# DIM-VEX: Exploiting Design Time Configurability and Runtime Reconfigurability

Jeckson Dellagostin Souza[1(✉)], Anderson L. Sartor[1], Luigi Carro[1], Mateus Beck Rutzig[2], Stephan Wong[3], and Antonio C. S. Beck[1]

[1] Instituto de Informática, Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, Brazil
{jeckson.souza,alsartor,carro,caco}@inf.ufrgs.br
[2] Departamento de Eletrônica e Computação, Universidade Federal de Santa Maria (UFSM), Santa Maria, Brazil
mateus@inf.ufsm.br
[3] Computer Engineering Laboratory, Delft University of Technology, Delft, The Netherlands
j.s.s.m.wong@tudelft.nl

**Abstract.** Embedded processors must efficiently deliver performance at low energy consumption. Both configurable and reconfigurable techniques can be used to fulfill such constraints, although applied in different situations. In this work, we propose DIM-VEX, a configurable processor coupled with a reconfigurable fabric, which can leverage both design time configurability and runtime reconfigurability. We show that, on average, such system can improve performance by up to 1.41X and reduce energy by up to 60% when compared to a configurable processor at the cost of additional area.

**Keywords:** Reconfigurable accelerator · Configurable processor
Binary translation · Binary compatibility

## 1 Introduction

For the past decades, embedded processors gained a vast market share. These processors can be found in devices ranging from mobiles to IoT nodes. They must meet (sometimes conflicting) requirements such as high performance, low energy consumption, and be small in size. One way of achieving such trade-offs is through configurable processors. A processor is configurable when it allows the design of different versions of the same processor, varying a significant number of features, such as the number of instructions it can issue, the size of its register file, special or customized instructions, and so on. It is done before deployment and gives the designer enough flexibility to build the processor according to a given set of constraints (e.g.: area and power) and the applications it will execute.

An example of a configurable processor is the $\rho$-VEX Very Long Instruction Word (VLIW) [14]. It is based on the VEX Instruction Set Architecture


N. Voros et al. (Eds.): ARC 2018, LNCS 10824, pp. 367–378, 2018.
https://doi.org/10.1007/978-3-319-78890-6_30

(ISA) with extended reconfigurable custom instructions. $\rho$-VEX also allows the parameterization of several hardware modules, such as the issue-width, register file size, type and amount of functional units (FUs), and memory buses, allowing a huge design space exploration, not only for performance and energy, but also for other requirements such as fault tolerance [10].

However, this flexibility may not always be enough to meet system requirements. Reconfigurable organizations emerge as an alternative since they can be coupled to processors to boost performance and save energy [3]. These systems can adapt themselves to the application at hand, reconfiguring their datapaths to maximize Instruction-Level Parallelism (ILP) exploitation and improve execution times [1] over classic processors. A particular advantage of such systems is: as it is highly regular, it is possible to couple multiple simple ALUs (Arithmetic and Logic Units) in sequence to execute several dependent operations in one processor cycle, without reducing the operating frequency [2].

In this work, we propose to leverage the advantages of reconfigurable hardware to expand the ILP exploitation capabilities of configurable processors even further, using the $\rho$-VEX as a case study. To maintain binary compatibility with the VEX ISA, we also use a binary translation system capable of identifying hotspots in the application code and dynamically creating configurations to be executed on the reconfigurable fabric at runtime. Therefore, we created a system which can be parameterized (configurable) during design time - not only the processor features but also the reconfigurable fabric - and reconfigurable during runtime as well, named as DIM-VEX (Dynamic Instruction Merging for VEX).

We show a diverse analysis of systems composed of differently sized reconfigurable fabrics and distinct $\rho$-VEX processors. Our results show that, while the DIM-VEX may require a considerable additional amount of area, it is possible to increase the performance and reduce the energy consumption of the configurable processors in many scenarios. On average, it is possible to achieve speedups of 1.41X and reduce energy consumption by up to 60% when using a reconfigurable fabric.

