RC-SIMD: Reconfigurable Communication SIMD Architecture for Image Processing Applications

Hamed Fatemi[1,2], Bart Mesman[2], Henk Corporaal[2], Twan Basten[2], and Richard Kleihorst[3]

[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,
Tel: (+31)-40-2474741, Fax: (+31)-40-2433066.

Abstract. During the last two decades, Single Instruction Multiple Data (SIMD) processors have become important architectures in embedded systems for image processing applications. The main reasons are their area and energy efficiency. Often the processing elements (PEs) of an SIMD processor are only locally connected. This may result in a communication bottleneck (only access to direct neighbors). One way to solve this is to use a fully connected communication network (FC-SIMD) between PEs. However, this solution leads to an excessive communication area cost, low communication network utilization, and scalability problems. E.g., the area overhead of an FC-SIMD is more than 100% when the number of PEs gets bigger than 64.
In this paper, we introduce a new type of SIMD architecture, called RC-SIMD, with a reconfigurable communication network. It uses a delay-line in the instruction bus, causing the accesses to the communication network to be distributed over time. This architecture requires only a very cheap communication network while performing almost the same as expensive FC-SIMD architectures.
However, the new architecture causes irregular resource conflicts. We therefore introduce a conflict model that existing schedulers are able to cope with. Experimental results show that, on average (compared to locally connected SIMDs), RC-SIMD require 21% fewer cycles than architecture without the delay-line, while the area overhead is at most 10%.

Keywords: Reconfigurable architectures, SIMD processors, Scheduling, Image processing algorithms, Parallel architectures, Low cost architectures.

1 Introduction

In the last few years, media processing has become one of the dominant computing workloads. Media processing refers to the computation required for the

creation, encoding/decoding, manipulation, display, and communication of digital multimedia information such as images, audio, video, and graphics. Major obstacles for implementing these applications on a general-purpose processor are their high computational demands, the stringent real-time, scalability, and low power consumption constraints [8].

Parallel architectures help to solve these problems to some extent. Therefore, there has been an increasing demand to exploit parallelism in applications. It is possible to explore parallelism along three axes: task-level parallelism (TLP), instruction-level parallelism (ILP) and data-level parallelism (DLP, or vector parallelism). Trimedia [16] is an example of a VLIW architecture which can be used to exploit ILP in applications. The bottlenecks of VLIW architectures are concerned with the code-size, register file cost, and the costly communication network between functional units. The Cell-based architecture [9] is an example of MIMD (Multiple Instruction Multiple Data). It exploits both TLP (between the processing units) and, to a limited extent, DLP (inside the vector units). SIMD architectures exploit DLP, usually to a very large extent. E.g., the XETAL [1] architecture has 320 parallel operating PEs, such that it can process (almost) complete image lines at once.

Although the SIMD concept has troubled application development and mapping, it offers undeniable advantages in control efficiency (e.g., 1 instruction word for 320 parallel PEs [1]) and its repetitive architecture relieves floorplan and lay-out design. SIMD architectures are cheaper in area and energy (in comparison with e.g., MIMD and VLIW), assuming that applications fit to this type of parallelism. The latter is true for the image processing domain due to the huge amount of DLP inherent in pixel-type operations. Nowadays, SIMD-based processors are available to offer a wide range in the number of PEs working in parallel.

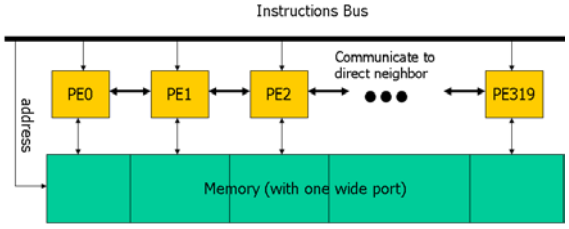


Fig. 1. Locally connected SIMD (LC-SIMD) architecture: each PE can only communicate with direct neighbors, and has only access to its (private) memory slice.

Although the utilization of the memory and PEs in SIMD architectures is essentially determined by the vectorizing compiler, the utilization of the communication resources is much more architecture dependent. We illustrate this with the two classes of communication architectures that existing SIMD processors employ. The first one is the locally connected SIMD (LC-SIMD) architecture (Fig. 1). Each PE can receive data of a single column in a video frame and PEs can work on the same video line in parallel. Furthermore, each PE can get data from its direct neighbors. However, many image processing algorithms (like FFT, image sub-sampling, and filtering) need to get data not only from direct

neighbors, but also from more distant PEs. Suppose that a PE needs to get some data from another PE that is 6 blocks away (i.e., 5 intermediate PEs) to its right; then each PE has to shift 5 times and only then the PE will be able to load the data from its direct neighbor. These shifts severely reduce performance. So, this communication architecture is cheap but rather inefficient.

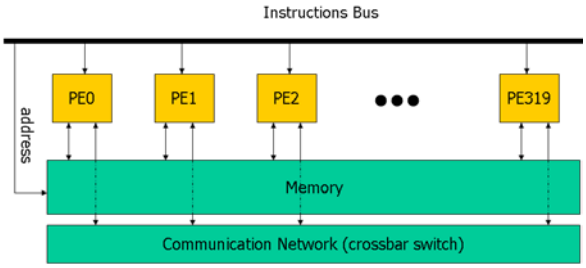


Fig. 2. Fully connected network SIMD (FC-SIMD) architecture: each PE is connected to all other PEs (simultaneously).

To overcome this inefficiency, one can separate computation and communication, as illustrated in Fig. 2, and implement a fully connected SIMD (FC-SIMD). Each PE can send/receive data to/from all other PEs by means of a fully connected crossbar switch. The problem of this architecture is that when the number of PEs increases too much (e.g., more than 64 PEs), the area of this communication network dominates the whole chip area [5]. So this is not feasible for massively parallel SIMDs. Between these two extremes (locally and fully connected) exist many other communication network solutions [11]. However they are difficult to use efficiency when using an SIMD execution mode, due to communication conflicts.

