Construction and exploitation of VLIW ASIPs with heterogeneous vector-widths

Citation for published version (APA):

Diken, E., Jordans, R., Corvino, R., Jozwiak, L., Corporaal, H., & Chies, F. A. (2014). Construction and exploitation of VLIW ASIPs with heterogeneous vector-widths. *Microprocessors and Microsystems*, 38(8), 947-959. <https://doi.org/10.1016/j.micpro.2014.05.004>

DOI:

[10.1016/j.micpro.2014.05.004](https://doi.org/10.1016/j.micpro.2014.05.004)

Document status and date:

Published: 01/01/2014

Document Version:

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Construction and exploitation of VLIW ASIPs with heterogeneous vector-widths

Erkan Diken^{a,*}, Roel Jordans^a, Rosilde Corvino^a, Lech Jóźwiak^a, Henk Corporaal^a, Felipe Augusto Chies^b

^a Eindhoven University of Technology, Den Dolech 2, 5612 AZ Eindhoven, The Netherlands

^b Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, Brazil

ARTICLE INFO

Article history:

Received 11 November 2013

Revised 15 April 2014

Accepted 8 May 2014

Available online 15 May 2014

Keywords:

VLIW

ASIPs

Vector processing

DLP

SIMD

ABSTRACT

Numerous applications in important domains, such as communication and multimedia, show a significant data-level parallelism (DLP). A large part of the DLP is usually exploited through application vectorization and implementation of vector operations in processors executing the applications. While the amount of DLP varies between applications of the same domain or even within a single application, processor architectures usually support a single vector width. This may not be optimal and may cause a substantial energy inefficiency. Therefore, an adequate more sophisticated exploitation of DLP is highly relevant. This paper proposes the use of heterogeneous vector widths and a method to explore the heterogeneous vector widths for VLIW ASIPs. In our context, heterogeneity corresponds to the usage of two or more different vector widths in a single ASIP. After a brief explanation of the target ASIP architecture model, the paper describes the vector-width exploration method and explains the associated design automation tools. Subsequently, experimental results are discussed.

© 2014 Elsevier B.V. All rights reserved.

1. Introduction

Computing platforms embedded in various modern devices are often required to satisfy high performance demands when processing data intensive applications from such fields as communication, multimedia, image processing or signal processing. Moreover, embedded systems of a mobile or autonomous equipment must also ensure a low energy consumption, due to a limited battery life. Very often embedded systems can also profit from flexibility, in the form of adaptability and programmability of their computing platforms, to accommodate the late design changes or tune the design to the application needs. The low energy consumption and high performance are often achieved through usage of highly specialized hardware processors realized as application specific integrated circuits (ASICs). These processors can be very efficient, but their flexibility is very limited. In contrast, application specific instruction-set processors (ASIPs) are programmable and, due to their customization to a specific application, can deliver high performance and energy efficiency. Moreover, ASIPs can be re-used

for different application versions or even for different applications in the same or similar domain due to their programmability. Therefore, they are becoming a more preferred alternative than the hard-wired processors. Modern system-on-chip solutions (e.g. [1,2]) targeting mobile computing platforms include such programmable and customized ASIP-based sub-systems.

The computing effectiveness and efficiency provided by ASIPs can be boosted by an adequate exploitation of the intrinsic parallelism of a given application. Coarsely speaking, the intrinsic parallelism of an application corresponds to the number of its independent operations that can be executed simultaneously. In the context of single-instruction multiple-data (SIMD)/very long instruction word (VLIW) architectures, the following two forms of parallelism are the subject to be exploited at the instruction level: instruction-level parallelism (ILP) and data-level parallelism (DLP). ILP refers all kinds of independent operations that can be concurrently executed. ILP is realized through parallel hardware units, as for instance issue slots in VLIW architectures or custom instruction set extensions. Realizing ILP through parallel issue slots has a limited scalability [3] due to the required high connectivity between the computing units and data storage units (e.g. register files and local memories). Moreover, it requires a wider program memory, with more complex instruction encoding and decoding, which results in a higher area and higher energy consumption of the program memory.

* Corresponding author.

E-mail addresses: e.diken@tue.nl (E. Diken), r.jordans@tue.nl (R. Jordans), r.corvino@tue.nl (R. Corvino), l.jozwiak@tue.nl (L. Jóźwiak), h.corporaal@tue.nl (H. Corporaal), felipe.chies@inf.ufrgs.br (F.A. Chies).

DLP refers to multiple occurrences of the same operation that can be independently executed on different data sub-sets. DLP is usually exploited through design and implementation of SIMD instructions, also called vector instructions. Vector processing is one of the main enablers of computing effectiveness and efficiency due to its regular structure, and low control and interconnect overhead. On the other hand, the usage of vector units in the ASIP hardware is effective and efficient only when the vector width of the hardware units matches the intrinsic DLP of the application. All the other cases result in the loss of either efficiency or effectiveness.

Fig. 1 exemplifies the effect of a mismatch between the hardware vector width and application DLP. It illustrates the energy consumption of a 2-tap filter (cf. Listing 1) executed on an ASIP with different vector width configurations. The example kernel exhibits the maximum DLP of 16. The term *maximum DLP*, corresponds to the maximum number of data items possible to be processed in parallel (e.g. the number of image pixels that can be processed in parallel). The dynamic energy is reduced by the increase of the vector width from 2 to 16 due to the reduction of the number of operations related to the control-flow (e.g. address generation, loop branches) of the kernel. When the vector width is higher than 16, the dynamic energy is constant due to the limitation imposed by the maximum DLP of the kernel (it is assumed that a clock/power gating is applied in order to disable the unneeded part of the vector function units and corresponding register files). The static energy is proportional to the area of the ASIP and the execution time of the kernel. The static energy slightly increases by the growth of the vector width from 2 to 16. It does not increase much because the increase in the area is compensated by the reduction in the execution time. When the vector width is equal to 16, the execution time reaches its lowest value. However, the area of the ASIP increases due to the increase of the vector widths. In the presence of the power gating [4], the static energy has more or less the same value. Otherwise, it tends to increase due to the wider units than needed.

Former research on application analysis [3,5,6] has shown that different application kernels in important domains, such as communications (e.g. FFT/IFFT, STBC, LDPC) and multimedia (e.g.

MPEG4 audio/video decoding, 3D graphics rendering, H.264) have different maximum natural DLPs. Table 1 presents the DLP analysis of various applications and some kernels being part of the these applications. Serving these kernels or applications with an architecture which has a single vector width may not be optimal and may cause a substantial energy and performance inefficiency. Therefore, adequate exploitation of DLP is highly relevant. We argue and experimentally confirm that the heterogeneity imposed by varying DLP can be much more efficiently served with heterogeneous vector widths. However, to realize this, a new method is needed to explore and decide the heterogeneous hardware architecture.

In this paper, we propose and discuss a new method that aims at exploring and deciding the architectural parameters of heterogeneous vectorization, i.e. the number, type and width of SIMD function units. The contributions of the research reported in this paper includes the following:

- analysis of the problem of VLIW ASIP construction with heterogeneous vector units;
- a new method of heterogeneous vector-width exploration for VLIW ASIPs;
- a design automation tool for selecting the right composition of vector widths for a given application;
- experimental analysis and demonstration of the applicability of our method for a set of kernels with different DLPs.