The remaining of this paper is organized as follows. Section 2 discusses the background of reconfigurable computing and the state-of-the-art. Section 3 describes the proposed DIM-VEX architecture. Section 4 explains the methodology used in this work, while Sect. 5 presents the results and analysis of the tested systems. Finally, Sect. 6 draws conclusions and future works.

## 2 Background and Related Work

Reconfigurable systems can adapt themselves to provide acceleration for specific applications. This is achieved through an additional circuit that offers reconfigurability, like a Field-Programmable Gate Array (FPGA) or a reconfigurable array of functional units. These organizations provide performance gains and energy savings over General Purpose Processors (GPP), at the cost of extra area. The reconfigurable logic can be classified by how it is connected to the main processor as follows [3]:

- loosely connected to the processor as an I/O peripheral (communication done through the main memory);
- attached as a coprocessor (using specific protocols for communication);
- tightly coupled as a functional unit (reconfigurable logic is inside the processor and share its resources, like its register file).

Furthermore, the granularity of the reconfigurable logic determines its level of data manipulation. A fine-grained logic is implemented at bit level (like Look-Up Tables in FPGAs) while a coarse-grained logic implements word level circuits (like ALUs and multipliers) [1]. Current implementations of reconfigurable systems usually favor coarse-grain reconfigurable arrays (CGRAs), as they present the following advantages:

- they can be tightly coupled to the main processor, avoiding significant penalties in communication;
- as configuration is applied at word level, the size of the contexts holding configuration bits is much smaller than those from fine-grained architectures;
- they have smaller reconfiguration latencies than fine-grained (e.g., FPGAs) fabrics, even when one considers only partial reconfiguration in the latter [1].

It is possible to use CGRAs as a generic accelerator by providing tools to dynamically analyze and translate regions of code for execution on the array. To achieve this, it is common to combine reconfigurable architectures with dynamic Binary Translation (BT) [6] techniques - in which the system is responsible for monitoring and transforming parts of the binary code, at runtime, in order to accelerate it; and Trace Reuse [4], which relies on the idea that a sequence of instructions will execute repeatedly using the same operands during the application execution. By associating these strategies, one can maintain the binary compatibility between the main processor and the reconfigurable fabric, while avoiding re-translating repetitive instruction block. An extensive characterization and classification study on reconfigurable architectures is presented in [1,3], and we discuss next a variety of works using reconfigurable architectures with dynamic binary translation.

Early studies on dynamically reconfigurable processors include the Warp Processor [9]. This system is based on a complex System-on-a-Chip (SoC), composed of a microprocessor that executes the regular application, a second microprocessor responsible for running simplified CAD algorithms, local memory, and an FPGA fabric. A profiler monitors the execution of the application to detect hotspots. Subsequently, the CAD software decompiles the application code into a control flow graph and synthesizes a circuit to execute the hotspot flow into the FPGA. Finally, the original binary code is modified to use the synthesized circuit. KAHRISMA [8] is an example of a completely heterogeneous architecture. It supports multiple instruction sets (RISC, 2- 4- and 6-issue VLIW, and EPIC) and fine and coarse-grained reconfigurable arrays. Software compilation, ISA partitioning, custom instructions selection, and thread scheduling are made by a design-time tool that decides, for each part of the application code, which assembly code will be generated, considering its dominant type of parallelism and resources availability. A run-time system is responsible for code binding and for avoiding execution

collisions in the available resources. Although both systems are dynamic and software compatible, they are heavily dependent on compilers and CAD tools, which increase software deployment time and the execution overhead.

More recent works have been proposed such as the DIM [2], HARTMP [12] and DORA [13]. DIM is a coarse-grained reconfigurable array (CGRA) tightly coupled to a RISC processor and a dynamic BT system. The BT executes parallel to the RISC pipeline, monitoring hotspots and its data dependencies, and allocating instructions to run in parallel inside the array. When the hotspot is re-executed, the RISC pipeline is stalled and execution is transferred to the reconfigurable fabric. DIM supports speculation and the execution of multiple ALU operations in the same cycle, which significantly increases the IPC of the processor. HARTMP is based on a similar strategy as DIM. However, it uses multicore systems with individual arrays for each core. The arrays are dimensioned distinctly to create heterogeneous environments, in which applications can schedule less demanding workloads on energy efficient arrays. DORA is a BT system coupled with a GPP and a CGRA (based on the DySER [5] system). The BT is realized through a microprocessor, which is responsible not only to transform hotspots for the reconfigurable fabric but also to use dynamic optimizations on the parts of application code that run on the GPP. Both the code transformation and optimization are saved for reuse.

