Construction and Exploitation of VLIW ASIPs with Multiple Vector-Widths

Erkan Diken, Roel Jordans, Lech Jóźwiak and Henk Corporaal
Eindhoven University of Technology
Den Dolech 2, 5612 AZ, Eindhoven, The Netherlands
Email: {e.diken, r.jordans, l.jozwiak, h.corporaal}@tue.nl

Abstract—**Many applications in important domains, such as communication, multimedia, etc. 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 and performance inefficiency. Therefore, an adequate more sophisticated exploitation of DLP is highly relevant. This paper studies the construction and exploitation of VLIW ASIPs with multiple vector widths.**

I. Introduction

Computing platforms embedded in various modern devices are often required to satisfy high performance demands when processing data intensive applications. Moreover, embedded systems of a portable equipment must also ensure a low energy consumption, due to the limited battery life. The low energy consumption and high performance can be achieved through the usage of application specific instruction-set processors (ASIPs). Since ASIPs are programmable, ASIPs can be re-used for different application versions or even for different applications in the same or similar domain. 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 the exploitation of the intrinsic parallelism of a given application. In the context of single-instruction multiple-data (SIMD)/very long instruction word (VLIW) architectures, one of the important form of parallelism to be exploited is the data-level parallelism (DLP). At the instruction level, DLP refers to independent occurrences of the same operation that can be executed on different data sub-sets. DLP can be 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 natural DLP of the application. All the other cases result in a loss of either efficiency or effectiveness. Former research on application analysis ([3], [4], [5]) has shown that different application kernels in important domains, such as communications (e.g. FFT/IFFT, STBC, LDPC), multimedia (e.g. MPEG4 audio decoding (AAC), 3D graphics rendering, H.264) etc. have different maximum natural DLPs. Table I 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, and specifically adequate construction and exploitation of ASIPs with multiple vector widths, is highly relevant. In this paper, we study the construction and exploitation of VLIW ASIPs with multiple vector widths.

TABLE I: Maximum natural DLP analysis of some multimedia and communication kernels and applications.

Kernel / application	DLP
FFT/IFFT (fast/inverse-fast fourier transform) [4], AAC [3]	1024
STBC (space time block coding) [4]	4
LDPC (low-density parity check) [4]	96
Deblocking filter, inverse transform, motion compensation ([4])	8
Intra-prediction [4]	16
3D graphics rendering [3]	128

The paper is structured as follows. In the next section, the motivation of the work presented in this paper is explained. The target ASIP architecture model used and its form with multiple vector widths are explained in Section III. Section IV briefly explains our new method and the related design automation flow. Section V experimentally demonstrates the applicability of our method and discusses the experimental results. Related research is discussed in Section VI. Finally, Section VII concludes the paper.

II. Motivation

Deciding the architectural ASIP parameters related to vectorization should not only consider the DLPs of involved tasks (kernels), but also should take the task-level parallelism (TLP) and task mapping into account. Figure 1 is used to explain the motivation of the work presented in this paper. For a given set of tasks (T1, T2, T3, T4), each exhibiting a DLP, and prototype processors (P1, P2), the following three possible cases can be considered. In the case of Figure 1-(a), one processor is allocated to run all the tasks in a sequential manner. In this case, the processor prototype needs to be configured, by considering all the tasks, in order to satisfy the required performance, energy consumption and area. Vectorization related architectural parameters (e.g. vector width ($w1$)) also needs to be chosen appropriately by considering the DLPs of tasks and required design metrics. For instance, $w1$ can be set to the minimum DLP (8 for T3) or maximum DLP (64 for T4) of

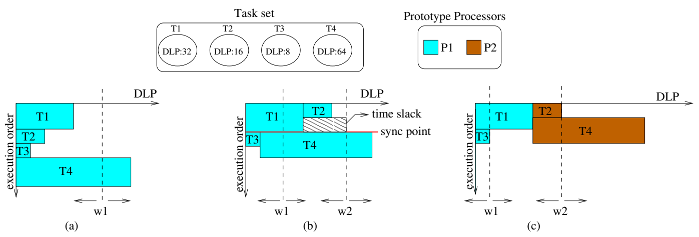

Fig. 1: (a) Sequential execution of tasks with different DLPs on a single processor system (b) parallel execution of tasks on a single processor system (c) parallel execution of tasks on a multi-processor system

given tasks or any value in between. While executing these four kernels, each selected $w1$ value will result in different energy consumption and performance.

The Figure 1-(b) exemplifies another case in which tasks are executed in a parallel and sequential order on a single processor. As it can be seen from the Figure 1-(b), the task pairs (T1, T2) and (T3, T4) are arranged to be executed in sequential order, while the tasks in each pair run in parallel to each other. This arrangement imposes mapping of T1 and T3 to the same resources, while T2 and T4 exploit different resources in the processor. In this case, the hardware units which execute the task pair (T1, T3) can be specialized for these tasks. Similarly, the hardware units which execute the tasks T2 and T4 can be specialized only for these tasks. This enables us to have two different vector widths ($w1$, $w2$) in a single processor. In this paper, we study the construction and exploitation of such ASIP processors which include two or more vector widths.

