A low-energy wide SIMD architecture with explicit datapath

Citation for published version (APA):

Waeijen, L. J. W., She, D., Corporaal, H., & He, Y. (2015). A low-energy wide SIMD architecture with explicit datapath. *Journal of Signal Processing Systems*, 80(1), 1-22. <https://doi.org/10.1007/s11265-014-0950-8>

DOI:

[10.1007/s11265-014-0950-8](https://doi.org/10.1007/s11265-014-0950-8)

Document status and date:

Published: 01/01/2015

Document Version:

Accepted manuscript including changes made at the peer-review stage

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

A Low-Energy Wide SIMD Architecture with Explicit Datapath

Luc Waeijen · Dongrui She · Henk Corporaal · Yifan He

Received: date / Accepted: date

Abstract Energy efficiency has become one of the most important topics in computing. To meet the ever increasing demands of the mobile market, the next generation of processors will have to deliver a high compute performance at an extremely limited energy budget. Wide *single instruction, multiple data* (SIMD) architectures provide a promising solution, as they have the potential to achieve high compute performance at a low energy cost. We propose a configurable wide SIMD architecture that utilizes *explicit datapath* techniques to further optimize energy efficiency without sacrificing computational performance. To demonstrate the efficiency of the proposed architecture, multiple instantiations of the proposed wide SIMD architecture and its automatic bypassing counterpart, as well as a baseline RISC processor, are implemented. Extensive experimental results show that the proposed architecture is efficient and scalable in terms of area, performance, and energy. In a 128-PE SIMD processor, the proposed architecture is able to achieve an average of 206 times speed-up and reduces the total energy dissipation by 48.3% on average and up to 94%, compared

to a *reduced instruction set computing* (RISC) processor. Compared to the corresponding SIMD architecture with automatic bypassing, an average of 64% of all register file accesses is avoided by the 128-PE, explicitly bypassed SIMD. For total energy dissipation, an average of 27.5%, and maximum of 43.0%, reduction is achieved.

Keywords Wide SIMD, Explicit Datapath, Configurable, Low Energy

1 Introduction

There is an increasing demand for running applications with high performance requirements on systems that have relatively limited resources. For example, a smart camera may combine high resolution video sensing, low-level to high-level vision processing, and communication within a single embedded device. All these applications require a large amount of computation, and yet system designers often have to meet these requirements with a very limited energy budget. *Energy efficiency* is thus becoming a dominant determinant in the design, especially for systems that have to run on restricted energy sources like batteries.

To provide the required high performance under a limited energy constraint, often a solution is found in wide *single instruction, multiple data* (SIMD) architectures [1, 15, 18, 32]. In SIMDs, a *single* instruction operates on *multiple* data in parallel. This enables SIMDs to exploit the data-level parallelism present in an application. Because multiple operations are carried out simultaneously, high computational throughput can be delivered at a very low clock frequency and thus low voltage, thereby greatly improving energy efficiency [9, 22].

Another important low-power feature of wide SIMDs is that a significant portion of the datapath and control path can be shared between multiple *processing elements* (PEs).

This article has been partially presented at the 13th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIII) [28].

L. Waeijen
Eindhoven University of Technology, The Netherlands
E-mail: l.j.w.waeijen@tue.nl

D. She
Eindhoven University of Technology, The Netherlands
E-mail: d.she@tue.nl

H. Corporaal
Eindhoven University of Technology, The Netherlands
E-mail: h.corporaal@tue.nl

Y. He
Eindhoven University of Technology, The Netherlands
& Recore Systems B.V., The Netherlands
E-mail: y.he@tue.nl

A wide SIMD typically consists of a *control processor* (CP) and a large number of PEs. Since the PEs execute the same instruction in each cycle, the instruction fetch and decode hardware can be shared between all PEs. The energy usage of the instruction memory, which contributes a significant part to the total energy usage of a single core processor, is amortized over multiple PEs. Furthermore the programs' control flow is also effectively shared, as it is managed by the CP. For a wide SIMD with hundreds of PEs, the energy used by these shared parts (i.e., instruction fetch, instruction decode, and control flow) becomes negligible. The largest amount of energy is dissipated in the PE datapath, which performs most of the useful computations, resulting in high energy efficiency. Because of this, techniques which can improve the energy efficiency of the PE datapath will be especially effective in wide SIMDs.

Since the energy usage of a wide SIMD is dominated by PEs [9, 22], and the *register file* (RF) is one of the major contributors to the PE's energy dissipation, as will be shown in Section 4, reducing the energy dissipation of the RF will have a large impact on the overall energy usage of wide SIMD architectures. Therefore, reducing the register file's energy usage in the context of a wide SIMD processor is of great importance.

Explicit bypassing is a technique that can be used to reduce the energy usage of a register file [5, 6, 10, 33]. Traditional pipelined architectures usually have hardware bypassing mechanism that can mitigate the penalty of read-after-write hazards. Such bypassing is transparent to the software, thus also called *automatic bypassing*. On the contrary, explicit bypassing directly controls the bypassing (forwarding) network in software. Explicit bypassing has the potential to greatly reduce the number of accesses to the RF, resulting in a much more energy efficient RF [5, 6, 10].

We propose a programmable, highly energy-efficient, configurable wide SIMD architecture that exploits the explicit datapath concept. A complementary tool flow composed of compiler, simulator, and *hardware description language* (HDL) automatic generator, is also developed for the proposed architecture. To demonstrate the efficiency of the proposed architecture and the effectiveness of explicit bypassing in wide SIMDs, multiple instantiations of the proposed wide SIMD architecture and its automatic bypassing counterpart, as well as a baseline RISC processor, are synthesized with a TSMC 40nm low-power library. Firstly, eleven representative kernels are chosen to examine the proposed architecture. These kernels contain different types of communication and memory access patterns, namely *point-to-point*, *neighborhood-to-point*, *global-to-point*, and *global-to-global* (described in Section 3), which represent a wide range of applications. Secondly, *Fast Focus on Structures* (FFoS) [11, 12], a complete computer vision application, is mapped onto the proposed architecture.

The experimental results show that in a 128-PE SIMD processor, the proposed architecture is able to achieve an average of 206 times speed-up and reduces the total energy dissipation by 48.3% on average and up to 94%, compared to a (RISC) processor. Compared to the corresponding SIMD architecture with automatic bypassing, an average of 64% of all register file accesses is avoided by the 128-PE, explicitly bypassed SIMD. For total energy dissipation, an average of 27.5%, and maximum of 43.0%, reduction is achieved.

The key contributions of this work are:

- We propose and discuss in detail a configurable, highly energy-efficient, wide SIMD processor architecture with explicit datapath.
- Using kernels with different communication and memory access patterns, we comprehensively analyze the area, performance, energy, and scalability of the proposed architecture.
- As a case study, we also map and analyze a complete industrial application to demonstrate the effectiveness of the proposed architecture.

The remainder of this paper is organized as follows: Section 2 introduces the proposed architecture and elaborates the differences between explicit and automatic bypassing. Section 3 describes the experimental setup and benchmarks. Experimental results that show the effectiveness of the proposed design are given in Section 4. Related work is discussed in Section 5. Finally, Section 6 concludes our findings and discusses future work.

2 Proposed Wide SIMD Architecture

As discussed in Section 1, a wide SIMD architecture is an excellent candidate for an energy efficient platform. In a wide SIMD, many *processing elements* (PEs) execute the same instruction. Hence instruction fetch and decode overhead are amortized over all PEs. Furthermore, because wide SIMDs naturally support *data-level parallelism* (DLP) by executing multiple operations in parallel, the architecture can meet high compute demands at low clock frequencies and voltages. This allows realistic applications, especially those with high DLPs, to be executed in an energy efficient manner.

The proposed processor architecture consists of two parts, a *control processor* (CP) and a wide *one dimensional* (1-D) array of processing elements (PEs), which run in lock-step. It results in a *very long instruction word* (VLIW) processor with one scalar issue slot for the CP and one (wide) vector issue slot for the PE array. Fig. 1 depicts the proposed architecture. Data-level parallelism is exploited in the PE array, while *instruction level parallelism* (ILP) is exploited through issuing scalar and vector operations simultaneously. This is particularly effective in an SIMD context,

Fig. 1 High level structure of a wide SIMD

where without the exploitation of ILP by the CP, a single scalar instruction would require the whole PE-array to execute this one instruction. By executing scalar instructions in parallel to vector instructions, an ideal speedup of two can be gained over any speedup already obtained by exploiting DLP. The CP and PE array operate in lock-step, such that the control flow for both entities is uniform. This enables the CP to handle the control flow, while the PE array processes data in parallel.

