Near-Optimal Microprocessor and Accelerators Codesign with Latency and Throughput Constraints

ANGELIKI KRITIKAKOU, University of Patras

FRANCKY CATTHOOR, Katholieke Universiteit Leuven

GEORGE S. ATHANASIOU, VASILIOS KELEFOURAS, and COSTAS GOUTIS,
University of Patras

A systematic methodology for near-optimal software/hardware codesign mapping onto an FPGA platform with microprocessor and HW accelerators is proposed. The mapping steps deal with the inter-organization, the foreground memory management, and the datapath mapping. A step is described by parameters and equations combined in a scalable template. Mapping decisions are propagated as design constraints to prune suboptimal options in next steps. Several performance-area Pareto points are produced by instantiating the parameters. To evaluate our methodology we map a real-time bio-imaging application and loop-dominated benchmarks.

Categories and Subject Descriptors: C.1.3 [Processor Architectures]: Other Architecture Styles—Heterogeneous (hybrid) Systems; C.3 [Special-Purpose and Application-Based Systems]: Real-time and embedded systems

General Terms: Design

Additional Key Words and Phrases: FPGA, near optimal, area reduction, real-time behavior

ACM Reference Format:

Kritikakou, A., Catthoor, F., Athanasiou, G. S., Kelefouras, V., and Goutis, C. 2013. Near-Optimal microprocessor and accelerators codesign with latency and throughput constraints. *ACM Trans. Archit. Code Optim.* 10, 2, Article 6 (May 2013), 25 pages.

DOI: <http://dx.doi.org/10.1145/2459316.2459317>

1. INTRODUCTION

Embedded systems usually have hard real-time constraints which require custom HW designs. Although they improve the performance, they have a high design cost and very limited flexibility, even when they are made partly configurable. The SW designs provide the required flexibility for a wide range of applications at the cost of reduced performance. Hence, a hybrid SW/HW approach is a promising solution, as it balances the SW flexibility with the HW performance [Kornaros 2010]. Existing design tools offer a partially automatic customization of soft microprocessors. These tools

This article is an extension of a conference paper Kritikakou et al. [2012].

The results were cofinanced by Public Welfare Foundation “Propondis” research funds, Hellenic and European Regional Development Fund (ERDF) under ESPA 2007-2013 (MICRO2-SE-G) and European Social Fund (ESF) and Greek national funds (Heracleitus II-NSRF).

The machine vision algorithm and SW model are patented by Micro2gen [Demiris and Blionas 2011].

Authors’ addresses: A. Kritikakou, Department of Electrical and Computer Engineering, University of Patras, Greece; email: akritikakou@ece.upatras.gr; F. Catthoor, IMEC and Department of Electrical Engineering, Katholieke University of Leuven, Belgium; G. S. Athanasiou, V. Kelefouras, and C. Goutis, Department of Electrical and Computer Engineering, University of Patras, Greece.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.

© 2013 ACM 1544-3566/2013/05-ART6 \$15.00

DOI: <http://dx.doi.org/10.1145/2459316.2459317>

usually require a high exploration time as they may explore a wide range of design instances, which are, however, restricted and focused on a relatively limited area of the Design Space (DS) (Section 2). When the application characteristics match less with the explored area, this approach leads to suboptimal designs. A very broad Design Space Exploration (DSE) corresponds to a very difficult and time-consuming task due to the high number of SW and HW design parameters. Hence, designers usually create SW/HW designs by following ad hoc or trial-and-error ways based on their expertise [Milder et al. 2012], which usually require costly design iterations. Hence, a systematic methodology is desired to support a scalable DSE for near-optimal SW/HW codesign [Sheldon and Vahid 2009].

The main contribution of this article is a scalable DSE methodology to near-optimally map an application domain to a SW/HW FPGA design with a microprocessor core and HW accelerators. The proposed methodology splits the codesign process into sequential mapping steps connected with unidirectional propagation of design constraints, avoiding nonscalable steps and needless design iterations. The steps are the inter-organization between the microprocessor and the HW accelerators, the Fore-Ground (FG) memory management (i.e., the memory attached to the datapath), and the DataPath (DP) mapping. They are described by parametric templates, that is, a scalable structure with the relevant parameters, equations, and functions connected with propagation of constraints, which support efficient and scalable exploration. By giving valid values to the parameters and applying the functions, a Pareto curve with performance-area trade-off points is produced per step. The decisions in a mapping step are propagated as design constraints to the next mapping steps to prune incompatible options. An option is incompatible when, in order to be feasible, it requires to overlook the design constraints propagated from the previous steps. For instance, scheduling decisions for the critical part are propagated to decide the scheduling of the not-critical part. The options which require to change the scheduling of the critical part are incompatible and thus pruned from the DS. In this way, only the valid and promising DS part is explored based on an efficient pruning through a what-if analysis of the parameters and a scalable DSE is achieved from the early mapping steps of the codesign processing. Scalability is maintained as illustrated in Section 5 where the application size increase still allows linear exploration time. The design objectives are to reduce the area and, thus, the required gates for the design, and indirectly the energy consumption, meeting the real-time constraints. Our second main contribution is to demonstrate how the proposed methodology is effectively applied in a set of benchmarks. The most detailed results are provided for a real-time bio-imaging application of a microfluidic-based FPGA. The design gain is 47.11% on performance and 72.29% on area compared with pure SW and HW implementations, respectively. Further experimental results for 10 PolyBench benchmarks [Pouchet et al. 2012] are presented to substantiate the effectiveness and sufficiently broad applicability of our approach.

The article organization is as follows. Section 2 presents existing codesign approaches, Section 3 describes the proposed methodology, Section 4 demonstrates the approach by a real-time bio-imaging application, Section 5 shows experimental results for Section 4 and for the PolyBench benchmarks. Section 6 concludes this study.

2. RELATED WORK

Both academic and commercial customizable microprocessors, for example, Microblaze or NiosII, allow the designer to insert HW components for SW/HW implementations. Several approaches exist in the literature that modify the HW platforms to provide more efficient ways to insert HW accelerators. Some approaches provide a more generic HW accelerator that is capable of executing several application domains but less efficient for a specific application domain, as it includes redundant resources. For

instance, a generic custom unit extension is presented in Vassiliadis et al. [2009]. Ref [Melpignano et al. 2012] presents a generic accelerator that includes several processor clusters and shared memory. In Neumann et al. [2008] a parameterizable embedded FPGA architecture is used as a HW accelerator and the flow to create the layout, vhd code, and the configuration is described.

However, the tools and development methodologies are less supportive for a scalable exploration of the efficient mapping options of the applications to (re-)configurable embedded systems [Jozwiak et al. 2006]. Several design tools exist to partially customize soft microprocessors. Synopsys Symphony C compiler [Synopsys 2012] creates accelerators from sequential code. CriticalBlue cascade [Criticalblue 2012] is an automated coprocessors synthesis solution. Cosmos, Handel-C, and ImpulseC (a survey is available in Compton and Hauck [2002]), provide RTL extensions to C for FPGA design, but providing less efficient results than custom HW designs. The tools identify automated design flows and implement the custom instructions, but they also require specification of new HW resources and rewriting part of the application [Kornaros 2010]. In Dimond et al. [2005], the compiler generates custom instructions by finding the instruction datapaths that can be reused across similar pieces of code and adding them to the customizable processor. However, the exploration and verification time induces a significant overhead [Kornaros 2010], while the design may still remain quite sub-optimal [Dimond et al. 2005]. ROCCC [Guo et al. 2008] generates VHDL code for the datapath and the control flow of operations for the HW execution in the FPGA. These approaches usually require a high exploration time to create the accelerators, focusing on a relatively limited part of the design space and applying costly design iterations. They typically avoid exploration of the options in the organization of the cores and the FG memory, narrowing their search space, and thus promising solutions in the unexplored design space cannot be identified. When the application characteristics match less with the search space, the tools can produce suboptimal solutions. Designers' effort is required to improve the codesign. The alternative broad DSE is difficult and time consuming due to the high number of SW and HW parameters [Palermo et al. 2005].

The designers in industry typically propose designs for the specific applications by following ad hoc or trial-and-error ways, increasing the costly design iterations. They propose several designs using FPGA manufacture tools, such as Xilinx EDK and Altera SOPC Builder. For instance, a design of an object tracking coprocessor was proposed in Shahzad and Zahid [2009] and a design for an object detection application in Flatt et al. [2010]. Since substantial time and effort are required to evaluate a design, they usually evaluate few final choices, usually the ones evaluated quickly based on previous experience [Gajski et al. 1998]. Hence, potentially promising options are easily overlooked. In Callahan et al. [2000] a reconfigurable HW is used as a general-purpose accelerator. Application blocks are mapped using a module library partially exploring the design options, as custom unit details are overlooked.

Several of the existing DSE methodologies mainly focus on a step of the overall mapping process without taking into account the design constraints of the other mapping steps. For instance, a methodology based on evolutionary Multi-Objective Optimization (MOO) creates the DP of a HW accelerator for a JPEG algorithm [Ferrandi et al. 2007]. The compiler in Liao et al. [2003] extracts loop parameters to decide memory code transformations, such as unrolling and SW pipelining. Most of the DSE methodologies are recursive approaches, which usually require too much exploration time and are less scalable. Stochastic approaches require too much exploration time to reach near-optimal solutions in a large exploration space. For instance, a Quantum-inspired Evolutionary Algorithm (QEA) is proposed for the multiprocessor mapping problem in Ahn et al. [2008]. The QEA is an improved heuristic wherein,

however, the optimality of the solution is highly based on the number applied generations, since the increase in the number of generations increases the chances to reach a near-optimal solution. A DSE methodology with stochastic algorithms is proposed in Palermo et al. [2005]. The independence of the parameters is used to prune the space, which usually is quite restricted, and to derive the Pareto curve in Platune [Palermo et al. 2005]. A simulated annealing approach for DSE of object detection accelerators is proposed in Huang and Vahid [2011]. Another iterative method starts from the designer's base configuration, changes the value of one parameter each time, and uses the results to predict the optimal design [Sheldon et al. 2006]. It may lead to less efficient designs when a high number of parameters and interdependencies exists. Ref Sheldon and Vahid [2009] sorts the parameters based on the impact determined by the maximum parameter value change. All combinations of the first two high-impact parameters are considered.