In this paper, we introduce a novel solution for communication and instruction distribution in highly parallel SIMD architectures that overcomes the communication bottleneck present in current SIMD processors. The resulting architecture is called RC-SIMD, because it has a reconfigurable communication network. The essential idea is that the instructions for successive processing elements are distributed over time, resulting in a time-interleaved and therefore, more balanced utilization of communication and computational resources. The cost of the proposed architecture is almost the same as LC-SIMD. Experiments on RC-SIMD show that the schedules for our benchmarks often perform well as on an FC-SIMD.

Furthermore, we show that RC-SIMD is reconfigurable, allowing for run-time adaptation to the characteristics of an image processing algorithm. RC-SIMD however, poses new resource constraints that cannot be handled by currently available scheduling algorithms. Therefore we also developed a resource conflict model for RC-SIMD which can be used by any scheduler. Experiments with our scheduler result in high-quality software-pipelined schedules, featuring a very good network utilization without much shifting overhead, for image processing loop kernels that are present in the most demanding image processing applications.

The rest of the paper is organized as follows: In Section 2, we give an overview of related work. The RC-SIMD architecture is explained in Section 3. Section 4 devises a suitable conflict model for the architecture that can be used by any scheduler. We present our experimental results in Section 5 and conclusions in Section 6.

2 Related work

Several commercial SIMD machines were introduced in the 1970s [13], but they were not widely used. Interest in this class of machines was renewed in the early 1980s with the introduction of the ILLIAC IV [10], the Connection Machine (CM-1) [17], and the MasPar MP-1 [6].

The ILLIAC IV was an SIMD computer for array processing. It included 64 PEs. An 8 by 8 grid interconnect joined each PE to 4 neighbors. Non-neighbor communication requires extra shifts. The CM-1 was based on 1-bit processors. 16 processors were implemented on a single chip. Within a chip, processors were connected through a 4-by-4 grid, and up to 4096 chips were connected via a 12-dimensional hypercube, which causes quite some communication area overhead. The MasPar MP-1 was introduced a few years after the CM-1. It also had a very narrow datapath, but it could process 4 data bits instead of 1 bit at a time. One of the interesting aspects of the MP-1 is that, there are two separate communication systems, and programmers can alternate between them to choose the best performance for different parts of their algorithms. One interconnection network is known as X-net. It connects each processor to its 8 nearest neighbors in a 2D torus (potentially causing shift overhead). The other connection is a global router, which provides point-to-point communication between each two PEs. The router is implemented by a 3-stage switching network, where each stage in a 1024-processor machine contains a crossbar, together, the three stages comprise a crossbar (which causes area overhead and has lower speed).

XETAL [1] and IMAP [7] are more recent and interesting SIMD processor examples (for video processing). They consist of 320 and 256 PEs, respectively, arranged as a 1-dim, linear array. Each PE has only the ability to access its neighbors (left and right) and when it wants to get some data from its $n'th$ neighbor ($n > 1$), the corresponding data should be shifted to become accessible. The communication bottleneck is essentially the same as in Fig. 1, which may cause a significant increase in the cycle count of the program.

Imagine [12] is an SIMD which consists of 8 PEs (each PE is a VLIW). Each PE has the ability to get data from all PEs using a fully connected network. This architecture is not scalable. If the number of PEs is increased beyond 64 (supporting increased data-level parallelism), the area related to this communication network will dominate the total area [5].

It seems that an SIMD processor needs either many cycles to perform non-neighbor communication, or a very rich interconnection structure with a high cost. The latter possibly results into high latency and energy consumption. In

contrast we propose a new reconfigurable architecture and a scheduler which aims at providing higher connectivity without the additional cost incurred.

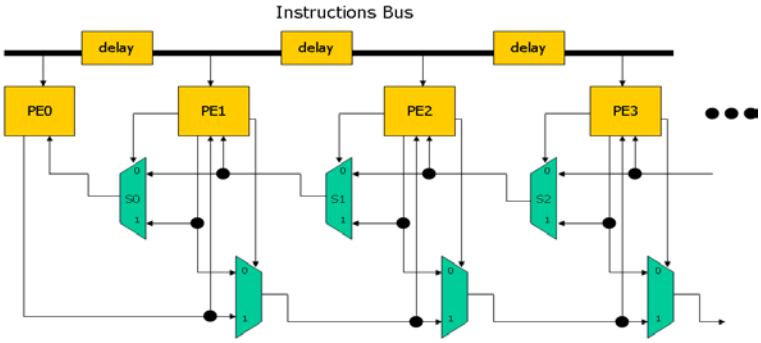


Fig. 3. Initial reconfigurable communication SIMD (RC-SIMD) architecture.

3 A reconfigurable SIMD

In this section, we propose a new SIMD architecture for solving the communication problems between PEs which are explained in the previous sections.

3.1 Basic architecture

As outlined in the previous sections current SIMD architectures are either locally connected (LC-SIMD) or have a rich and almost fully connected inter-PE communication network (FC-SIMD). The first ones may cause shift overheads when performing non-local communication while the latter are not scalable. Fig. 3 shows our architecture that can overcome this communication bottleneck. In this architecture, there are two segmented unidirectional communication busses (bottom of Fig. 3), one for right and the other for left communication. The segmented bus allows multiple concurrent communications by effectively using one bus. By controlling the multiplexors, each PE can choose to forward the data which is already on the bus, or to put itself new data on the bus. This allows e.g. to have communication from PE1 to PE3 and from PE3 to PE5 over different segments of the same bus at the same time.