The research work presented in this paper was performed in the scope of the European project ASAM (Architecture Synthesis and Application Mapping for heterogeneous MPSoCs based on adaptable ASIPs) of the ARTEMIS program. The general aim of the ASAM project is to enhance the design efficiency of the ASIP-based MPSoCs for highly demanding applications, while improving the result quality. This aim is being realized through the development of a coherent system-level design-space exploration and synthesis flow including automatic analysis, synthesis and rapid prototyping. The flow and its implementation have to provide efficient exploration of the architecture and application design alternatives and

Fig. 1. Energy consumption trend of a 2-tap filter executed for different vector width configurations of an ASIP. The kernel exhibits the maximum DLP of 16.

```

for (h=0; h<height-1; h++)
{
    for (w=0; w<width; w++)
    {
        image_out[h+1][w]=(image_in[h][w]+image_in[h+1][w])>>1;
    }
}

```

Listing 1. A 2-tap filter.

Table 1

Data-level parallelism analysis of multimedia and communication kernels and applications (FFT/IFFT: fast fourier transform, AAC: MPEG4 audio decoding, STBC: space time block coding, LDPC: low-density parity check). Deblocking filter, inverse transform, motion compensation and intra-prediction kernels are parts of the H.264 decoding application.

Kernel/application	Maximum DLP
FFT/IFFT [5], AAC [3]	1024
STBC [5]	4
LDPC [5]	96
Deblocking filter, inverse transform, motion compensation [5]	8
Intra-prediction [5]	16
3D graphics rendering [3]	128

Fig. 2. ASAM design flow.

trade-offs. The ASAM overview paper [7] briefly explains the results of the analysis of the main problems and challenges to be faced in the design of such heterogeneous MPSoCs. It explains which system, design, and electronic design automation (EDA) concepts seem to be adequate to resolve the problems and address the challenges. Moreover, it introduces and discusses the design-flow, its main stages and the tools proposed by the ASAM project consortium to enable an effective and efficient solution of these problems. It also shows the application of the ASAM tools to a real-life case study. The ASAM design flow involves the following main stages (see Fig. 2): micro-level DSE, macro-level DSE, and communication and memory DSE. The method and design automation tool presented in this paper constitute a part of the micro-level DSE stage. The micro-level DSE stage is responsible for designing the ASIPs for the given task or tasks.

This paper is structured as follows. In the next section, related research is discussed. In Section 3, the target architecture model used and its heterogeneous forms are explained. Section 4 focuses on our new method and its corresponding design automation flow. Section 5 experimentally demonstrates the applicability of our method and discusses the experimental results. Finally, Section 6 concludes the paper.

2. Related work

Traditionally, DLP is implemented using vector processing units with a single vector width, as in the cases of 32-wide vector SODA

[8], 8-wide vector Imagine [9] and 16-wide vector NXP EVP [10] processors. In these architectures, parts of the application where the DLP amount exceeds the vector width may be served through several parallel issue slots or by sequential iterations over the same vector unit.

Research of the heterogeneous vector processing is quite new. We were able to find only a very limited set of publications targeting this specific topic. In [5], an analysis of computational characteristics of 4G wireless communication and high-definition video algorithms is carried out. The analysis showed that different algorithms in the same application domain have different intrinsic DLPs. In the same paper, an example architecture, referred to as *anySP*, with configurable SIMD data-path which supports wide and narrow vector widths is proposed. Moreover, the paper suggests some other architectural enhancements such as the temporary buffer with the bypass network and the swizzle network to support data reordering. However, it does not focus on any method for exploring the heterogeneous vector widths, as we do in our work. Another work presented in [11], referred to as *Libra*, also focuses on the heterogeneous construction of architectures with different vector widths. It considers dynamic reconfiguration of SIMD-width of the architecture based on the DLP characteristic of loops. Dynamic configurability enables lane resource to execute as a traditional SIMD processor, be re-purposed to behave as a clustered VLIW processor, or combinations of both. In our work, we focused on the static configuration of an ASIP architecture tailored to specific kernels or an application.

Moreover, several concepts were presented to support flexible architecture construction to serve different kinds of parallelisms. The SIMD-Morph [12] architecture uses transition modes to exploit both DLP and ILP. The Vector-Thread (VT) architecture [13] can execute in multiple modes in order to support both DLP and TLP, while TRIPS architecture [14] exploits ILP, DLP and TLP. However, no one of these works addressed the heterogeneous vectorization being the subject of this paper.

3. Architecture model

3.1. Target architecture model

The target ASIP architecture is a VLIW machine capable of executing parallel software with a single thread of control. Fig. 3 depicts a simplified view of the corresponding generic ASIP architecture template. It includes a VLIW data-path controlled by a sequencer that uses status and control registers, and executes a program stored in a local program memory. The data-path contains function units organized in several parallel scalar and/or vector issue slots (IS) connected via a programmable interconnect network to register files (RFs). The register files and issue slots can be organized in clusters. The function units perform computation operations on intermediate data stored in the register files. Only function units in different issue slots can execute parts of an application simultaneously. Local memories, collaborating with particular issue slots, enable scalar access for the scalar slots, and vector or block access for the vector slots. The target architecture model is configurable and extensible. The parameters to be explored and set to create a new ASIP configuration include: the number and type of issue slots and (scalar or vector) instructions inside the issue slots, the number and type of issue slot clusters to optimize parallelism exploitation and communication between the issue slots, the number and size of register files, the type, data width, and size of local memories, the architecture and the parameters of the local communication structure, etc.

This architecture model corresponds to some actual industrial ASIP architectures used in modern MPSoCs for mobile applications, as for instance, to a VLIW ASIP architecture of Intel Benelux

Fig. 3. Generic ASIP architecture template.

being a major industrial participant of the ASAM project. The vector-width exploration method explained in Section 4 constitutes a part of our design automation tool-flow for this industrial ASIP technology.

3.2. Heterogeneity of the architecture

The targeted generic ASIP architecture allows us to construct architecture instances involving heterogeneous architecture structures. For vector units the heterogeneity is represented by two parameters: operation type and vector width, meaning that it is possible to have different function units in the processor data-path and these units can have different (vector) widths. Fig. 3 also depicts the heterogeneous architecture structure of the VLIW data-path. Cluster 2 and Cluster N correspond to two heterogeneous components of the data-path. The execution units in each cluster can support different functionalities (FU_1 and FU_N) and have different widths (w_1 and w_2). This structure provides both DLP and ILP. DLP is realized inside each issue slot through vector function units, and the parameters $w_{1,2,...,n}$ define the corresponding vector widths. ILP is enabled by having several parallel issue slots. This architecture can be used for parallel executable kernels (tasks) with different DLP and different functionality.

4. Vector-width exploration method

In this section, our new heterogeneous vector-width exploration method is explained and discussed. The method aims at exploring and deciding the set of heterogeneous vector widths specific to a given set of tasks. Each task corresponds to a kernel (i.e. a system of nested loops which realizes a particular computation). To propose adequate solutions, the vector-width exploration has to consider the HW/SW partitioning (hardware allocation and task mapping) and a coarse scheduling, as well as, the estimation of the relevant design metrics, as energy consumption, area occupation and performance. Furthermore, application analysis is required to characterize the application regarding its parallelism and analytical models [15] are needed for a fast estimation of the design metrics.

The method combines the use of two different abstraction levels, for which two different input specifications are used.

1. *High-abstraction level*: An adequate increase of the abstraction level of the program representation and corresponding program analysis eliminates the irrelevant program details, and in result, reduces the design space size and the exploration time. Due to the exploration time reduction, the exploration at this level can efficiently account for the whole initial set of the most promising coarse architectures to be considered for a further design refinement.