The proposed methodology describes a scalable DSE that finds the near-optimal codesigns of an application domain to an FPGA with one microprocessor and several HW accelerators. This highly supports the design process since the near-optimal designs can be identified even for large benchmarks. The final design can be selected based on the requirements from early stages of the design process. The proposed methodology is scalable as it consists of unidirectionally ordered mapping steps, which are required to be applied once and they propagate the design constraints to the next step to prune suboptimal options, as explained in the next section. This is substantiated by experimental results (Section 5.3) where we show a near-optimal result as opposed to what can be achieved with local iterative improvement techniques.

3. SYSTEMATIC TEMPLATE-BASED MAPPING METHODOLOGY

The proposed methodology is a scalable DSE to provide near-optimal designs including the design of the required HW accelerators that meet timing and resource constraints under performance and area objectives (Pareto curve). The input of the methodology is the application and the hardware platform, and their characteristics are incorporated as domain constraints that restrict the behavior and design of HW accelerators. For instance, the application deadline gives the maximum time that can be allowed to execute the application in the HW accelerators. Hence, any design that has a performance above the deadline constraint is incompatible and pruned. Although constraints exist, the platform provides a significant flexibility in the codesign. The methodology steps are described by a parametric template. A parametric template is created by finding the relevant SW and HW parameters, the functions, and how the parameters and the functions affect each other in order to define the direction of propagating the design constraints. The methodology explores per mapping step valid options inside the available flexibility by giving specific values to the not-constraint parameters of the step (template instantiation). The partial mappings and decisions are unidirectionally propagated as design constraints to the next step to systematically prune suboptimal and overconstraint design options from the large exploration space based on constraint analysis. The remaining potentially optimal options are mainly trade-offs that are explored based on a scalable what-if analysis of template parameters. In this way costly design iterations are avoided. At the final step, near-optimal designs are depicted in a Pareto curve, where the Pareto points are placed quite close to each other, and from where the designer selects a high-quality SH/HW codesign based on the specifications each time.

The main design objective is area reduction. It reduces the number of gates and indirectly the leakage energy consumption. The dynamic energy consumption is not proportional to the active area. Hence, when constraints are met, the methodology tries to further improve the activity in order to reduce dynamic energy consumption

Fig. 1. The flow of the proposed methodology.

as a second objective. The real-time constraints imposed from the application context should always remain guaranteed, though.

The methodology flow chart is depicted in Figure 1 and the mapping steps are explained in the remaining section following the design constraints propagation order. The application and platform domain analysis step identifies the SW and HW parameters, for examples, the real-time constraints, the critical kernels, and their characteristics, which are propagated as constraints to the inter-organization step to decide the microprocessor and the HW accelerators connection. The result is propagated to the FG memory management step and then to the DP mapping, where the final SH/HW design is composed. In the scope of this article, the communication of the BackGround (BG) memory is organized by the SW executed on the microprocessor, using the HW of the memory and the cache controllers of the target platform. Hence, the array accesses in SW are compiled into load/store operations and the cache controller handles the data [Hennessy and Patterson 2006]. Application-platform-independent transformations have been upfront applied.

The application domain under study is described by embedded systems applications with one thread frame. The thread frame has deterministic behavior, that is, consists of several condition statements and nested loops, but without including any

ALGORITHM 1: Application & Platform Domain Analysis

```

\*Step 1.1: Platform Analysis*\
Determine HW parameters();
\*Step 1.2: Application Analysis*\
Identify basic kernels;
for (i=0; i<NumOfKernels; i++) do
    Profile  $t_i$ ;
    Identify SW parameters();
    Kernels=Sort based on  $t_i$ ;
    Identify control flow;
\*Step 1.3: Decide SW & HW execution*\
\*Step 1.3.1: Application Constraints*\
 $f_{PC} = \max(Av_{f_{PC}})$ ;
 $t_{SW} = \text{Assign}(\text{Application}, PC)$ ;
 $t_{tot} = t_{SW}$ ;
if ( $t_{tot} < D$ ) then
    Exit;
High level estimate( $t_{TR,Opt,CDFG}$ );
if ( $t_{TR,Opt,CDFG} > t_{HW}$ ) then
    Exit(Change HW parameters);
High level estimate( $CP_{HW,Opt,CDFG}$ );
if ( $CP_{HW,Opt,CDFG} > t_{HW}$ ) then
    Exit(Change HW parameters);

High level estimate( $t_{TR,Opt,CDFG}$ );
if ( $t_{TR,Opt,CDFG} > t_{HW}$ ) then
    Exit(Change HW parameters);
\*Step 1.3.2: Kernel Constraints*\
while ( $t_{tot} > D$ ) do
    k=select i from Kernels;
     $t_{SW} = t_{tot} - t_i$ ;
     $t_{HW} = (1 - \text{slack}) * (D - t_{SW})$ ;
    Assign(k, HW);
    Compute  $BandW$ ,  $ABandW_{BG,Opt}$ ;
    if ( $BandW > ABandW_{BG}$ ) then
        Repeat=1; break;
    Compute  $ABandW_{PC-HW,Opt}$ ;
    if ( $BandW > ABandW_{PC-HW,Opt}$ ) then
        Repeat=1; break;
    High level estimate( $CP_{HW,Opt}$ );
    if ( $CP_{HW,Opt} > t_{HW}$ ) then
        Repeat=1; break;
    if (Repeat==1 && SWsolution==1) then
         $t_{tot} = t_{SW} + t_{HW}$ ;
    else
        Exit(Change HW parameters);

```

event-triggered task generation or nondeterministic elements. The application is highly data dominated with increased computation requirements. The application real-time constraints are expressed as latency constraints, that is, the allowed time between two parts of the application, and as throughput constraints, that is, the number of application executions that should be completed in a given time interval, or as their combination. The throughput is translated into a latency constraint and the kernel is unrolled based on the iteration interval [Lam 1988]. The derived latency constraint and the transformed kernel are inputs to the methodology. When a combination of latency and throughput constraints exists, the throughput constraint is translated to a latency constraint and the most restricting one is selected. For instance, wireless applications usually have a throughput constraint per incoming sample and a latency constraint over the payload processing of the wireless baseband. Multiple real-time constraints for different parts of the application are handled by distributing their effect over the different parts/kernels of the application code. When they focus on the same part, their requirements are combined into a common delay constraint set. The platform domain is a heterogeneous FPGA with a microprocessor core and parallel HW accelerators. When the platform or application domain is partly modified with similar characteristics, the main principles will remain valid and they can be reprojected to produce the mapping methodology. For larger modifications to domains with clearly different characteristics, a more extensive exploration of the new principles and projections has to be initiated.

3.1. Step 1: Application and Platform Domain Analysis

The pseudocode of the application and platform domain analysis step is depicted in Algorithm 1 and explained in the next paragraphs.

Table I. Main Application and Platform Domain Parameters Used by the Proposed Method

SW Parameters		HW Parameters	
Parameter	Description	Parameter	Description
D	Execution Deadline	f_{HW}	Hardware Frequency
TP	Throughput	$Av_{f_{HW}}$	Available HW Frequencies
t_{Tot}	Total Execution Time	f_{PC}	MicroPC Frequency
t_{SW}	Exec. Time in MicroPC	$Av_{f_{PC}}$	Av. MicroPC Frequencies
t_{HW}	Exec. Time in HW acc.	f_{Mem}	Memory Frequency
For each kernel i		f_{MemBus}	Memory Bus Frequency
Regular	Data-Dependent execution	W_{MemBus}	Memory Bus Width
Data	Number of transfered Data	$f_{BusPC-HW}$	MicroPC & HW acc. Bus Freq.
DType	Type of Data	$W_{BusPC-HW}$	MicroPC & HW acc. bus Width
RF	Iterations between Data reuse	$BusPC-HW$	MicroPC & HW acc. buses
RData	Number of reused Data	$Av_{BusPC-HW}$	Av. MicroPC & HW acc. buses
RType	Type of RData	$Size_{FG}$	Size of FG memory
Res	Number of results	$Av_{Size_{FG}}$	Av. Sizes of FG memory
ResType	Type of results	W_{PLB}	PLB bus Width
Var_{Data}	Variables for Ops.	f_{PLB}	PLB bus Frequency
$VType_{Data}$	Type of variables for Ops.	MemBus	Memory Buses
Var_{Cntr}	Variables for control flow		
$VType_{Cntr}$	Type of control flow variables		
OPs_{Arithm}	Number of Arithmetic Ops.		
$OPsType_{Arithm}$	Type of Arithmetic Ops.		
$Cost_{OPsType_{Arithm}}$	Arithmetic Op Cost		
$Occ_{OPsType_{Arithm}}$	Arithmetic Op Occupation		
OPs_{Cntr}	Number of Control Flow Ops.		
$OPsType_{Cntr}$	Type of Control Flow Ops.		
$Cost_{OPsType_{Cntr}}$	Control Flow Cost		
t	execution time		
iter	Loop iterations		

3.1.1. Step 1.1: Platform Analysis Step. The platform is analyzed to determine the HW parameters, which describe the physical constraints of the main memory, memory buses, microprocessors, the buses between the cores, the local memory, and the HW accelerator. The main HW parameters are depicted in Table I.