In the proposed architecture each PE has a private *data memory* (DMEM) with its own address generator, i.e., per-PE addressing. Though the memory is a bit more complex compared to global addressing for all PEs, per-PE addressing results in much better programmability [14]. Since many applications, such as histogram and Hough transform, can benefit from independent address generation [13, 14], the proposed architecture supports this feature. The CP also has its own scalar data memory.

The *instruction memory* (IMEM) is shared between the CP and the PE array. Each instruction stored in the IMEM contains a pair of operations, i.e., one CP (scalar) operation and one PE-array (vector) operation. The vector part of instruction is fetched from the shared IMEM, partially decoded in the shared instruction decode stage, and broadcasted to all PEs. The broadcasting of these signals across the chip could in a silicon implementation affect the maximum achievable clock frequency. In this work only synthesis is performed, which showed the target frequency could be reached without any such issues. For more accurate results, place and route is planned as future work. If place and route reveals that these long broadcast wires negatively affect the maximum clock frequency, a possible solution is to pipeline the wires, at a penalty of more branch delay slots. This is however beyond the scope of this work.

The *instruction set architecture* (ISA) of both the CP and the PE is based on a 24-bit RISC-like ISA, similar to the one used by D.She et al. [25], but with two extra bits for neighborhood communication (Section 2.2.1) and two extra bits for predication (Section 2.4). The instruction width for

both the CP and the PE therefore becomes 28 bits instead of 26.

2.1 Processing Element (PE) Datapath

The proposed wide-SIMD framework supports different PE configurations (Section 2.5), e.g., 4-stage pipeline or 5-stage pipeline. As an example, we focus on the 4-stage RISC-like datapath in this work.

One of the properties of SIMDs is that the *instruction fetch* (IF) and part of the *instruction decode* (ID) logic are shared among all PEs. The remaining parts of the decoding logic, such as RF access and operand selection, are done locally in each PE (Fig. 2 and Fig. 3). The *execution stage* (EX) contains an *arithmetic logic unit* (ALU), a *multiplier unit* (MUL), and a *load store unit* (LSU). The *writeback* (WB) stage commits the results to the RF if necessary. To optimize the datapath for low energy usage, each *functional unit* (FU) in the execution stage has its own input registers. This is to isolate/clock-gate FUs, such that an FU does not dissipate dynamic power if it is not the target FU of the current operation. Another, probably more important, reason to introduce input registers for each FU is to extend the available time of FU outputs in the bypassing network. This can improve the efficiency of the explicit datapath as described in Section 2.1.2 [8]. Before moving to the main focus of this work, i.e., the *explicit datapath*, the next section first presents a conventional *automatically bypassed datapath*.

2.1.1 Automatically Bypassed Datapath

Conventional pipelined architectures typically have an automatic bypassing mechanism that can mitigate the penalty of read-after-write hazards. Such bypassing is transparent to the software. In an automatically bypassed datapath, hardware keeps track of all the uncommitted results in the pipeline and determines whether a bypass is required. Fig. 2 depicts a 4-stage datapath with automatic bypassing. In this datapath there are two bypass sources, i.e., the result of the EX stage and the output of the WB stage.

Despite being widely used, automatic bypassing has two major disadvantages with respect to energy efficiency:

1. Speculative Reads

The detection of bypass situations is performed in the ID stage, in parallel with the operand fetching from the RF. Therefore the operands specified by an instruction are always fetched from the RF. If then a bypass is required, the operand fetched from the RF is invalid and will be discarded. The energy used to fetch the discarded operand from the RF is thus wasted.

2. Unnecessary Writes

In programs with short-lived values, often a calculated

Fig. 2 Datapath with automatic bypassing

result is consumed through a bypass. If all uses of this result are bypassed, writing it back to RF is not necessary. In an automatically bypassed datapath, however, the hardware cannot determine whether a variable is going to be referenced in the future or not. Therefore, the *dead* result is still written back to the RF. Writing a variable to the RF that is never referenced again wastes energy.

Speculative reads are caused by the lack of time to detect such bypass situations dynamically before the RF is accessed. The unnecessary writes find their origin in the fact that hardware has no liveness information of the variables. After all, hardware can only observe the variables that are currently in the pipeline.

2.1.2 Explicitly Bypassed Datapath

Explicit bypassing is a technique that can be used to reduce the energy usage of a register file [5, 6, 10, 33]. The key idea of explicit datapath architectures is to expose more details of the datapath to software, thereby enabling fine-grained control over the datapath in the software.

The disadvantages of the automatically bypassed datapaths originates from the lack of liveness information and limited time to detect bypass situations. Therefore, it makes sense to shift the task of detecting and handling bypassing from the hardware to the compiler. If the compiler is given information on the architecture of the datapath, the situations that require bypassing can be easily detected at compile time. By keeping track of where all the variables are in the pipeline, the compiler can determine what bypasses are required.

Furthermore, the compiler has a full view of the liveness of all the variables. Given that the compiler can detect and control what values/operands to bypass in the pipeline, it can also determine whether a variable is dead before it is committed to the register file. In that case it can prevent the unnecessary write that automatic bypassing would have performed.

To enable the compiler to control bypassing, the actions required to perform a bypass need to be encoded in the instructions. In the proposed datapath this is done by mapping the bypass sources to the register address space. When a bypass is needed from a certain bypass source, the compiler inserts the address associated with that source into the corresponding operand field. Furthermore, if a variable does not need to be written to the register file, the compiler will replace the destination address with `r0`, or the alias `--` which is used for readability in this paper. This register always reads zero, and when the hardware encounters it as a destination, no actual write is performed.

The proposed architecture implements the explicit datapath with the guideline of maximizing energy efficiency. Fig. 3 depicts the explicit datapath of the proposed architecture. There are four bypass sources, namely ALU output, MUL output, LSU output, and WB output. These sources are mapped to the top of the 5-bit register space. To enhance readability of code, an alias is defined for each of the bypasses. ALU indicates the result of the ALU, MUL of the multiplier, LSU of the load store unit, and WB indicates the writeback stage.

Compared to explicit datapath implementation with less bypass sources, such as the work of J. Yan et al. [33], multiple bypass sources can further reduce the traffic to/from the RF, as will be shown in the example later in this section. In the proposed architecture, since the bypass sources are mapped to the top of the 5-bit register space and RF-write removal is achieved by *writing* to a virtual register `r0`, which is constant 0, no extra bit(s) need to be added to the instruction format. Another optimization is that each *functional unit* (FU) in the execution stage has its own input registers. These input registers are only updated when the FU is active. The advantage of this is that when a FU is not used, the old inputs remain present and the output of the FU is maintained. This extends the available time of a result at the FU output, increasing the bypass opportunities. In the remainder of this sub-section, these features are further discussed.

Fig. 3 Datapath with explicit bypassing

As an example of explicit datapath, consider the bypass situation in Code 1. In instruction 1 the first operand (r1) needs to be bypassed from the multiplier (MUL). The compiler will thus insert the register address associated with the MUL bypass source. The hardware will then fetch the operand from the output of the multiplier. Because the bypass situation can be directly extracted from the instruction, the RF does not need to be accessed. Additionally the result of instruction 0 does not need to be written to the RF, because after the bypass, the variable is dead. The compiler can analyze this and insert the -- address as the destination, which internally is converted to r0 such that the hardware will automatically not perform the write. The explicitly bypassed version of Code 1 is given in Code 2

```
0: mul r1, src1, src2
1: add dest, r1, src2
```

Code 1 Simple bypassing situation

```
0: mul --, src1, src2
1: add dest, MUL, src2
```

Code 2 Explicitly bypassed version of Code 1

The proposed architecture increases bypass opportunities with more bypass sources. Essentially the bypass source of the execution stage in Fig. 2 is split into three separate sources, ALU, MUL and LSU, as shown in Fig. 3. The input operands of these units are only updated when a given unit needs to be active. This avoids unnecessary toggling of inactive units. It also ensures the output of a unit remains unchanged until the next operation that uses it. As the input operands do not change, the output of the logic holds the output value of the previous operation. This enables bypassing of these outputs as long as the associated unit is not used, thus, increasing bypass opportunities.

To demonstrate the effect of adding more bypass sources, consider Code 3, which would only have one bypass with the bypass sources from Fig. 2. By adding the bypass sources as in Fig. 3 this code is transformed into the code in Code 4. The difference is significant, from only one read avoided by bypassing in Code 3, the added bypass sources avoid another six reads, yielding a total of seven avoided reads. On top of this the additional sources also avoid four writes to the RF, halving the number of writes.