Fig. 4(a) shows an SIMD schedule for a 4-tap filter. LD+2 and LD-2 operations denote acquiring data from the second neighbors to the right and to the left, respectively. However, this schedule is invalid because, for example, at (clock) cycle 4, PE0 uses multiplexors S0 and S1 and PE1 uses S1 and S2. So both PEs (0 and 1) want to use S1 at the same time (with different input).

This problem is caused by the fact that all PEs, when organized in an SIMD fashion, need to access remote PEs (not only direct neighbors) in the same cycle. This is essentially due to the SIMD distribution of instructions to PEs. Although sufficient communication capacity may be available, even for remote communication, we can not use it for high peak performance.

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)

Fig. 4. Schedule of a 4-tap filter on the SIMD architecture of Fig. 3, (a) without and (b) with delay line in the instruction distribution. The schedule for (a) is invalid because of the communication bottleneck. Note that an FC-SIMD also solves this problem.

We solve this problem by executing the same SIMD instruction on different PEs at different points in time. We put a delay line in the instruction distribution (top of Fig. 3). Successive PEs receive the same instruction (like in SIMD), but in successive cycles. Fig. 4(b) shows the 4-tap filter schedule on the architecture, with the instruction delay line activated. There is no communication conflict in cycle 4. This architecture has the advantages (small code-size, simple communication) of the LC-SIMD architecture, but overcomes the communication problem.

The most important parameter for measuring performance in image processing algorithms is the initiation interval (the interval between the start time of successive iterations), which is the same (10 cycles in this example) in both cases of Fig. 4 (only the latency of the schedule of Fig. 4 (b) is longer than Fig. 4 (a)). Although this delay line solves the fundamental communication problem, it introduces some other unexpected resource conflicts. E.g., in Fig. 4(b) PE0 and PE2 use multiplexor S2 at the same time in cycle 6. We deal with these conflicts in Section 4.

3.2 Updated architecture

There is still another problem with our proposed architecture as shown in Fig. 3. If we have an SIMD processor which can operate on a whole (or part of a) video line by using e.g. 320 PEs, it takes 320 cycles for PE319 to execute its first instruction. This is often not acceptable, not only because of the extra latency. It would also mean that we have to store more image lines. when PE319 still operates on the first line, PE0 may already be far ahead in the image. Therefor, we adapt the architecture as shown in Fig. 5. By adding multiplexors in the instruction delay line each PE can be configured to receive either the instruction delayed by the previous PE or the un-delayed instruction from the instruction bus. This brings a major performance improvement (it also reduces conflict graph and therefore, scheduling complexity). Suppose that the maximum neighborhood communication in an algorithm is k, we then configure the multiplexors I0,

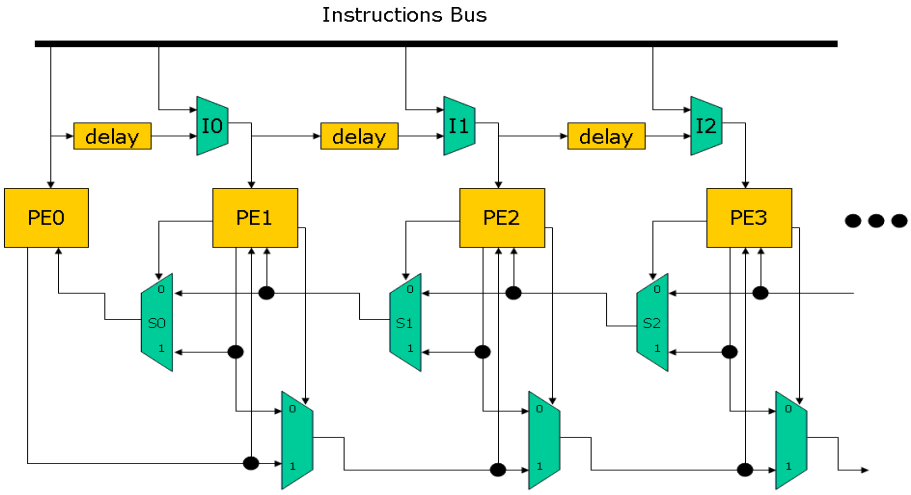


Fig. 5. Update of the RC-SIMD architecture, adding configurable instruction delay.

I1,.....I318 such that PEn receives the instructions with $n \ mod \ k$ cycles delay. Fig. 6 shows the instruction distribution when $k = 4$.

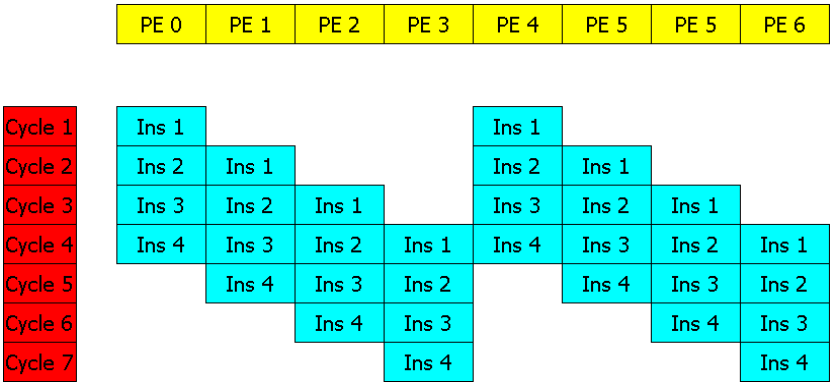


Fig. 6. Instruction distribution when the maximum neighborhood communication is 4.

3.3 Flexible clock frequency

Most segmented busses contain registers or latches at equidistant intervals. In this way the clock frequency is upper bounded by the length of the maximum wire segments between successive registers. In general, more registers imply shorter wires and a higher clock frequency. There are no registers or latches in the communication busses of our RC-SIMD architecture (Fig. 5). The clock frequency is upper bounded by the longest delay through the segments and multiplexors (the maximum neighborhood communication).