3.1.2. Step 1.2: Application Analysis Step. This steps determines the SW parameters by analyzing the application. The main SW parameters are described in the left part of Table I. The application real-time constraints determine the throughput TP and the deadline D , which defines the maximum bound on the application execution time t_{Tot} , $t_{Tot} \leq D$. For instance, in video applications the TP derives from the Frame Rate (FR), that is, frames per second, and $D = 1/TP$.

Based on the application structure, the loops, basic kernels, and kernel execution time t_i are profiled. The kernel type is defined through the parameter *Regular*, that is, when *Regular*=1 it is executed in every loop iteration, otherwise the execution depends on the specific values of the data in the control flow operations. The kernel arithmetic operations are identified and characterized by the cost (in terms of number of gates) and by the occupation factor. The number and type of data required from the memory, results, variables of the arithmetic operations, their dependencies, etc., are identified. The control flow operations (e.g., parameters $OPs_{Cntr}(p)$) and the corresponding variables (e.g., $Var_{Cntr}(k)$) are identified (Table I). The main SW parameters are depicted in left part of Table I. The kernels are sorted by t_i to identify the most critical ones.

3.1.3. Step 1.3: Decide SW and HW Execution. This step decides the kernels to be mapped to SW and to HW. The microprocessor frequency f_{PC} is set to the maximum available. The time required to execute the application in SW, that is, in the microprocessor, is computed. If the t_{tot} is smaller than the available time D , the SW design meets the timing constraints. A further exploration to decrease f_{PC} while timing constraints are met can be applied. If t_{tot} is larger than D , SW/HW designs are required. Then, a lower-frequency for the microprocessor is less efficient, because in order to meet D , parallelization of HW accelerators would be required, increasing the FPGA active area and energy consumption. This option will be efficient if the TP and the D of the application requirements are quite low. Then, the microprocessor probably also meets the real-time constraints. To motivate a microprocessor lower frequency, the platform should provide a fine-grained voltage selection or have a frequency reduction by at least a factor of 2 for significant gain.

- **Step 1.3.1.: Application Constraints.** In this step it is verified if the overall design is possible based on constraint reasoning. In case operator strength reduction [Cooper et al. 2001] has not been applied during platform-independent transformations, we replace the constant operations with simpler operations, for example, a constant multiplication is replaced by a sequence of Shift-Add (SA) operations [Novo et al. 2010]. The bandwidth required to transfer the data from the memory is given by Eq. (2), where the number and the type of data per kernel are accumulated ($Data_{CDFG}$ given by Eq. (1)) and divided by the available time D . The available bandwidth is given by Eq. (3). If D is less than the time required to transfer the data, the problem is overconstraint. Hence, changes in the platform characteristics, that is, parameters changes, are required. Otherwise, SW/HW designs are explored.

$$Data_{CDFG} = \sum_{i=0}^{kernels} \sum_{j=0}^{DTypes} Data(j) * DType(j) * Iter(i) \quad (1)$$

$$BandW = \frac{Data_{CDFG}}{D} \quad (2)$$

$$ABandW_{BG,Opt} = MemBus * W_{MemBus} * \min(f_{Mem}, f_{MemBus}) \quad (3)$$

The critical path of the application $CP_{CDFG,Opt}$ is estimated in a high-level way considering an optimal mapping to the HW accelerator and taking into account potential HW area constraints, for example, based on the method of Diguët et al. [2000]. If the estimated critical path is higher than the available time D , the problem is overconstraint and HW parameter changes should be applied.

- **Step 1.3.2.: Kernel Constraints.** The SW/HW DSE is mainly based on the basic kernels, which take nearly all of the required workload on the platform resources. The remaining noncritical part of the code, that is usually dedicated to initializations, is assumed to be absorbed in the slack that is introduced during the mapping steps. For instance, the preamble/postamble code, which may introduce nonregularity, is mapped to the microprocessor when the speedup factor required to meet real-time constraints is quite low. When the speedup factor is significantly high or the communication overhead is highly increasing [Kim et al. 2012], it is mapped to the HW accelerators. In this way we can restrict the design time spent on the entire application and the effort of industrial designers. The available time is then fully focused on obtaining a near-optimal result for the kernels. That part of the design effort should remain scalable because real-life applications will still involve a substantial amount of code (several kernels) to be dealt with. The proposed methodology ensures these characteristics.

The control flow operations can be executed either in the Microblaze, that is parameter $OPs_{Cntr} \in SW$ or implemented in the HW accelerator, $OPs_{Cntr} \in HW$. In the first case, the complexity is moved to the microprocessor, whereas in the second case, dedicated control and diverse FUs are inserted to the HW accelerator. By propagating the design objectives to these two options, the control statements are selected to be executed in the microprocessor. In this way, the HW accelerator design complexity and the synchronization between microprocessor and HW accelerators is reduced, which allows a very efficient and high-performance HW accelerator CP design dedicated only to the execution of the arithmetic operations.

The design objectives, that is, area reduction, are propagated to select the kernels for mapping in the HW accelerators. Hence, the smallest number of kernels should be mapped in the smallest number of HW accelerators. The kernel with the highest t_i is initially selected. When two kernels have similar t_i , the kernel with *Regular* = 1 is favored, due to the high HW accelerator use, the execution regularity, and the simpler synchronization scheme between microprocessor and accelerators. The irregular part of the application is mapped to the microprocessor.

When kernel i is selected, the HW part has in the worst case $t_{HW} = (1 - slack) * (D - t_{SW})$ using a *slack* percentage for the noncritical code. The kernel constraint verification is performed based on the HW parameters and the optimal design. Hence, pipelining between the data fetching from the memory, the data transferring to the HW accelerator, and the computation of the HW accelerators is assumed. The number of data required from BG memory are given by Eq. (4), where $Data(j)$ is the number of data with data type j for one iteration of the kernel i . The total number of data is given by Eq. (5), where $Iter(i)$ are the loop iterations of kernel i in the available time t_{HW} . The required (available) transfer bandwidth is given by Eq. (6) (Eq. (7)). Similar equations exist for the the microprocessor to the HW accelerator transfer assuming optimal design, that is, the bus with the maximum bandwidth (Eq. (8)). If the required bandwidth from the memory (to the HW accelerator) is larger than the available one, the problem is overconstraint. In this case two potential options exist: (1) HW solution: change the HW parameters, or (2) SW solution: increase the available time t_{HW} . The latter is expressed by mapping in HW the best candidate to reduce t_{SW} , and thus increase t_{HW} and reduce the required bandwidth.

$$Data = \sum_{j=0}^{DTypes} Data(j) * DType(j) \quad (4)$$

$$Data_{Tot} = Data * Iter(i) \quad (5)$$

$$BandW = \frac{Data_{Tot}}{t_{HW}} \quad (6)$$

$$ABandW_{BG,Opt} = MemBus * W_{MemBus} * \min(f_{Mem}, f_{MemBus}) \quad (7)$$

$$ABandW_{PC-HW,Opt} = \max(Av_{Bus,PC-HW,max}) * W_{Bus,PC-HW,max} * f_{Bus,PC-HW,max} \quad (8)$$

The next step is to estimate in a high-level way the critical path $CP_{HW,Opt}$ of the HW accelerator DP. This is achieved considering an optimal HW accelerator design that the platform constraints allow based on the minimum possible path in terms of delay for the given target technology [Diguët et al. 2000]. In this way, we can verify that the optimal HW accelerator can support the execution of the kernel. If the optimal estimated critical path is larger than the available time, either the HW parameters have to be modified or the t_{HW} has to be increased.

ALGORITHM 2: Microprocessor & HW Accelerators Inter-Organization

```

Dependent=1;CoProcessor=1;Control=0;
Compute  $ABandW_{PC-HW}$ ;
if ( $BandW > ABandW_{PC-HW}$ ) then
  Dependent=0;NPI=0;
  Compute  $ABandW_{PC-HW_{PLB}}$ ;
  if ( $t_{TR,PC-HW_{PLB}} > t_{HW}$ ) then
    NPI=1;
    Explore(SCIP-NPI);
  High level estimate of  $t_{TR,PC-HW}$ ;
if ( $t_{TR,PC-HW} << CP_{HW,OPT}$ ) then
   $t_{HW} = t_{HW} - t_{TR,PC-HW}$ 
Compute  $Bus_{PC-HW}$ ;

```

Fig. 2. Inter-Organization of the microprocessor and the HW accelerator parametric template.

Table II. Truth Table of Microprocessor and HW Accelerators Inter-Organization

Parameters				HW Accelerator Connection
Dependent	Coprocessor	Control	NPI	
0	Not-care	Not-care	1	Standalone Custom IP-NPI
0	Not-care	Not-care	0	Standalone Custom IP-PLB
1	0	Not-care	Not-care	Internal FU
1	1	0	Not-care	Base Coprocessor
1	1	1	Not-care	Control Coprocessor

3.2. Step 2: Microprocessor and HW Accelerators Inter-Organization

This step decides the organization of the microprocessor and the accelerators and the transferring of the data between the cores. The pseudocode is depicted in Algorithm 2.

3.2.1. Step 2.1: Microprocessor and HW Acceleration Connection. This step decides the organization of the microprocessor and accelerators. A HW accelerator can be integrated into the platform in several ways as depicted in the parametric template of Figure 2. Table II shows the corresponding design options.