As a possible drawback, adding extra bypass sources takes up more locations in the register file address space. The automatic datapath has 32 registers, while the explicit datapath has only 28, because four locations are used to address the bypass sources. However, as has already been shown in many related works, explicit bypassing can largely

```
0: lw r2, r1, 0
1: add r1, r1, 1
2: mul r3, WB, 4 /* r2 is bypassed with WB */
3: lw r4, r2, 0
4: sw r0, r1, 1
5: add r5, r3, r1
6: mul r6, r3, r4
```

Code 3 Code snippet that has only one bypass when there is only a single bypass source for the EX stage as described in Fig. 2

```
0: lw --, r1, 0
1: add --, r1, 1
2: mul --, LSU, 4
3: lw --, LSU, 0
4: sw r0, ALU, 1
5: add r5, MUL, ALU
6: mul r6, MUL, LSU
```

Code 4 Same code as in Code 3, but now bypassed with the additional bypass sources as in Fig. 3

mitigate the increased RF pressure as no registers need to be reserved for variables that are consumed through bypassing [5, 20, 21, 26, 33]. This can also be observed when comparing Code 3 and Code 4. In the original situation six registers are used. After adding the additional bypass sources only three registers are required.

A long-standing issue of explicit datapath is that much larger states need to be saved for interrupts and context switching. One possible solution is to use a scan chain to stream out the current state efficiently. This is however out of the scope of this work. Please refer to the thesis of H. Corporaal for more details regarding this topic [3].

2.2 Interconnect

One of the main challenges of wide SIMDs is the interconnect between different processing elements. Most practical applications require some form of communication between the processing units. For example to access input data or processed results stored in the DMEM of a different PE, or for synchronization between the PEs. From an application point of view, the most convenient form of interconnect is a fully connected network. Such networks scale extremely poorly, however, because of their complexity and long wires. A number of interesting solutions have been proposed for wide-SIMD in recent years, such as a modified crossbar [23], an X-RAM swizzle network [24], and dynamic communication for SIMDs [4]. Although these networks mitigate the problem somewhat, either they are optimized for only a very few specific applications (e.g., DC-SIMD [4] is mainly tuned for kernels such as lens distortion correction), or a large amount of additional hardware is required [23, 24]. In all cases scalability is limited.

2.2.1 Circular Neighborhood Communication Network

In the proposed architecture, a circular neighborhood network is used, which is very similar to the neighborhood network presented in the Xetal processor [1]. The main difference is that the neighborhood network in the Xetal processor has constrained access to data of its direct neighbors, e.g., a PE can access data in the data memory of its direct neighbor, but all PEs have to access with the same address/index to its neighbor's memory. This is because the Xetal processor does not support per PE addressing and it also lacks predication support. The circular neighborhood network in the proposed architecture has more flexible access to data of its direct neighbors as both per PE addressing and predication are supported.

In a circular neighborhood network, the units are logically connected in a large circle and each unit can communicate with its left and right neighbors, resulting in exclusively short wires for the communication network. This allows virtually unlimited scaling of this type of network, which is backed up by the results presented in section 4.1 and section 4.2. Fig. 4 illustrates the architecture of this network. To support neighborhood communication, two additional bits are added to the instruction set, indicating from which source (i.e., left neighbor, right neighbor, or itself) the operand should be read.

Fig. 4 Circular neighborhood communication network

The neighborhood network performs extremely well for local, short distance communication as encountered in many typical kernels on SIMDs, such as box filters and motion estimation. In the target application domains of wide SIMDs, such as image, video and vision applications, many processing kernels show high locality in their communication, making a neighborhood network perfectly suited.

When two PEs which are not direct neighbors need to communicate, data must be shifted through all units in between. This is especially costly if long-distance communication is involved extensively. Unfortunately, these kinds of applications are by no means rare. For example, *partial histogram merging*, *row projection* (sub kernels in the

FFoS application described in Section 3.2), *Find Maximal Element in a Vector*, and *Sum of Vector Elements* (categorized as *global-to-point* kernels in Section 3.2). To efficiently handle these kernels on a wide SIMD with only a circular neighborhood network, we introduced two novel reduction algorithms, *pipelined reduction* and *diagonal access reduction*, which do not rely on complex communication networks or any dedicated hardware [29]. The key idea of both approaches is to utilize inter-vector parallelism instead of intra-vector parallelism. The experimental results show that using the proposed algorithms, the performance is comparable to the performance when dedicated reduction hardware is equipped. For details please refer to the work of L.Waeijen et al. [29].

As can be seen from Fig. 4, the CP takes a special place in the circular neighborhood network to facilitate both communication among PEs and between PE-array and CP. By dynamically programming the border PEs of the PE-array, it is possible to eliminate the CP from the circular network. This is useful when the PEs need to wrap around data. Furthermore the border PEs can be programmed to not read from each other, but reading a fixed value instead. This is useful for automatic insertion of image borders for example.

2.3 CP Broadcast

In addition to the circular neighborhood network described in the previous section, direct broadcasting from the CP to all the PEs in the PE-array is also introduced. In some cases it is required to send the exact same data from the CP to all the PEs. For example a centrally computed threshold. Although it would be possible to use the neighborhood communication network for this purpose, it would be much more efficient using direct broadcasting. The data broadcasted by the CP can be read by PEs in a similar manner as the neighboring operands.

2.4 Predication

Strictly speaking, the SIMD paradigm applies exactly the same instruction to multiple data elements in parallel. This however is not always desired. Sometimes it is required by the program flow that some PEs do not perform a certain operation, while other PEs do. To facilitate this diverseness in program flow, each instruction is prefixed with two predication bits. Each PE can set and reset two predication flags with compare operations in the ALU. An instruction will only be executed on a PE if its corresponding flags are set, otherwise a NOP is inserted. This enables for example the implementation of *if-then-else* constructs on the PEs. Each PE can determine whether it needs to execute the *then* or *else* part.

2.5 Configurable Framework

To enable fast design space exploration and to tailor the proposed architecture for different target applications, a design framework is developed to easily generate different instantiations of the architecture with, for example, varying number of PEs, 4-stage or 5-stage datapath, explicit or automatic bypassing, as well as different datapath widths. Fig. 5 shows the high-level diagram of this framework.

Fig. 5 The framework to generate instances of the proposed wide-SIMD architecture based on a configuration file

The architecture configuration file is a human readable JSON file that specifies an instance of the wide-SIMD architecture. The hardware toolflow is visualized at the top half of Fig. 5. An HDL template is combined with the architecture configuration file to generate corresponding HDL code of the instance specified in the configuration file. After this step, conventional hardware design tools can be used for simulation, synthesis, and post-synthesis analysis. For the software toolflow, an efficient compiler is developed [27], which supports both C and OpenCL. This compiler takes the same architecture configuration into account and produces the proper binary. Besides that, also a cycle-accurate simulator is generated based on the configuration file.

3 Experimental Setup

This section describes the experimental setup used to quantify the effectiveness of the proposed architecture. Section 3.1 describes the configurations of the target architecture as well as a reference RISC architecture. Section 3.2 presents the benchmarks used for the evaluation.

3.1 Architecture configurations

To evaluate the effectiveness of the proposed design, both the explicitly and automatically bypassed versions of the SIMD architecture proposed in Section 2 are implemented in HDL. The configurations used in the experiments are shown in Table 1. Since the proposed architecture is configurable, the HDL code of each configuration is automati-

cally generated from the architecture template. For a complete analysis, we also compared the proposed SIMD architecture to a reference RISC architecture. The configuration of the reference RISC processor is shown in Table 2.

To exclude any interference a memory hierarchy could introduce in the measurements due to data misses, the evaluated designs assume only one level of memory. This memory can be accessed within a single cycle. The sizes of the data memories of the RISC and SIMD are chosen such that all data of the benchmarks can be contained.

Datapath	{automatic, explicit}
Data width	32 bits
Pipeline stages	4
Number of PEs	{8, 16, 32, 64, 128}
Instruction memory	56b × 2k
PE data memory	32b × 1k
CP data memory	32b × 1k

Table 1 Configurations of the target architecture

Data width	32 bits
Pipeline stages	4
Instruction memory	24b × 4k
Data memory	32b × 4k

Table 2 Configuration of the reference RISC processor

The core part of each configuration, that is the whole system with the exception of the memories, is synthesized for 1.1V, 25°C, typical case, with a 40nm TSMC low-power CMOS digital standard cell library. The target frequency is set to 100MHz. Energy dissipation is estimated using the physical information in the technology library and circuit toggle rate generated by post-synthesis simulation on the gate-level netlist. The energy dissipation of the memory part is estimated with CACTI [19]. The CACTI tool provides an average access energy for a given memory configuration. The estimated access energy of the corresponding memory configuration is given in Table 3. The number of accesses to each memory is extracted from simulation.