2. *Cycle-accurate level*: Each coarse architecture solution provided by the high-abstraction level is refined through actually building the precise design of the corresponding processor, code compilation for this processor and HW/SW simulation, followed by an estimation step which analyzes the activity counts from the cycle-accurate simulation.

The **input** of our design flow includes: the ANSI-C application behavior specification, the requirements on delay, energy consumption and area, and the processor architecture template (PAT). The **output** of the design flow is a set of ASIP designs that optimize the quality metrics w.r.t. the selected vector processing architecture. Fig. 4 graphically represents the design flow which implements the proposed method. It has the following two main parts: the pre-exploration (at the high-abstraction level) and the actual vector-width exploration. The details of the design flow are explained below.

4.1. Pre-exploration (at the high-abstraction level)

Former research (e.g. [16,17]) has shown that a large part (e.g. 50–80%) of the total cost of an information processing sub-system for a data-intensive application is due to the data storage and transfer. The data storage and transfer are strictly related to exploitation of the task-level and data-level parallelism. In our VLIW ASIP design method, the exploration of the task-level parallelism and coarse exploration of the data parallelism are performed before the exploration of the vector parallelism and result in a coarse architecture of the ASIP-based sub-system, deciding the coarse memory, communication and data-path architecture. In order to find a set of the most promising coarse architecture solutions, all the possible resource allocations and their corresponding mapping solutions are explored by another partial tool, earlier-developed and implemented [18,19]. This tool accepts a task-graph specification of an application, processor architecture template and application requirements as inputs. It explores the task and data-level parallelisms by applying several transformations in order to construct the most promising coarse architecture solutions w.r.t. the quality indicators. The pre-exploration phase includes the following three main steps:

1. It infers an abstract array-oriented model (Array-OL) from C specification. Array-OL is used to represent the task-graph model of the application. It is a data-flow based formalism able to represent the data intensive applications as a pipeline of parallel tasks performed on multidimensional data arrays. More details are given in [20].
2. It applies P2CS (parallel processing, communication and storage) exploration tool in order to explore possible restructuring of the array-oriented model (i.e. combinations of task fusion, tiling and paving change).

Fig. 4. Tool-flow for exploring the heterogeneous vector widths.

3. It uses a set of allocation, mapping and scheduling rules in order to infer the correspondingly modified (restructured) C code and the corresponding initial coarse ASIP architecture description from an Array-OL instance.

The output of the pre-exploration phase is composed of the restructured C code, including mapping of data to the local memories, and the corresponding initial coarse ASIP-based sub-system architecture. The initial coarse architecture is further explored by the vector-width exploration tool, implementing the method being the subject of this paper, to decide the parameters of the vector processing and related hardware.

```

// Kernel 1
for(ht = 0; ht < height; ht++) {
    for(wd = 0; wd < width; wd++) {
        T1: image_out1[ht,wd] = image_in1[ht,wd];
    }
}

// Kernel 2
for(ht = 0; ht < height; ht++) {
    for(wd = 0; wd < width; wd++) {
        T2: image_out2[ht,wd] = image_in2[ht,wd];
    }
}

```

Listing 2. An example of input C code.

```

vector ON(VMEM0) image_in1[height][width];
vector ON(VMEM1) image_in2[height][width];
// Merged Kernels
for(ht = 0; ht < height; ht++) {
    for(wd = 0; wd < width; wd++) {
        T1: image_out1[ht,wd] = image_in1[ht,wd];
        T2: image_out2[ht,wd] = image_in2[ht,wd];
    }
}

```

Listing 3. An example of restructured C code (merged kernels for parallel execution) and memory mappings.

The code listed in Listing 2 (computations carried out by the tasks are not shown for the sake of simplicity) is an example of an input C code. The code includes two nested loops. Each loop processes two different images (*image_in1* and *image_in2*) of certain heights and widths. First, the kernels are translated into their task-graph model, then several transformations are explored using the model. Listing 3 represents a possible output of the exploration. The original kernels are merged into one kernel as a result of the task fusion in the model. Moreover, the code is annotated using the *ON* keyword in order to specify the data mapping to local memories (e.g. *ON(VMEM0)*, *ON(VMEM1)*). The keyword *vector* is used to designate the input data for the vector processing. The corresponding coarse ASIP sub-system architecture, including the number of issue slots, the number and size of register files and data memories, is also generated.

4.2. Vector-width exploration

The refinement of the coarse ASIP architecture regarding the vector processing is decided in the vector-width exploration phase. Vector-width exploration focuses on finding the best possible set of vector widths for a given restructured C code, coarse ASIP architecture, and a data mapping solution. Fig. 5 depicts the basic system setup for starting the exploration. It consists of host code (*host.c*) and kernel code (*kernel.c*). The host code is responsible for initiating and controlling the execution of the kernel code. The host code manages storing data from the host memory to the local memories of the ASIP processor, starting the kernel code and eventually loading the processed data back to the host. The kernel code includes the main task to be executed by the ASIP.

4.2.1. Enabling heterogeneous vectorization

Being able to construct and exploit a processor with two different vector widths requires accomplishment of the following three tasks. First of all, having a second vector width requires definition of a second vector type (*vector2*), in addition to default vector type (*vector*). The width (*w*) of a vector type corresponds to the product of *nways* (*number of lanes*) and *element precision*. In this way, definition of *nway1*, *nway2*, etc., each having different values, decides the width of each vector type. The code presented in *definitions.h* (cf. Fig. 5) shows the usage of two different *nways* (*eva_nway1*, *eva_nway2*) that are two of many ASIP configuration parameters. We assume that *element precision* is fixed (e.g. 32 bits). Moreover, the kernel code exemplifies the usage of both vector types in the kernel code. Secondly, processor building blocks (e.g. operations, function units, issue slots) that are compatible with the new vector type need to be constructed. Subsequently, these building blocks can be instantiated in the processor description files. Finally, *application programming interface (API)* support is required for transferring the *vector2* type of data between the host and processor. The host code provides an example usage of these functions (*_pack_store_vector2()* and *_load_vector2()*).

Fig. 5. Initial system setup for exploration.

4.2.2. Exploration

Algorithm 1 presents the pseudo-code of the script that automates the vector-width exploration. The vector width set (N) to be explored, the directories that include application (kernel and host C files) and the processor description files are provided as exploration algorithm inputs. The exploration includes ASIP building, data packing/storing and synchronization, code compilation, simulation and estimation steps for each vector width to be explored. These steps are further explained below.

Algorithm 1. Vector-width exploration script pseudo-code

```

1: procedure VectorWidthExploration
2:   for all nway1, nway2  $\in N$  do
3:     asip  $\leftarrow$  buildASIP(nway1, nway2);
4:     dataPackingAndStoring(nway1, nway2);
5:     updateLoopIterationsAndSynchFactor();
6:     schedule  $\leftarrow$  compile(kernel, asip);
7:     activity  $\leftarrow$  simulate(asip, schedule, inputstimuli);
8:     estimates  $\leftarrow$  estimate(activity, component cost);
9:   end for
10: end procedure

```

4.2.3. ASIP building

The ASIP architecture template is configured specific to the vector width selected. The width of function units, register files and vector memories are adjusted and the rest of the processor building blocks are tailored accordingly. ASIP builder compiles and builds the adjusted template in order to create an ASIP instance of the architecture. The parameters $CORE_NWAY1$ and $CORE_NWAY2$ specified in *definitions.h* are updated accordingly. This way application is made aware of the actual vector widths setting in the target architecture.

