Dynamic-SIMD for lens distortion compensation

Bart Mesman, Hamed Fatemi, Henk Corporaal, and Twan Basten
Eindhoven University of Technology, The Netherlands
{b.mesman, h.fatemi, h.corporaal, a.a.basten}@tue.nl

Abstract. *An increasing computational demand is placed on the image processing capacity of current and future smart cameras. SIMD processor architectures provide an efficient solution because their repetitive structure matches the data-parallel execution pattern inherent in pixel-type processing. But the lack of support for communicating pixel data over variable distances has forced designers to allocate dedicated hardware or FPGAs for compensating lens distortion and other non-linear operations. We propose a hardware extension to SIMD processors that enables dynamic communication. Using detailed area cost models and a high-level simulator we optimize the extension with regard to the number of busses, bus arbitration policies, and local instruction buffer sizes.*

1 Introduction

In the pursuit of Smart-Cameras [3] an increasing demand is placed on the computational capabilities of image processors. For reasons of power consumption and cost, SIMD architectures are especially efficient for image processing because their repetitive structure matches the data-parallel execution pattern inherent in pixel-type processing.

However, available SIMD processors lack support for indirectly addressing processing elements (dynamic communication), a necessary property for lens distortion compensation (LDC) and other highly non-linear algorithms. Non-linear transformations require pixel values to be transferred over the image by a distance that is different from those of other pixels, and potentially require run-time computations. The SIMD principle however states that all processing elements perform the exact same operation, including the transfer of pixel values (with equal distances).

In this paper we will consider an extension to the communication infra-structure of SIMD processors that enables dynamic communication of pixel data over the processing elements. Area models of the architecture components indicate less than 30% increase in area for an architecture configuration that has sufficient capacity to support real-time LDC and many other algorithms in parallel. Although the mapping of LDC is a sufficient motivation for accepting the area overhead, we have kept the architecture generic such that both static algorithms and other dynamic algorithms can be programmed.

This paper is organized as follows. Section 2 and Section 3 discuss our architecture extension and the corresponding area model. The architecture is evaluated using LDC in Section 4.

2 Dynamic-SIMD Architecture

To support dynamic communication in SIMD processors, a flexible network is needed between processing elements (PEs). The communication bottleneck, as illustrated in the introduction, is caused by the fact that PEs organized in an SIMD fashion need access to the communication infrastructure at the same time and over the same distance. From the application requirements we can easily motivate the following global architectural choices.

Segmentations: The busses in our architecture are segmented using registers to enable parallel access to different parts of the bus and in order to constrain the critical path for a high clock frequency.

Destination PE address: Communicated data packets are annotated with the destination PE address. Receiving PEs require an address comparator for fetching the correct data packets from a bus.

Source PE address: Because the order of packet arrival is dynamic, the receiving PE also needs the source address for identifying the data packets.

Valid bit: To check the validity of the data in busses and also reduce the energy of decoding the destination address, a valid bit in the data packet is used.

Separate busses: The direction of data (to the left or right in the PE array) is always clear since we apply separate busses for communication to the left and right. Since most image processing algorithms have symmetrical properties, this separation yields an efficient load balancing of communication tasks.

Simple PEs: To minimize the PE area overhead, PEs have a simple organization that does not support out-of-order execution.

Local instruction buffers: Dynamic communication requires arbitration of each bus. As a result, a PE sometimes has to wait for availability of the corresponding bus segment. Each PE has a local instruction buffer responsible for the next instructions (or communications) not to wait un-

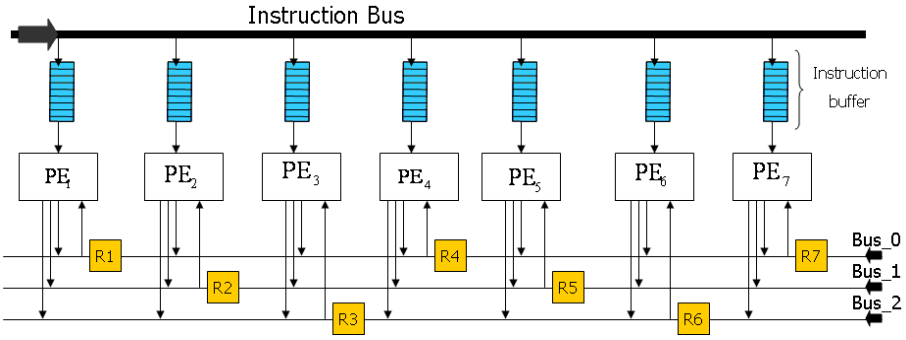

Figure 1. Dynamic SIMD architecture, where each PE can write to all busses (only left communication).