Differently from previous works, which use reconfigurable systems to accelerate either RISC or superscalar processors, our work proposes to accelerate configurable VLIW processors and exploit the available design space offered by this configurability. For that, we have adapted the DIM CGRA to work together with the $\rho$-VEX, creating the DIM-VEX. As already mentioned, the advantage of using $\rho$-VEX for this work is its ability to parameterize its FUs, issue-width, memory and register file. We use the parameterizable issue-width to create multiple configurations for design space exploration. As for the DIM, its ability to execute various dependent ALU operations in the same cycle perfectly exploits the low frequencies of VLIW processors, as more of these FUs can be nested in sequence. In this work, we create a variety of systems composed of different $\rho$-VEX configurations and different DIM-VEX versions. We show how a dynamic CGRA can further expand the ILP exploitation of VLIW processors and analyze the impact on performance, energy, and area of such systems.

## 3 The Proposed Architecture

A general overview of the proposed architecture is given in the Fig. 1, which is an adaptation of the original DIM system proposed in [2]. Initial data flow is exactly the same as the original $\rho$-VEX: instructions are fetched, decoded and executed in the pipelanes of the processor. However, in the DIM-VEX system, instructions are also translated and saved for future execution on the CGRA. Block 1 in Fig. 1 shows the structure of the reconfigurable datapath. It is composed of registers for input and output context and a matrix of functional units. The matrix is a combinational block with ALUs (Arithmetic and Logic Units), Multipliers, Memory access ports and multiplexers. The matrix is composed of levels that

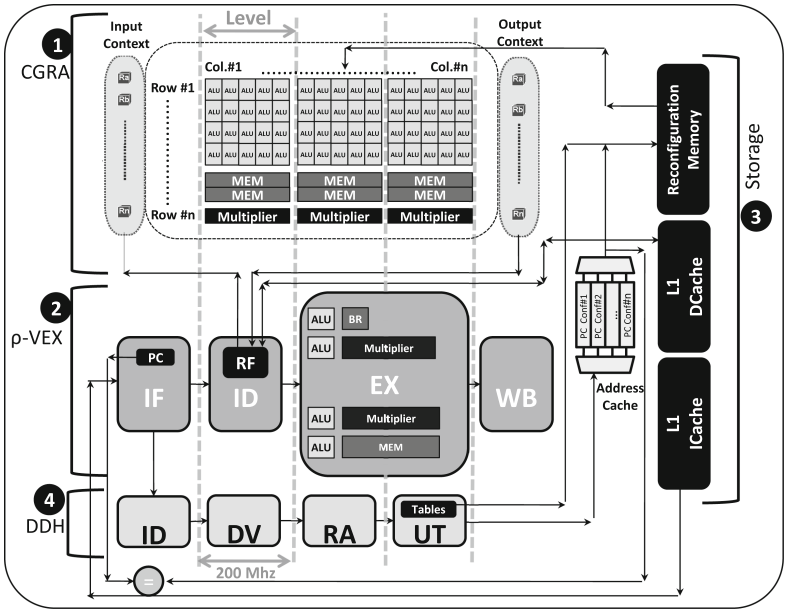


**Fig. 1.** DIM-VEX: a VLIW processor tightly coupled to a reconfigurable logic and a binary translator.

run in parallel with the VLIW pipeline (block 2). Each level has columns and rows. The columns have units that can run in parallel, executing instructions that do not have data dependency. As the multiplier and load operations stand as the critical path of the level, it is possible to align multiple ALUs in each row and keep the base frequency of the processor unchanged. In the given example, each ALU row has five columns and can execute five data dependent ALU instructions in the same level. During the reconfiguration process, a basic block is mapped to the matrix, to execute the whole block in a combinational fashion.