	32b × 1k	24b × 4k	32b × 4k	56b × 2k
pJ/Access	2.02	2.49	2.92	3.37

Table 3 Energy dissipation of different memory accesses

3.2 Benchmarks

To have a comprehensive evaluation of the proposed design across various types of applications, a total of eleven representative kernels are chosen, which are divided into four categories based on their communication and memory access patterns (Table 4). The four kernel categories are:

1. *point-to-point*

Binarization is a typical example of this type of kernel. To calculate an output at $D_{out}(x_a, y_b)$, a PE only needs to read the input data, $D_{in}(x_a, y_b)$, at the same location. Since this type of kernels does not need to access data in the other memory banks, no communication to neighboring PEs is required.

2. *neighborhood-to-point*

To calculate an output at $D_{out}(x_a, y_b)$, a PE only needs to read the current input data, $D_{in}(x_a, y_b)$, and its surrounding data, $D_{in}(x_a \pm m, y_b \pm n)$, where m and n are small. These data are either in the PE's own memory bank, or in its nearby memory banks, so only local communication is required. A *3x3 low-pass filter* is an example of this type of kernel.

3. *global-to-point*

To calculate an output at $D_{out}(x_a, y_b)$, a PE needs to read the data far away from its corresponding input data, i.e., global communication. This global communication is either in the X direction, i.e., $D_{in}(x_a \pm p, y_b)$, or in the Y direction, $D_{in}(x_a, y_b \pm q)$, where p and q are large. One paradigm that falls into this category is *reduction*, in which all the elements of a vector are combined into a single element by a certain operator such as add, min/max, logic and/or.

4. *global-to-global*

To calculate an output at $D_{out}(x_a, y_b)$, a PE needs to read the data far away from its corresponding input data, i.e., global communication. This global communication can be in both X and Y directions, i.e., $D_{in}(x_a \pm p, y_b \pm q)$, where p and q are large. For this type of kernels, every input data element is still mapped to an output data element, i.e., no reduction operator is involved. These kernels are often mapped in such a way that one dimension is handled when reading the required input data, and the other dimension is handled when writing the output. *Matrix rotation of 90 degrees* is an example of this type of kernel.

Evaluation based on such patterns is interesting in the context of wide SIMDs with neighborhood communication network. This is because kernels with only local communication can be efficiently mapped onto such a design, while kernels with global access will spend a significant amount of cycles on data transfers between PEs that are far apart. To reduce the overall cost of long-distance communication, we introduced two algorithms, *pipelined reduction* and *diago-*

point-to-point	global-to-point
Binarization	Find Max. Element in a Vector
Color Conversion	Sum of Vector Elements
	Vector-Vector Addition
neighborhood-to-point	global-to-global
Convolution 3x3	Matrix Rotation
Erosion 3x3	Matrix Mirroring
5-tap FIR	Matrix Transpose

Table 4 Benchmark kernels and the categories they belong to

nal access reduction, which do not rely on complex communication networks or any dedicated hardware [29]. The key idea of both approaches is to utilize inter-vector parallelism instead of intra-vector parallelism, which can be applied to both *global-to-point* and *global-to-global* kernels.

Besides kernel-level evaluation, we also map the Fast Focus on Structures (FFoS) application [11] to profile the proposed architecture. FFOs is the complete vision processing pipeline of an industrial application, *Organic Light Emitting Diode* (OLED) screen printing. Its purpose is to find the center of OLED cells at high speed in the manufacturing process. It consists of the following four parts:

1. *Otsu*

With the input image shown in Fig. 6a, the optimal threshold for binarization is determined by means of Otsu's method [17]. Otsu's method exhaustively searches for a class that minimizes the intra-class variance. In order to achieve this, partial histograms per column are first calculated in the PE array, which are then merged into a combined histogram. The optimum threshold is calculate based on this histogram. In order to achieve this 256 divisions are required, which are performed in parallel on the PE-array.

2. *Binarization*

Once the optimal threshold is determined, the input image is binarized to value 0 or 1. The result of this process can be seen in Fig. 6b.

3. *Erosion*

In order to remove noise and small objects from the binarized image, an erosion kernel is applied to the binarized image. The eroded output image is shown in Fig. 6c.

4. *Row and column projection*

Finally, by summation of the rows/columns and determining the local peaks in the resulting vectors, the boundaries of the OLED cells can be determined. In Fig. 6d, bonding boxes are drawn around the detected cells.

As the number of cores scales up, it is important to decide how to scale the problem size accordingly. One option is to keep the problem size fixed while the number of parallel cores scales. This methodology is in line with the well-known Amdahl's law [2]. Amdahl's law predicts a maxi-

Fig. 6 FFoS images at different stages

num speedup as given by equation 1, where s is the sequential fraction of a program, p is the parallel fraction and N is the number of cores in the system. The implication of this law is that the achieved speed-up rapidly diminishes even for small s . Therefore, if applied to the proposed SIMD, the expected speed-up for 128 cores is relatively small.

$$speedup = \frac{1}{s + \frac{p}{N}} \quad (1)$$

Not only is the speedup predicted by Amdahl's law small, it is also unrealistic for practical purposes as reasoned by J.Gustafson [7]. Gustafson argues that for realistic applications of multi core systems, it is unlikely that the problem size is kept constant. Instead, a higher number of cores is typically used to solve bigger problems. For example in video processing a higher resolution can be used, or a weather prediction system can be applied to a larger area. Assuming the problem size scales with the number of cores leads to an alternative to Amdahl's law know as Gustafson's *scaled speedup*, which is given in equation 2.

$$Scaled\ speedup = N + (1 - N) \times s \quad (2)$$

Because Gustafson's scaled speedup is more appropriate for real-life applications than Amdahl's law, in this work we choose to scale the problem size with the number of cores. In particular the kernels operate on an $N \times N$ matrix, where N is the number of PEs. Note that in Gustafson's original paper it is approximated that the amount of work scales linearly with the number of processors, yet in this work the input scales quadratically with the number of PEs. This is done because for our image and matrix oriented benchmarks this is a more natural choice. When both the width and height are scaled with N , the mapping of the problem to the processor can remain unchanged. For example if the row of an input image matches the number of PEs, it will match across all scaled versions. Otherwise some form of wraparound would be required, which would introduce a variable overhead in some problem-processors mappings, but not in others. This would lead to unwanted noise in the speedup measurements, and skew any speedup results.

As an exception to the quadratic scaling, FFoS's input is chosen to scale linearly. Since FFoS detects the centers of OLEDs in a production process, it is only interesting to increase the number of cells which are detected in the width of the input. The cells are moved underneath the camera so it is not useful to detect cells at a large number of rows simultaneously, yet detecting the centers of all cells in a row is the real goal of the application. Therefore it operates on input images of size $1024 \times N_{PE}$, where N_{PE} is the number of PEs. In this way the FFoS application is modeled in the most realistic way, and remains true to Gustafson's original approximation that the problem size increases linearly with the number of cores.

4 Results and Analysis

In Section 4.1, the proposed architecture is first compared to a reference RISC processor in terms of area, performance, and energy dissipation. Performance and energy dissipation figures are obtained using the four types of kernels described in Section 3.2. The purpose of this comparison is to demonstrate the scalability and energy efficiency of the proposed architecture. To examine the energy and performance impact of explicit bypassing in SIMDs, the proposed explicitly bypassed architecture is compared to its automatically bypassed counterpart in Section 4.2. The energy and performance analysis are done for all kernel categories as well as a realistic application, FFoS. By varying the number of PEs, the effects of explicit bypassing in an SIMD architecture are discussed in detail.

4.1 SIMD vs. RISC

In this section the performance, area, and energy dissipation of the proposed explicit SIMD architecture is compared with that of a reference RISC architecture, which is described in Section 3.1.

In terms of running time, the proposed SIMD has a significant speedup over the RISC in each kernel category as is shown in Fig. 7. For the *point-to-point* kernels as can be seen in Fig. 7a, the relation between the number of PEs and the speedup is almost linear. This is to be expected as the point-to-point kernels have no data dependencies between different PEs.

Fig. 7b shows the speedup for the *neighborhood-to-point* kernels. Interestingly, the speedup of the neighborhood-to-point kernels is greater than that of the point-to-point kernels. This is because instead of only exploiting more DLP by adding more PEs, also the locality of the loaded data is exploited in a more efficient manner. If a neighborhood is to be processed on the RISC, the processor will load every pixel inside the neighborhood window and calculate a

result. When the window shifts, a couple of pixels can be reused, and do not need to be reloaded, but also some pixels will be lost that are needed for a future computation. These pixels will have to be reloaded by the RISC once an overlapping window is processed. On the SIMD, the image is loaded row by row, and all windows in a line are processed in parallel. Since each PE has its own register file, it is possible to keep a couple of complete rows in the register file at the same time. Therefore, once a pixel is loaded, it is used in all relevant neighborhood calculations it belongs to, and does not need to be loaded again. This saves extra operations and memory accesses, which will also be visible later in the energy dissipation.

