Designing Area and Performance Constrained SIMD/VLIW Image Processing Architectures

Hamed Fatemi[1,2], Henk Corporaal[2], Twan Basten[2], Richard Kleihorst[3], and Pieter Jonker[4]

[1] h.fatemi@tue.nl
[2] Eindhoven University of Technology,
PO Box 513, NL-5600 MB Eindhoven, The Netherlands,
[3] Philips Research Laboratories,
Prof. Holstlaan 4, NL-5656 AA Eindhoven, The Netherlands,
[4] Delft University of Technology,
Lorentzweg 1, NL-2628 CJ Delft, The Netherlands.

Abstract. Image processing is widely used in many applications, including medical imaging, industrial manufacturing and security systems. In these applications, the size of the image is often very large, the processing time should be very small and the real-time constraints should be met. Therefore, during the last decades, there has been an increasing demand to exploit parallelism in applications. It is possible to explore parallelism along three axes: data-level parallelism (DLP), instruction-level parallelism (ILP) and task-level parallelism (TLP).
This paper explores the limitations and bottlenecks of increasing support for parallelism along the DLP and ILP axes in isolation and in combination. To scrutinize the effect of DLP and ILP in our architecture (template), an area model based on the number of ALUs (ILP) and the number of processing elements (DLP) in the template is defined, as well as a performance model. Based on these models and the template, a set of kernels of image processing applications has been studied to find Pareto optimal architectures in terms of area and number of cycles via multi-objective optimization.

1 Introduction

Recently, vision based human interfaces, robotic, inspection or surveillance systems have gained more and more importance, and real-time image processing is essentially necessary for these applications. Therefore, during the last decade, the exploitation of parallelism in applications has been increased [1].

Image processing operations can be classified as low-level (e.g. smoothing, sharpening and filtering), intermediate-level (e.g. Hough transform and object labeling) and high-level (e.g. position estimation and object recognition) [2]. Low-level operations and some medium-level operations can be implemented efficiently in single instruction multiple data (SIMD) processors to exploit data-level parallelism (DLP). High-level operations and some medium-level operations

can be mapped onto very long instruction word (VLIW) processors which exploit the instruction-level parallelism (ILP) [3].

During the last decade, many systems have been designed to exploit parallelism (DLP and ILP). Xetal [4] is an SIMD processor which includes 320 processing elements (PEs), each with one ALU. It is suitable for many low-level operations to exploit DLP. Trimedia [5] is a VLIW example; it can execute five operations per cycle. It is suitable to exploit ILP in high-level operations.

There are also some processors which combine DLP and ILP together, like Imagine [6], which includes eight PEs, with each PE including six ALUs. By increasing the number of PEs, it is possible to exploit more DLP in applications which leads to better performance (decrease in number of cycles). It is also possible to increase the potential for ILP by increasing the number of ALUs per PE, which again causes better performance. In both cases, the area (cost) of the architecture is increased. So, there is a trade-off between area and performance.

In this paper, the relationship between the number of processing elements and ALUs per PE, on the one hand, and the area and performance of the architecture, on the other hand, is studied. For this purpose, an area model based on the number of PEs and ALUs is defined, as well as a performance model. We use multi-objective optimization to find Pareto optimal architectures for several image processing kernels.

The paper is organized as follows. Section 2 explains the architecture on which our measurements are based. The area and performance models for this architecture are studied in Sections 3 and 4. The implementation of the kernels and the design-space exploration via multi-objective optimization are studied in Section 5. Conclusions and future work are discussed in Section 6.