A HW accelerator can be fully independent or partially dependent on the microprocessor (parameter *Dependent*). When the parameter *Dependent* = 1, the HW accelerator partially depends on the microprocessor. It reuses the microprocessor resources, for example, memory interface, which reduces the custom design complexity, the area,

and the energy consumption. When $Dependent = 1$, the HW accelerator can be implemented as an extension of the internal FUs of the microprocessor or as external coprocessor (parameter $CoProcessor$). When $CoProcessor = 0$, the HW accelerator is implemented as an internal FU. This implementation affects the critical path of the microprocessor, potentially reducing the $f_{HW}(f_{PC})$, which is not acceptable since real-time constraints may be not met. The implementation of the HW accelerator as a coprocessor ($CoProcessor = 1$) removes these limitations. The microprocessor and the coprocessor can execute different parts of the application at the same time. Hence, the microprocessor can be used for memory address generation and the fetching of data from memory, while the coprocessor executes the kernel operations. The interconnection of the microprocessor and the coprocessor is quite fast, a small latency (one or two cycles) is required to write (read) the data to (from) the coprocessor, which makes this option Pareto (near-)optimal for our domain, when constraints allow it.

When parameter $Dependent = 0$, the HW accelerator is connected independently from other HW resources to the communication channel (Stand-alone Custom IP - SCIP). In this design, apart from the FG memory and the DP, the HW accelerator has a BG memory interface, similar to “fire and forget” model. The BG interface can be implemented by: (1) a custom design, for example, Native Port Interface (NPI) [Xilinx 2011] (expressed through parameter $NPI=1$) designed to control the memory in the most efficient way, or (2) using a common bus protocol, for example, Processor Local Bus (PLB) ($NPI=0$). The first case is optimal, but with higher area and design effort, which is acceptable when other approaches are overconstrained (see what follows). The second case is less efficient due to the bus protocol bottleneck. In the case when the bus option cannot provide the required bandwidth, a NoC-based communication topology can be explored as valid option to replace the bus option.

In the coprocessor implementation two further options exist for the control, that is, loop organization, control operations, and structure. It can be common or different between microprocessor and coprocessor. When parameter $Control = 0$, the microprocessor takes care of the control of the kernel operations that are executed on the DP of the coprocessor. The responsibility for the synchronization of the data between the two cores resides on the microprocessor. This allows a more efficient, smaller area and lower-energy consumption design of the coprocessor (base coprocessor). This option fully meets the design objectives, so it is Pareto (near-)optimal for simple synchronization schemes. The microprocessor is responsible for invoking the coprocessor, which executes the operations and writes back the result. If $Control = 1$, different control structures can exist in the processor and in the coprocessor. The coprocessor requires a HW control unit, for example, Finite State Machine (FSM), to support the correct functionality of the DP and the communication from/to the microprocessor. By propagating the design objectives, this option is potentially suboptimal due the increased coprocessor area. However, when the design synchronization is more complex, this option should be valid.

3.2.2. Step 2.2: Microprocessor and HW Acceleration Communication. This step is dedicated to organizing the transferring of data between cores. The available bandwidth to transfer the data between the cores for the less costly design is verified if it is sufficient. The available bandwidth of the base coprocessor design is given by Eq. (9) considering optimal data transfer, that is, the maximum number of buses between the cores. When the bandwidth is insufficient, alternative designs are explored. The next less costly option is the Stand-alone Custom IP with PLB and last the high design effort option of the Stand-alone Custom IP with NPI.

$$ABandW_{TR,PC-HW} = \max(Av_{Bus,PC-HW}) * W_{Bus,PC-HW} * f_{PC} \quad (9)$$

If the bandwidth of the optimal parallel transfer is acceptable, the time required to transfer the data $t_{TR,PC-HW}$ is computed. The available time t_{HW} is updated based on the time required to transfer the data from the background memory and it is verified if enough time is available to transfer the data. The number of minimum required parallel transfers, that is, the $Bus_{PC-HW,min}$, derives from Eq. (10). If $t_{TR,PC-HW}$ is negligible compared to the estimated critical path of the DP, sequential data transfer can be selected to reduce the design effort. In this case, the lifetime of the variables in the FG memory is increased, which restricts the next mapping step. By giving different values to the number of transfers (which still satisfy the minimum requirements) different Pareto points are produced. The available time for computation is updated, $t_{HW} = t_{HW} - t_{TR,PC-HW}$.

$$Bus_{PC-HW} = (int) \frac{BandW}{ABandW_{TR,PC-HW}} \quad (10)$$

The application is transformed accordingly with the corresponding operations to write and read the data to the bus connecting the microprocessor and the HW accelerator. The scheduling of the transfers and their address generation (e.g., DMA or load/store instructions) is taken care of by the microprocessor compiler.

3.3. Step 3: Foreground Memory Management

The dimensioning and management of the FG memory is decided in this step. The corresponding pseudocode is depicted in Algorithm 3. The size of the FG memory of the microprocessor is given by the platform parameters, whereas the FG memory of the HW accelerator is determined by the required data for the kernel execution. The FG dimensioning and the operations executed in the HW accelerator depend on the constraints propagated from the previous steps: When *Dependent* = 1 and *Coprocessor* = 0, the FG of the HW accelerator is the FG of the microprocessor. When the *Coprocessor* = 1, the scalars to be stored in the FG memory are based on the *Control* parameter. If *Control* = 0 only the scalars for arithmetic operations are stored in the FG, that is, the data from BG memory, the arithmetic variables, the intermediate results, and the final results. If *Control* = 1, the scalars for the control flow are also stored in the FG, since the control flow operations are executed in the HW accelerator. The FG should be sufficiently large to support all the control and flow operations, increasing energy consumption and reducing the opportunities for DP parallelism. When the parameter *Dependent* = 0, the FG memory of the HW accelerator should store the scalars required for control and arithmetic operations.

The bandwidth and the energy requirements for BG memory are reduced by replacing several BG accesses with FG ones. In the Pareto-optimal case, only the new *Data* are transferred from the BG memory and the intermediate results are kept in the FG memory avoiding register spilling. Otherwise the most used scalars are maintained in the FG memory and the remaining ones are stored back to the BG memory, increasing the latency and the number of future accesses to FG memory. The spilling option is acceptable if HW parameters restrict the optimal option.

Initially, a cost operations analysis takes place, as it affects the scalars required to be stored in the FG memory. Each operation type of the kernel is evaluated based on the number of gates and the occupation factor. If an operation type is characterized as costly (during application analysis) and has a low occupation factor, exploration is applied to replace it by smaller and simpler operations, which require less gates to be

ALGORITHM 3: Foreground Memory Management

```

 $OPs = \sum_{n=0}^{OPsType_{Arithm}} OPs_{Arithm}(n)$ 
if ( $Coprocessor==0$ ) then
     $Size_{FG} = Size_{RegFile}$ ; Exit;
if ( $Dependent==0$ ) || ( $Dependent==1$  &&
 $Coprocessor==0$  &&  $Control=1$ ) then
     $OPs = OPs + \sum_{n=0}^{OPsType_{Cntr}} OPs_{Cntr}(n)$ 
    Evaluate( $OPs$ );
    Select primitive operations;
     $UF_{FG}=1$ ;
    Evaluate Reg( $UF_{FG}$ );
    Compute  $U_{FG}$ ;
     $Num_{Reg}=FG$ ;
    Exploration( $UF_{FG}, \infty$ );
    if ( $Num_{Reg} < Num_{Reg,Max}$ ) then
         $Num_{Reg}=FG$ ;
        Exploration( $UF_{FG}, Num_{Reg,Max}$ );
     $Accesses_{FG} = Count(RD, WR, MLT)$ ;
    FG Exploration( $U_{FG}, Max$ ) {
        while ( $U_{FG} < U_{Thr}$ ) do
            Update  $UF_{FG}$ ; Evaluate Reg( $UF_{FG}, Max$ );
            Compute  $U_R$ ;

```

```

        while ( $CP_{FG} < t_{HW}$ ) do
            Reduce  $Num_{Reg}$ ;
            Evaluate Reg( $UF_{FG}, Num_{Reg}$ );
        }
        Evaluate( $OPsType$ ) {
            for ( $j=0; j < OPsType; j++$ ) do
                if ( $Cost_j > Threshold$ ) then
                    Explore for transformation of
                    operation( $j$ );
                    Update(SW Parameters);
                    Transform(Code);
                }
            Evaluate Reg( $UF, PF, Max$ ) {
                Compose HDFG( $UF, PF$ );
                Compute life time(HDFG);
                SLT=Sort(life time);
                MLT=Left edge algorithm(SLT,Max);
                 $Num_{Reg}=Count(MLT)$ ;
            }

```

executed. In this way the area is reduced and the use of simple resources is highly increased. The kernel loop is transformed accordingly.

The next step is to define the HW accelerator primitive operations, that is, operations used in the datapath. This step is similar to the instruction set selection process in microcoded processors [Koes and Goldstein 2008], but modified for the HW accelerator operations. The primitive operation selection can be achieved through exploration based on techniques that use cost functions related to the utilization of the operations, the area cost, the critical path, etc. After the primitive operation set selection, the application code is modified accordingly and a new estimation of the critical path is used to verify that the real-time constraints are met.