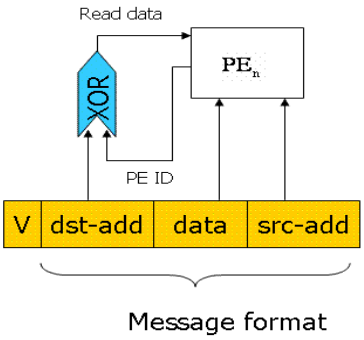

Figure 2. Matching PE-ID (for reading data).

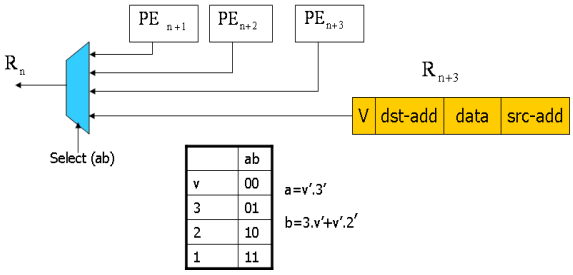

Figure 3. Writing data to register (when $N_{bus} = 3$).

til all PEs finish their communication. Only if a buffer is full will the instruction dispatcher interrupt the instruction stream.

One way to realize this idea is with the architecture depicted in Fig. 1. In this architecture, there are separated unidirectional segmented busses for left and right communication (N_{bus}, busses per direction). Fig. 1 gives a schematic view of this architecture when $N_{bus} = 3$. (In the remainder of this section, we only mention left communication. All our arguments hold for right communication as well.) In this architecture each PE can only read data from one bus (e.g. PE_n can read data on bus number $(n-1)$ *mod* N_{bus}) but it can write to all busses. For each PE, there is a specific register for reading data. PE_n can only read data from R_n but it can write data to R_{n-1}, R_{n-2},..., $R_{n-N_{bus}}$. For example, in the architecture of Fig. 1, if PE_6 wants to send data to PE_1, it should put the data in R_4; then this data is shifted to R_1 and subsequently PE_1 can read this data from R_1.

Fig. 2 (bottom) shows the message format of data (including PE-ID's of receiver/sender, data and valid bit).

For reading data from a register, each PE should execute the following steps: *Step 1*: Each PE should check the valid bit of a register. *Step 2*: Check the destination address of the message with its own PE-ID. *Step 3*: If the destination address matches the PE-ID, read the data and reset the valid bit (to zero). Fig. 2 depicts the reading procedure (hardware logic) from a register.

Before each register R_n, there is a multiplexor to select the data from PE_{n+1}, PE_{n+2},..., $PE_{n+N_{bus}}$ and the preceding register $R_{n+N_{bus}}$. Fig. 3 shows the writing procedure (and logic) when $N_{bus} = 3$ by giving highest priority to the preceding register, then to PE_{n+3}, PE_{n+2} and PE_{n+1}. Deriving the select signal of the multiplexor is simple (needs only four gates: 3 ANDs and 1 OR). Consequences of assigning different priorities for writing data to the busses are studied in Section 4.

Note that reading from registers is done in the first half of the cycle time and writing to registers is done in the second half of the cycle.

Because of the dynamic nature of communication in many applications, it sometimes happens that a PE wants to write data in the register and at the same time, the preceding register or a higher priority PE wants to transfer data to the next register (e.g. PE_5 wants to put data in R_4 and the valid bit of R_7 is one, see Fig. 1). Because this architecture gives priority to the register, the PE should wait till the preceding register will become empty. A local instruction buffer is needed to keep instructions (top of Fig. 1).

Parameter	Value
N_{PE}(Number of PEs in the SIMD processor)	1...512
$A_{reg-bit}$(Area of a 1-bit D flipflop)	5
N_{PE-bit}(Number of bits which represents the PE-ID)	9
A_{XOR}(Area of a 1-bit XOR (2-input))	3
A_{AND}(Area of a 1-bit AND (2-input))	1
$A_{mux2\times1}$(Area of a 2x1 multiplexor)	2
b(Data width (bits))	16
$b1$(width of instruction (bits))	16
$N_{instr-size}$(Size of instruction buffer in each PE)	1...400
N_{bus}(Number of busses (for each direction)	1...9
k(Number of gate for the priority logic in Dynamic-SIMD)	-

Table 1. Parameters used in the area model (normalized to 1-bit AND in CMOS 0.18)

