Run-Time Reconfiguration of Communication in SIMD Architectures

Hamed Fatemi[1], Bart Mesman[1], Henk Corporaal[1], Twan Basten[1], and Pieter Jonker[2]
[1]Eindhoven University of Technology [2]Delft University of Technology
{h.fatemi, b.mesman, h.corporaal, a.a.basten}@tue.nl p.p.jonker@tnw.tudelft.nl

Abstract

SIMD processors are increasingly used in embedded systems for multi-media applications because of their area- and energy-efficiency. Communication between the processing elements (PEs) in an SIMD processor has remained a cause of inefficiency however; the SIMD concept prescribes that all PEs communicate in the same clock cycle. Existing SIMD architectures solve this problem either by multi-hop communication (causing cycle overhead), or by a fully connected communication network (causing area overhead).

To solve the communication bottleneck, we propose a reconfigurable SIMD architecture (RC-SIMD) with a set of delay-lines in the instruction bus, distributing the accesses to the communication network over time. We can (re-)configure the size and number of delay-lines, a specific configuration representing a trade-off between the number of clock cycles and the length of a clock period. Reconfiguration time is typically much less than 1% of the execution time of an algorithm, and the extra configuration hardware is less than 2%. Experiments show that our reconfigurable architecture achieves (on average) more than 10% performance improvement over a non-reconfigurable architecture.

1 Introduction

Video, graphics, and gaming applications increasingly pose some of the most challenging demands on the computational capabilities of consumer-oriented devices. For reasons of excessive power consumption and cost, general-purpose processors do not offer viable solutions matching these requirements. Solutions should therefore be sought in MIMD (Multiple Instruction Multiple Data) (e.g., the cell-based architecture for PlayStation3 [6]) or SIMD (Single Instruction Multiple Data) architectures. SIMD architectures are especially efficient for the mentioned applications because their repetitive structure matches the data-parallel execution pattern inherent in pixel-type processing.

One of the most debatable aspects of an SIMD architecture however, is its communication infrastructure. To illustrate the problem, consider two extremes of an SIMD communication architecture. In the first architecture (locally connected SIMD: LC-SIMD), 320 sequential PEs can communicate with their direct neighbors only. Mapping a 13-tap horizontal filter to this architecture, each PE n requires pixel values from PE $n-6$ to PE $n+6$. In that case, almost 50% of time and resources is spent on shifting values to neighboring PEs, which is an intolerable overhead. The other extreme (fully connected SIMD: FC-SIMD) has a much richer communication architecture, supporting single-cycle communication within a neighborhood of 6 PEs. It requires at least a network with 6 (segmented) busses, since all 6 PEs in a neighborhood access the bus network at the same time. Because it is dimensioned for high peak bandwidth, this yields low bus-utilization and a rigid bus structure that eludes the possibility to adapt to application characteristics (e.g., maximum neighborhood communication).

In [5] we introduced a new SIMD architecture (RC-SIMD) with a delay line in the instruction bus. The delay line distributes bus accesses of consecutive PEs over successive clock cycles, drastically reducing peak bus bandwidth requirements. We showed that this architecture requires only 10% more area than a (cheap) LC-SIMD, whereas the performance (in number of cycles) is almost equal to that of an (expensive) FC-SIMD. We solved the corresponding scheduling problem via a precise resource conflict model. We also mentioned that our architecture can in principle be adapted to a specific communication neighborhood size. In this paper we focus on these (re-)configuration aspects of RC-SIMD, and determine the cost and gain of this flexibility by comparing area and performance to that of an RC-SIMD with a fixed neighborhood size.

Our paper is organized as follows. The RC-SIMD architecture is explained in Section 2. In Section 3, we explain the reconfigurability aspects of RC-SIMD. The programming of the reconfigurable part is studied in Section 4. We present experimental results and a comparison with non-reconfigurable architectures in Section 5, and end with conclusions.

2 Architecture

The communication dilemma in SIMD processors is well-recognized. It stems from the time-consuming