For the *global-to-point* category, the speed up is typically less than point-to-point and neighborhood-to-point kernels, as can be seen in Fig. 7c. Especially the *max* and *reduction* kernels have less speed up. This is explained by the fact that when the number of PEs increases, the data from these kernels need to travel farther to reach the final destination point. Both the max and reduction kernels operate on data which is spread out across the PEs. More PEs means a longer path through the neighborhood network, which somewhat counteracts the gain of exploiting more parallelism. The *vectoradd* kernel is the exception in this category. This is explained by the fact that max and reduction kernels have data movement across the PE array, i.e., reduction of the elements in a row, while the *vectoradd* kernel only has data movement within the PEs, i.e., reduction of the elements in a column. While the max and reduction kernel combine an element from every PE in the Array, the *vectoradd* kernel only combines elements which are within the data memory of a PE. Therefore, when the array becomes larger, there is no communication penalty such as the ones for the max and reduction kernels. As mentioned in the previous sections, to reduce the overall cost of long-distance communication, we introduced two algorithms that exploit inter-vector parallelism instead of intra-vector parallelism [29]. These approaches can also be applied to the *global-to-global* kernels.

Finally the *global-to-global* kernels show the least amount of speedup by adding more PEs, as is shown in Fig. 7d. For these kernels data elements need to move both between PEs and inside the data memories of the PEs (i.e., row and column wise). The communication patterns that arise from this are not always regular, making the neighborhood network the main bottleneck when more PEs are added. Because of the more irregular patterns, the global-to-global kernels pay an even high penalty.

The core area of different instantiations of the explicitly bypassed SIMD, as well as that of the reference RISC processor is shown in Table 5. The area of the 8-PE SIMD is slightly larger than eight times that of the RISC processor. This is because an 8-PE SIMD consists of a vector array of eight PEs and a control processor (CP). The area of a PE

Fig. 7 Speedup compared to RISC

itself is smaller than its RISC counterpart as the instruction fetch (IF) and part of the instruction decode (ID) logic are

#PE	RISC	Explicit Bypass				
	–	8	16	32	64	128
Cell	10884 (1.0)	88663 (8.1)	165809 (15.2)	320198 (29.4)	629034 (57.8)	1246670 (114.5)
Net	23509 (1.0)	205614 (8.7)	384332 (16.3)	741679 (31.5)	1457081 (62.0)	2889653 (122.9)
Total	34392 (1.0)	294277 (8.6)	550140 (16.0)	1061876 (30.9)	2086115 (60.7)	4136323 (120.3)

Table 5 Core area of explicitly bypassed SIMD compared to RISC

shared among all PEs. When the number of PEs increases, the CP area is amortized over more PEs. Table 5 shows that the proposed SIMD architecture also scales well in area.

In this work, the energy of both the core and memory are considered. The reduction of the overall energy dissipation of the proposed SIMD architecture compared to the reference RISC is shown in Fig. 8.

Fig. 8a shows the results of the *point-to-point* kernels. It can be seen that the reduction in energy dissipation of the binarization kernel keeps on increasing when more PEs are used. For the color conversion kernel, however, increasing the number of PEs beyond 32 starts to degrade the efficiency. It seems that for the color conversion kernel, after a certain number of PEs the energy overhead of additional hardware, such as neighborhood communication network and predication logic, is not compensated sufficiently by the speed up it provides.

For the *neighborhood-to-point* kernels, it can be seen in Fig. 8b that increasing the number of PEs always leads to an increased energy efficiency. The reason for this is twofold: on one hand the speed up of these kernels is slightly higher than for the color conversion kernel. On the other hand, because of better exploitation of the locality of the data, the number of external memory accesses of the neighborhood-to-point kernels decreases more than for the color conversion kernel when the number of PEs increases. Accessing the data memory is expensive in terms of energy, so reducing the accesses to this memory has a profound effect on the overall energy dissipation.

The *global-to-point* kernels seem to benefit especially when going from a relative low number of PEs to a higher amount. In particular the kernels that gather elements from all different PEs exhibit this effect. In Fig. 8c, the biggest increase in energy efficiency is observed when increasing the number of PEs from 16 to 32. For lower numbers of PEs, there is only a small reduction of the energy dissipation. The overhead of the control of the array is hardly compensated by the exploited DLP at this point. When the number of PEs increases, more DLP can be exploited, while the control overhead remains similar. This leads to an increased energy efficiency. When more and more PEs are added, the neighborhood network starts to become a bottleneck. The positive effect of exploiting DLP is partly counteracted by the longer communication distances across the neighborhood network.

(a) point-to-point

(b) neighborhood-to-point

(c) global-to-point

(d) global-to-global

Fig. 8 Energy dissipation compared to RISC

Finally the *global-to-global* kernels show the least reduction in energy dissipation, which is shown in Fig. 8d. However, this is to be expected as their overall speedup as shown in Fig. 7d is also much less than the other kernel cate-

gories. More importantly, unlike the RISC processor, which can directly access its complete memory space, a PE within an SIMD processor requires extra operations to access data in other memory banks. This communication overhead significantly reduces the benefit of the increased exploitation of DLP. The mirror kernel is an extreme in these kernels, the SIMD actually performs worse than its RISC counterpart. These kernels exhibit a similar behavior as the *max* and *reduction* kernels of the global-to-point category. When going up from a small number of PEs, first the efficiency is low. Then it increases due to the exploitation of DLP. And in the end this is counteracted by the increased communication distances across the PE Array.

Overall, the examined kernels show a significant reduction in energy dissipation compared to RISC. In all of these kernels, a significant speed up in performance is achieved in the proposed SIMD architecture due to efficient DLP exploitation. For a design that focuses on low energy, techniques like *dynamic voltage and frequency scaling* (DVFS) can be applied to further improve the energy efficiency, while still meeting the same performance requirement as the RISC. DVFS is however out of the scope of this work.

4.2 Explicitly Bypassed SIMD vs. Automatically Bypassed SIMD

The goal of this section is to analyze the effectiveness of explicit bypassing over automatic bypassing in SIMDs. First energy breakdowns are presented in section 4.2.1, which give insight into where energy is being dissipated in both the explicit and transparent SIMD. Furthermore five aspects are discussed and analyzed per kernel category, namely number of RF accesses, RF energy dissipation, overall energy dissipation, performance, and area.

4.2.1 Energy Breakdowns

This section presents the energy breakdowns for one selected kernel out of each category in the benchmark. The breakdowns are given for each tested number of PEs and provide a comparison between the automatic and explicit bypassed SIMDs.

Each breakdown features six parts, the *MEM* accounts for the energy dissipated in both the instruction memory and all the data memories. *PE_RF* represents the energy dissipated in the RFs of all the PE's. Similarly *PE_EX* and *PE_ID* represent the energy dissipated in the execution stage and local decode stage of all the PEs respectively. The *PE_IF_ID* category includes both the energy dissipated in the instruction fetch for the PEs, and the shared part of the instruction decode. Finally the part labeled *CP* represents the energy dissipated by the CP.

The energy breakdown for binarization from the point-to-point category is given in Fig. 9. As expected, the shared parts, such as the instruction fetch, global decode and CP become less important when the number of PEs increases. In Fig. 9a, it can be seen that for the automatically bypassed SIMD the RF starts to play a bigger role in the overall energy dissipation as the number of PEs increases. In the explicit SIMD the energy dissipation in the RF is reduced to such an extent that it is no longer the dominating part in the overall energy dissipation, see Fig. 9b.

(a) Energy Breakdown on Automatically Bypassed SIMD

(b) Energy Breakdown on Explicitly Bypassed SIMD

Fig. 9 Energy Breakdown for Binarization

For the Convolution kernel from the neighborhood-to-point category, the energy breakdown is shown in Fig.10. From the figure it is clear that the RF dominates the energy dissipation even more, especially in the automatically bypassed SIMD. The neighborhood-to-point kernels generally store data elements from the neighborhood in the RF, and update it as the neighborhood window slides over the input. This explains why the RF in this case is used more, and thus dissipates relatively more energy than in the point-to-point kernels. The positive effect of explicit bypassing on the energy dissipation of the RF can be clearly seen in Fig. 10b.

The RF still accounts for a larger portion of the total energy dissipation as the number of PEs increases, but the the explicit datapath techniques significantly reduce the energy dissipation relative to the automatically bypassed SIMD.