Block 2 shows the configurable processor coupled with the matrix. In this work, we use different $\rho$-VEX configurations all running at 200 MHz. This processor has a five-stage pipeline, in which the execution stage needs two cycles for the multiplication and load operations. We have also considered this same latency for multiplications and loads inside the CGRA. In block 3, the necessary storage components are illustrated. Apart from the usual L1 caches, two other memories are used. The address cache holds the address for each basic block decoded by the dynamic detection hardware (block 4) and is used as an index (and to check existence) for the datapath configurations. The reconfiguration memory holds the bits necessary to reconfigure the datapath into a basic block indexed by the address cache.

The Dynamic Detection Hardware (DDH), represented in block 4, does the binary translation and data dependency check of the instructions in a basic block. DDH is a four-stage pipelined circuit that runs in parallel to $\rho$-VEX, and thus being out of the critical path of the system. The Instruction Decode (ID) stage is responsible for decoding the operands in the base processor instruction to

datapath code, while the Dependence Verification (DV) checks if these operands have any dependency with the instructions already stored in the configuration being built. The Resource Allocation (RA) stage uses the DV analysis to determine the optimal functional unit for the given operation inside the array. Finally, the Update Tables (UT) phase saves the new allocation in the reconfiguration memory for future use. Every time a branch or an incompatible instruction is detected, a new configuration is started by the DDH, and a new entry is created in the address cache. Moreover, the DDH can also manage speculation for branch instructions to increase the matrix usage. Further details of the DDH and how it interacts with the reconfigurable array can be found in [2].

During the Instruction Fetch (IF) stage of the base processor, the Program Counter (PC) is compared to the values in the address cache. A hit in this cache means that the following sequence of instructions was already translated to a configuration. In this case, the processors pipeline is stalled, and the configuration is executed in the reconfigurable datapath, greatly exploiting the ILP of the application.

## 4 Methodology

We have designed three different configurations of the $\rho$-VEX processor, each using a specific issue-width (2, 4 and 8). Subsequently, we use these same $\rho$-VEX configurations to attach DIM components and create DIM-VEX platforms. We also vary the sizes of the CGRA to explore a wider design space. In our early experiments, we tested the $\rho$-VEX with different FUs as well; however, results showed negligible variation in cycle count. Details on the tested configurations, DIM-VEX and $\rho$-VEX standalone, are shown in Tables 1 and 2, respectively.

**Table 1.** Configurations for DIM-VEX. Columns include $\rho$-VEX issue width, levels in the CGRA, number of parallel ALUs in each level, number of sequential ALU in each row level and number of multipliers and memory access in each level

| Config | VLIW issues | Levels | ALUs | Seq ALUs | MULs | MEMs |
|---|---|---|---|---|---|---|
| DIM-VEX1 | 2 | 3 | 4 | 5 | 1 | 2 |
| DIM-VEX2 | 4 | 3 | 4 | 5 | 1 | 2 |
| DIM-VEX3 | 8 | 3 | 4 | 5 | 1 | 2 |
| DIM-VEX4 | 2 | 6 | 4 | 5 | 1 | 2 |
| DIM-VEX5 | 2 | 9 | 2 | 5 | 1 | 2 |

To extract power dissipation and area occupation of DIM-VEX, we have implemented the circuits of $\rho$-VEX and the FUs of the CGRA in VHDL and synthesized to CMOS 65 nm cells using a library from STMicroelectronics on the Cadence Encounter RTL compiler. The operation frequency of the $\rho$-VEX was set for 200 MHz. We were able to synthesize ALUs that run at 1/5 of the $\rho$-VEX cycle time; thus five of these units are nested in sequence to run in one clock

**Table 2.** Configurations for the $\rho$-VEX standalone. Columns include the issue width, number of pipelanes that contain an ALU, a multiplier and a memory port and number of registers in the RF.

| Config | VLIW issues | ALUs | MULs | MEMs | RF |
|---|---|---|---|---|---|
| $\rho$-VEX1 | 2 | 2 | 2 | 1 | 64 |
| $\rho$-VEX2 | 4 | 4 | 4 | 1 | 64 |
| $\rho$-VEX3 | 8 | 8 | 4 | 1 | 64 |

cycle inside the reconfigurable fabric of DIM-VEX, as shown in the column Seq ALUs in Table 1.