A Hierarchical Data Flow Graph (HDFG) with primitive operations is introduced, which includes the data dependencies (satisfying the control dependencies), the operations of both branches in a control flow operation, and the loop structure information. The HDFG provides a relevant ordering of the operations based on a realistic optimistic scheduling, that is, As Soon As Possible (ASAP) of the critical path and As Late As Possible (ALAP) of the remaining operations. This realistic optimistic scheduling leads to smaller scalar lifetimes. The next step computes the lifetime of the scalars of the HDFG, for example, using the technique in Poletto and Sarkar [1999]. The result is sorted per slot in increasing scalar lifetime to enable an efficient register allocation algorithm, for example, left edge [Sant'Anna et al. 2004]. It merges the scalars with nonoverlapping lifetime. It starts from the second slot and moves the scalars to the left to be merged with scalars whose lifetime has terminated. The result Num_{Reg} provides the minimum number of registers for realistic optimistic scheduling.

The HDFG of one execution of the kernel i is explored and the minimum number of registers is computed (Num_{Reg}) for this maximally parallel execution order. The utilization of the scalars U_{FG} is computed by Eq. (11). If the scalars are distributed in a less balanced way and several holes exists, unrolling by a factor of UF is explored

ALGORITHM 4: DataPath Mapping

```

\*Step 4.1: DP Mapping*\
Allocate(Primitive Ops,Single HW);
Schedule, Assign(UFFG,Single HW);
UFHW=1;
Compute UOp;
while (UOp < UThr) do
    Update UFHW;
    Schedule, Assign(UFHW,Single HW);
    Compute UR;
if (fHW > fPC) then
    Estimate PL stages;
    Pipeline(PL);
if (CPHW < tHW) then
    Parallelize(PF,Single HW);
if (CPHW < tHW) then
    Pipeline(PL);
\*Step 4.2: FG Memory Connection*\
Estimate ABandWFG;
NumFG,Ports =  $\frac{BandW}{ABandW_{FG}}$ ;
Allocate ports;
Connect(DP,FG);

```

to increase the utilization of FG resources. The UF_{FG} is estimated by Eq. (12) and the FG dimensioning process is repeated.

$$U_{FG} = \frac{\sum_{i=0}^{Num_{Reg}} Holes(i)}{Num_{Reg} * Slots} \quad (11)$$

$$UF_{FG} = \frac{1}{U_{FG}} \quad (12)$$

The FG critical path is computed based on the control and data dependencies and on an estimation of the f_{HW} , for example, the lower bound of f_{PC} . If the critical path is lower than t_{HW} , the time slack can be used to increase the schedule length and to reduce the number of registers. In case a HW parameter constrains the number of available registers ($Num_{Reg,max}$) and it is lower than the Num_{Reg} of the realistic optimistic scheduling with explored potential time slack, the left edge algorithm is repeated by taking the $Num_{Reg,max}$ constraint into account. The scalars with small lifetimes are allocated in the available registers, whereas the scalars with long lifetimes are the first candidates for spilling to BG memory. Although the register spilling increases the latency and the number of FG accesses, it also reduces the maximal size of the simultaneously alive scalars and, thus, also the FG memory size, as desired. Based on the requirements, different design points for the FG memory management are produced. The register width is $max(DType, VType, Rtype)$. The number of accesses, that is, reads and writes, to the FG memory are computed by Eq. (13). The available FG memory bandwidth is also required to determine the ports of FG memory $Num_{FG,Port}$. However, we can postpone the computation of the available bandwidth as it is not essential for DP mapping. The number of ports and the final FG memory scheduling are decided after the datapath mapping, where the required information is available.

$$Accesses_{FG} = \sum_{i=0}^{Num_i} \sum_{j=0}^{Scalars(i)} (WR_j + RD_j) \quad (13)$$

3.4. Step 4: Datapath Mapping and Final Design

The datapath design and the connection to FG memory are decided in this step. The pseudocode is depicted in Algorithm 4.

3.4.1. Step 4.1: DP Mapping. This step determines the datapath design based on propagated design constraints of previous steps. When the parameter *Dependent* = 0,

the DP of the HW accelerator executes both control and arithmetic operations. When *Dependent* = 1 and *Coprocessor* = 0, the DP executes only the arithmetic operations and the control is at the microprocessor. When *Coprocessor* = 1, the size of the DP and the executed operations are based on the *Control* parameter. If *Control* = 1, the control is executed on the coprocessor. If *Control* = 0, the HW accelerator control is executed on the processor DP and the arithmetic operations on the HW accelerator DP. The size of the DP of both the processor and the HW accelerator should support the maximum length of the *Data* and the *Variables*.

The FG memory management step propagates the kernel *i* potentially unrolled by the estimated UF_{FG} . The primitive operators are allocated, the kernel is scheduled and assigned on the single HW accelerator (a survey of techniques is available in Kritikakou et al. [2013]). The utilization of the primitive operators U_{HW} is given by Eq. (14) and used to decide on further unrolling to better utilize the HW operators. The scheduling and assignment step is reapplied with UF_{HW} .

$$U_{HW} = \frac{\sum_{i=0}^{Num_{HWOps}} Holes(i)}{Num_{HWOps} * Slots} \quad (14)$$

$$UF_{HW} = \frac{1}{U_{HW}} \quad (15)$$

The critical path of the single HW accelerator CP_{HW} is computed by accumulating the latency of the operators in the critical path of the design, and the f_{HW} is determined ($f_{HW} = \frac{1}{CP_{HW}}$). If $f_{HW} \geq f_{PC}$, the f_{HW} is set equal to f_{PC} in order not to unnecessarily increase the area. If a further increase is required, it will be verified in the next step. If $f_{HW} < f_{PC}$, pipelining is inserted to increase the f_{HW} up to f_{PC} . The number of pipeline stages PL (Eq. (16)) is determined by a balanced split of the critical path. An unbalanced split is a less efficient trade-off option since the unbalanced “fast” pipeline stages consume more energy than required. If the $CP_{HW} \leq t_{HW}$, where t_{HW} describes the available time left, the single HW accelerator design meets the real-time constraints. Otherwise parallelization across multiple HW accelerators (each with their own FG memory access ports) is considered to meet the real-time constraints. The Parallelization Factor PF is given by Eq. (17). The critical path is reduced (Eq. (18)), since the different iterations are executed in PF HW accelerators.

$$PL = \frac{f_{PC}}{f_{HW}} \quad (16)$$

$$PF = \frac{t_{HW}}{CP_{HW}} \quad (17)$$

$$CP_{HW} = \frac{CP_{HW}}{PF} \quad (18)$$

If HW constraints exist over the PF and we still cannot meet the application timing requirements, the last option is to use a different frequency in the HW accelerator and the microprocessor. This introduces additional overhead due to the required synchronization of the cores, which increases the area and the operations. However, it is a valid option when the other design options are overconstrained. A further exploration is to produce a design by selecting the next kernel in the sorted list of candidates to be assigned to a HW accelerator. The time required on the microprocessor (which is in the overall critical path) is reduced and hence it allows to increase the available time

for the HW accelerator execution. The objective function results of these two alternatives should then be compared to identify the most Pareto-optimal one. Hence, different Pareto points are finally developed and the overconstrained ones are removed (as illustrated in Section 5).

3.4.2. Step 4.2: FG Memory Connection. This step determined the FG memory connection. The FG memory bandwidth is determined and the available bandwidth per register with one port is computed by Eq. (20). The required bandwidth derives from Eq. (19) and thus the required number of ports is given by Eq. (21). The allocation of ports and the final scheduling of the FG memory is performed, for example, by using the technique in Capitanio et al. [1995]. The final connections between FG registers and the accelerator logic are inserted [Bjerregaard and Mahadevan 2006].

$$RBandW_{FG} = \frac{Accesses_{FG}}{t_{HW}} \quad (19)$$

$$ABandW_{FG} = f_{HW} * Num_{Reg} \quad (20)$$

$$Num_{FG,Port} = \frac{RBandW_{FG}}{ABandW_{FG}} \quad (21)$$

4. DEMONSTRATOR DESIGN: REAL-LIFE MICRO-FLUID APPLICATION

This section illustrates the proposed methodology by deriving a near-optimal SW/HW mapping of a bio-imaging application on an FPGA board. For different application characteristics, the proposed methodology develops different near-optimal designs which create the Pareto curve (Section 5).

4.1. Step 1: Application and Domain Analysis

4.1.1. Step 1.1: Platform Analysis. Our target HW platform is the Virtex-5 FPGA ML-507 evaluation platform with one Microblaze soft processor set. A SDRAM DDR2 main memory, a data, and an instruction cache of 16KB and a local memory of 32KB are used. The data and the instructions are fetched by the HW cache controller. The HW parameters are identified from the board characteristics, for example, $W_{BusPC-HW} = 32$ bits, $BusPC-HW = 16$, $Av_{BusPC-HW} = 1$ to 16, etc.