4.2.4. Data packing and storing

Depending on the vector width setting, data need to be packed accordingly and stored into the corresponding local memories. As

mentioned before, data mapping is decided in the pre-exploration phase. The local memories used are *scalar addressable* type of vector memories. The load/store unit of the processor accesses the data by using *base address + offset* formulation. Each access to the local memories loads/stores the aligned packed data and takes two/one clock cycle(s). For instance, an image with *height * width* pixels requires $(height * width)/nway$ accesses to the memory in order to load the whole image. Moreover, loop iteration counts in the kernel code need to be adjusted accordingly. In the host code, the store function (*_store(height1, height2, width1, width2)*) is used to update the kernel about the new widths and heights of the input image. For the sake of paper brevity, the code that computes the height and width parameters is not presented.

4.2.5. Synchronization

In the case of a parallel execution of several kernels, the synchronization of the kernels has to be handled explicitly by introducing an additional synchronization loop. The inner-most loop in the kernel code (3rd level loop) corresponds to the manually added synchronization loop, which does not exist in the input C code (cf. Listing 3). This loop ensures that both input images are completely processed when the program ends. The synchronization loop is only required when the total numbers of iterations are not equal for both kernels. This difference occurs if either the two kernels process data in different sizes or the processor data paths that execute the two kernels differ in widths (nways). The parameter (*sync_factor*) represents the factor of such difference, if it exists. Eq. (1) shows the computation of the required number of iterations (*iter1*, *iter2*) of the two different tasks when processing two input images with two different nways (*nway1*, *nway2*). The *sync_factor* is calculated depending on the computed number of iterations as shown in Eq. (2).

$$iter1 = \frac{(width * height)_{image_in1}}{nway1}, \quad iter2 = \frac{(width * height)_{image_in2}}{nway2} \quad (1)$$

$$sync_factor = \begin{cases} iter1/iter2, & iter2 \leq iter1 \\ iter2/iter1, & iter2 > iter1 \end{cases} \quad (2)$$

Fig. 6 is used to exemplify the computation of the synchronization factor. The figure shows two input images of the same height and

width. Each rectangle in the figures corresponds to a pixel in the images. The size of both images is equal to 32 pixels ($height * width$). We assume that only pixels in the same row are allowed to be processed in parallel. Therefore, maximum DLP of these kernels is 8 ($width$). If the vector width of the first cluster, which processes *image_in1*, is 4 then processing of one row of the image requires $width1 = width/nway1$ iterations. Therefore, $width1$ is set as a new width of *image_in1*. The value of $height1$ does not change and equals to $height$. The total number of iterations required to load *image_in1* is 8 ($height1 * width1$). Similar computation can be performed for the second kernel which processes *image_in2*. Since $nway2$ is 2, processing of one row of the image requires $width2 = width/nway2$ iterations. The new width of *image_in2* is set to $width2$. The value of $height2$ does not change and equals to $height$. The total number of iterations required to load *image_in2* is 16 ($height2 * width2$). Since the total numbers of iterations of the two kernels are not equal, when processing of these two images in parallel, the kernel which processes the first image needs to synchronize with the second kernel. Therefore, a third loop is introduced and iteration count ($sync_factor$) is set to 2 ($16/8$).

The new loop introduced creates additional overhead caused by the control operations of the loop. In order to minimize the overhead caused by the control operations of the synchronization loop and to increase the overall throughput of the kernel, *unrolling* is applied to the synchronization loop. Unrolling replicates the statements in the loop body so that the loop actually disappears. In the case of the full loop unrolling, the basic blocks of the 2nd and 3rd level loops are merged into one basic block. This may provide more opportunities for the parallel execution of the operations and may result in an increase of the ILP. On the other hand, if the trip-count of the loop is high, the full loop unrolling may increase the number of instructions. In result, the required program memory capacity also increases. If unrolling takes place, the 2nd level loop becomes vulnerable for *software pipelining* [21]. Software pipelining requires the control-flow free loop body and the independent loop iterations. Software pipelining is an important throughput enhancement technique used when scheduling the application code for the execution on parallel architectures. With software pipelining, an increased utilization of parallel resources is achieved by overlapping the execution of multiple iterations of a loop body. However, software pipelining is not always beneficial. For instance, when the trip count of the software pipelined loop is smaller than the number of copies of the loop body, the software pipelining is not beneficial anymore. The prolog and epilog code introduce extra operations which are not actually needed. Since the compiler is not able to evaluate the usefulness of such optimizations, the **profile-guided optimization** mechanism is used to assist the compiler when taking such decisions. Compiler directives (`#pragma unroll`, `#pragma pipeline`) are used to suggest the compiler that the associated loop is a good candidate for unrolling and software pipelining.

4.2.6. Compilation, simulation and estimation

The retargetable compiler compiles the synchronized version of the C code for the target ASIP in order to generate the scheduled

Fig. 6. Processing of two input images with two different nways.

assembly code. The scheduler reports the average ILP and the total number of instructions of the compiled and scheduled kernel code. Moreover, it reports the *initiation interval (II)* of the software pipelined loops. The II of a software pipelined loop is the distance, in cycles, between the start of two consecutive loop iterations. A host compiler is also used to compile the host code. The cycle-accurate simulation of the mapped code is carried out in order to collect the activity counts for the various components of the target ASIP during the simulated execution of the program. Simulation reports the total cycle count and total number of operations of the program execution. The collected activity counts and the cost of each ASIP component are used to estimate the dynamic energy consumption. Moreover, the analytical models are used to estimate the area and static energy consumption of the program. The estimator reports the energy consumption and area metrics for each run of the program on the target ASIP. Additionally, the estimator can be configured to enable **profile-guided estimation** mechanism. This mechanism is used to imitate the effects of the clock and power gating for the energy estimation. In order to achieve this, the profile-guided estimation takes the maximum achievable DLP ($maxDLP$) of kernels into account during the estimation. Fig. 7 is used to explain this mechanism. When the width (w) of a function unit (FU) and register file (RF) is greater than $maxDLP$, all the units of the FU and RF are not used for data processing. The unused units are marked as *passive units* in the figure. Since these units are subject to be clock/power gated on the actual chip, we imitate the effect of the clock/power gating by neglecting the static and dynamic energy caused by these passive units.

In our total ASIP design flow, the instruction-set architecture (ISA) of the ASIP is also explored. Vector-width exploration tool-flow is able to work in collaboration with the ISA exploration tool. More detailed information on our ISA exploration can be found in [22,23].

5. Experimental evaluation

This section demonstrates the applicability of our method and discusses the experimental results. Experiments focus on the vector-width exploration phase that accepts as its inputs: an initial coarse ASIP architecture to be explored, vector width set to be considered and restructured C code corresponding to the initial architecture.

5.1. Experiment setup

For the experimental research the **kernels** listed in Table 2 are used. The F2T kernel performs 2-tap filtering on two vertical successive pixels of an input image. It creates a blurred output image. Down-sampling kernel (DownS_VH) performs vertical and horizontal down sampling on four neighboring pixels of an input image. It produces a down-scaled output image. Computational

Fig. 7. Profile-guided mechanism allows to neglect the unused units for the dynamic and static energy estimation.

Table 2
Kernels used for exploration.

Kernels	maxDLP	Input (height × width)
F2T_1	32	64 × 32
F2T_2	64	64 × 64
DownS_VH1	64	64 × 128
DownS_VH2	128	64 × 256

intensity of the F2T and down-sampling kernels are different. The F2T kernel performs one addition and one shift operation, while the down-sampling performs three additions and three shift operations. Moreover, the down-sampling kernel requires data reorganization on its packed vector data before it applies the actual processing on pixels. This adds another two data shuffling operations. Therefore, the down-sampling is more compute-intensive than the F2T filter. The table also provides maximum achievable DLPs (*maxDLP*) of each kernel. The value of *maxDLP* is limited by the maximum number of pixels that can be processed in parallel. The restructured input C code of the kernels corresponds to column-wise vectorization. Therefore, *maxDLP* is limited to the width of the input images for 2-tap filtering kernels. The maximum DLPs of the down-sampling kernels are equal to the half of the width of the input image.