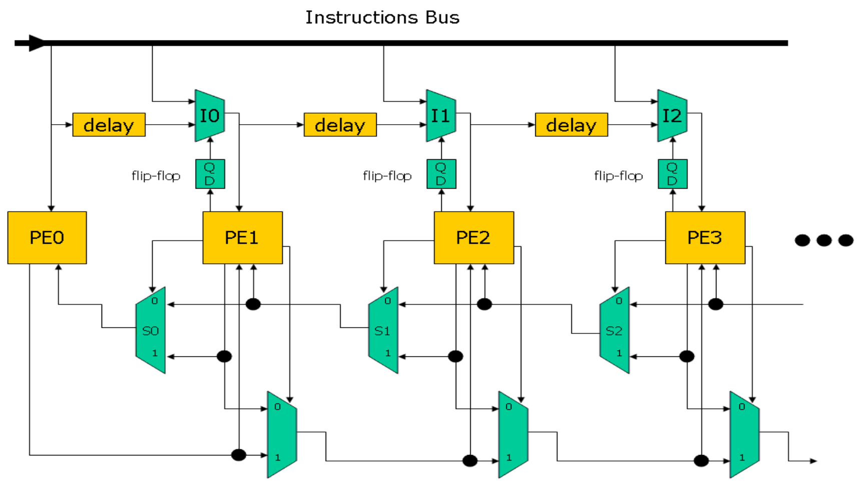

Figure 1. Reconfigurable Communication SIMD (RC-SIMD) architecture.

shift overhead in locally connected and the costly network in fully connected communication architectures. As illustrated in [4], the problem is intensified by the fact that all PEs used in an SIMD fashion need to access the communication infrastructure at the same time. This bottleneck can thus be overcome by ensuring that an SIMD instruction reaches the various PEs at different points in time. One way to realize this idea is with the architecture depicted in Figure 1.

The architecture has two segmented unidirectional communication busses (bottom of Figure 1), one for left-hand and one for right-hand traffic. Using the multiplexors, each PE can choose to pass the value already on the bus, or to put itself a value on the bus. This allows for simultaneous communication, for example from PE1 to PE3 and from PE3 to PE5, over different segments of the same bus. The real novelty of this architecture is the delay line in the instruction distribution (top of Figure 1). Assuming that the multiplexors receive their inputs from the delay registers, successive PEs receive the same instructions (like any SIMD), but in successive clock cycles. This architecture therefore has the same advantages as other SIMD architectures (i.e., small code size), but the communication bottleneck identified above is no longer present. This is illustrated in Figure 2, where an algorithm is scheduled on an SIMD architecture without and with a delay line in the instruction distribution. In these schedules, operation LD+2 on PE0 (clock cycle 4) means loading a value from PE2 (PE0+2). The attentive reader may notice that the schedule in Figure 2 (b) is actually also invalid (PE0 and PE2 use multiplexor S2 at the same time in cycle 6). In [5], this problem is solved by defining a proper resource conflict model for an instruction scheduler. Figure 2 (c) shows a valid schedule for this example when using an extended version of the FACTS scheduler [3] for this architecture.

3 A Reconfigurable Architecture

Reconfigurability The architecture with a fixed delay line has several drawbacks. The most severe one is the extreme run-in phenomenon. In an architecture with, e.g., 320 PEs, it takes 320 cycles for the first instruction to reach PE319, which is clearly undesirable. Another issue is the size of the conflict model, which may cause severe run-time problems for the compiler.

A relatively simple idea shown in Figure 1 can solve all these problems. Using the multiplexor for selecting the instruction and the flip-flop to store the control-signal of the multiplexor, each PE can be configured to either take the instruction delayed by the previous PE or the un-delayed instruction from the instruction bus. Suppose that the maximum neighborhood communication in an algorithm is k. We then configure the multiplexors I0, I1, ... such that PEn receives instructions with $n \ mod \ k$ clock cycles delay.

Flexible clock frequency The clock frequency is upper bounded by the longest delay through the segments and multiplexors (the maximum neighborhood communication). The architecture itself does not pose any limits on that; in theory it allows 1 clock-cycle communication between PE0 and PE319, albeit with a very low clock frequency. The maximum clock frequency is determined by the maximum neighborhood communication as allowed by the compiler (or vice versa). The compiler is therefore parameterizable with respect to the maximum neighborhood communication.

To determine the clock frequency in RC-SIMD, we motivate this assumption that PEs can do the computation and communication in one clock cycle. It means that each PE can receive data from the other PEs, use

cycle	PE 0	PE 1	PE 2	PE 3
0	LD 0	LD 0	LD 0	LD 0
1	* C0	* C0	* C0	* C0
2	LD +1	LD +1	LD +1	LD +1
3	* C1	* C1	* C1	* C1
4	LD +2	LD +2	LD +2	LD +2
5	* C2	* C2	* C2	* C2
6	LD +3	LD +3	LD +3	LD +3
7	* C3	* C3	* C3	* C3
8	sum	sum	sum	sum
9	ST	ST	ST	ST