This shows an interesting novel aspect of our RC-SIMD architecture. By changing the maximum neighborhood communication, it is possible to change the clock frequency for different algorithms. For instance, if an algorithm needs less

communication, it is possible to reduce the maximum neighborhood communication and increase clock frequency to achieve better performance. Alternatively we could reduce the voltage for lower energy consumption. The communication network is reconfigurable and this can be exploited by the compiler. Our compiler is parameterizable with respect to the maximum neighborhood communication distance (k). If a longer communication distance ($> k$) is needed, our compiler automatically inserts fetch and put instructions to segment communication (e.g. if $k = 6$ and we need to communicate to the 11th neighbor, the compiler segments this communication into two communications with 6 and 5). We estimate that for a 200 MHz clock in .13 micron technology, a maximum neighborhood communication of 6 PEs is achievable.

4 Automatic scheduling

As illustrated in Section 3, our proposed RC-SIMD architecture can potentially solve the fundamental communication problem explained in the introduction by using different bus segments concurrently, while still operating in SIMD mode. However, this causes other resource conflicts, as shown in Fig. 4(b). In cycle 6 of the shown schedule PE0 and PE2 both require the same resource, multiplexor S2 (also in cycle 7, between PE1 and PE3).

If the PEs would operate independently, like in a VLIW, a parallel scheduler can easily solve the resource conflicts. However, in our RC-SIMD architecture an instruction arriving at PE0 arrives at PE2 two cycles later. Therefore, scheduling operations on PE0 necessarily implies scheduling the same operations on PE2 two cycles later. These type of constraints are not handled by a parallel VLIW scheduler. Furthermore, it is not necessary to treat the PEs independently (i.e., they only receive delayed or un-delayed instructions but in the same order). So by modelling the resource constraints carefully for a scheduler, it is possible to schedule only for a single PE and automatically derive the schedule for other PEs. The overall schedule is still in the SIMD fashion (only a single instruction stream is required) thus, it shares all the advantages of an SIMD.

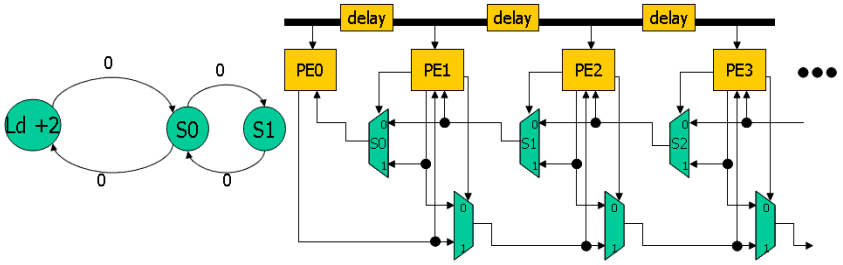


Fig. 7. The resource model for a LD+2 operation.

4.1 Conflict model

To model the resource conflicts for the scheduler for a single PE, it is necessary to consider which exact data communication resources (i.e., the multiplexors)

are used in each cycle. The resource model of a LD+2 instruction is shown in Fig. 7. A node in this graph represents the use of a resource. Edges represent timing dependencies. An edge with delay d implies that the destination node of the edge has to be scheduled at least d cycles after the source node. Fig. 7 shows that the nodes are all connected with edges with zero delay (LD+2 in PE0 uses multiplexors S0 and S1 in the same cycle). Fig. 8 extends this conflict model to include the resource use in the succeeding cycle. In Fig. 8, two extra resource uses are added, S1 and S2. The edges with delay 1 and delay -1 together imply that S1 and S2 are used at least and at most 1 cycle after S0 and S1, respectively. This conflict model describes the following steps:

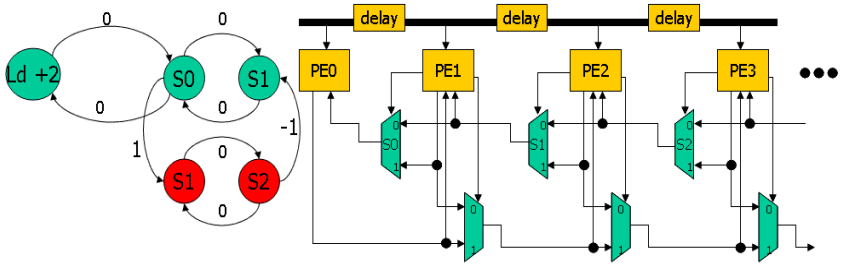


Fig. 8. The extended resource model for a LD+2 operation.

- At cycle n: Multiplexors S0 and S1 are set to pass the value from PE2 to PE0, and PE0 loads the value from the bus.
- At cycle n+1: Multiplexors S1 and S2 are set to pass the value from PE3 to PE1. PE1 loads the value from the bus, but this is not explicit in the conflict model (because we only consider PE0 as mentioned earlier).

Similarly, multiplexors S2 and S3 are occupied at cycle n+2, S3 and S4 at cycle n+3, and so on, till the maximum neighborhood communication.

Note that the conflict model should cover the conflicts over two segments of the maximum neighborhood communication size in order to take into account the resource conflicts over the borders of two segments of the maximum neighborhood communication (for left or right communication). So given a maximum neighborhood communication k, the scheduler now generates a conflict model that considers conflicts up to $2k$ cycles, and the scheduler guarantees a valid schedule for this particular configuration.

When all operations are scheduled on PE0, the schedule for all other PEs is derived by applying the appropriate delay. It is guaranteed that no resource conflict arises between the PEs, and the schedules for all PEs are valid. Fig. 9 shows the final valid schedule for our 4-tap filter. In Section 5 we use this conflict model for our scheduler to schedule several image processing loop kernels.