3. Area

In order to calculate the area (cost) of dynamic communication in an SIMD processor, we developed an area model. This area model is meant to give an area estimation for the region containing PEs, multiplexors, registers in communication busses, instruction buffers and logic for implementing the priorities. The parameters used for the area model are described in Table 1 [5, 4]. (All area values are normalized to the area of 1-bit AND. We assume that the area of a 1-bit AND-gate is the same as that of a 1-bit OR-gate.)

$$A_{Dynamic-SIMD} = N_{PE} * [A_{PE} + 2 * A_{read-part} + 2 * A_{write-part} + 2 * A_{reg} + A_{instruction-buffer}] \quad (1)$$

Note that parts of the area are multiplied by 2 because of the presence of left and right communication. $A_{read-part}$ includes the area of a XOR (which is used for comparing the destination address and the PE-ID) and the area of registers which are used for storing data (one register per PE).

$$A_{reg} = (b + 2 * N_{PE-bit} + 1) * A_{reg-bit} \quad (2)$$
$$A_{instruction-buffer} = N_{instr-size} * b1 * A_{reg-bit}$$
$$A_{read-part} = N_{PE-bit} * A_{XOR} + (N_{PE-bit} - 1) * A_{AND} + (b + N_{PE-bit}) * A_{reg-bit}$$
$$A_{write-part} = A_{mux(N_{bus}+1)\times1} * b + k * A_{AND} + A_{OR}$$

In the calculation of $A_{write-part}$, the area of AND gates is multiplied by k to account for the priority logic for writing data to the register ($k = 4$ when $N_{bus} = 3$, Fig. 3).

Fig. 4 shows the area-overhead of the Dynamic-SIMD architecture in comparison to a locally connected(LC)-SIMD [1] when $N_{bus} = 3$ (the horizontal axis shows the number of PEs in each architecture). In LC-SIMD, each PE has only one bus for communication and it can communicate only to its direct neighbors.

4 Evaluation

Most vision applications use cameras for capturing images or video data. Wide-angle lenses are often used in environments where there is little room to position cameras to record an activity of interest. Also, often, cheap compact lenses are used to cut down the cost of the device. Such devices produce images with a radial distortion. Lens Distortion correction (LDC) can be applied to correct these images. Fig. 5 shows a distorted image and the image after correction. LDC is one of the complex low-level algorithms [2] that cannot be mapped on current SIMD processors for real-time execution as discussed in Section 1. Because of its importance for smart cameras this application is chosen to inspect the possibility of mapping it on the architecture presented in Section 2. We consider different bus arbitration priorities, different number of busses, and different buffer sizes for instructions.

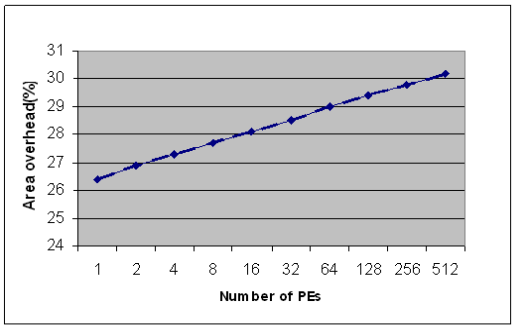

Figure 4. Area overhead (compared to an LC-SIMD).

(a) (b)

Figure 5. (a) Lens distorted image , (b) Corrected image.

Priority: Fig. 3 shows the architectures by giving higher priority to registers than to PEs. Other alternatives exist which give higher priority to PEs (by changing the Dynamic-SIMD hardware).

Table 2 shows the number of cycles required for communication by the implementation of LDC in the architecture Dynamic-SIMD with different priorities (with $N_{bus} = 3$ and $N_{instr-size} = 32$). The result shows that the best performance is gained by giving the highest priority to registers. In this case, the communication is about twice faster

priority	Number of cycles
register, PE_{n+1}, PE_{n+2}, PE_{n+3}	10035
register, PE_{n+3}, PE_{n+2}, PE_{n+1}	11641
PE_{n+1}, PE_{n+2}, PE_{n+3}, register	19601
PE_{n+3}, PE_{n+2}, PE_{n+1}, register	20281

Table 2. Cycle count for different priorities

than the communication with the other priority assignment (giving the priority to PEs). The total time for implementing LDC is $T_{execution} = T_{communication} + T_{processor-load}$. $T_{communication}$ is around $0.1ms$ (=$10035 * 10ns$, freq = 100 MHZ). Total number of cycles needed for calculating destination address for each pixel of input image is 86400 cycles (image size: 640×480) which is $0.86ms$ (=$86400 * 10ns$, freq = 100 MHZ). Accordingly, the total time needs for implementing LDC is less than $1ms$ which is less than 3% of frame rate (30 frames per sec). The result is the same for different values of N_{bus} and $N_{instr-size}$.