(a)

cycle	PE 0	PE 1	PE 2	PE 3
0	LD 0	-	-	-
1	* C0	LD 0	-	-
2	LD +1	* C0	LDP0	-
3	* C1	LD +1	* C0	LD 0
4	LD +2	* C1	LD +1	* C0
5	* C2	LD +2	* C1	LD +1
6	LD +3	* C2	LD +2	* C1
7	* C3	LD +3	* C2	LD +2
8	sum	* C3	LD +3	* C2
9	ST	sum	* C3	LD +3
10	-	ST	sum	* C3
11	-	-	ST	sum
12	-	-	-	ST

(b)

cycle	PE 0	PE 1	PE 2	PE 3
0	LD +3	-	-	-
1	LD +1	LD +3	-	-
2	LD 0	LD +1	LD +3	-
3	* C0	LD 0	LD +1	LD +3
4	LD +2	* C0	LD 0	LD +1
5	* C1	LD +2	* C0	LD 0
6	* C2	* C1	LD +2	* C0
7	* C3	* C2	* C1	LD +2
8	sum	* C3	* C2	* C1
9	ST	sum	* C3	* C2
10	-	ST	sum	* C3
11	-	-	ST	sum
12	-	-	-	ST

(c)

Figure 2. Schedule of a 4-tap filter on the SIMD architecture of Figure 1 without (a) and with (b) delay line in the instruction distribution. (c) A valid schedule without any conflicts.

this value as an input for the ALU, and compute and write the result in a register in one clock cycle. The clock cycle time is determined by the maximum length of the critical path which is the time of the computation plus communication over the PEs:

$$\begin{array}{l} T_{clockcycle} = T_{computation} + T_{communication} \ \ \& \\ T_{computation} = T_{read-reg} + T_{ALU} + T_{write-reg} \end{array} \quad (1)$$

$T_{computation}$ is constant but $T_{communication}$ depends on the architecture configuration. Changing the maximum neighborhood communication distance k, allows to tune the clock frequency for different applications ($T_{communication} = k * T_{mux_{2\times1}}$). In CMOS 0.18 micron, the delay of a multiplexor and computation are about 0.11 ns and 2.5 ns, resp. [7, 1]. Section 5 shows that by changing k, it is possible to tune the architecture for different algorithms to improve performance.

4 Programming

Configuration All PEs are controlled by a control processor that reads the program from program memory and distributes instructions to all PEs. Before distributing instructions to PEs the control processor has to configure the multiplexors (I0, I1,...) based on the k which is determined by the compiler and also set the clock frequency to achieve the best performance for each kernel. For programming the PEs, we add two instructions to the instruction set: a) RESET: By receiving this instruction, each PE configures its multiplexor (I0, I1,...) to receive instructions from the undelayed instruction bus. Therefore, all PEs can receive the new configuration at the same time. b) SET: The control processor sends the value of k via this instruction. Each PE compares the value of k to its PE-ID (the number of the PE: 0 for PE0, 1 for PE1,...; hence each PE needs one hard-wired register to store its PE-ID). If the PE-ID mod k equals zero (multiple of k), the PE configures its multiplexor to receive its instructions from the un-delayed bus; otherwise, it receives them from the delayed bus.

RC-SIMD supports reconfiguration at run-time, e.g., in between different kernels in one application. Before sending the instructions for a new kernel to PEs, the control processor sends the RESET instruction to reset the multiplexors and then sends the SET instruction to configure multiplexors. The number of cycles needed to program the multiplexors is:

$$Ncycles_{prog} = Ncycles_{RESET} + Ncycles_{SET} \quad (2)$$

In most kernels, we observed that, the maximum neighborhood communication is less than 25, so the number of cycles spent on receiving the RESET instruction by all the PEs is less than 25 (due to the delay registers in the instruction bus). Furthermore, the SET instruction (PE-ID *mod* k) needs about 10 cycles. Therefore, the total cycle overhead for programming the multiplexors is usually less than 35 cycles which is far less than 1% of the typical execution time of kernels.

Timing The compiler determines for each loop kernel an appropriate neighborhoods size k. Using equation 1, it computes the corresponding clock frequency. With regard to clock generation, we assume that an RC-SIMD processor is applied in a system-on-chip with multiple components. In these systems a master clock is generated (using a PLL) with a very high clock frequency. Individual clock signals for the different components are derived from the master clock using clock division. Especially power-aware systems have the means to scale down the clock by adjusting the divider. This adjustment is typically performed in 1 or 2 clock cycles. We assume that we have the same means at our disposal, and for that reason do not describe clock generation or adjustment in more detail.