Since there is a single control unit in a single processor, efficient exploitation of the parallel hardware structure of the architecture requires an adequate application code restructuring. Listing 1 gives an example of such restructured code. As it can be observed from the code, the tasks T1 and T2 are placed in the body of one nested loop in order to have a parallel execution of these tasks (computations carried out by the tasks are not shown for the sake of simplicity). It is assumed that there is no dependency between these tasks. The code represents processing of two different images ($image\_in1$ and $image\_in2$) of certain heights and widths. The control unit of the processor controls the execution of the code (e.g. address computation, loop-flow control) and the data-path of the processor realizes the actual computation (loop body). Having architecture with different vector widths brings some new challenges. Since the target ASIP architecture is a VLIW machine capable of executing parallel software with a single thread of control, in some circumstances, synchronization of tasks is required. Due to the synchronization, time slacks may be involved between tasks, as exemplified in Figure 1-(b). Synchronization of the kernels has to be handled explicitly by introducing an additional synchronization loop, as shown in Listing 2. This loop ensures that the both input data sets are completely processed when the program ends. Synchronization loop is only required when the total numbers of iterations are not equal for both kernels. This difference occurs if either two kernels process data in portions of different sizes or the data paths that execute two kernels differ in widths. The parameter

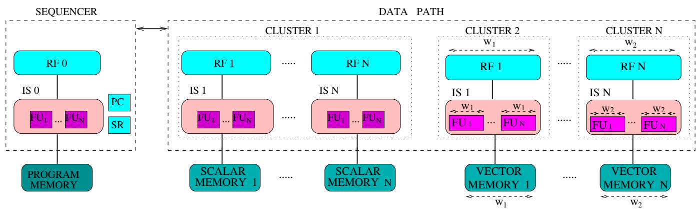

Fig. 2: Generic ASIP architecture template

(*sync_factor*) represents the factor of such difference. Equations 1 and 2 show the calculation of the sync_factor.

$$iter1 = \frac{(width*height)_{image\_in1}}{w1}, iter2 = \frac{(width*height)_{image\_in2}}{w2} \quad (1)$$

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

Figure 1-(c) demonstrates yet another case involving two processors. It corresponds to a multi-processor sub-system. The task arrangement and ordering are the same as in the case (b). However, since parallel execution of the tasks is realized with two independent threads of control, no synchronization is needed in this case. As it can be seen from the Figure 1-(c), the task T4 can be initiated just after the task T2 finishes. Therefore, no time slack is involved. Moreover, vector widths of each processor can be configured to be specific for the executed kernel sets. The multi-processor system case is not further discussed in this paper, as it is limited to construction of a single ASIP.

Listing 1 An example of restructured C code

```
int image_in1[height][width];
int 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 2 Merged kernels with synchronization loop