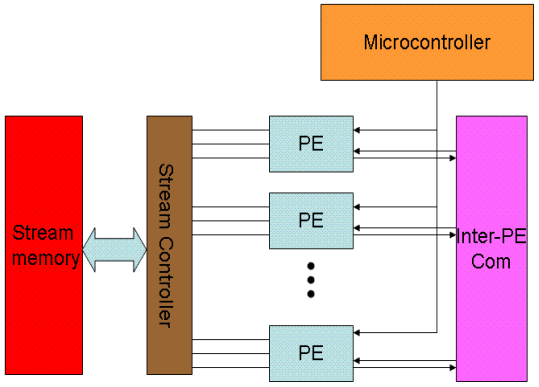

Fig. 1. SIMD Architecture Template (each PE can be a VLIW)

2 Architecture

Fig. 1 shows the template (processor) which is used for our measurements. The template includes the following parts: Processing Elements (PEs); micro con-

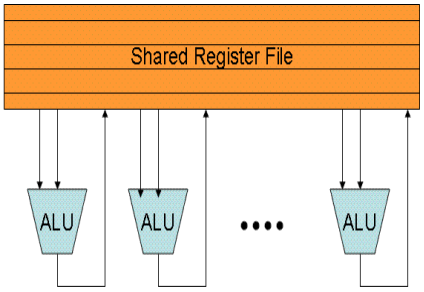

Fig. 2. PE with shared register file

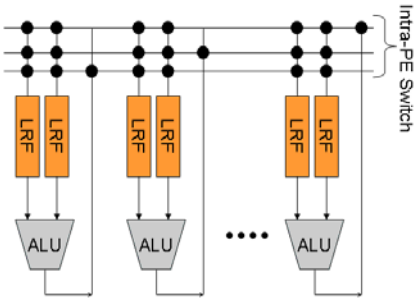

Fig. 3. PE with local register file

troller unit; inter-PE communication unit; stream memory unit (including the stream controller).

Fig. 2 and 3 show the inside of a processing element which includes ALUs and register file(s). The number of ALUs is one of the template parameters determining the number of operations that can be performed at the same time by a single PE. Each ALU has two inputs and one output. Two kinds of register files can be used. One type is a shared register file. It means that each PE has one register file and each ALU is attached to this shared register file by three ports (two inputs and one output) (Fig. 2). Another kind of register file is the local register file. It means that each ALU input has a separate register file and there is an intra-PE switch that connects the ALU outputs to the inputs (Fig. 3). The stream controller unit is responsible for reading data (e.g. pixel values) from the stream memory and transferring it to the register files and vice versa. The size of the register file is also one of the template parameters. The micro controller unit has two parts: 1- the memory part for storing PE instructions and 2- the decoder for decoding PE instructions and sending them to PEs. The inter-PE communication unit is responsible for connecting PEs together to send/receive data among each other. This unit is an N*N switch (with N being the number of PEs). The stream memory unit is the connection between external memory and I/O (outside the template) and the PEs. It takes data from off-chip memory and sends it to the PEs via the stream controller and vice versa. Input and output data are stream based. Each PE iteratively reads elements from input streams and writes elements to the output streams. The stream memory contains N (the number of PEs) single ported memory banks and each PE is able to access its own part in parallel to other PEs. This implies that inter-PE communication requires a separate communication network (inter-PE communication unit) and cannot be done by the stream memory.

3 Area Model

In order to compare different parallel configurations, we developed an area model. We derive a formula for area, which is based on the number of ALUs (functional

units) in each PE and the total number of PEs. This area model is meant to give an area estimation for the region containing the ALUs, PEs, micro controller and stream memory of the template. Template parameters are described in Table 1. The total area of the template is (in mm^2):

$$A_{total} = N_{PE} \cdot A_{PE} + A_{inter-PEcomm} + A_{stream-memory} + A_{micro} \quad (1)$$

Variable	Value	Description
N	-	Number of <subscript> (i.e., Npe= number of PEs)
A	-	Area of <subscript> (i.e., Ape= area of a PE)
b	32	Data width of the architecture
w	1.86	Wire pitch (typically 0.64 – 2μm/wire in a 0.18μm process)

Table 1. Parameters used in the area model

Description	Values
Number of PEs in the template	1,2,4,8,...1024-
Number of ALUs per PE	1,2,3,4,....10
Type of register file	1: shared & 2: local
Size of register file	3,4,6,8,12,16....256
Mem size of microcontroller	256,512,.....2048

Table 2. Parameters of DSE

The inter-PE communication unit allows each PE to send/receive data to/from other PEs. The area of this switch (assuming a wire limited design) grows quadratically with the number of PEs inside the template. The basic formula for the switch size is:

$$A_{inter-PEcomm} = N_{PE}^2 \cdot b^2 \cdot w^2 \quad (2)$$

To estimate the size of this switch, we used a wire pitch of 1.86 μm to make enough room for power, ground, and noise shielding wires [7].