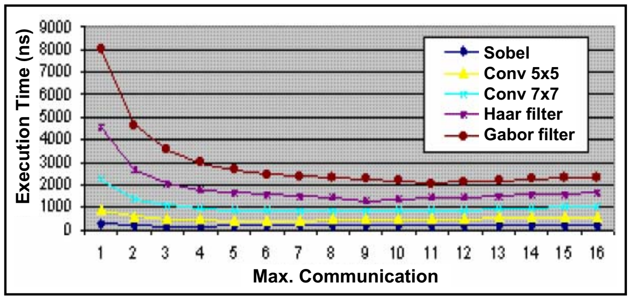

Figure 3. Performance for different k values.

Kernel	Best execution time (ns)	Execution time when k=6 (ns)	Improvement (%)
Sobel	158.48	176.96	10.4
Conv 5x5	402.6	417.12	3.5
Conv 7x7	784.8	853.2	8.0
Haar filter	1326.2	1561.04	15.0
Gabor filter	2047.97	2471.12	17.1
Average			***10.8***

Table 1. Performance improvement.

5 Experiments

As performance benchmarks, we chose several frequently used communication kernels from the image processing domain (Haar filter, sobel, convolution, Gabor filter) [2]. Each kernel was implemented on RC-SIMD architectures with different maximum neighbor communication $k, 0 < k \leq 16$. Our compiler [5] inserts fetch and put instructions to segment communication when it exceeds k. Figure 3 shows the results. The best performance for a kernel is when k equals the maximum neighbor communication in the kernel. By changing the value of k, it is possible to reconfigure RC-SIMD to get a better performance for different kernels. Table 1 shows that, by using RC-SIMD and tuning the clock frequency, it is possible to achieve a performance gain (on average) of more than 10% in comparison with a fixed clock frequency non-reconfigurable architecture. The reference configuration has $k = 6$, which is the average of the best points for all kernels.

The reconfigurable part of RC-SIMD includes the multiplexors I0, I1,... and the one-bit flip-flops to store the multiplexor configuration during the execution of each kernel. Figure 4 shows the area overhead of the reconfigurable architecture in comparison with the non-reconfigurable architecture with the number of PEs ranging from 1 to 256. It shows that the overhead of the reconfigurable architecture is less than 2%.

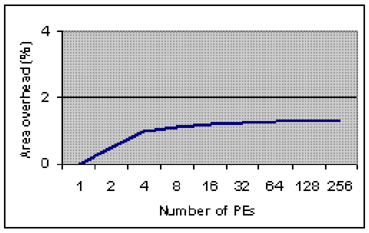

Figure 4. Area overhead.

6 Conclusions

We examined the (re-)configuration aspects of RC-SIMD, a new SIMD architecture proposed to solve the communication bottleneck in current SIMD architectures, caused either by timing overhead in locally connected architectures or area overhead in more richly connected architectures. RC-SIMD solves the bottleneck with a set of delay lines in the instruction bus, distributing bus accesses over time. We can (re-)configure the amount of unit-delay (per PE) via a simple mechanism that adds less than 2% area. Since different configurations correspond to both different schedules and different clock frequencies, each configuration represents a trade-off between cycle time and number of cycles. We have demonstrated the process of (re-)configuration, the required hardware, and the extension to the instruction-set of the PEs. Experiments show (on average) more than 10% performance improvement over a non-reconfigurable architecture.

References

[1] A. Abbo and R. Kleihorst. Smart Cameras: Architectural Challenges. In *Proceedings of ACIVS*, pages 6–13, Gent, Belgium, September 2002.

[2] W. Caarls, P. Jonker, and H. Corporaal. Benchmarks for SmartCam Development. In *Proceedings of ACIVS*, pages 81–86, Gent, Belgium, September 2003.

[3] K. v. Eijk, B. Mesman, A. A. Carlos Pinto, and Q. Zhao. Constraint Analysis for Code Generation: Basic Techniques and Applications in Facts. *ACM Transactions on Design Automation of Electronic Systems*, 5(4):774–793, October 2000.

[4] H. Fatemi, H. Corporaal, T. Basten, R. Kleihorst, and P. Jonker. Designing Area and Performance Constrained SIMD/VLIW Image Processing Architectures. In *Proceedings of ACIVS*, pages 689–696, Antwerp, Belgium, September 2005.

[5] H. Fatemi, B. Mesman, H. Corporaal, T. Basten, and R. Kleihorst. RC-SIMD: Reconfigurable Communication SIMD Architecture for Image Processing Applications. *Journal of Embedded Computing*, To appear in 2006.

[6] H. Hofstee. Power Efficient Processor Architecture and The Cell Processor. In *Proceedings of 11th International Conference on HPCA*, pages 258–262, San Francisco, CA, February 2005.

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