```
for(ht = 0; ht < height; ht++) {
  for(wd = 0; wd < width; wd++) {
    T1: image_out1[ht,wd] = image_in1[ht,wd];
     for(k = 0; k < sync_factor; k++) {
      T2: image_out2[ht, wd, k, sync_factor]=
          image_in2[ht, wd, k, sync_factor];
    }
  }
}
```

III. Architecture Model

Figure 2 depicts a simplified view of the corresponding generic ASIP architecture template. It includes a VLIW data-path controlled by a sequencer (control unit) that uses status (SR) and control registers (PC), and executes a program stored

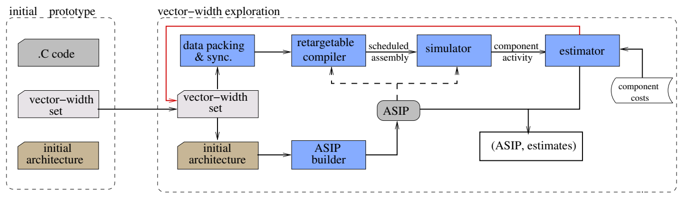

Fig. 3: Tool-flow for exploring the (multiple) vector widths

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 (RF). The function units perform computation operations on intermediate data stored in the register files. Only function units in different issue slots can be triggered and execute parts of an application in parallel. Local memories, collaborating with particular issue slots, enable scalar access for the scalar slots and vector or block access for the vector slots. The numbers and kinds of function units, issue slots, register files, memories, interfaces, etc. can be freely selected by specifying component configuration parameters. The targeted generic ASIP architecture allows us to construct architecture instances involving multiple vector widths. In Figure 2, *Cluster 1* and *Cluster N* correspond to two components of the VLIW data-path with multiple vector widths. The execution units in each cluster can support different functionalities (FU_1 and FU_N) and have different widths (w_1 and w_2).

IV. Vector-Width Exploration Method

The method focuses on finding the best possible set of vector widths for a given parallelized version of the application C code and coarse ASIP architecture. Figure 3 presents the tool-flow used for the vector-widths exploration. The vector width set to be explored and the directories that include application and processor description files are provided as inputs. The exploration includes the ASIP building, data packing/storing and synchronization, code compilation, simulation and estimation steps for each vector width to be explored. A cycle-accurate simulation of the C program is carried out in order to estimate the performance value. We implemented our new vector width exploration method as an EDA-tool, and used this tool to perform a set of ASIP synthesis experiments. The results of these experiments are discussed in the following section.

V. Experimental Evaluation

The kernels listed in Table II are used for the exploration. The F2T kernel performs a 2-tap filtering on two vertical successive pixels of an input image.The down-sampling kernel (DownS) performs vertical and horizontal down sampling on four neighbouring pixels of an input image. Table II lists two versions of the F2T and DownS kernel sets. The kernels in each set perform the same computation, but they exercise images with different maximum DLPs ($maxDLP$). In this way, it is aimed to demonstrate the relation between the $maxDLP$ of a particular kernel and vector width change of a processor. Therefore, exploration is carried out separately for each of the two kernel sets.

TABLE II: Kernels used for exploration

Set name	Kernels	$maxDLP$	Input image (height x width)
F2T	F2T_1	32	64 x 32
	F2T_2	64	64 x 64
DownS	DownS_1	64	64 x 128
	DownS_2	128	64 x 256

A base processor, which has one scalar IS and four vector ISs with corresponding four local vector memories (VM), is selected as the initial coarse processor. The pairs of ISs, VMs and corresponding register files are grouped into one cluster. This results in two clusters. The **parallel execution** of the kernels is carried out on the base processor. The parallel execution corresponds to running of the parallel versions (i.e. merged kernels + synchronization loop) of the kernels. In this way, two images can be processed in parallel. Clustering allows us to set $w1$ and $w2$ parameters at the cluster level. The exploration was carried out for all the possible vector width configurations, which resulted in 49 different ASIPs, from vector width of 2 to 128. First of all, the **synchronization factor** analysis of both kernel sets is carried out. Figure 4 presents the synchronization factor analysis of both F2T and DownS kernels for these 49 (P7-P55) 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 performance results than the designs which suffer from a high synchronization factor.