4.1.2. Step 1.2: Application Analysis. The demonstrator is a bio-image analysis used in a blood analysis application executed on a Lab-on-Chip (LoC) micro-fluid device. The image taken by the FPGA camera is depicted in Figure 3(b). During the setup, the frame (the continuous box of Figure 3(b)) and the coordinates of the micro-fluid pipes are detected. Based on the application specifications, the frame can be rotated only $\pm 3^\circ$. During LoC device normal function, an angle detection algorithm and a detection of the fluid's fronts coordinates algorithm is executed in each frame. The pseudocode of the application is depicted in Figure 3(a). The fluid velocity is computed based on the coordinates of the fronts determining the provision of required liquid quantity. The angle detection algorithm requires only the vertical line in the window where it is applied (dotted box of Figure 3(b)). It applies the Canny algorithm for finding the intensity gradient of the image using a horizontal contrast 3x3 Sobel (middle column multiplicands are 0) and a Hough transform version, since the vertical line computations are required with small angles, for example, $[-3^\circ, +3^\circ]$. If the Sobel kernel's result is an edge point, the Hough transform maps it to the Hough space and stores the results to the accumulator matrix. The final line is detected by suppressing the neighborhood lines. The fluid coordinates are derived by subtracting the micro-fluid pipes of two successive frames and by computing the centroid of the result. The SW parameters are defined, for example, for the intensity gradient kernel $Data = 6$, since

```

uint8 f[N][M];
int Coordinates, Mask[3][3], Gx[M]; Angle[M];
if (Set-up):
    Detect Frame position(f);
    Identify pipes coordinate(f);
/*AngleDetection*/
for (row=1; row<N-1; row++):
    Initialization();
    for (col=1; col<M-1; col++):
        for (rowOff=-1; rowOff≤1; rowOff++):
            for (colOff=-1; colOff≤1; colOff++):
                Gx[col] +=
                    f[row+rowOff][col+colOff]*Mask[rowOff+1][colOff+1];
Gx[col] = abs(Gx[col]);
if (Gx[col]! = 0):{Angle[col]=90;}
else:{Angle[col]=0;}
if (Angle[col-1]==90):
    if (Gx[col-2]<Gx[col] || Gx[col-1]<Gx[col-2] ):
        if (Gx[col-1]>Threshold): {HoughAccum.(row,col);}
LineSuppression;
FluidDetection(f);

```

(a)

(b)

Fig. 3. Real-time bio-imaging application:(a) Pseudocode and (b) image taken by the micro-fluid device camera: the box is the outline frame and the dotted box is the angle detection window.

6 pixels are required for the horizontal contrast 3x3 Sobel mask, $Res = 2$ for the angle and the gradient, $ResType = 32$ bits, etc.

We demonstrate the proposed methodology for a 200x16 window and $FR = 100$ frames/sec. The throughput is given by the video frame ratio, that is, $TP = 100$. The $D = 10$ msec to execute the angle detection algorithm and the algorithm for the detection of the fluid fronts, $t_{Tot} = t_{Angle} + t_{Fluid}$.

4.1.3. Step 1.3: Decide SW and HW Execution. The Microblaze frequency is set to the maximum allowed, that is, $f_{PC} = \max(Av_{f_{PC}}) = 125$ MHz, $t_{Fluid} = 4.87$ msec, and $t_{Angle} = 2.87$ msec. The SW design meets the deadline constraints and thus no HW accelerator is required. To demonstrate the next steps of the proposed methodology we use a frequency of $f_{PC} = 83,33$ MHz. In this case, the platform is compatible with the Avnet Spartan-6 LX150T Development Kit, where $\max(Av_{f_{PC}}) = 83,33$ MHz. The detection of the fluid fronts' coordinates depends on the number of fluids fronts and the frame resolution. For small frame resolution (640x480) and 2 fluids fronts, it executes in 1.23 msec in SW. For the minimum quality application specifications (small frame and 3 edges) the average execution time is estimated at 3.9 msec and for the maximum quality (1024x1024 and 7 edges) is 7.3 msec. The angle detection requires 4.31 msec (Section 4) when executed on the Microblaze soft microprocessor. The execution time of the application for a 200x16 window is $t_{Tot} = 4.31 + (3.9 \text{ to } 7.3) \text{ msec} = 8.21$ to 11.61 msec. Hence, real-time behavior is not always achieved ($t_{Tot,MinQ} < D < t_{Tot,MaxQ}$), and thus SW/HW designs are required.

—**Step 1.3.1: Application Constraints.** We apply strength reduction to remove the costly multiplications by constant, for example, the constant values of multiplications are analyzed, the multiplications with the value of 1 and 0 are removed, and the remaining ones are replaced by shift and add operations. Based on application profiling the most time-consuming regular task is the angle detection algorithm and thus its critical kernels should be explored for mapping on HW accelerators. Since the application time exceeds the D for a small value, it is possible by mapping part of it to the HW accelerator to meet the real-time constraints. The bandwidth required

to transfer the data for the kernels to the microprocessor $BandW_{CDFG} = 227,556$ bits/msec and the bandwidth provided by the BG memory of the platform in the optimistic case $ABandW_{BG} = 10,666,240$ bits/msec.

- *Step 1.3.2: Kernel Constraints.* To safely meet the real-time constraints, the available time for executing the angle detection is $t_{Angle} = D - t_{Fluid} = 10.0 - (3.9 \text{ to } 7.3) \text{ msec} = 6.1 \text{ to } 2.7 \text{ msec}$, and the worst case is considered, that is, $t_{Angle} = 2.7 \text{ msec}$. Based on the profiling of the application analysis, the main loop takes 68% to execute the kernel for the intensity gradient of the image and the kernel for creating the Hough accumulator array. The execution time of the two kernels is similar (31% and 33%, respectively) with the intensity gradient kernel regular. The application of the horizontal 3x3 Sobel mask is 90% of the intensity gradient kernel, so it is the first candidate for implementation in HW ($Regular_{Sobel} = 1$). When a slack of 10% is used for the noncritical part of the code, the available time is $t_{HW} = 0.9 * (0.9 * 0.31 * 2.7) = 0.677 \text{ msec}$. The data required to be transferred are $Data = 614,400$, when 32 bits are used to store the data of the image and the required bandwidth to transfer the data is $BandW = 906,235 \text{ bits/msec}$. The available bandwidth of the BG memory $ABandW_{BG}$ is 10,666,240 bits/sec when 128 bits are transferred and 2,666,560 bits/msec when 32 bits are transferred. The available bandwidth to transfer the data from the Microblaze to the HW accelerator is $ABandW_{PC-HW} = 42,664,960 \text{ bits/msec}$ when all FSLs are used in parallel. An estimation on the optimal critical path is $CP_{HW,Opt} = 0.028 \text{ msec}$ without area constraints using a $Latency_{ADD/SUB/COMP} = 2 \text{ msec}$ from the FPGA platform. Hence, mapping the most critical kernel with $Regular = 1$, that is, the 3x3 Sobel mask, on HW accelerator is enough to meet the real-time constraints.

4.2. Step 2: Microprocessor and HW Accelerators Organization

Since the design objectives are minimizing the area and thus reducing the energy consumption while the real-time constraints are met, the HW accelerator is efficiently connected through Fast Simplex Link (FSL) to the Microblaze soft processor as a coprocessor ($CoProcessor = 1$ and $Dependent = 1$). The Microblaze is responsible for the synchronization and the loop organization between the components ($Control = 0$). Hence, the application loops are modified accordingly to support the common control of both cores. The Microblaze provides the appropriate information for one execution of the coprocessor in the corresponding FSL, that is, the required pixels accessed from the memory. Then the coprocessor reads the data, executes the operations, and writes the results, that is, the gradient and the angle, to the FSL. The FSL width should be $W_{BusPC \rightarrow HW} = 32 \text{ bits}$ to support the maximum width of transferred data. The required bandwidth to transfer the data within t_{HW} is 906,235 bits/msec and the available bandwidth in the optimistic case, that is, 16 FSL are used, is 42,664,960 bits/msec, which is sufficient. The time required to transfer the data is $t_{TR} = 0.014 \text{ msec}$ when all FSL are available. The bandwidth of one FSL is 2,666,560 bits/msec. The minimum required parallel transfers is 1 and the transfer time is 0.23 msec.

4.3. Step 3: Foreground Memory Management

The cost operation analysis does not modify the kernel, since the operations after strength reduction are simple and highly used. In this case study, the inner kernel is mapped to a primitive operation. The HDFG is the degenerated case of 6 registers for the inputs, each one with 1 write and 1 read operation to the FG memory, and 2 registers for the results, each one with 1 write and 1 read. The registers width is 32 bits to support the maximum data type. The total $Accesses_{FG} = 56,000$ and one

Fig. 4. The HW platform architecture and the final design for the Microblaze and the HW accelerator of the demonstrator application.

register port provides 83,330 accesses/msec. The FG critical path is estimated by 11,200 accesses and the lower allowed frequency ($f_{PC} = 83,33$ MHz) at 0.134 msec.

4.4. Step 4: Datapath Mapping and Final Design

4.4.1. Step 4.1: Datapath Mapping. The control of the loops, the FG memory scheduling, and the initialization code are executed on the Microblaze DP. The primitive operator is described by the arithmetic operations executed on the coprocessor, that is, $OPs = Num_{SHIFT} + Num_{ADD/SUB/COMP} = 2 + 6$, which are implemented by 2 Shift FUs (S-FUs) and 6 Add/sub/comp FUs (A-FUs). To reduce the area, an S-FU can be combined with an A-FU in one Shift-Add (SA) FU and thus 4 SA-FUs and 8 A-FUs. The DP bus width should support the result and the operands of the primitive operation, that is, 32 bits. Further unrolling is not required since full utilization of the primitive operations is achieved. $CP_{HW} = 0.0177$ msec and $f_{HW} = 157,679$ MHz. The frequency is set to f_{PC} to avoid extra synchronization and meet the real-time constraints, $CP_{HW} = 0.0336$ msec. When the data transfer and the computation are executed sequentially the total time is 0.628 msec.

4.4.2. Step 4.2: FG Memory Connection. The required FG bandwidth $RBandW_{FG}$ is 82,600 accesses/msec, the available FG bandwidth of one port is 83,330 and thus 1 port is sufficient to meet the bandwidth.

5. EXPERIMENTAL RESULTS

In this section, we show a broad range of different designs derived from the proposed methodology based on the application characteristics to create a Pareto curve for mapping of application to FPGA. The performance is measured by execution and the microcode provided by SDK for ML-507 cx5vfx70t-ff1136 platform, the HW accelerator area from XST, and the total area from the XPS EDK Xilinx tool.