This work uses instruction traces generated directly from the VHDL simulation to feed an in-house cycle accurate simulator for the proposed architecture. We use this tool to reduce the simulation time of the applications and give us flexibility to analyze and parameterize the CGRA. The simulator emulates the behavior of the DIM algorithm, allocating instructions to be accelerated into configurations of the CGRA. With the configurations built, the simulator estimates power and performance of the application, while considering a realistic overhead for the CGRA reconfiguration time. We consider that the CGRA can perform power-gating of its functional units (a state of near 0 energy consumption) when they are not in use. We also consider both static power - the power dissipated by the circuit when it is on but not in use - and dynamic power - the power dissipated when the circuit is active - for energy consumption. When the CGRA is not in use, the system power is equal to the total power (static + dynamic power) of the $\rho$-VEX processor. While the CGRA is active, the energy of all its functional units in use is accounted for along with the static energy of the $\rho$-VEX processor (which is stalled). The energy consumption and area are evaluated for the whole DIM-VEX, which is composed of the $\rho$-VEX, DDH, and CGRA, without the caches and main memory. Even though this methodology simplifies the analysis, it may bias results in energy in favor of the $\rho$-VEX processor without the CGRA, since it hides an important source of energy savings in the DIM-VEX system: when the CGRA is active in, all instruction fetches are stalled. Therefore, the DIM-VEX reduces the pressure on the instruction memory (cache), while keeping the same access rate to the data memories.

The benchmark set is composed of a subset of 15 applications from the WCET [7] and Powerstone [11] benchmark suites: ADPCM, CJPEG, CRC, DFT, Engine, Expint, FIR, JPEG, LUDCMP, Matrix Multiplication, NDES, POCSAG, QURT, Sums (recursively executes multiple additions on an array), and x264. All benchmarks are compiled using the HP VEX compiler.

## 5 Results and Analysis

We present the results for performance, energy and Energy-Delay Product (EDP) of all benchmarks in our proposed system and the standalone $\rho$-VEX. All the data presented in the charts are normalized by the $\rho$-VEX3 configuration, which

is the 8-issue $\rho$-VEX processor. We chose this configuration as the baseline because it has the best performance among the standalone $\rho$-VEX systems. We also present the area overhead of each of the analyzed systems.

### 5.1 Performance

Figure 2 shows the normalized speedup for the tested benchmarks. Bars with values above 1 mean that the configuration improved performance; while for those under 1, performance slowed down. As expected, the $\rho$-VEX1 and $\rho$-VEX2 configurations have worse performance than the baseline ($\rho$-VEX3), as they work with smaller issue-widths. However, when we include DIM, most of the benchmarks show speedups, even when the $\rho$-VEX processor to which it is coupled to has fewer pipelanes. For instance, for the benchmarks DFT, Engine, Expint and x264, the best configuration for performance is the DIM-VEX5, which consists of a 2-issue processor coupled with a 9-levels array. For the benchmarks ADPCM, CRC, matrix multiplication, NDES, POCSAG and QURT, the best configuration is DIM-VEX2, a 4-issue processor. On the other hand, coupling the array with an 8-issue processor (DIM-VEX3) can still increase its capabilities for exploiting ILP. On average, configuration DIM-VEX3 has an speedup of 1.38X.

Table 3 shows the cycles and resource utilization for the benchmarks. Due to space constraints, we show these data only for configurations DIM-VEX1 and DIM-VEX5, which represent the most contrasting CGRA configurations (with the least and the most levels). We restrain into showing just ALU usage, as this is what impacts acceleration the most in the CGRA. We also position the benchmarks DFT, Expint, CJPEG, and x264 on top of the table for easy visualization, as we use them as examples for explanation.