The first set of experiments corresponds to the vector-width exploration for the F2T kernels. Figure 5 presents the cycle counts for all 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 the vector width eliminates several operations (e.g. for address computation, control-flow) otherwise required to execute the loop. This results in a cycle count reduction. 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 maxDLPs (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. The sync_factor increase from P14 to P19 leads to the increase of the total number of operations. In consequence, performance gets worse. When sync_factor equals to 1 (e.g. 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 rest of the design points follow a similar trend.

The second set of experiments corresponds to the vector-width exploration for the DownS kernels. Figure 5 also presents the cycle counts for the DownS kernels. It follows similar trend as F2T depending on the values of vector width and sync_factor. However, since maxDLPs of the down-sampling kernels are 64 and 128, we do not see the limitation due to 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.

The goal of the vector-width exploration is to find the best ASIP design which executes the four kernels effectively and efficiently. The total cycle count of the ASIP designs executing

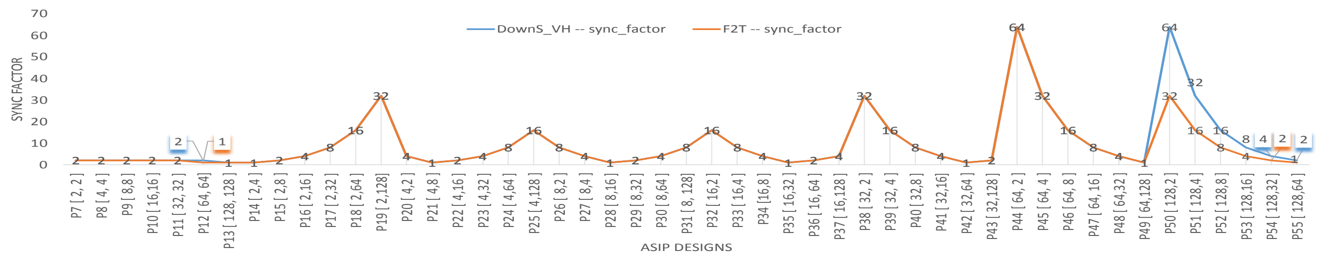

Fig. 4: Synchronization factor analysis of F2T and DownS kernels

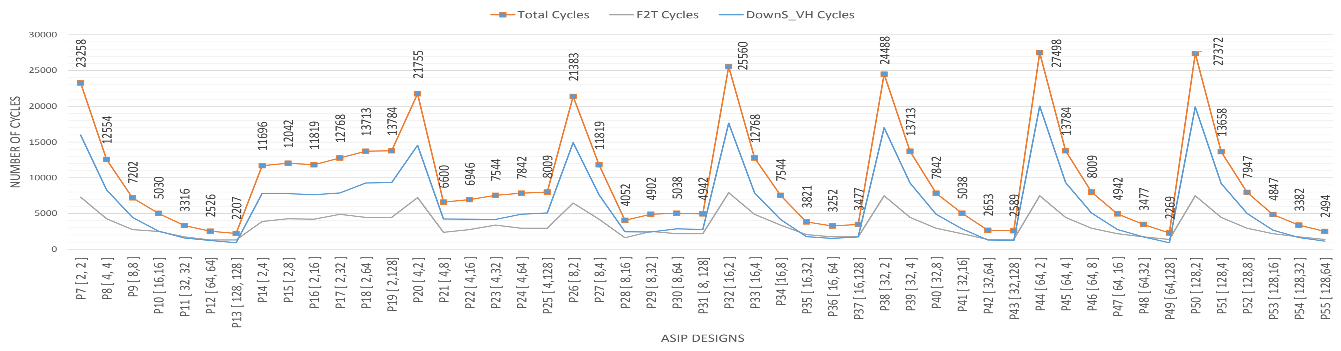

Fig. 5: Cycle counts of DownS, F2T kernels and total of them for different ASIP designs

all the kernels are presented in Figure 5. As a result, the ASIP design points P13[128,128] and P49[64,128] are selected as they both provide the two best performance values among all the designs. The experiments showed that performance is improved proportionally to vector width increase, but is inversely proportional to the sync factor.

VI. Related Work

Traditionally, DLP is implemented using vector processing units with a single vector width, as in the cases of 32-wide vector SODA [6], 8-wide vector Imagine [7] and 16-wide vector NXP EVP [8] processors. Since the vector processing with multiple vector widths is a quite new research topic, we were able to find only a very limited set of publications targeting this topic. In [4], an example architecture, referred to as *anySP*, with configurable SIMD data-path which supports wide and narrow vector lengths is proposed. However, it does not focus on any method for exploring multiple vector widths, as we do in our work. Another work presented in [9], referred to as *Libra*, also focuses on 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 repurposed to behave as a clustered VLIW processor, or combinations in between. In our work, we focus on the static configuration of an ASIP architecture tailored to specific kernels or application.

VII. Conclusion & Future Work

In this paper, we proposed and discussed a novel design method that aims at exploring and deciding the application-specific vector widths for VLIW ASIPs. We also demonstrated application of our method on a set of selected kernels. We implemented our new vector width exploration method as an EDA-tool, and used this tool to perform a set of ASIP synthesis experiments. The results of these experiments are demonstrated and discussed.

References

[1] Software Programmable Media Processor, Movidius Myriad SoC, “Project website.” [Online]. Available: http://movidius.com/

[2] Programmable Image Signal Processor, Intel Mobile SoCs (Medfield, Clover Trail), “Project website.” [Online]. Available: http://www.intel.com/

[3] Y. Park, S. Seo, H. Park, H. K. Cho, and S. Mahlke, “Simd defragmenter: efficient ilp realization on data-parallel architectures,” *SIGARCH Comput. Archit. News*, vol. 40, no. 1, pp. 363–374, Mar. 2012.

[4] M. Woh, S. Seo, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner, “Anysp: anytime anywhere anyway signal processing,” in *Proceedings of the 36th annual international symposium on Computer architecture*, ser. ISCA ’09, 2009, pp. 128–139.

[5] L. Jóźwiak and Y. Jan, “Design of massively parallel hardware multi-processors for highly-demanding embedded applications,” *Journal of Microprocessors and Microsystems*, September 2013.

[6] Y. Lin, H. Lee, M. Woh, Y. Harel, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner, “Soda: A low-power architecture for software radio,” *SIGARCH Comput. Archit. News*, vol. 34, no. 2, pp. 89–101, May 2006.

[7] J. H. Ahn, W. J. Dally, B. Khailany, U. J. Kapasi, and A. Das, “Evaluating the imagine stream architecture,” in *Proceedings of the 31st annual international symposium on Computer architecture*, ser. ISCA ’04, 2004, pp. 14–.

[8] K. van Berkel, F. Heinle, P. P. E. Meuwissen, K. Moerman, and M. Weiss, “Vector processing as an enabler for software-defined radio in handheld devices,” *EURASIP J. Appl. Signal Process.*, vol. 2005, pp. 2613–2625, Jan. 2005.

[9] Y. Park, J. Jong, K. Hyunchul, and 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.