Table 3 shows the selected initial coarse processors to be used as base processors for exploration. The eva3 processor has three issue slots (IS), namely one scalar and two vector slots. The scalar IS controls the execution of a kernel (e.g. address computation, loop-flow control) and the vector IS realizes the actual computation (loop body). The vector ISs are connected to their corresponding local vector memories (VM). The eva5 has one scalar IS and four vector ISs with corresponding four local VMs.

As listed in Table 2, two versions of the F2T and DownS_VH kernels are used. The F2T_1 and F2T_2 constitute the F2T kernel set, while the DownS_VH1, DownS_VH2 are in the DownS_VH kernel set. The kernels in each set perform the same computation, but they exercise images with different *maxDLP*. In this way, it is aimed to demonstrate the relation between the vector width change of a processor and *maxDLP* of a particular kernel. Therefore, exploration is carried out separately for each of the two kernel sets. After each exploration, correctness of the produced image is validated against the original reference image. The dimensions of the input images are set to small numbers in order to avoid an excessively long

simulation time. During all reported experiments basic compiler optimizations are applied.

5.2. Experiments and results

First of all, the **sequential execution** of the kernels on the initial coarse processor is carried out. The sequential execution corresponds to the execution of the non-merged versions of kernels. In other words, input images are processed one after the other. Fig. 8a represents one of the sequential orderings of the kernels. The sequential execution provides the initial results to be used as a reference base for assessing the results from the parallelized (merged) versions of the kernels. The initial processor eva3 is used for the sequential execution of the kernels. The eva3 has 2 vector ISs dedicated to execute the kernels. In our experiments, the same number of resources, 2 ISs, are allocated for execution of each kernel. Moreover, the eva3 processor has two VMs. This allows us to map the input and output data on different VMs in order to have the parallel access to the memories. Before presenting the results of the sequential execution, we will show the importance of the profile-guided software optimizations. In order to demonstrate this, the software pipelined and non-software pipelined versions of DownS_VH1 kernel (cf. Listing 4) is executed for seven configurations of the vector width, between 2 and 128. Software pipelining is applied to the inner-most loop. Table 4 reports the total number of operations, total number of instructions, average ILP, II, total number of cycles and dynamic energy consumption values for both versions. Table 4 shows that the software pipelined version of the code outperforms the non-software pipelined version regarding the energy consumption and performance for P0–P3 designs.

If we take a close look at the results from P0, we observe that although the numbers of operations are almost the same for both versions of the code, the difference in the energy consumption and cycle count between the two versions are significant. Software pipelining increases the ILP and, as a consequence, reduces cycle count. Moreover, increase of the ILP results in a more compact code and, consequently, reduces the number of accesses to the program memory. This has a significant impact on the energy consumption. Fig. 9 shows the source of the energy consumption difference of two versions of the code executed for P0.

It is shown that 75% of the energy is consumed by the program memory and decoder. Since energy consumption of the interconnect and clock tree is proportional to the cycle count, they contribute 24% of the energy consumption. The energy consumption value is computed by taking the profile-guided estimation into account during the estimation. On the other hand, software pipelining does not perform well for some design points such as P4, P5 and P6. This is due to the fact that when the trip count of the software pipelined loop is smaller than the number of copies of the loop body. In such cases, the software pipelining introduces prolog and epilog code

Table 3
The initial coarse processors.

Name	#IS	#VM
eva3	3 (1 scalar + 2 vector)	2
eva5	5 (1 scalar + 4 vector)	4

Fig. 8. Two different scenarios are considered: (a) sequential execution of kernels and (b) parallel execution of kernel sets.

```

vector v1, v2;
const int final_h = height>>1;
const int final_w = width>>1;
for (h2=h=0; h < final_h; h++,h2+=2) {
    #if ENABLE_SWP_L1
    #pragma pipeline
    #endif
    for (w2=w=0; w < final_w; w++, w2+=2) {
        v1 = (image_in[h2][w2] + image_in[h2+1][w2])>>1;
        v2 = (image_in[h2][w2+1] + image_in[h2+1][w2+1])>>1;
        image_out[h][w]=(vec_odd(v1,v2) + vec_even(v1,v2))>>1;
    }
}

```

Listing 4. DownS_VH1 kernel.

Table 4

Results for software pipelined and non-software pipelined executions of DownS_VH1 kernel on eva3 processor with different nways. The highlighted values in bold represent the lowest cycle count and energy consumption values of the same design point with respect to software pipelined or non-software pipelined versions of the code.

Processor	w/Software pipelining						wo/Software pipelining					
	#Oper.	#Instr.	Avg. ILP	II	#Cycles	Energy (nj)	#Oper.	#Instr.	Avg. ILP	II	#Cycles	Energy (nj)
P0[2]	9128	54	2.3	7	3908	305.2	9194	55	1.5	–	6053	439.5
P1[4]	4776	54	2.3	7	2116	196	4842	55	1.5	–	3237	267.2
P2[8]	2600	54	2.1	7	1220	141.5	2666	55	1.5	–	1829	181
P3[16]	1512	54	2	7	772	114.2	1578	55	1.4	–	1125	137.9
P4[32]	1512	54	2	7	772	180.8	1034	55	1.3	–	773	116.3
P5[64]	1386	53	2	7	709	310.4	908	52	1.4	–	646	176.7
P6[128]	1386	53	2	7	709	310.4	908	52	1.4	–	646	176.7

Fig. 9. Source of the energy consumption difference of the two versions of the code executed for P0.

which are not actually needed. Since the compiler is not able to evaluate the usefulness of such optimizations, the profile-guided optimization is used to assist the compiler. Table 5 shows the results of sequential execution separately for each kernel set. The presented results take profile-guided optimization and estimation into account. The best energy and performance values are highlighted for each kernel set. The P4 design point provides the best energy and performance results for the F2T kernel set, as well as, the best energy result for the DownS_VH kernel set. The P6 design point provides the best performance value for the DownS_VH kernel set.

The **parallel execution** of the kernels is carried out on eva5. Fig. 8b illustrates the kernels which are merged to be executed in parallel. The parallel execution corresponds to running of the parallel versions of the kernels (i.e. merged kernels and synchronization loop). Fig. 10 shows the input and output data mappings to the vector memories and task mappings to the clusters for the parallel execution of each kernel set. Each cluster can run one kernel. In other words, when executing the kernel set two images can be

processed at the same time. Clustering allows us to set *nway1* and *nway2* parameters at the cluster level. The exploration was carried out for all the possible vector width configurations, which resulted in 49 different ASIPs. Seven of these configurations correspond to homogeneous ASIPs. In order to build the homogeneous ASIPs, the parameters *nway1* and *nway2* are set to the same value, from vector width of 2–128. The parameters *nway1* and *nway2* are set to different values (e.g. (2, 4), (4, 16)) to create heterogeneous ASIPs. Since a fixed data mapping is considered for the whole exploration, we take all the possible permutations of vector widths into account. It results in 42 different ASIPs with different heterogeneous vector width configurations.