5.1. Real-Life Micro-Fluid Application

The angle detection algorithm is applied for a 640x480 frame resolution, a 200x16 window, and 100 frames/sec. The results for the different designs are depicted in Table III. The SW Sobel MUL in the execution of the reference angle detection routine with multiplications on the Microblaze soft processor, the SW/HW MUL design puts the critical kernel in a coprocessor with multipliers and it is representative for existing state-of-the-art HW/SW FPGA mapping techniques (Section 2). The estimated results based on the microcode provided from the Xilinx compiler lead to at least 225,900 cycles. The extra area is quite large, that is, 32 slices and 12 DSP48e slices. The DSP48e slices are more complex and the total area will be significantly larger. A lower bound is computed based on $DSPSlices = 4 * CLB_{Tot} = 4 * 2 * Slices$, that is, ≈ 128 slices. The SW/HW-1FSL

Table III. Performance and Area for Demonstration Case Study

Design	Performance 200x16 Window		Performance 300x75 Window		Area	
	Cycles	Time(ms)	Cycles	Time(ms)	HWAcc.	Total
					Slices	Slices
SW MUL Ops	358,996	4.31	1,458,006	17.50	0	4210
SW/HW MUL	231,444*	2.78*	765,901*	9.19*	≈128*	4284*
SW/HW-1FSL	189,864	2.28	439,591	5.27	114	4255
SW/HW-6FSL	176,004	2.11	330,821	3.97	114	4268
SW/HW-2PF-1FSL	174,618	2.09	319,944	3.84	231	4374
SW/HW-2PF-6FSL	167,688	1.98	265,559	3.19	231	4385
SW/HW Sobel&Hough-6FSL	98,705*	1.19*	196,576*	2.36*	314*	4505*
HW & Microblaze M.M.	>70,000*	>0.84*	>182,638*	>2.19*	>375*	>4581*
HW & Custom M.M.	>46,000*	>0.56*	>142,218*	>1.77*	>550*	>4721*

*estimated

Fig. 5. Pareto curve for 200x16 window.

design derived from our methodology achieves gain of 47.11% and the extra HW area is 231 slices. The proposed SW/HW design hides the overhead of the address generation and the memory accesses as they are executed by the processor, while the coprocessor computes the set of the Sobel masks. The DP of the coprocessor includes no idle cycles during the mask execution, since the data are already available and efficient mapping of the operations to the coprocessor is achieved. To compare the proposed design, we insert the HW design with the memory management performed by the Microblaze with a lower bound of 69,484 cycles and of 375 slices. The most performance-efficient design is the application-specific HW with custom memory management and DP dedicated to the specific application, which requires a very large design effort and time. These are not reusable across different applications so the NonRecurring Engineering (NRE) cost will be shared for a relatively small market volume. For deep submicron process technologies with hugely increasing NRE costs, this is a clear disadvantage. The proposed methodology design achieves gain of 79.29% in area compared to conventional HW design.

We modify the application requirements and briefly describe the different SW/HW designs derived from the proposed methodology to compose the Pareto curve depicted in Figure 5. When the frame rate is increased to 105 frames/sec, $D = 9.52$ msec, $t_{\text{Angle}} = 2.22$ msec, and $t_{\text{HW}} = 0.558$ msec (Algorithm 1). The time estimation of transferring the data is 0.23 msec from the BG memory (Eqs. (1)–(3)), 0.23 msec to the coprocessor (Eqs. (4)–(8)) and 1.168 msec is the critical path of the FG and the DP (Algorithm 2). Then, parallel FSL (Eq. (10)) are required to transfer the data to decrease the transfer time to 0.038 msec (SW/HW-6FSL). When the frame rate is

Table IV. Performance and Area for PolyBench Benchmark Suite

Bench.	Design	Performance (Cycles)	HW Acc.					
			Primitive Operations	CP (ns)	f (MHz)	Slices	DSP Slices	Total Slices
2mm	SW	6,000,208	-	-	-	-	-	-
	SW/HW	2,155,152	1-2PL	10.17	98.32	38	6	134
			2	10.11	98.87	24	3	
	SW/HW-2PF	1,298,043	1-3PL	10.44	95.8	71	12	263
			2	11.49	87.06	48	6	
3mm	SW	9,067,317	-	-	-	-	-	-
	SW/HW	3,227,592	1	10.13	98.67	22	3	46
	SW/HW-2PF	1,832,904	1	11.48	87.08	50	6	98
Bicg	SW	128,678	-	-	-	-	-	-
	SW/HW	48,256	1	10.11	98.91	42	6	90
Gemm	SW	3,648,404	-	-	-	-	-	-
	SW/HW	2,246,593	1-2PL	10.22	97.81	41	6	89
	SW/HW-3PL	2,190,881	1-2PF	10.65	93.9	73	12	169
Gesummv	SW	143,301	-	-	-	-	-	-
	SW/HW	70,753	1	10.11	98.87	24	3	48
	SW/HW-2PF	65,633	1	11.49	87.06	48	6	96
Atax	SW	139,420	-	-	-	-	-	-
	SW/HW	80,790	1	10.13	98.67	22	3	46
Cholesky	SW	374,334	-	-	-	-	-	-
	SW/HW	70,753	1	10.31	96.99	25	3	49
Gemver	SW	251,550	-	-	-	-	-	-
	SW/HW	96,650	1	11.48	87.08	50	6	203
			2-2PL	10.17	98.32	39	6	
			3	2.67	374.96	18	0	
Jacobi_2d	SW	1,333,510	-	-	-	-	-	-
	SW/HW	303,464	1	6.59	151.86	80	0	80
Correlation	SW	4,413,504	-	-	-	-	-	-
	SW/HW	1,084,299	1	2.65	376.65	16	0	1194
			2	11.91	83.97	32	6	
			3-3PL	44.82	22.31	1028	3	
			4	10.13	98.67	22	3	

SW: Software, SW/HW: Co-design, PF = Parallel Factor, PL = Pipeline stages.

increased to 115 frames/sec, $t_{\text{Angle}} = 1.39$ msec and $t_{\text{HW}} = 0.351$ msec. The time estimation is 0.23 msec for transferring from the BG memory (Eqs. (1)–(3)) and to the coprocessor (Eqs. (4)–(8)), which exceeds the available time. Hence, parallel FSL are required (Eq. (10)). Even then the required time is 0.439, which exceeds the available time. Parallelization is explored with a factor of $PF = 3$ (Eq. (17)). The total time is estimated at 0.32 msec (Algorithm 4). Since the PF is highly increased, the option of mapping the second kernel to a HW accelerator is explored. Then, the $t_{\text{HW}} = 0.765$ msec, the $BandW = 1,271,706$ bits/msec (Eq. (6)), the estimated critical path of the Hough kernel is $CP_{\text{HW,Hough}} = 0.039$ msec, and the total critical path based on the dependencies is $CP_{\text{HW,Tot}} = 0.082$ msec. When 1 FSL is used (Eq. (10)), the total time exceeds the available. With 6 parallel transfers for the Sobel kernel and a sequential transfer for the Hough, the total estimated time is 0.675 msec. The window of applying the angle detection is increased (300x75) and 8 bits are used to store the data. Then, more cycles are required for the execution and less bandwidth for the transfer. For 60 frames/sec, $D = 16.06$ msec, $t_{\text{Angle}} = 9.36$ msec, and $t_{\text{HW}} = 2.351$ msec. The required

Fig. 6. Pareto curves for 2mm, 3mm, and Gemm for data size 128.

Table V. Comparison Results for the Proposed and the Iterative Improvement Approach

Algorithm	Iterative Improvement			Proposed Methodology		
	Perf.	Area	Time	Perf.	Area	Time
	(ms)	(LUT)	(ms)	(ms)	(LUT)	(ms)
Seidel-2d	0.02020892	72	118,686.089	0.02020892	72	1,125.692
	0.02024000	64		0.02022202	64	
	0.02524778	56		0.02521426	56	
	0.02526088	48		0.02521426	48	
	0.03525846	40		0.03062306	40	
	-	-		0.04059266	32	
Gemm	0.1609017	40	50,451.987	0.1609017	40	256.695
	0.1609075	32		0.1609075	32	
	0.2418297	24		0.2418297	24	
Syr2k	0.3217926	40	55,155.037	0.3217926	40	265.931
	0.3217984	32		0.3217984	32	
	0.48365575	24		0.48365575	24	

bandwidth is $BandW = 115,303$ bits/msec (Eq. (6)), the estimated time for BG transfers is 0.40 msec (Eqs. (1)–(3)), for parallel FSLs is 0.067 msec (Eqs. (4)–(8)) and the critical path is 2.33 msec. Parallelization is explored (with $PF = 2$ (Eq. (17))) to meet real-time constraints.

Industrial design practices with experienced designers will potentially also reach these results, but with substantial design effort and without the guarantee of systematically finding the relevant Pareto points.

5.2. PolyBench Benchmark Suite

The PolyBench benchmarks [Pouchet et al. 2012] is a polyhedral benchmark suite with static control parts and has as purpose to make the execution and monitoring of kernels uniform. It includes linear algebra, data mining, and medley and stencils kernels. The PolyBench benchmarks are usually used as parts of more complex applications with real-time constraints, for example, matrix-matrix multiplication is used in signal processing applications, Jacobi is used to determine solutions of linear equations, etc. A selected set of SW reference designs and SW/HW codesigns for 10 different PolyBenchmarks derived from our methodology is depicted in Table IV. The Pareto curves for a set of PolyBenchmarks are summarized in Figure 6. They show the effectiveness and broad applicability of our approach. Notably we also produce a wide range of Pareto working points which are crucial in practical design contexts where trade-offs typically exist between the different objectives.