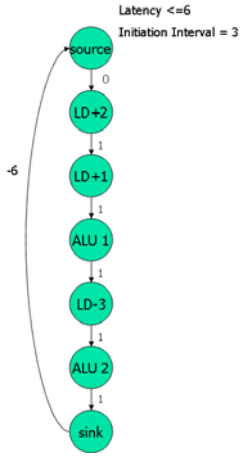
4.2 Facts tools

Although the conflict model introduced in the previous section is suitable for any resource-constrained scheduler, it fits very well with our in-house developed

cycle	PE 0	PE 1	PE 2	PE 3	PE 4	PE 5	PE6	PE 7
0	LD +3	-	-	-	LD +3	-	-	-
1	LD +1	LD +3	-	-	LD +1	LD +3	-	-
2	LD 0	LD +1	LD +3	-	LD 0	LD +1	LD +3	-
3	* C0	LD 0	LD +1	LD +3	* C0	LD 0	LD +1	LD +3
4	LD +2	* C0	LD 0	LD +1	LD +2	* C0	LD 0	LD +1
5	* C1	LD +2	* C0	LD 0	* C1	LD +2	* C0	LD 0
6	* C2	* C1	LD +2	* C0	* C2	* C1	LD +2	* C0
7	* C3	* C2	* C1	LD +2	* C3	* C2	* C1	LD +2
8	sum	* C3	* C2	* C1	sum	* C3	* C2	* C1
9	ST	sum	* C3	* C2	ST	sum	* C3	* C2
10	-	ST	sum	* C3	-	ST	sum	* C3
11	-	-	ST	sum	-	-	ST	sum
12	-	-	-	ST	-	-	-	ST

Fig. 9. A valid schedule for a 4-tap filter in RC-SIMD.

Facts tool [4], which has been transferred to Silicon Hive [14] for application in their core scheduler. The reason is that Facts has been developed in the past to schedule for architectures and applications that are highly constrained with respect to available resources (both computational and storage) and timing (both throughput and latency). This is certainly the case for image processing applications running on SIMD architectures. Unlike a VLIW, for an SIMD, a NOP on one processing element implies a NOP on all processing elements. In order to obtain a high resource utilization, we apply software pipelining. Most schedulers apply greedy scheduling techniques that are good for latency, but not for throughput. We illustrate this with the scheduling example in Fig. 10(a). Assume that the Initiation Interval (II) should be 3 cycles, meaning a new iteration should start every 3 cycles. Furthermore, the latency should not exceed 6 cycles. This is modelled in the graph by a timing edge from the sink to the source with delay -6. A PE in RC-SIMD architecture can perform a single ALU operation in parallel with a single load from a bus. For educational purposes, we omit the extensive conflict model from the previous section, and focus on scheduling a single processing element.



(a)

cycle	Iteration 1	Iteration 2
1	LD+2	-
2	LD+1	-
3	ALU1	-
4	-	LD+2
5	-	LD+1
6	LD-3	ALU1
7	ALU2	-
8		-
9		LD-3
10		ALU2

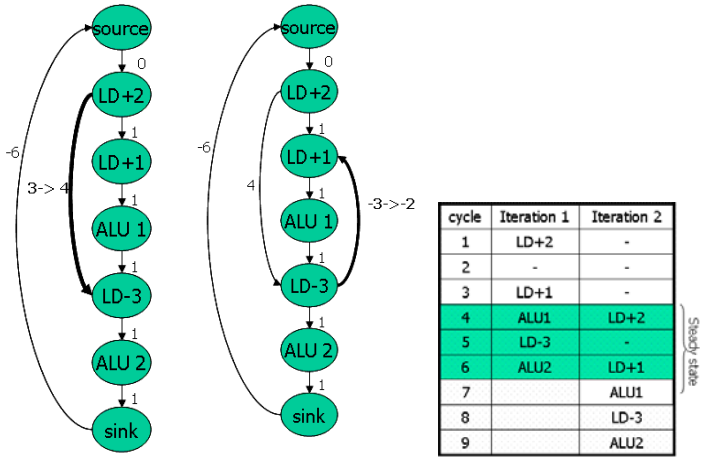
(b)

Fig. 10. (a) Schedule example for Facts and (b) Incorrect schedule.

As indicated in Fig. 10(b), a greedy scheduler would schedule as follows: The LD+2 at cycle 1, the LD+1 at cycle 2, and the ALU1 at cycle 3. Cycle 4 of the first iteration coincides with cycle 1 of the second iteration, since we start a new iteration every 3 cycles. So the LD-3 cannot be scheduled at cycle 4 since it would coincide with the LD+2 of the next iteration. The LD-3 can also not be scheduled at cycle 5 since it would coincide with the LD+1 of the next iteration. So the LD-3 is scheduled at cycle 6, and the ALU2 at cycle 7. One iteration now takes 7 cycles, thus violating our latency constraint of 6 cycles.

Our Facts scheduler is based on analyzing constraints rather than just greedy scheduling. We do not give a full account of Facts' constraint analysis capabilities, but using the above example we demonstrate a few techniques that turn out to be especially useful for the RC-SIMD architecture. The basic operation of Facts is to compute minimal timing distances between instructions over paths implied by edges in the data-flow graph (DFG). If a minimum timing distance between two instructions is infeasible, it will be increased by one cycle. We demonstrate this for the example above in three steps.