In Fig. 2, DFT shows a reduction in cycle counts over the baseline for both configurations DIM-VEX1 and DIM-VEX5. In fact, the benchmarks show speedups of 1.21X and 1.78X, respectively. As presented in Table 3, most of the instructions executed by DFT are performed inside the CGRA (80.88% and 83.49%), spread along 12 basic blocks in DIM-VEX1 and 7 in DIM-VEX5, which explains such speedups. A similar behavior is observed in Expint, which executes 92.14% of its instructions on the CGRA in DIM-VEX1, and 87.75% in DIM-VEX5. However, the cycle reduction in Expint is much higher, achieving speedups of near to 2.5x in DIM-VEX1 and 3.5x in DIM-VEX5 with respect to

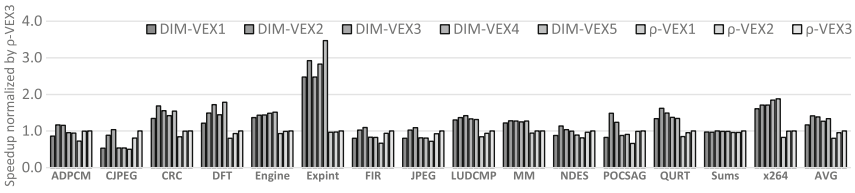


**Fig. 2.** Execution cycles for each benchmark normalized by the baseline ($\rho$-VEX3). Bars under 1 represent faster executions than baseline, while bars above 1 represent slower execution.

**Table 3.** Cycles and resource usage for the CGRA on all benchmarks. Cycles in CGRA is the total number of cycles executed inside the reconfigurable fabric (along with the ratio over the entire system). Significant BB shows the number of basic blocks which execute a meaningful part of the application and the average ALU usage on the reconfigurable fabric for these basic blocks.

| | DIM-VEX1 | | | | DIM-VEX5 | | | |
|---|---|---|---|---|---|---|---|---|
| | Cycles | Cycles in CGRA | Significant BB | | Cycles | Cycles in CGRA | Significant BB | |
| DFT | 26841 | 21708 (80.88%) | Number | Avg ALU usage | 18238 | 15226 (83.49%) | Number | Avg ALU usage |
| | | | 12 | 12.36% | | | 7 | 17.08% |
| Expint | 3677 | 3388 (92.14%) | 6 | 44.72% | 2620 | 2299 (87.75%) | 2 | 92.78% |
| CJPEG | 773 | 228 (29.5%) | 6 | 12.21% | 764 | 349 (45.68%) | 6 | 8.89% |
| x264 | 9362 | 8135 (86.89%) | 3 | 17.22% | 8029 | 7827 (97.48%) | 3 | 15.55% |
| ADPCM | 659 | 254 (38.54%) | 5 | 21.50% | 604 | 179 (29.64%) | 3 | 27.86% |
| CRC | 9866 | 6003 (60.85%) | 4 | 25.42% | 8591 | 4722 (54.96%) | 2 | 35.56% |
| Engine | 505871 | 244005 (48.23%) | 17 | 32.26% | 455695 | 206498 (45.31%) | 9 | 44.45% |
| FIR | 139520 | 44745 (32.07%) | 6 | 33.20% | 135464 | 40479 (29.88%) | 4 | 32.47% |
| JPEG | 1800718 | 525662 (29.19%) | 16 | 7.81% | 1794808 | 566096 (31.54%) | 14 | 6.52% |
| LUDCMP | 34281 | 20091 (58.61%) | 20 | 17.51% | 33925 | 20891 (61.58%) | 20 | 9.99% |
| MM | 90848 | 62241 (68.51%) | 9 | 18.29% | 87275 | 63046 (72.24%) | 8 | 14.98% |
| NDES | 31301 | 16069 (51.34%) | 6 | 12.48% | 30789 | 17803 (57.82%) | 7 | 8.96% |
| POCSAG | 22890 | 15616 (68.22%) | 4 | 25.42% | 20788 | 13330 (64.12%) | 3 | 43.71% |
| QURT | 13426 | 7607 (56.66%) | 14 | 19.93% | 13356 | 7546 (56.50%) | 17 | 28.08% |
| Sums | 328 | 40 (12.20%) | 2 | 0.80% | 325 | 101 (31.08%) | 2 | 6.11% |