(a) Energy Breakdown on Automatically Bypassed SIMD

(b) Energy Breakdown on Explicitly Bypassed SIMD

Fig. 10 Energy Breakdown for Convolution

The breakdown of the Vector-Vector Addition kernel from the global-to-point category is given in Fig. 11. Comparable with the Convolution kernel, the Vector-Vector Addition kernel uses the RF heavily to exploit locality. This is why the RF again accounts for such a large amount of the total energy dissipation in the automatically bypassed SIMD, as can be seen in Fig. 11a. However, the vast majority of variables in the Vector-Vector Addition kernel are short-lived, since each loaded element is just added to a sum without any other computations. This enables explicit bypassing to achieve high savings in energy dissipation, as a large number of accesses to the RF can be avoided. Given that the computation of the Vector-Vector Addition is so simple, and the RF is almost not accessed due to explicit bypassing, the memory accesses dominate the energy dissipation of the explicit SIMD in Fig. 11b.

The last kernel for which we present an energy breakdown is Matrix Transpose from the global-to-global category. In this kernel data predominantly moves between the PEs. Since the PEs communicate by accessing each other's

(a) Energy Breakdown on Automatically Bypassed SIMD

(b) Energy Breakdown on Explicitly Bypassed SIMD

Fig. 11 Energy Breakdown for Vector-Vector Addition

operands, every communication will result in a read and write of the RF in the automatically bypassed SIMD. This is why also here the RF plays such a dominant role in the automatic SIMD, as shown in Fig. 12a. Furthermore, when the number of PEs increases, so does the average communication distance. Therefore there are relatively more accesses to the RF for larger number of PEs, making the RF even more important for an increasing number of PEs. Explicit bypassing avoids a large amount of the RF accesses during long distance communication. After all, data is passed from neighbor to neighbor and never needs to be committed into the RF. This is why also here explicit bypassing is so effective at reducing the contribution of the RF to the total energy dissipation, as is visible in Fig. 12b.

Overall it can be seen that the RF plays an increasingly more dominant role in the automatically bypassed SIMD when the number of PEs increases. Yet the explicit bypassing techniques significantly reduce the contribution of the energy dissipation of the RF. In the following sections the automatically bypassed and explicitly bypassed SIMD will be compared for each of the kernel categories with respect

(a) Energy Breakdown on Automatically Bypassed SIMD

(b) Energy Breakdown on Explicitly Bypassed SIMD

Fig. 12 Energy Breakdown for Matrix Transpose

to the absolute number and type of RF accesses, energy dissipation in the RF, overall energy dissipation.

4.2.2 point-to-point

As can be seen later in this section, the main cause of reduction in energy dissipation in an explicitly bypassed SIMD processor is the reduction of traffic to the register file. Fig. 13a shows how many RF accesses (both RF reads and writes) are avoided compared to the automatically bypassed SIMD.

The point-to-point kernels only read and write to the private data memory of a PE. Only the data in the current location is needed to calculate a new one. Therefore, most variables are short-lived and can be bypassed. This is shown in terms of avoided accesses (Fig. 13a). For the binarization kernel, the number of remaining RF writes is even reduced to zero. Each pixel is loaded, compared to a threshold and written back to the main memory. The lifespan of the pixel is short enough to avoid involving the RF. The RF is only used to hold memory addresses and the threshold for binarization.

If a large kernel has only one RF access, avoiding that access hardly brings any reduction of energy dissipation.

(a) RF access reduction (128 PE), normalized to the automatically bypassed SIMD

(b) Reads per cycle

(c) Writes per cycle

Fig. 13 RF access of point-to-point kernels

Therefore it makes sense to analyze the effectiveness of explicit bypassing by looking at the average number of read/write accesses per cycle. Fig. 13b and Fig. 13c show the absolute number of reads and writes per cycle respectively. The red bars indicate the extra accesses required by the automatically bypassed SIMD over the explicitly bypassed SIMD.

Fig. 13b shows that for the color conversion kernel, relatively more reads are avoided than for the binarization kernel. However, binarization almost completely avoids all writes as is shown in Fig. 13c. Since RF writes consume more energy than RF reads, this explains why binarization

saves more RF energy as is shown in Fig. 14a. For the overall energy dissipation, i.e., including the core, RF and memories, color conversion kernel has a higher reduction as shown in Fig. 14b. This is because the color conversion has both a larger number of reads and a larger number of writes per cycle to start with (Fig. 13b and Fig. 13c), which means the percentage of RF energy dissipation within the complete processor is higher. Although the reduction of energy dissipation in the RF is less than for binarization, the total amount of reduction of energy dissipation in the color conversion kernel is still higher.

Fig. 14 Energy usage reductions by explicit bypassing: point-to-point kernels

Explicit bypassing results in significant reduction in energy dissipation for the point-to-point kernels. The reduction of overall energy usage increases when the number of PEs increases. This is because, as previously shown in Section 4.2.1, the datapath, including the RF, of an SIMD processor plays an increasingly more important role in the overall energy dissipation when more PEs are added. This again shows that reducing the RF energy dissipation is particularly effective in (wide) SIMDs.

4.2.3 neighborhood-to-point

For the neighborhood-to-point kernels behavior similar to that of the point-to-point kernels is observed. Roughly

(a) RF access reduction (128 PE), normalized to the automatically bypassed SIMD

(b) Reads per cycle

(c) Writes per cycle

Fig. 15 RF access of neighborhood-to-point kernels

around 60 to 70% of the original RF accesses are avoided when explicit bypassing is applied (Fig. 15a). The largest reduction in accesses is observed for the erosion kernel.

This is also reflected in the number of accesses per cycle, as shown in Fig. 15b and Fig. 15c. Among these kernels, the number of writes per cycle of the convolution kernel decreases the most. This is also the reason the convolution kernel has the largest reduction of energy dissipation, both in the RF and overall, as can be seen in Fig. 16a and Fig. 16b respectively.

Fig. 16 Energy usage reductions by explicit bypassing: neighborhood-to-point kernels

Compared to the point-to-point category, it can be seen in Fig. 14b and Fig. 16b that in the neighborhood-to-point category the overall energy dissipation reduces the most. This is due to the inherent nature of the neighborhood-to-point category. Since the kernels in this category gather surrounding pixels and merge them into a single value, a lot of short-lived variables exist. Pixels are moved around the neighborhood and absorbed quickly. Moving the pixels around typically requires a large number of RF accesses in the automatically bypassed datapath. In the explicit datapath, these short-lived variables provide an excellent opportunity to reduce RF accesses, incurring a large reduction in the overall energy usage. This can also be observed by comparing the initial number of accesses per cycle of the neighborhood-to-pixel category in Fig. 15b and Fig. 15c to the corresponding figures of the other kernel categories. No other category has such a large amount of accesses per cycle in the automatic datapath and reduces the number of accesses by this much.

4.2.4 global-to-point

From the register file access numbers in Fig. 17, it can be seen that the max and reduction kernel both reduce the number of accesses by a significant amount. The vectoradd kernel is an outlier however, and avoids almost all accesses.

This is because the max and reduction kernels combine data elements that are spread out across the PE-array. Therefore, they cause a large amount of communication and control overhead, in order to coordinate the data transfers. The vectoradd kernel, however, only combines data elements that are already located in the same PE data memory. Because all data elements are already located in the private data memory of a PE, the values only need to be loaded and added to a sum variable located in the RF. In the automatic datapath, each load induces a write to the RF, and summing the loaded value causes two reads. Since loading and adding a pixel can be done in just a couple of instructions, these accesses to the RF can be almost completely avoided. This is why in Fig. 17a vectoradd reduces the RF accesses much more than the other two kernels. It is therefore no surprise that the vectoradd kernel reduces the energy usage most of all kernels in the global-to-point category (Fig. 18). The energy dissipated in the RF is significant with automatic bypassing for the vectoradd kernel, but with explicit bypassing the energy used in the RF is reduced by more than 90%.

It should be noted that for 128 PEs, the reduction in overall energy usage for these kernels is less than the reduction for 64 PEs (Fig. 18b). After detailed analysis it was not possible to conclusively determine the cause of this unexpected drop in the energy usage reduction. A similar drop is observed for FFoS in Fig. 22b. Further research is needed to find a satisfactory explanation for these drops.

4.2.5 global-to-global

The global-to-global kernels require a large amount of long distance communication. In PE-to-PE communications, variables only pass through the PEs, hence they do not need to be stored in the RF. This is the reason that explicit bypassing avoids a significant amount of the RF accesses in the global-to-global kernels, as can be seen from Fig. 19. When the number of PEs increases, the percentage of PE-to-PE communications increases accordingly. This directly translates into an increasing energy efficiency for the global-to-global kernels, as is shown in Fig. 20.

4.2.6 FFoS

In this section, an industrial application, *Fast Focus on Structures* (FFoS), is benchmarked. The size of the input image is $1024 \times N_{PE}$, where N_{PE} is the number of PEs in a particular SIMD instantiation. Because the number of rows of the input image is fixed, the number of avoided RF reads and RF writes per cycle is hardly influenced when more PEs are added (Fig. 21b and Fig. 21c). As a result, the reduction of the energy dissipation in the RF is around 48%, as is shown in Fig. 22a.