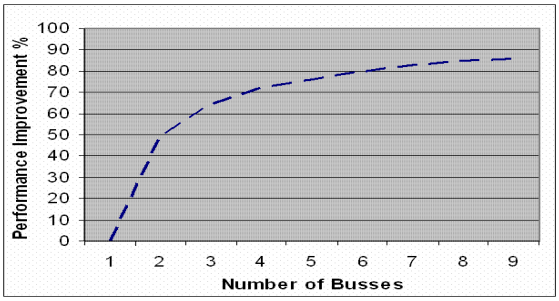

Figure 6. Performance improvement with different N_{bus}.

Number of busses: One of the parameters in this architecture is the number of busses for left and right communication. By increasing the number of busses the architecture becomes expensive (because the number of multiplexors and the arbitration logic are increased, Eq. 2). Fig. 6 shows the performance improvement by increasing the number of busses (N_{bus}) in comparison to the architecture with $N_{bus} = 1$ (we assume that $N_{instr-size} = 32$). It shows that by increasing the number of busses to 3, it is possible to improve performance by almost 65%. More busses (> 3) make the architecture expensive without much gain in performance.

Buffer size: Fig. 7 shows the number of cycles with different buffer sizes ($N_{instr-size}$) when $N_{bus} = 3$. It illustrates that by increasing the buffer size to 32 ($= 2^5$) instructions, we can gain upto almost 10% improvement. Larger buffer sizes (> 32) make the architecture expensive without much gain.

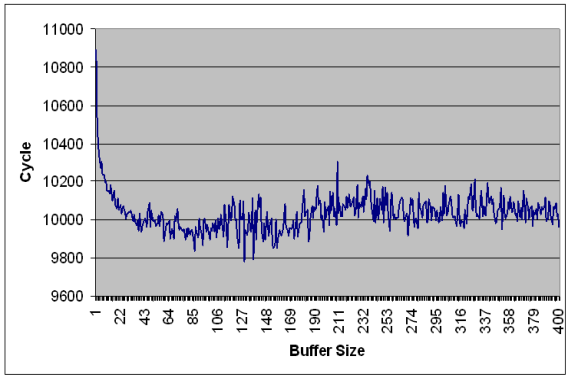

Figure 7. Different instruction buffer sizes.

5 Conclusions

In this paper we have proposed an alternative communication architecture for SIMD processors that allows dynamic indirect addressing of processing elements. This is a necessary property for compensating lens distortion and other future smart camera functions that display non-linear behavior. The required architectural components, including a bus access arbiter, address comparators, local instruction buffer, and the PEs, are designed for simplicity to constrain the area cost, as verified by detailed cost models. Architecture parameters like arbitration protocols, the number of busses, and the buffer sizes have been explored to yield a configuration that supports real-time lens distortion compensation in parallel to many other functions with an increase of less than 30% in area. This is an attractive alternative to the allocation of dedicated processors or FPGA for LDC.

References

[1] A. Abbo and R. Kleihorst. Smart Cameras: Architectural Challenges. In *Proc. of Advanced Concepts for Intelligent Vision Systems (ACIVS)*, pages 6–13, Gent, Belgium, September 2002.

[2] W. Caarls, P. Jonker, and H. Corporaal. SmartCam Design Framework. In *Proc. of PROGRESS 2003, 4th seminar on embedded systems*, pages 1–8 (CD–ROM), Nieuwegein, The Netherlands, October 2003.

[3] H. Fatemi, R. Kleihorst, H. Corporaal, and P. Jonker. Real time face recognition on a smart camera. In *Proc. of Advanced Concepts for Intelligent Vision Systems (ACIVS)*, pages 222–227, Gent, Belgium, September 2003.

[4] S. Mathew, R. K. Krishnamurthy, M. A. Anders, R. Rios, and K. Soumyanath. Sub-500-ps 64-b ALUs in 0.18 SOI/Bulk CMOS: Design and Scaling Trends. *IEEE Journal of Solid-State Circuits*, 36(11):1636–1646, November 2001.

[5] T. Szymanski, H. Wu, and A. Gourgy. Power complexity of multiplexer-based optoelectronic crossbar switches. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions*, 13:604–617, May 2005.