First of all, the **synchronization factor** analysis of both kernel sets is carried out. Fig. 11 presents the synchronization factor analysis of both F2T and DownS_VH kernels for these 49 (P7–P55) ASIPs. The first 7 (P7–P13) ASIPs are homogeneous ones. The remaining 42 (P14–P55) designs correspond to heterogeneous ASIPs. As can be seen from the graph, the *sync_factor* varies between 1 and 64, and it has the same values for both kernels for the most of the design points. The designs which have lower *sync_factor* values are expected to provide better results regarding energy and performance than the designs which suffer from high synchronization factor.

Experiments for the F2T kernels: The first set of experiments corresponds to the vector-width exploration for the F2T kernels. Table 6 presents the results for all homogeneous design points. For the designs (P7–P11) where the *sync_factor* is constant (2), the number of operations decreases with increase of the vector width. The increase of vector width eliminates several operations (e.g. for address computation, control-flow) otherwise required to execute the loop. It also results in a cycle count reduction as the cycle count is proportional to the ILP and the number of operations. For the designs (P12–P13) where the *sync_factor* is 1, the operation counts do not change anymore. This is due to the fact that max-DLPs (32 and 64) of the kernels are lower or equal to the vector widths. Therefore, the vector width increase from 64 to 128 does not improve the performance.

Table 5

Data from the scheduler and simulator for sequential execution of kernel sets. The lowest cycle count and energy consumption values are highlighted in bold.

Processor	F2T_1 & F2T_2						DownS_VH1 & DownS_VH2					
	#Oper.	#Instr.	Avg. ILP	II	#Cycles	Energy (nj)	#Oper.	#Instr.	Avg. ILP	II	#Cycles	Energy (nj)
P0[2]	19,840	77	1.9	3, 3	10,444	712	26,942	96	2.4	7, 7	11,445	895.4
P1[4]	10,768	77	1.8	3, 3	5908	478.5	13,886	96	2.3	7, 7	6096	567.9
P2[8]	6232	77	1.7	3, 3	3640	361.7	7358	96	2.2	7, 7	3381	404.3
P3[16]	3964	77	1.6	3, 3	2506	303.6	4094	96	2.0	7, 7	2037	322.4
P4[32]	2770	79	1.3	–, 3	2130	286.6	2528	96	1.7	–, 7	1527	297.2
P5[64]	2332	83	1	–, –	2258	288.7	1924	94	1.4	–, –	1401	361.2
P6[128]	2332	83	1	–, –	2258	288.7	1798	92	1.4	–, –	1275	489.9

Fig. 10. Input and output data mappings to the vector memories and task mappings to the clusters for the parallel execution of each kernel set. **Fig. 11.** Synchronization factor analysis of F2T and DownS_VH kernels.

The instruction count is also decreased from 69 (P10) to 62 (P11) and 56 (P12). This results from the fact that, when the loop's iteration count equals to 1, the compiler discards all operations related to the loop control. The elimination of such operations may increase the ILP, by breaking dependences, and may decrease the instruction count. Another metric that affects performance is the initiation interval (II) of a software pipelined loop. The II is limited by the available resources and inter-iteration dependences of a loop. Since II corresponds to the minimum cycle required to initiate the loop iterations, the lower it is, the better for performance. Since the profile-guided optimization is considered for the exploration, software pipelining is not applied for some design points, such as P11, P12 and P13. The corresponding II values of these design points are marked with (–) sign.

Table 6 also presents results for seven ASIPs which are selected from among the 42 heterogeneous design points. The sync_factor

increase from 2 to 8 (P15–P17) leads to the increase of the total number of operations. Moreover, it leads to the increase of the number of instructions and II due to the loop unrolling. In consequence, performance gets worse. When sync_factor equals to 1 (P28–P55), the increase of the vector width reduces the operation counts as expected, until the maxDLPs of the kernels are lower or equal to the vector widths. The average ILP of the homogeneous designs is 2.17, while this value is only 2.06 for the heterogeneous designs.

Experiments for the DownS_VH kernels: The second set of experiments corresponds to the vector-width exploration for the down-sampling kernels. Table 6 presents the results for all homogeneous design points. For the designs (P7–P11) where the sync_factor is constant (2), the number of operations decreases with increase of the vector width as expected. Since maxDLPs of the kernels are 64 and 128, we do not see the limitation due to

Table 6

Data from the scheduler and simulator for parallel execution of kernel sets. The lowest cycle counts are highlighted in bold.

Processor	Sync	#Oper.	#Instr.	Avg. ILP	II	#Cycles	Sync	#Oper.	#Instr.	Avg. ILP	II	#Cycles
<i>F2T – homogeneous</i>							<i>DownS_VH – homogeneous</i>					
P7[2,2]	2	24,236	69	3.3	6	7282	2	27,183	89	4.0	12	6760
P8[4,4]	2	12,644	69	3.0	6	4258	2	13,871	89	3.8	12	3688
P9[8,8]	2	6848	69	2.5	6	2746	2	7215	89	3.4	12	2152
P10[16,16]	2	5399	69	2.3	6	2368	2	4017	72	2.3	–	1769
P11[32,32]	2	2693	62	1.5	–	1803	2	2353	72	2.0	–	1193
P12[64,64]	1	1688	56	1.3	–	1301	2	2259	70	2.1	–	1098
P13[128,128]	1	1688	56	1.3	–	1301	1	1493	62	1.9	–	780
<i>F2T – heterogeneous</i>							<i>DownS_VH – heterogeneous</i>					
P15[2,8]	2	24,236	69	3.3	6	7282	2	27,247	81	3.5	14	7784
P16[2,16]	4	34507	74	2.6	12	13,267	4	43,634	110	3	27	14,536
P17[2,32]	8	54671	92	2.4	21	22,339	8	76,474	121	2.2	–	34,921
P28[8,16]	1	4139	54	2.6	3	1612	1	5646	89	3.2	7	1768
P35[16,32]	1	3320	54	2.3	3	1423	1	2704	63	2.1	–	1289
P42[32,64]	1	1814	57	1.3	–	1364	1	1648	63	1.8	–	905
P55[128,64]	1	1814	57	1.3	–	1364	2	2259	71	2	–	1130

Fig. 12. Dynamic energy consumption of DownS_VH, F2T kernels and total of them for different ASIP designs. **Fig. 13.** Cycle counts of DownS_VH, F2T kernels and total of them for different ASIP designs.

the DLP, as it was the case for F2T kernels. Therefore, the number of operations is decreased and performance is improved from P7 to P12. Table 6 also presents results for seven heterogeneous design points. As it can be observed from the table, the number of operations are reduced from the P28 to P42. However, the number of operations are increased for the design P55. This is due to the increase of the sync_factor from 1 to 2. Therefore, cycle count is also increased. The profile-guided optimization is also considered for this exploration, and therefore, the software pipelining is not applied to some design points. The average ILP of the homogeneous designs is 2.78, while it is only 2.36 for the heterogeneous designs.

Evaluation of the ASIP designs for all kernels: The goal of the heterogeneous vector-width exploration is to find the best ASIP design which executes the four kernels effectively and efficiently. Performance is an important metric, but it is not sufficient to assess an ASIP design quality. Therefore, both the energy and performance are used to evaluate the ASIP designs. The activity counts and costs of each ASIP component are used to estimate the dynamic energy consumption. The energy estimation considers all ASIP components, including memories, ISs, register files and interconnects. Fig. 12 presents the dynamic energy consumption of DownS_VH and F2T kernels for all homogeneous and

heterogeneous design points. Moreover, the total cycle count of ASIP designs executing all kernels are presented in Fig. 13. In result, the ASIP design points P13[128, 128] and P49[64, 128] are selected as they both provide the best performance and dynamic energy consumption among the homogeneous and heterogeneous designs. Based on the experiments, the following conclusions can be drawn:

- The dynamic energy consumption and performance are improved proportionally to the vector width increase, but inversely proportionally to the increase of the sync factor
- The dynamic energy consumption is proportionally to the decrease of the number of operations. However, for some configurations, where sync factor is high, loop unrolling increases the width of and number of accesses to the program memory, resulting in an increase of the dynamic energy consumption (e.g. P14–P19)

Moreover, many applications do not require the peak performance from the processor. For those applications, the frequency and voltage scaling can be applied in order to further save the active energy and to reduce the power consumption ([24,25]).