– Step 1: In the first step, we compute the distance in the DFG from LD+2 to LD-3 via the LD+1 and ALU1 instructions. This distance is 1+1+1=3 cycles. It is, however, not feasible to actually schedule the LD+2 and LD-3 exactly 3 cycles apart: because the Initiation Interval is 3, the LD-3 of the first iteration coincides with the LD+2 of the second iteration, and we can only perform one bus load per cycle. Since the minimum distance of 3 is infeasible, it is increased to 4, see Fig. 11 (a).



cycle	Iteration 1	Iteration 2
1	LD+2	-
2	-	-
3	LD+1	-
4	ALU1	LD+2
5	LD-3	-
6	ALU2	LD+1
7		ALU1
8		LD-3
9		ALU2

Steady state

Fig. 11. (a) DFG transform because of conflict LD+2 vs. LD-3, (b) DFG transform because of conflict LD-3 vs. LD +1 and (c) Correct schedule.

– Step 2: In the second step, we compute the distance in the DFG from the LD-3 to the LD+1 via the ALU2, sink, source, and LD+2 instructions. This distance is 1+1-6+0+1=-3 cycles. It is however, not feasible to actually schedule the LD-3 and the LD+1 exactly -3 cycles apart: again, the LD-3 of

the first iteration coincides with the LD+1 of the second iteration. Since the minimum distance of -3 is infeasible, it is increased to -2 cycles, as indicated in Fig. 11 (b).
- Step 3: In the last step, we simply combine the results of step 1 and 2. Using the two added edges, we find a path from LD+2 via LD-3 to LD+1 of 4-2=2 cycles. We conclude that the LD+2 and LD+1 instructions should be at least two cycles apart. Since there are no instructions to fill the gap, a gap remains at cycle 2. The resulting schedule is depicted in Fig. 11 (c). It is easy to verify that this schedule satisfies all constraints.

For a more extensive explanation of Facts we refer to [4]. Although we did not demonstrate the constraint-analysis techniques in the context of the conflict model of the previous section (the DFG would not even fit on one page) we hope that the reader understands that Facts treats all the resources in the conflict model in a way similar to the LD+1, LD+2, and LD-3 instructions in the example. Facts will find highly constrained software-pipelined schedules for all processing elements in the RC-SIMD architecture, taking into account the effects of the instruction delay-line, as well as the effects of communications that cross neighborhood borders, as modelled in the previous section.

5 Evaluation

In Section 3 and 4, we explained the RC-SIMD architecture and its scheduler. In this section, we evaluate RC-SIMD respect to performance and area.

5.1 Cycle count comparison

To compare the proposed architecture with an LC-SIMD architecture (like XETAL and IMAP) and an FC-SIMD (like Imagine), we selected several frequently used communication kernels from image processing applications (FFT, convolution, sub-sampling, and Haar filter) [3]. We use a modified version of the Facts tools which includes the resource conflict model for multiplexors (We assumed maximum 6 neighborhood communication in our template and the clock frequency is the same in all architectures).

Kernel	Num operations	Num cycles in LC-SIMD	Num cycles in FC-SIMD	Num cycles in RC-SIMD	Comm. Overhead in LC-SIMD	Cycle Improvement (compared to LC-SIMD)
4-tap filter	8	10	8	8	2	20%
Image sub-sampling	21	26	21	21	5	19%
Convolution 7x7	98	126	98	98	28	22%
Haar filter	162	216	162	162	54	25%
FFT	26	-	26	28	-	-

Table 1. Cycle-count improvement, comparing LC-SIMD, FC-SIMD and RC-SIMD.

Table 1 shows the results of implementing 4 kernels on different SIMD architectures (LC-SIMD, FC-SIMD and RC-SIMD architectures). The second column of Table 1 represents the number of operations per pixel for each of these kernels; columns 3 to 5 show the number of cycles per pixel when these kernels are implemented on LC-SIMD, FC-SIMD, and RC-SIMD, respectively. The implementation of these kernels on an LC-SIMD architecture (Fig. 1) leads to a communication overhead, because of the limited access to the direct neighbors only (6'th column in Table 1). This overhead decreases the performance in the kernels. The proposed RC-SIMD architecture in Fig. 5 solves this problem by providing access to remote neighbors. The last column of Table 1 shows that RC-SIMD schedules have a better cycle-count than LC-SIMD schedules, with an average improvement of 21%. From a cycle-count point of view, an FC-SIMD has hardly any advantage over RC-SIMD, at least not for the used kernels.

We have also implemented the FFT kernel on RC-SIMD, which is not possible to be implemented efficiently on an LC-SIMD like XETAL and IMAP (because each PE may require access to other PEs at different distances at the same time, e.g., one requires access to PE+2 while another one to PE+3). To compare RC-SIMD with an FC-SIMD (like Imagine), we implemented a radix-2 FFT butterfly, decimation in frequency [2], with 1024 points. The implementation shows that RC-SIMD has less than 8% cycle $((28-26)/26)$ overhead in comparison to an FC-SIMD.

Note that the required conflict model extensions add less than 5% to the scheduling time.The schedule time is less than 2 minutes for each benchmarks (on an Intel Pentium processor running at 1.70 GHz).