Fig. 7. Exploration time comparison when the application size is increased by a factor.

5.3. Relative Comparison

Existing methodologies and frameworks mainly use different architectural assumptions and valid options in the designs. To provide a useful comparison in terms of performance, area, and the exploration time, we have implemented an Iterative Improvement (II) approach, similar to the existing in the current literature (as much as possible mimicking it). The comparison is performed for one of the most complicated steps of our methodology, that is, the ForeGround memory management step. The remaining steps have in the worst case similar behavior and thus the relative comparison remains representative for the other steps also. The II approach applies a register scheduling and assignment step and an improvement step. The first step is based on the mobility of the FG memory operations. After the first step and the scheduling and assignment of the selected operations, the improvement step is applied. The improvement step selects different nodes in previous iteration steps to search for potential improvements. The results of our approach and the II approach for a set of test cases are depicted in Table V. For the seidel-2d benchmark the II has less optimal results because it selects to schedule later a node whose successors affect the critical path, as it has the same mobility with the other ready-to-be-scheduled nodes. The proposed methodology uses different types of nodes. It schedules and assigns based on the node type and unidirectionally propagates constraints, that is, scheduling decisions, to the nodes of the next-to-be-scheduled type. In this way it can identify points that are outside the local scope of the II. For smaller benchmarks, the II potentially has similar quality with the proposed approach but the points require a much higher exploration time to be produced. When the number of nodes in the application graph is increased the exploration time of the II is strong polynomially increased and the proposed methodology remains linear (Figure 7).

6. CONCLUSIONS

A systematic stepwise template-based methodology is described to compose a Pareto curve for near-optimal mapping of an application to a SW/HW design with a processor and a (set of) HW accelerator(s) taking into account the SW/HW organization, the FG memory management, and the DP mapping. The suboptimal options are pruned early in the design process based on scalable what-if analysis and the constraints propagation of each step lead to a scalable and efficient approach.

REFERENCES

- Ahn, Y., Han, K., Lee, G., Song, H., Yoo, J., Et al. 2008. Socdal: System-On-Chip design accelerator. *Trans. Des. Autom. Electron Syst.* 13, 1, 17:1–17:38.
- Bjerregaard, T. and Mahadevan, S. 2006. A survey of research and practices of network-on-chip. *ACM Comput. Surv.* 38, 1, 1–51.
- Callahan, T., Hauser, J., and Wawrzynek, J. 2000. The garp architecture and c compiler. *J. Comput.* 33, 4, 62–69.

- Capitanio, A., Nicolau, A., and Dutt, N. 1995. A hypergraph-based model for port allocation on multiple-register-file vliw architectures. *Int. J. Parallel Program.* 23, 6, 499–513.
- Compton, K. and Hauck, S. 2002. Reconfigurable computing: A survey of systems and software. *ACM Comput. Surv.* 34, 171–210.
- Cooper, K. D., Simpson, L. T., and Vick, C. A. 2001. Operator strength reduction. *ACM Trans. Program. Lang. Syst.* 23, 5, 603–625.
- Criticalblue. 2012. Criticalblue cascade, programmable application coprocessor generation. <http://www.criticalblue.com>.
- Demiris, A. and Blonas, S. 2011. Integrated system for the visual control, quantitative and qualitative flow measurement in microfluidics. Hellinic Industrial Property Organisation patent 20110100390.
- Diguet, J. P., Chillet, D., and Sentieys, O. 2000. A framework for high level estimations of signal processing vlsi implementations. *J. VLSI Signal Process. Syst.* 25, 3, 261–284.
- Dimond, R., Mencer, O., and Luk, W. 2005. Custard - A customisable threaded fpga soft processor and tools. In *Proceedings of the International Conference on Field Programming Logic and Applications*. IEEE, 1–6.
- Ferrandi, F., Lanzi, P. L., Palermo, G., Pilato, C., Sciuto, and Tumeo, A. 2007. An evolutionary approach to area-time optimization of fpga designs. In *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*. IEEE, 145–152.
- Flatt, H., Blume, H., and Pirsch, P. 2010. Mapping of a real-time object detection application to a configurable risc/coprocessor architecture at full hd resolution. In *Proceedings of the Reconfigurable Computing and FPGAs (ReConFig)*. IEEE, 452–457.
- Gajski, D., Vahid, K., Narayan, S., and Gong, J. 1998. Specsyst: An environment supporting the specify-explore-refine paradigm for hardware/software system design. *IEEE Trans. VLSI Syst.* 6, 1, 84–100.
- Guo, Z., Najjar, W. A., and Buyukurt, B. 2008. Efficient hardware code generation for fpgas. *ACM Trans. Archit. Code Optim.* 5, 1, 6:1–6:26.
- Hennessy, J. and Patterson, D. 2006. *Computer Architecture: A Quantitative Approach* 4th Ed. Morgan Kaufmann Publishers, San Francisco, CA.
- Huang, C. and Vahid, F. 2011. Scalable object detection accelerators on fpgas using custom design space exploration. In *Proceedings of the Symposium on Application Specific Processors*. IEEE, 115–121.
- Jozwiak, L., Gawlowski, D., and Slusarczyk, A. 2006. Multi-objective optimal controller synthesis for heterogeneous embedded systems. In *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*. 177–184.
- Kim, Y., Lee, J., Mai, T. X., and Paek, Y. 2012. Improving performance of nested loops on reconfigurable array processors. *ACM Trans. Archit. Code Optim.* 8, 4, 32:1–32:23.
- Koes, D. R. and Goldstein, S. C. 2008. Near-optimal instruction selection on dags. In *Proceedings of the International Symposium on Code Generation and Optimization*. ACM Press, New York, 45–54.
- Kornaros, G. 2010. A soft multi-core architecture for edge detection and data analysis of microarray images. *J. Syst. Archit.* 56, 48–62.
- Kritikakou, A., Catthoor, F., Athanasiou, G. S., Kelefouras, V., and Goutis, C. 2012. A template-based methodology for efficient microprocessor and fpga accelerator co-design. In *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*. 15–22.
- Kritikakou, A., Catthoor, F., Kelefouras, V., and Goutis, C. 2013. A systematic approach to classify global scheduling techniques. *ACM J. Comput. Surv.* 45, 2, Article 14.
- Lam, M. 1988. Software pipelining: An effective scheduling technique for vliw machines. *SIGPLAN Not.* 23, 7, 318–328.
- Liao, J., Wong, W.-F., and Mitra, T. 2003. A model for hardware realization of kernel loops. In *Proceedings of the International Conference on Field Programmable Gate Arrays*. Vol. 2778, Springer, 334–344.
- Melpignano, D., Benini, L., Flamand, E., Jego, B., Lepley, T., et al. 2012. Platform 2012, a many-core computing accelerator for embedded socs: Performance evaluation of visual analytics applications. In *Proceedings of the 49th Annual Design Automation Conference (DAC)*. ACM Press, New York, 1137–1142.
- Milder, P., Franchetti, F., Hoe, J. C., and Puschel, M. 2012. Computer generation of hardware for linear digital signal processing transforms. *ACM Trans. Des. Autom. Electron. Syst.* 17, 2, 15:1–15:33.
- Neumann, B., Von Sydow, T., Blume, H., and Noll, T. G. 2008. Designflow for embedded fpgas based on a flexible architecture template. In *Proceedings of the Conference on Design, Automation and Test in Europe*. ACM Press, New York, 56–61.

- Novo, D., Kritikakou, P., Raghavan, P., Van Der Perre, L., Huisken, J., and Catthoor, F. 2010. Ultra low energy domain specific instruction-set processor for on-line surveillance. In *Proceedings of the IEEE 8th Symposium on Application Specific Processors (SASP)*. IEEE Computer Society Press, 30–35.
- Palermo, G., Silvano, C., and Zaccaria, V. 2005. Multi-objective design space exploration of embedded systems. *J. Embedded Comput.* 1, 305–316.
- Poletto, M. and Sarkar, V. 1999. Linear scan register allocation. *ACM Trans. Program. Lang. Syst.* 21, 5, 895–913.
- Pouchet, L.-N., et al. 2012. Polybenchmarks benchmark suite. <http://www.cse.ohio-state.edu/~pouchet/software/polybench/>.
- Sant’Anna, R., De Lima, M. E., and Mariel, P. R. M. 2004. A left-edge algorithm approach for scheduling and allocation of hw contexts in dynamically reconfigurable architectures. In *Proceedings of the ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays (FPGA)*. ACM Press, New York, 259–259.
- Shahzad, M. and Zahid, S. 2009. Image coprocessor: A real-time approach towards object tracking. In *Proceedings of the International Conference on Digital Image Processing*. IEEE, 220–224.
- Sheldon, D., Kumar, R., Lysecky, R., Vahid, F., and Tullsen, D. 2006. Application-specific customization of parameterized fpga soft-core processors. In *Proceedings of the International Conference on Computer-Aided Design*. IEEE, 261–268.
- Sheldon, D. and Vahid, F. 2009. Making good points: Application-specific pareto-point generation for dse using statistical methods. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*. ACM Press, New York, 123–132.
- Synopsys. 2012. Synopsys symphony - high level synthesis solution. <http://www.synopsys.com>.
- Vassiliadis, N., Theodoridis, G., and Nikolaidis, S. 2009. The arise approach for extending embedded processors with arbitrary hardware accelerators. *Trans. VLSI* 17, 2, 221–233.
- Xilinx. 2011. Logicore ip multi-port memory controller.

Received June 2012; revised October 2012; accepted December 2012