(a) RF access reduction (128 PE), normalized to the automatically bypassed SIMD

(b) Reads per cycle

(c) Writes per cycle

Fig. 17 RF access of global-to-point kernels

The FFoS application is particularly memory intensive. In order to clearly show the effects of explicit bypassing, which does not affect energy used in the data memories, Fig. 22b only shows the reduction in energy dissipation in the core/logic part of the processor. It is interesting that the FFoS application shows an overall improved energy efficiency for the logic part when the number of PEs is increased, even though the number of avoided RF reads and writes per cycle is hardly influenced. This is due to the fact that the register file, percentage-wise, contributes a larger part to the total energy dissipation, because the instruc-

(a) RF energy usage reduction

(b) Overall energy usage reduction

Fig. 18 Energy usage reductions by explicit bypassing: global-to-point kernels

tion fetch and decode are amortized over more PEs. This makes the register file's contribution to the energy dissipation larger, so even though the reduction in the register file is nearly constant, overall the energy usage reduces as the number of PEs increases.

4.2.7 Performance

Table 6 shows the cycle count of each kernel on both the 128-PE SIMD with explicit bypassing and the 128-PE SIMD with automatic bypassing. The result shows that explicit bypassing introduces almost no performance loss.

4.2.8 Area

The core areas of the automatically bypassed SIMDs are shown in Table 7. Compared to Table 5, we can see that the explicitly bypassed SIMDs occupy slightly less area. This is because the explicitly bypassed SIMDs have slightly smaller physical register files and simpler bypassing logic.

5 Related Work

Reducing the energy dissipation of the register file has always been considered important in improving processor energy efficiency [10, 32]. About 15% of the core energy

(a) RF access reduction (128 PE), normalized to the automatically bypassed SIMD

(b) Reads per cycle

(c) Writes per cycle

Fig. 19 RF access of global-to-global kernels

within a typical single-issue RISC processor is dissipated by the register file, and an even higher percentage for processors that exploit more instruction-level or data-level parallelism [9, 30, 32]. Earlier work has shown that optimizing the bypassing network can reduce this large energy usage [6]. For example, in VLIWs it has been shown that storing short-lived values in pipeline registers can reduce energy usage while sustaining the compute performance [5]. Similarly, in a *transport triggered architecture* (TTA), which is considered to be a superset of the VLIW architecture [3], the reduction of energy dissipation in the RF induced by explicit

(a) RF energy usage reduction

(b) Overall energy usage reduction

Fig. 20 Energy usage reductions by explicit bypassing: global-to-global kernels

Table 6 Cycle count of each kernel on 128-PE SIMDs

Kernel	Explicit	Automatic
Binarization	527	526
Color Conversion	4383	4382
Erosion 3x3	747	747
5-tap FIR	1411	1411
Convolution 3x3	2452	2452
Find Max. Element in a Vector	1100	1099
Sum of Vector Elements	844	843
Vector-Vector Addition	268	268
Matrix Rotation	5671	5542
Matrix Mirror	11273	11015
Matrix Transpose	5606	5477
FFoS	32961	32961
Average performance loss	0.62 %	

bypassing has been shown to be as much as 80%, leading to a reduction of the overall energy dissipation of 11% [10]. A compiler was developed for this TTA [26]. It fully automates explicit bypassing and achieves the same amount of energy reduction. This proves the practical value of explicit bypassing. However, none of the related works provide a detailed head-to-head comparison in terms of energy efficiency between explicit and automatic bypassing in an SIMD setting.

Explicit bypassing is also used to improve performance by mitigating RF pressure on both size and number of read-

(a) RF access reduction (128 PE), normalized to the automatically bypassed SIMD

(b) Reads per cycle

(c) Writes per cycle

Fig. 21 RF access of FFoS

/write ports. The MOVE work [20] and the TCE work [21], both of which are TTAs, studied this thoroughly. J. Yan et al. introduced a similar concept, called virtual register, which exploits the short-lived variables and the data bypassing network to minimize the demand on real registers [33]. Instead of focusing on power dissipation, this work mainly aimed at achieving higher performance without enlarging the RF physically. Compared to this work, the proposed architecture exploits the same principle, but with a more compact instruction format, resulting in smaller instruction memory and less expensive memory access in terms of energy dissipation. Moreover, in the proposed architecture, data stays available longer for bypassing. This is because input latches

(a) RF energy usage reduction

(b) Energy usage reduction in logic

Fig. 22 Energy usage reductions by explicit bypassing: FFoS

	RISC	Automatic Bypass				
#PE	—	8	16	32	64	128
Cell	10884 (1.0)	90489 (8.3)	169267 (15.6)	329952 (30.3)	645903 (59.3)	1275403 (117.2)
Net	23509 (1.0)	205219 (8.7)	383131 (16.3)	747418 (31.8)	1454670 (61.9)	2894902 (123.1)
Total	34392 (1.0)	295708 (8.6)	552398 (16.1)	1077370 (31.3)	2100573 (61.1)	4170305 (121.3)

Table 7 Core area of automatically bypassed SIMD compared to RISC

are introduced to each functional unit (FU), such that an FU output is preserved till the same FU is used by another instruction. Since data is available longer for bypassing, more variables can be bypassed, reducing the traffic from/to the RF [10].

Wide SIMD architectures are widely used in embedded processors. The Xetal from NXP [1] is an SIMD processor with 320 PEs that is designed for smart camera data processing. The PEs in Xetal are connected by a neighborhood network. However, Xetal has one large vector memory (called frame memory). It does not have any small memory (e.g., a register file) in between to exploit data locality. Because of this, energy dissipation of data accesses is high. Y. He et al. addressed this issue in Xetal-Pro. By introducing an extra level of memory, as well as aggressive voltage scaling, a much more efficient architecture is obtained [9, 22]. This work shows that to achieve ultra-low power, improving the efficiency of data movement is of crucial importance in

SIMDs, which motivated us to introduce explicit datapath techniques in this work. The IMAPCAR from NEC [16] is another example of a wide SIMD processor. The IMAPCAR has 128 PEs connected with a ring network. A key difference in IMAPCAR compared to Xetal is that it has independent address generation for each PE. While the memory is more complex in such a configuration, it also results in much better programmability. Since many applications, such as histogram and Hough transform, can benefit from independent address generation [13, 14], the proposed architecture supports independent address generation for each PE.

M. Woh et al. proposed AnySP, a wide SIMD targeting wireless and multimedia applications [32]. The PE interconnect in AnySP is a reconfigurable RAM-based crossbar, which is more flexible compared to Xetal, IMAPCAR, and this work. The energy usage of the vector register file in AnySP is reduced by introducing an extra 4-entry small register file. AnySP also uses explicit bypassing. However, instead of using a small RF to increase the bypass opportunities and reduce the RF pressure, we achieve these goals by increasing the number of bypassing sources in this work.

In another work of M. Woh et al., the evolution from SODA to Ardbeg is presented [31]. It is noted that the RF is the largest power consumer in SODA, accounting for 30% of the total power. To mitigate this problem, Ardbeg introduces 2-issue *long instruction word* (LIW) support, allowing a restricted set of operations to run in parallel. In order to facilitate LIW, the RF requires two read and two write ports, making the RF more complex, and therefore presumably more power hungry. Yet the performance gained by the 2-issue LIW results in an overall better energy-delay product. This technique is orthogonal to the explicit datapath approach evaluated here, and it would be interesting to investigate how much power can be reduced by combining the two techniques.

In our work, the proposed architecture is similar to the Xetal-Pro [9]. The main differences are that the proposed architecture uses per-PE register files, independent addressing, and a PE datapath with explicit bypassing. Compared to the PE micro architecture of Xetal-Pro, which supports limited operation types due to its simplicity [9], the PE micro architecture of this work is RISC-like and supports more operation types.

6 Conclusions

In this work, a low-energy, wide SIMD architecture with explicit datapath is proposed. The proposed architecture is fully programmable and features a configurable number of processing elements and pipeline stages. Scalar operations and (wide) vector operations are issued in parallel to exploit DLP and ILP at the same time.

To show the effectiveness of the proposed architecture, an instantiation of the explicitly bypassed architecture with 128 PEs is compared with a reference RISC architecture. The experimental results show that the SIMD processor reduces the energy dissipation by up to 94% in the erosion kernel and by 48.3% on average for the total of eleven tested kernels. The proposed SIMD processor also achieves an average of $206.1\times$ speed up compared to the reference RISC, even though it only has 128 PEs. This is because in the proposed SIMD architecture, scalar operations and (wide) vector operations are issued in parallel to exploit DLP and ILP at the same time, and enhanced exploitation of data locality.