5.3. Discussion and future work

As it can be observed from the discussed experiments, the average ILP values for homogeneous designs are higher than for the heterogeneous designs. This is mainly due to the extra limitations imposed by having issue slots with two different widths in the ASIP data-path. For heterogeneous designs, the scheduler has less freedom regarding the resource allocation. This creates an advantage for homogeneous designs. Moreover, since scheduler can map an operation of a task on any issue slot, even though some particular resources are meant to be used only by another task, the activity count of some ASIP components may be miscalculated. Therefore, we forced the scheduler to map operations of a task on certain resources in order to apply the profile-guided estimation for homogeneous designs. This technique is applied to the homogeneous designs P12 and P13 for the mappings where the vector widths of the ASIPs are greater than maximum DLPs of the kernels. In this work, we used profile-guided optimization for deciding the application of software pipelining. A similar analysis is however also required for the loop unrolling, because loop unrolling may not be beneficial for some designs. Furthermore, since we have a single sequencer in an ASIP, some design points suffer from the high synchronization overhead. These design points can benefit from the heterogeneous multi-ASIP system implementation instead of the single heterogeneous ASIP. Furthermore, a corresponding multi-core system of the P42 design can be built in order to compare the single-core and multi-core solutions regarding the performance, area and energy consumption.

6. Conclusion

In this paper, we proposed and discussed a novel ASIP design space exploration method that aims at exploring and deciding the heterogeneous application-specific vector widths for a VLIV ASIP. We also demonstrated application of our method to a set of selected kernels. We implemented our new heterogeneous exploration method as an EDA-tool, and used this tool to perform a set of ASIP synthesis experiments. The experimental results demonstrated that our new method is able to efficiently exploit the heterogeneous vector widths.

Acknowledgments

This work was performed as part of the European project ASAM [7] that has been partially funded by ARTEMIS Joint Undertaking, Grant No. 100265.

References

- [1] Software Programmable Media Processor, Movidius Myriad SoC, Project website, 2011. <<http://movidius.com/>>.
- [2] Programmable Image Signal Processor, Intel Mobile SoCs (Medfield, Clover Trail), Project website, 2012. <<http://www.intel.com/>>.
- [3] Y. Park, S. Seo, H. Park, H.K. Cho, S. Mahlke, Simd defragmenter: efficient ilp realization on data-parallel architectures, SIGARCH Comput. Archit. News 40 (2012) 363–374.
- [4] Z. Hu, A. Buyuktosunoglu, V. Srinivasan, V. Zyuban, H. Jacobson, P. Bose, Microarchitectural techniques for power gating of execution units, in: Proceedings of the 2004 International Symposium on Low Power Electronics and Design, ISLPED '04, ACM, New York, NY, USA, 2004, pp. 32–37. <http://dx.doi.org/10.1145/1013235.1013249>.
- [5] M. Woh, S. Seo, S. Mahlke, T. Mudge, C. Chakrabarti, K. Flautner, Anysp: anytime anywhere anyway signal processing, in: Proceedings of the 36th Annual International Symposium on Computer Architecture, ISCA '09, ACM, New York, NY, USA, 2009, pp. 128–139. <http://dx.doi.org/10.1145/1555754.1555773>.
- [6] L. Jóźwiak, Y. Jan, Design of massively parallel hardware multi-processors for highly-demanding embedded applications, J. Microprocess. Microsyst. (2013).
- [7] L. Jóźwiak, M. Lindwer, R. Corvino, P. Meloni, L. Micconi, J. Madsen, E. Diken, D. Gangadharan, R. Jordans, S. Pomata, P. Pop, G. Tuveri, L. Raffo, Asam: automatic architecture synthesis and application mapping, J. Microprocess. Microsyst. (2013).
- [8] Y. Lin, H. Lee, M. Woh, Y. Harel, S. Mahlke, T. Mudge, C. Chakrabarti, K. Flautner, Soda: a low-power architecture for software radio, SIGARCH Comput. Archit. News 34 (2006) 89–101.
- [9] J.H. Ahn, W.J. Dally, B. Khailany, U.J. Kapasi, A. Das, Evaluating the imagine stream architecture, in: Proceedings of the 31st Annual International Symposium on Computer Architecture, ISCA '04, IEEE Computer Society, Washington, DC, USA, 2004, p. 14. <<http://dl.acm.org/citation.cfm?id=998680.1006734>>.
- [10] K. van Berkel, F. Heinle, P.P.E. Meuwissen, K. Moerman, M. Weiss, Vector processing as an enabler for software-defined radio in handheld devices, EURASIP J. Appl. Signal Process. 2005 (2005) 2613–2625.
- [11] Y. Park, J. Jong, K. Hyunchul, P.S. Mahlke, Libra: tailoring SIMD execution using heterogeneous hardware and dynamic configurability, in: Proceedings of the 2012 IEEE ACM 45th International Symposium on Microarchitecture (MICRO-45), 2012.
- [12] G. Dasika, M. Woh, S. Seo, N. Clark, T. Mudge, S. Mahlke, Mighty morphing power-simd, in: Proceedings of the 2010 International Conference on Compilers, Architectures and Synthesis for Embedded Systems, CASES'10, ACM, New York, NY, USA, 2010, pp. 67–76. <http://dx.doi.org/10.1145/1878921.1878934>.
- [13] R. Krashinsky, C. Batten, M. Hampton, S. Gerding, B. Pharris, J. Casper, K. Asanovic, The vector thread architecture, SIGARCH Comput. Archit. News 32 (2004) 52.
- [14] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. Huh, D. Burger, S.W. Keckler, C.R. Moore, Exploiting ilp, tlp, and dlp with the polymorphous trips architecture, in: Proceedings of the 30th Annual International Symposium on Computer Architecture, ISCA '03, ACM, New York, NY, USA, 2003, pp. 422–433. <http://dx.doi.org/10.1145/859618.859667>.
- [15] E. Diken, R. Corvino, L. Jóźwiak, Rapid and accurate energy estimation of vector processing in vliv asips, in: ECyPS 2013 – EUROMICRO/IEEE Workshop on Embedded and Cyber-Physical Systems, Budva, Montenegro, 2013, pp. 33–37 (<http://dx.doi.org/10.1109/MECO.2013.6601350>).
- [16] P.R. Panda, F. Cathoor, N.D. Dutt, K. Danckaert, E. Brockmeyer, C. Kulkarni, A. Vandercappelle, P.G. Kjeldsberg, Data and memory optimization techniques for embedded systems, ACM Trans. Des. Autom. Electron. Syst. (TODAES) 6 (2001) 149–206.
- [17] K. Danckaert, K. Masselos, F. Cathoor, H.J. DeMan, C. Goutis, Strategy for power-efficient design of parallel systems, IEEE Trans. Very Large Scale Int. (VLSI) Syst. 7 (1999) 258–265.
- [18] R. Corvino, A. Gamatié, M. Geilen, L. Jóźwiak, Design space exploration in application-specific hardware synthesis for multiple communicating nested loops, in: SAMOS XII – 12th International Conference on Embedded Computer Systems, Samos, Greece, 2012, pp. 1–8. <<http://www.samos-conference.com/>> (<http://dx.doi.org/10.1109/SAMOS.2012.6404166>).
- [19] R. Corvino, E. Diken, A. Gamatié, L. Jóźwiak, Transformation based exploration of data parallel architecture for customizable hardware: a JPEG encoder case study, in: DSD 2012 – 15th Euromicro Conference on Digital System Design, Cesme, Izmir, Turkey, 2012, pp. 774–781. <http://www.univ-valenciennes.fr/congres/dsd2012/> (<http://dx.doi.org/10.1109/DSD.2012.133>).
- [20] C. Glitia, P. Dumont, P. Boulet, Array-ol with delays, a domain specific specification language for multidimensional intensive signal processing, Multidim. Syst. Signal Process. 21 (2010) 105–131.