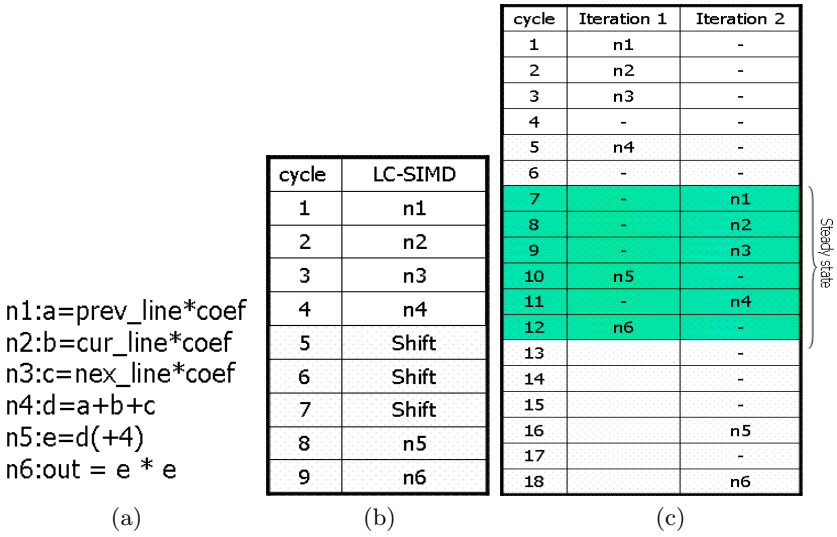
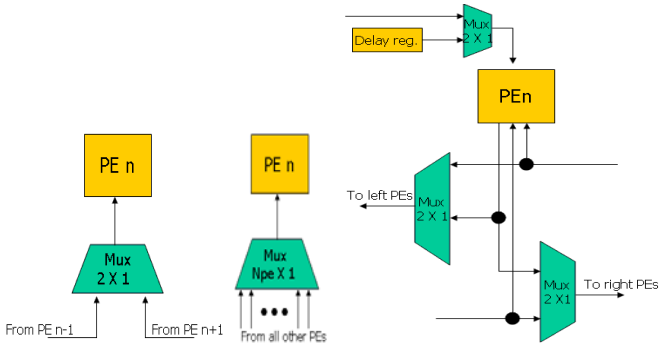


Fig. 12. (a) Example for dependency loop kernel, (b) schedule for the LC-SIMD and (c) schedule for RC-SIMD with initiation interval 6.

5.2 Dependency kernels

Some SIMD image processing kernels require that, each PE gets the result of one of its immediately preceding instructions from some distance PEs. For example, in Fig. 12 (a), to execute instruction n5, each PE needs to load the result of n4 from the PE located 4 PEs to its right. Fig. 12(b) shows the schedule of this loop kernel when it is implemented on an LC-SIMD. It needs 9 cycles (instead of 6 cycles) to start the next iteration (i.e., its initiation interval is equal to 9), each PE requires to shift 3 times to access the result from its direct neighbor. In RC-SIMD, PE0, for instance, needs to wait 4 cycles to load the result of instruction n4 in PE4 (because of the delay line in the instruction bus, see Fig. 5). However, our scheduler can solve this problem by finding a schedule (Fig. 12(c)) with an initiation interval of 6 cycles (equal to the number of instructions). This gives the same throughput as an FC-SIMD. The only disadvantage is that in RC-SIMD, the latency of each iteration is 12 cycles. This is usually less important in image processing kernels.



(a) (b) (c)

Fig. 13. The multiplexor area overhead for each PE in (a) LC-SIMD, (b) FC-SIMD, (c) RC-SIMD.

5.3 Area estimation

In order to compare the area of the three considered SIMD architectures (LC-SIMD, FC-SIMD, and RC-SIMD), we use the area model which is proposed in [5]. The area of an LC-SIMD (Fig. 13(a)) includes the area of PEs and multiplexors for acquiring the data from direct neighbors (Eq. 1) (A and N represent area and number).

$$A_{LC-SIMD} = N_{PE} * A_{PE} + N_{PE} * A_{mux2} \tag{1}$$

The area of an FC-SIMD (Fig. 13(b)) includes the area of PEs and the area of a crossbar switch (Eq. 2)(growing with $(N_{PE})^2$).

$$A_{FC-SIMD} = N_{PE} * A_{PE} + N_{PE} * A_{muxNpe} \tag{2}$$

The area of RC-SIMD includes the area of PEs, multiplexors for data communication between PEs, delay registers in the instruction path, and multiplexors in the instruction path (Eq. 3) (Fig. 13(c)).

$$A_{RC-SIMD} = N_{PE} * A_{PE} + 2 * N_{PE} * A_{mux2} + (N_{PE} - 1)(A_{delay-register} + A_{mux2}) \quad (3)$$

The used parameters for the area model are described in Fig. 14(a) [15]. We assume PEs are the same for all architectures. The first PE in RC-SIMD does not need a delay register and multiplexor in the instruction bus.

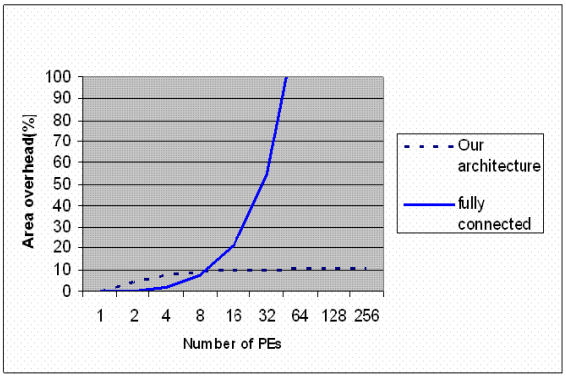
Fig. 14(b) shows the area-overhead (Eq. 4) of an FC-SIMD and RC-SIMD in comparison with an LC-SIMD (the horizontal axis shows the number of PEs in each architecture).

$$overhead_{FC-SIMD} = (A_{FC-SIMD} - A_{LC-SIMD})/A_{LC-SIMD}$$
$$overhead_{RC-SIMD} = (A_{RC-SIMD} - A_{LC-SIMD})/A_{LC-SIMD} \quad (4)$$

When the number of PEs equals 1, Fig. 14 (b) shows that all architectures have the same area (because communication between PEs is not needed). The LC-SIMD and FC-SIMD cover a smaller area than RC-SIMD when the number of PEs is less than 8 PEs because of the delay line and the multiplexors in the instruction bus. When the number of PEs increases to more than 8, Fig. 14(b) shows that RC-SIMD has at most 10% overhead compared with an LC-SIMD, but the overhead of FC-SIMD increases to more than 100% when the number of PEs gets larger than 64 (because the area of a crossbar switch grows with $(N_{PE})^2$).