The PE area depends on the area of the ALUs, the register file and the intra-PE switch (in case of local register files). Based on the register file, the PE area can be calculated in two ways:

- PE with shared register file: In this case, the area includes the register file and the ALUs. For our estimation of the register file size, we borrow the model described by Rixner in [7]. In his model, the area of a register file is the product of the number of registers, the number of bits per register and the size of a single-bit register cell. The size of each register cell is a function of the width and height of the register cell without ports (estimated to be w^2) and the number of ports squared. As each ALU needs 3 ports (2 inputs and 1 output), the area of the register file is a function of the number of ALUs squared. The total area of a PE is:

$$A_{PE} = N_{register} \cdot b \cdot (3 \cdot N_{ALU})^2 \cdot w^2 + N_{ALU} \cdot A_{ALU} \quad (3)$$

- PE with local register files: In this case, the area includes the local register files, the ALUs and the intra-PE switch. Each ALU has two local register files with two ports (one input and one output). The area of a local register

file is the product of the number of registers, the number of bits per register and the size of a single-bit register cell (with two ports). Regarding (2), the area of the intra-PE switch is proportional to the number of ALUs squared. Therefore, the area of a PE is:

$$A_{PE} = N_{register} \cdot 2 \cdot b \cdot A_{lrf\ bit} + N_{ALU} \cdot A_{ALU} + N_{ALU}^2 \cdot b^2 \cdot w^2 \quad (4)$$

Since every PE receives the same instruction, the micro controller size is constant as the DLP degree is increased. Even when the number of ALUs per PE increases, the code size does not change dramatically. The total number of operations remains roughly constant (assuming not too much speculative code). Only the number of NOP will increase (a well known VLIW problem). However, using NOP code compression will compensate for that. Therefore, the memory storage part of the microcontroller can remain constant, but the control logic and instruction decoders should be increased as we scale ILP:

$$A_{micro} = A_{micromemory} + A_{decoder} \cdot N_{ALU} \quad (5)$$

The area of the stream memory unit contains a constant part for the stream controller plus the product of the number of PEs, the number of ALUs per PE, the memory size for each ALU (when increasing the number of ALUs, more data storage is needed to keep them busy; we assume a constant memory size required per ALU), data width and the area of a 1-bit SRAM:

$$A_{stream-memory} = N_{PE} \cdot N_{ALU} \cdot b \cdot memory\ size\ per\ ALU \cdot A_{SRAM\ bit} + A_{stream-controller} \quad (6)$$

4 Performance Model

For calculating the number of cycles of an application kernel, we used an adapted version of the Imagine tools. It is possible to simulate varying degrees of instruction-level parallelism by changing the number of ALUs in each PE. From the microcode file (output of the Imagine kernel compiler), we can directly determine the number of instructions in each basic block of a kernel. Furthermore, we know which of these blocks correspond to the kernel loop body and which are outside the loop. For kernels with one loop, we model its number of cycles as:

$$N_{cycle} = N_{loop-cycle} \cdot N_{loop-iter} + N_{nonloop-cycle} \quad (7)$$

$N_{loop-cycle}$ and $N_{nonloop-cycle}$ are extracted from the microcode file. The number of iterations ($N_{loop-iter}$) depends on N_{data} (the amount of data sent to the kernel, e.g. image size) and N_{PE}, as expressed in the following formula.

$$N_{loop-iter} = \lceil \frac{N_{data}}{N_{PE}} \rceil \quad (8)$$

The cycle calculation is easily adapted to kernels with multiple loops.