- [21] M. Lam, Software pipelining: an effective scheduling technique for vliw machines, *SIGPLAN Not.* 23 (1988) 318–328.
- [22] R. Jordans, R. Corvino, L. Jóźwiak, H. Corporaal, Instruction-set architecture exploration strategies for deeply clustered vliw asips, in: *ECyPS 2013 – EUROMICRO/IEEE Workshop on Embedded and Cyber-Physical Systems*, Budva, Montenegro, 2013, pp. 38–41. <<http://www.embeddedcomputing.me/>>.
- [23] R. Jordans, R. Corvino, H. Corporaal, L. Jóźwiak, Exploring processor parallelism: estimation methods and optimization strategies, in: *DDECS 2013 – 16th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, Karlovy Vary, Czech Republic, 2013, pp. 18–23. <<http://www.fit.vutbr.cz/events/ddecs2013/>> (received best paper award).
- [24] Y. He, Y. Pu, R. Kleihorst, Z. Ye, A.A. Abbo, S.M. Londono, H. Corporaal, Xet al-Pro: an ultra-low energy and high throughput SIMD processor, in: *Proceedings of the 47th Design Automation Conference, DAC '10*, ACM, New York, NY, USA, 2010, pp. 543–548. <http://dx.doi.org/10.1145/1837274.1837409>.
- [25] Y. Pu, Y. He, Z. Ye, S.M. Londono, A.A. Abbo, R.P. Kleihorst, H. Corporaal, From Xetal-II to Xetal-Pro: on the road toward an ultralow-energy and high-throughput SIMD processor, *IEEE Trans. Circ. Syst. Video Techn.* 21 (2011) 472–484.

Erkan Diken is a PhD student in the Electronic Systems Group of the Electrical Engineering Department at the Eindhoven University of Technology, The Netherlands. He received the MSc degree in Embedded Systems Design from the Advanced Learning and Research Institute in collaboration with ETH Zurich and Politecnico di Milano, Switzerland, in 2010, and the BSc degree in Computer Engineering from the Gebze Institute of Technology, Turkey, in 2008. His research interests include automatic instruction-set architecture synthesis and application mapping on heterogeneous multi-processor embedded systems. His research focuses on efficient realization and exploitation of data-level parallelism (DLP) on VLIW/SIMD architectures and MPSoCs. He is a member of the IEEE and the HIPEAC.

Roel Jordans (M'13) received the MSc degree in field of Electrical Engineering from Eindhoven University of Technology in 2009. He worked within the PreMaDoNA project on the MAMPS tool flow as a researcher afterwards. As of September 2010 he continues his education as a PhD student at the Electronic Systems group of the Department of Electrical Engineering. His research interest include VLIW architectures and the automatic synthesis of application specific instruction-set processors.

Rosilde Corvino is a research scientist and project manager in the Electronic Systems group at the Department of Electrical Engineering at Eindhoven Technical University, The Netherlands. She is currently a project manager and work-package responsible in the European project ASAM – Automatic Architecture Synthesis and Application Mapping. In 2010, she was a post-doctoral research fellow in DaRT team at INRIA Lille Nord Europe and was involved in Gaspard2 project. She earned her PhD in 2009, from University Joseph Fourier of Grenoble, in micro and nanoelectronics. In 2005/06, she obtained a double Italian and French M.Sc in electronic engineer. Her research interests involve design space exploration, parallelization techniques, data transfer and storage mechanisms, high level synthesis, application specific processor design for data intensive applications. She is author of numerous research papers and a book chapter. She serves on program committees of DSD and ISQED.

Lech Jozwiak is an Associate Professor, Head of the Section of Digital Circuits and Formal design Methods, at the Faculty of Electrical Engineering, Eindhoven University of Technology, The Netherlands. He is an author of a new information driven approach to digital circuits synthesis, theories of information relationships and measures and general decomposition of discrete relations, and methodology of quality driven design that have a considerable practical importance. He is also a creator of a number of practical products in the fields of application-specific embedded systems and EDA tools. His research interests include system, circuit, information theory, artificial intelligence, embedded systems, re-configurable and parallel computing, dependable computing, multi-objective circuit and system optimization, and system analysis and validation. He is the author of more than 150 journal and conference papers, some book chapters, and several tutorials at international conferences and summer schools. He is an Editor of "Microprocessors and Microsystems", "Journal of Systems Architecture" and "International Journal of High Performance Systems Architecture". He is a Director of EUROMICRO; co-founder and Steering Committee Chair of the EUROMICRO Symposium on Digital System Design; Advisory Committee and Organizing Committee member in the IEEE International Symposium on Quality Electronic Design; and program committee member of many other conferences. He is an advisor and consultant to the industry, Ministry of Economy and Commission of the European Communities. He recently advised the European Commission in relation to Embedded and High-performance Computing Systems for the purpose of the Framework Program 7 preparation. In 2008 he was a recipient of the Honorary Fellow Award of the International Society of Quality Electronic Design for "Outstanding Achievements and Contributions to Quality of Electronic Design". His biography is listed in "The Roll of Honour of the Polish Science" of the Polish State Committee for Scientific Research and in Marquis "Who is Who in the World" and "Who is Who in Science and Technology".

Henk Corporaal (M'09) received the M.S. degree in theoretical physics from the University of Groningen, Groningen, The Netherlands, and the Ph.D. degree in electrical engineering, in the area of computer architecture, from the Delft University of Technology, Delft, The Netherlands. He has been teaching at several schools for higher education. He has been an Associate Professor with the Delft University of Technology in the field of computer architecture and code generation. He was a Joint Professor with the National University of Singapore, Singapore, and was the Scientific Director of the joint NUS-TUE Design Technology Institute. He was also the Department Head and Chief Scientist with the Design Technology for Integrated Information and Communication Systems Division, IMEC, Leuven, Belgium. Currently, he is a Professor of embedded system architectures with the Eindhoven University of Technology, Eindhoven, The Netherlands. He has co-authored over 250 journal and conference papers in the (multi)processor architecture and embedded system design area. Furthermore, he invented a new class of very long instruction word architectures, the Transport Triggered Architectures, which is used in several commercial products and by many research groups. His current research interests include single and multiprocessor architectures and the predictable design of soft and hard real-time embedded systems.

Felipe Augusto Chies is currently a hardware designer at IMS Soluções de Energia Ltda, Brazil. In 2013, he obtained a double degree in computer engineering from Universidade Federal do Rio Grande do Sul (Brazil) and Grenoble INP (France). His Dipl.-Ing was focused on embedded systems and the subject of his thesis was about design space exploration applied to ASIPs.