To demonstrate the effectiveness of explicit bypassing in an SIMD environment, multiple instantiations of the proposed architecture are implemented. Eleven representative kernels and one industry application are mapped onto all these instantiations, as well as their automatically bypassed counterparts. Detailed comparison and analysis are carried out. The experimental results show that, compared to the automatic bypassing counterpart, a considerable number of RF accesses, 64% on average for 128 PEs, are avoided by using explicit bypassing. For total energy dissipation, an average of 27.5%, and maximum of 43.0%, reduction is achieved.

Future work includes exploring low-power interconnects for wide SIMDs to improve both performance and energy efficiency. Place and route is also considered to investigate the effect of the architectural changes on wiring, both in terms area and power. It would also be interesting to use software-hardware co-exploration to improve the efficiency of kernels with irregular communication patterns, or with long distance communication patterns. In addition, further changes in the architecture, e.g., clustering PE memory banks to reduce memory energy dissipation, introducing multiple-issue PE micro architecture, and supporting complex operations with special function units, are also part of our future work.

Acknowledgements This work is supported by the Ministry of Economic Affairs of the Netherlands, project EVA PID07121, and the Dutch Technology Foundation STW, project NEST 10346.

References

1. Abbo, A., Kleihorst, R., Choudhary, V., Sevat, L., Wielage, P., Mouy, S., Vermeulen, B., Heijligers, M.: Xetal-II: A 107 GOPS, 600 mW massively parallel processor for video scene analysis. *IEEE Journal of Solid-State Circuits (JSSC)* **43**(1), 192–201 (2008)
2. Amdahl, G.M.: Validity of the single processor approach to achieving large scale computing capabilities, reprinted from the afips conference proceedings, vol. 30 (atlantic city, n.j., apr. 18 x2013;20), afips press, reston, va., 1967, pp. 483 x2013;485, when dr. amdahl was at international business machines corporation, sunnyvale, california. *Solid-State Circuits Society Newsletter, IEEE* **12**(3), 19–20 (2007). DOI 10.1109/N-SSC.2007.4785615
3. Corporaal, H.: *Microprocessor Architectures: From VLIW to TTA*. John Wiley & Sons (1998)

4. Frijns, R., Fatemi, H., Mesman, B., Corporaal, H.: DC-SIMD : Dynamic communication for SIMD processors. In: Proceedings of International Symposium on Parallel and Distributed Processing (IPDPS), pp. 1–10 (2008)
5. Goel, N., Kumar, A., Panda, P.: Power reduction in VLIW processor with compiler driven bypass network. In: Proceedings of the 20th International Conference on VLSI Design (VLSID), pp. 233–238 (2007)
6. Guan, X., Fei, Y.: Reducing power consumption of embedded processors through register file partitioning and compiler support. In: Proceedings of International Conference on Application-Specific Systems, Architectures and Processors (ASAP), pp. 269–274 (2008)
7. Gustafson, J.L.: Reevaluating amdahl's law. *Commun. ACM* **31**(5), 532–533 (1988). DOI 10.1145/42411.42415. URL <http://doi.acm.org/10.1145/42411.42415>
8. He, Y.: Low Power Architectures for Streaming Applications. PhD Thesis (2013)
9. He, Y., Pu, Y., Ye, Z., Londono, S., Kleihorst, R., Abbo, A., Corporaal, H.: Xetal-Pro: An ultra-low energy and high throughput SIMD processor. In: Proceedings of the 47th Design Automation Conference (DAC), pp. 543–548 (2010)
10. He, Y., She, D., Mesman, B., Corporaal, H.: MOVE-Pro: A low power and high code density TTA architecture. In: Proceedings of the 11th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), pp. 294–301 (2011)
11. He, Y., Ye, Z., She, D., Mesman, B., Corporaal, H.: Feasibility analysis of ultra high frame rate visual servoing on fpga and simd processor. In: Proceedings of Advances Concepts for Intelligent Vision Systems (ACIVS), pp. 623–634 (2011)
12. He, Y., Ye, Z., She, D., Pieters, R., Mesman, B., Corporaal, H.: 1000 fps visual servoing on the reconfigurable wide SIMD processor. In: Proceedings of the 16th Annual Conference of the Advanced School for Computing and Imaging (ASCI), pp. 302–309 (2010)
13. He, Y., Zivkovic, Z., Kleihorst, R., Danilin, A., Corporaal, H.: Real-time implementations of hough transform on simd architecture. In: Proceedings of the ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC), pp. 1–8 (2008)
14. He, Y., Zivkovic, Z., Kleihorst, R., Danilin, A., Corporaal, H., Mesman, B.: Real-time hough transform on 1-d simd processors: Implementation and architecture exploration. In: Proceedings of the International Conference Advanced Concepts for Intelligent Vision Systems (ACIVS), pp. 254–265 (2008)
15. Kapasi, U., Dally, W., Rixner, S., Owens, J., Khailany, B.: The imagine stream processor. In: Proceedings of International Conference on Computer Design: VLSI in Computers and Processors (ICCD), pp. 282–288 (2002)
16. Kyo, S., Okazaki, S.: IMAPCAR: A 100 GOPS in-vehicle vision processor based on 128 ring connected four-way VLIW processing elements. *Journal of Signal Processing Systems* pp. 1–12 (2008)
17. Otsu, N.: A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics* **11**, 285–296 (1975)
18. Prengler, A., Adi, K.: A reconfigurable SIMD-MIMD processor architecture for embedded vision processing applications. In: SAE World Congress, pp. 1–9 (2009)
19. CACTI: cacti 5.3, rev 174. <http://quid.hpl.hp.com:9081/cacti/>
20. Delft University of Technology: MOVE project. <http://ce.et.tudelft.nl/MOVE/>
21. Tampere University of Technology: TTA-based Codesign Environment (TCE). <http://tce.cs.tut.fi/>
22. Pu, Y., He, Y., Ye, Z., Londono, S., Abbo, A., Kleihorst, R., Corporaal, H.: From Xetal-II to Xetal-Pro: On the road toward an ultra low-energy and high-throughput SIMD processor. *IEEE Transactions on Circuits and Systems for Video Technology (TCAS-VT)* **21**(4), 472–484 (2011)
23. Raghavan, P., Munaga, S., Ramos, E., Lambrechts, A., Jayapala, M., Cathoor, F., Verkest, D.: A customized cross-bar for data-shuffling in domain-specific SIMD processors. In: Proceedings of Architecture of Computing Systems (ARCS), pp. 57–68 (2007)
24. Satpathy, S., Foo, Z., Giridhar, B., Dreslinski, R., Sylvester, D., Mudge, T., Blaauw, D.: A 1.07 tbit/s 128x128 swizzle network for SIMD processors. In: Proceedings of IEEE Symposium on VLSI Circuits (VLSIC), pp. 81–82 (2010)
25. She, D., He, Y., Corporaal, H.: Energy efficient special instruction support in an embedded processor with compact isa. In: Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES), pp. 131–140 (2012)
26. She, D., He, Y., Mesman, B., Corporaal, H.: Scheduling for register file energy minimization in explicit datapath architectures. In: Proceedings of Design, Automation Test in Europe Conference Exhibition (DATE), pp. 388–393 (2012)
27. She, D., He, Y., Waeijen, L., Corporaal, H.: OpenCL code generation for low energy wide SIMD architectures with explicit datapath. In: Proceedings of International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), pp. 322–329 (2013)
28. Waeijen, L., She, D., Corporaal, H., He, Y.: SIMD made explicit. In: Proceedings of International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), pp. 330–337 (2013)
29. Waeijen, L., She, D., Corporaal, H., He, Y.: Reduction operator for Wide-SIMDs reconsidered. In: Proceedings of the 51st Design Automation Conference (DAC) (2014)
30. van de Waerdt, J., et al.: The TM3270 media-processor. In: Proceedings of the 38th International Symposium on Microarchitecture (MICRO), pp. 331–342 (2005)
31. Woh, M., Lin, Y., Seo, S., Mahlke, S., Mudge, T., Chakrabarti, C., Bruce, R., Kershaw, D., Reid, A., Wilder, M., Flautner, K.: From soda to scotch: The evolution of a wireless baseband processor. In: Microarchitecture, 2008. MICRO-41. 2008 41st IEEE/ACM International Symposium on, pp. 152–163 (2008). DOI 10.1109/MICRO.2008.4771787
32. Woh, M., Seo, S., Mahlke, S., Mudge, T., Chakrabarti, C., Flautner, K.: AnySP: Anytime anywhere anyway signal processing. *IEEE Micro* **30**(1), 81–91 (2010)
33. Yan, J., Zhang, W.: Virtual registers: Reducing register pressure without enlarging the register file. In: Proceedings of High Performance Embedded Architectures and Compilers (HiPEAC), pp. 57–70 (2007)