5 Evaluation

In Sections 2, 3 and 4, we studied the template and the area/performance model related to it. In this section, the search for an optimal solution in terms of area and cycles is discussed.

5.1 Multi-Objective Optimization

It is obvious that the number of cycles needed for the execution of a program can be decreased by increasing the number of PEs. By increasing the number of PEs, the area of the template is also increased (1). Therefore, improving the number of cycles leads to an increase in area and vice versa. To investigate this trade-off, we have used multi-objective optimization [8]. A general multi-objective optimization problem can be described as a vector function f that maps a number of m decision variables (in our case the template parameters; see Table 2) to a number of n objectives (in our case area and performance). Formally:

$$\begin{array}{l} min/max \;\; Y = f(X) = (f_1(X), f_2(X), \ldots, f_n(X)) \\ subject\ to : X = (x_1, \ldots, x_m), Y = (y_1, \ldots, y_n) \end{array} \tag{9}$$

where X is called the decision vector from the parameter space and Y is the objective vector from the objective space. In our measurements, the objective vector consists of area and number of cycles. The set of solutions for a multi-objective optimization problem consists of all decision vectors for which the corresponding objective vectors cannot be improved in any dimension without degradation in another. These vectors are known as Pareto optimal. Mathematically, the concept of Pareto optimality is defined as follows. Assume, without loss of generality, a maximization problem and consider two decision vectors a, b. Then, a is said to dominate b if and only if:

$$f_i(a) \geq f_i(b) \;\; \forall i = 1, \ldots, n \quad \bigwedge \quad f(a) \neq f(b) \tag{10}$$

All decision vectors which are not dominated by any other decision vector of a given set, are called non-dominated regarding to this set. The decision vectors that are non-dominated within the entire search space are Pareto optimal and constitute the so-called Pareto-optimal set.

5.2 Measurements

Table 2 shows the template parameters. Our design space has 13200 points and for finding Pareto points, we could still search the complete design space. It takes around 10 minutes (on an Intel Pentium processor 1.70 GHZ) to perform this exhaustive search. The objective vector includes area and cycles for each point of the design space.

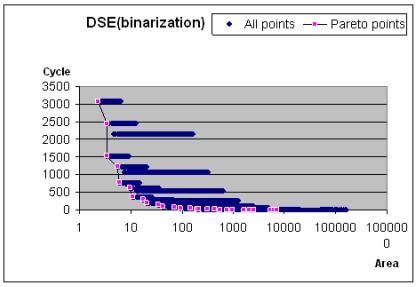

Fig. 4. DSE for binarization kernel

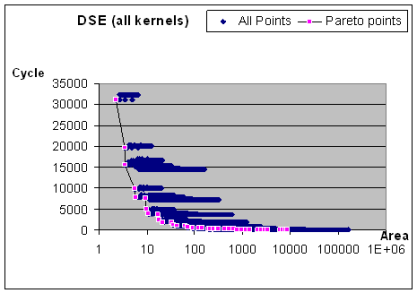

Fig. 5. DSE for merged kernels

For measurement, we selected three of the most popular kernels from image processing applications [9] (color conversion, binarization, convolution, with image size 640×480). The result for binarization is shown in Fig. 4 (for the other kernels, results are similar). We used a normalized logarithmic scale for the horizontal axis. The big gaps in the figure are caused by changes in the number of PEs in the template. The Pareto points represent the trade-off between area and number of cycles.