Description	Value
Data width of architecture (bit)	16
Width of instruction (bit)	24
Area of 2-1 Mux (normalized to area of a NAND in CMOS 0.18)	2
Area of 1 bit D flip flop (normalized to area of a NAND in CMOS 0.18)	5
Number of PES	1,2,4,8,... 256

(a)



(b)

Fig. 14. (a) Parameters used in the area model (CMOS 0.18) (b) Area overhead (compared to an LC-SIMD).

6 Conclusions

In this paper, we studied the communication bottleneck present in currently available SIMD architectures. The problem comes from the limited access to

other PEs in the SIMD architecture. Although this bottleneck can be solved by over-dimensioning the communication architecture, as in a fully connected SIMD (FC-SIMD), it leads to an increase in the communication area, low interconnect utilization, and scalability problems.

We introduced a novel architecture with segmented data communication busses, and with a delay-line in the instruction distribution which enables to time-interleave bus accesses by the processing elements. The area of this architecture is almost the same as the cheapest, locally connected SIMD (LC-SIMD), while its performance is nearly the same as the most expensive FC-SIMD.

We showed that this new architecture poses some irregular resource constraints that available schedulers cannot directly cope with. However, a carefully constructed resource conflict model enables available schedulers to generate valid schedules for RC-SIMD. Furthermore, we demonstrated a reconfigurable version of the architecture that allows to exploit the maximum neighborhood communication in an algorithm. Experiments on industrially relevant image processing kernels show that for RC-SIMD our scheduler generates code requiring on average 21% fewer cycles than LC-SIMD architectures, while the area overhead of our architecture (compared to an LC-SIMD) is at most 10%.

As part of future work we are interested to add (compiler controlled) delay and energy models, enabling the comparison of RC-SIMD with other SIMD architectures on these important aspects as well. Furthermore, for extremely intensive and non-local communicating kernels we will consider to add multiple segmented data communication busses.

Acknowledgment: The authors would like to thank Patrick Groeneveld for his support in the area estimation part.

References

1. Anteneh Abbo and Richard Kleihorst. Smart Cameras: Architectural Challenges. In *Proceedings of Advanced Concepts for Intelligent Vision Systems (ACIVS)*, pages 6–13, Gent, Belgium, September 2002.
2. E.Oran Brigham. *The fast Fourier transform and its applications*. Prentice Hall International, 1988.
3. Wouter Caarls, Pieter Jonker, and Henk Corporaal. Benchmarks for SmartCam Development. In *Proceedings of Advanced Concepts for Intelligent Vision Systems (ACIVS)*, pages 81–86, Gent, Belgium, September 2003.
4. Koen van Eijk, Bart Mesman, A. Alba Carlos Pinto, Qin Zhao, Marco Bekooij, Jef van Meerbergen, and Jochen Jess. Constraint Analysis for Code Generation: Basic Techniques and Applications in Facts. *ACM Transactions on Design Automation of Electronic Systems*, 5(4):774–793, Octobor 2000.
5. Hamed Fatemi, Henk Corporaal, Twan Basten, Richard Kleihorst, and Pieter Jonker. Designing Area and Performance Constrained SIMD/VLIW Image Processing Architectures. In *Proceedings of Advanced Concepts for Intelligent Vision Systems (ACIVS)*, pages 689–696, Antwerp, Belgium, September 2005. Springer-Verlag, Berlin, Germany, 2005.
6. J.R. Fischer and J.E. Dorband. Applications of the MasPar MP-1 at NASA/Goddard. In *Proceedings of COMPCON*, pages 278–282, San Francisco, CA, February 1991. IEEE Computer Society.

7. Yoshihiro Fujita, Sholin Kyo, Nobuyuki Yamashita, and Shin'ichiro Okazaki. A 10 GIPS SIMD Processor for PC-based Real-Time Vision Applications — Architecture, Algorithm Implementation and Language Support. In *Proceedings of the Computer Architectures for Machine Perception (CAMP)*, pages 22–32, Washington, DC, USA, October 1997. IEEE Computer Society.
8. Patrick Gelsinger. Microprocessors for the New Millennium: Challenges, Opportunities and New Frontiers. In *Proceedings of International Solid-State Circuits Conference (ISSCC)*, pages 22–25, San Francisco, CA, February 2001. IEEE Computer Society.
9. H.P. Hofstee. Power Efficient Processor Architecture and The Cell Processor. In *Proceedings of 11th International Conference on High-Performance Computer Architecture (HPCA)*, pages 258–262, San Francisco, CA, February 2005. IEEE Computer Society.
10. R.M Michael Hord. The ILLIAC IV, the first supercomputer. *Computer Science Press*, 1982.
11. Kai Hwang and Faye Briggs. *Computer Architecture and Parallel Processing*. McGraw-Hill, USA, 1984.
12. Brucek Khailany, William J. Dally, Scott Rixner, Ujval J. Kapasi, Peter Mattson, Jinyung Namkoong, John D. Owens, Brian Towles, and Andrew Chang. Imagine: Media Processing with Streams. *IEEE Micro*, 21(2):35–46, April 2001.
13. David J. Kuck. A Survey of Parallel Machine Organization and Programming. *ACM Comput. Surv*, 9(1):29–59, 1977.
14. Silicon Hive. http://www.siliconhive.com.
15. T.H Szymanski, Honglin 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.
16. TriMedia Technologies. http://www.semiconductors.philips.com.
17. S. A. Zenios and R. A. Lasken. The connection machines CM-1 and CM-2: solving nonlinear network problems. In *Proceedings of the 2nd International Conference on Supercomputing (ICS)*, pages 648–658, Saint Malo, France, 1988.