genetic algorithms. The proposed methodology has been validated on a Xilinx-ml510 board that includes an FPGA and a PPC as computing elements.

ACKNOWLEDGMENT

This work was supported in part by the EC under the grant HARPA FP7-612069, and by the Swiss National Science Foundation under the grant PBF1P2-142333.

REFERENCES