In order to investigate the Pareto points for a *domain*-specific (i.e for all three kernels) architecture instead of an *application*-specific architecture, we merge these three kernels into one kernel (Fig. 5). The most interesting part of this graph is when the area is not much larger than 100 (larger chip area becomes too costly). Our measurements show the most interesting part is when the number of PEs is between 2 and 64 and the number of ALUs in each PE is between 1 and 4. More ALUs per PE causes an increase in the intra-PE switch area (needed for inter ALUs communication). All Pareto points turn out to have local register files, even if the PE contains only one ALU. The reason is that a shared register file needs (many) more ports (even for a single ALU, it needs three ports). By increasing the number of ALUs in a PE, it is possible to reduce the size of a local register file. For example in convolution, it is possible to reduce the size of each register file from 24 registers (PE with 1 ALU) to 8 (PE with 4 ALUs). The size of the microcontroller in all Pareto points is 256; it turns out that this is sufficient to store all kernels. By comparing these Pareto points, we can observe that specialized architectures (Pareto points of each kernel) do not perform much better than the generalized architectures (Pareto points of merged kernels) because of our limited design space. By adding other parameters to the template like inter-PE communication, specialized function units, etc., it is possible to observe the trade-off between specialization and generalization. This is a topic for our future research.

One of the ratios that can be used for comparing the Pareto points is performance per unit area. The optimal templates (better performance/area) have between 4 and 64 PEs and each PE has 1 or 2 ALUs. These Pareto points might be good candidates if the designer is interested in getting the most performance out of area.

6 Conclusions and Future Work

In this paper, we studied a suitable hybrid SIMD/VLIW processor for image processing kernels, regarding area and cycle numbers. The parameters which we studied are the number of ALUs, the number of PEs, the type of the register file, the size of the register file and the micro controller. To study this problem, area and performance models have been defined. For finding the Pareto-optimal solutions, we used multi-objective optimization. Regarding the design space, we used the full search method. By looking at the Pareto points in the design space, it is observed that most interesting points have 4-64 PEs, one or two ALUs per PE with local register files.

By increasing the number of PEs beyond 64, the area of the inter-PE communication unit dominates the area (2). For solving this problem, in the future, we will study other processor parameters such as the number of connections and the bandwidth between PEs. Furthermore, we will add parameters like the number of ports to the stream memory, special function units and the number of load/store units. This creates a larger design space. For finding Pareto points, in this extended space, we already developed a multi-objective optimization by using evolutionary algorithms (full search takes too much time). We also want to look at the delay (cycle-time) and energy of the template, and investigate a multi-processor template (combining multiple instances of our current template) for image processing applications.

References

1. W. Caarls, P. Jonker, and H. Corporaal, "Smartcam: Devices for embedded intelligent cameras," in *PROGRESS 2002, 3rd seminar on embedded systems, Proceedings*, (Utrecht, The Netherlands), pp. 1–4 (CD–ROM), 24 October 2002.
2. E. Komen, *Low-level Image Processing Architectures*. PhD thesis, Delft University of Technology, 1990.
3. H. Fatemi, H. Corporaal, T. Basten, P. Jonker, and R. Kleihorst, "Implementing face recognition using a parallel image processing environment based on algorithmic skeletons," in *Proceedings of the 10th Annual Conference of the Advanced School for Computing and Imaging*, (Port Zelande, The Netherlands), pp. 351–357, June 2004.
4. A. Abbo and R. Kleihorst, "Smart cameras: Architectural challenges," in *Proceedings of ACIVS 2002 (Advanced Concepts for Intelligent Vision Systems)*, (Gent, Belgium), pp. 6–13, 2002.
5. TriMedia Technologies. http://www.semiconductors.philips.com.
6. B. Khailany, W. J. Dally, S. Rixner, U. J. Kapasi, P. Mattson, J. Namkoong, J. D. Owens, B. Towles, and A. Chang, "Imagine: Media processing with streams," *IEEE Micro*, pp. 34–46, April 2001.
7. S. Rixner, W. J. Dally, B. Khailany, P. Mattson, U. J. Kapasi, and J. D. Owens, "Register organization for media processing," in *Proceedings of the 6th International Symposium on High-Performance Computer Architecture*, (Toulouse, France), pp. 375–386, Jan 2000.
8. C. M. Fonseca and P. J. Fleming, "An overview of evolutionary algorithms in multiobjective optimization," *IEEE Micro*, vol. 3, pp. 1–16, 1995.
9. W. Caarls, "Testbench algorithms for smartcam," tech. rep., Delft University of Technology, The Netherlands, 2003.