the baseline. These speedups are due to the high usage of the CGRA functional units, mostly caused by the large number of ALU operations in the basic blocks. DIM-VEX5 is able to best use the ALU resources (92.78% against 44.72% in DIM-VEX1) because it has only two rows of ALUs executing in parallel, which is a perfect match for this application. In DIM-VEX1, there are four rows, but two of them are mostly not used during the application execution. Nevertheless, executing most of the code on the CGRA does not necessarily translates into high speedups. In the x264 application, almost the whole application is executed in the CGRA, but speedup is of only 1.61X in DIM-VEX1 and 1.88X in DIM-VEX5 when compared to the baseline. The x264 is composed of few (3) and small significant BBs with low ALU usage, which translates into lower speedups. The CJPEG is an example of an application in which the CGRA cannot provide better execution time than the 8-issue $\rho$-VEX. That is, the CGRA configurations are small and do not support speculation - they are constantly broken by unsupported instructions instead of branch operations -, which limits the acceleration of such application.

### 5.2 Energy

Figure 3 shows the normalized energy consumption for the tested systems in all benchmarks. In this chart, bars above 1 represent higher energy consumption and bars under 1 lower energy consumption than the baseline. The 8-issue $\rho$-VEX ($\rho$-VEX3) processor operates at a higher power usage than its 2- and 4-issue counterparts. This reflects into the results, with $\rho$-VEX2 reducing energy by half (49.4%)

and $\rho$-VEX1 reducing energy by 70.4% when compared to the baseline. When the CGRA is added, many new components are integrated to the system, highly increasing its power consumption. However, the CGRA also provides speedups, reducing the time needed to execute the application. As the energy depends on the execution time as well as power, most of the CGRA configurations can provide lower energy consumption than the 8-issue processor alone. The only exception is the DIM-VEX5 configuration running the POCSAG application. Under this scenario, a 2-issue processor is coupled with a big CGRA, but no speedups are obtained. As can be seen in Table 3, POCSAG activates the array in 64.12% of its instructions with a usage of 43.71% of the CGRA ALUs. This results in a power hungry system that cannot provide any speedups, increasing the energy consumption without giving any improvements. On average, the best energy configuration is the $\rho$-VEX1 (70.4% energy reduction), as it is an extremely low power processor. Among the DIM-VEX systems, DIM-VEX1 presents the best energy consumption, reducing it by 60% with respect to the baseline.

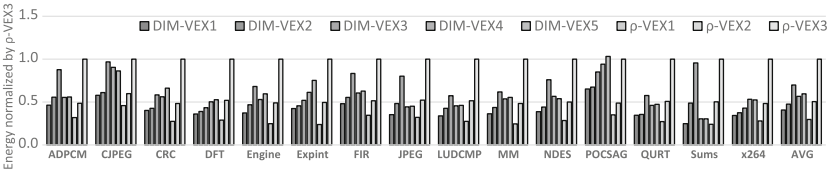


**Fig. 3.** Energy consumption for each benchmark normalized by the baseline ($\rho$-VEX3). Bars under 1 represent lower energy consumption than baseline, while bars above 1 represent higher consumption.

Table 3 can also be used to explain the energy results. For example in the Expint benchmark, the speedups for the application are huge in DIM-VEX5 (almost 3.5x); nonetheless, the energy reductions are much more restrained: only 25%. This small reduction is also explained by the usage of the CGRA resources. DIM-VEX5 executes 87.75% of its instructions in the CGRA using an average of 92.78% of its ALUs. This represents an average usage of 83 ALUs (see Table 1 for ALUs per configuration) during 87.75% of the execution time, which results in a considerable power consumption for the system. Resource usage also explains the reason DIM-VEX configurations can still provide energy gains for the CJPEG application, even when such systems provide slowdowns for the applications (DIM-VEX1, DIM-VEX2, DIM-VEX4 and DIM-VEX4 cases). Apart from these configurations using processors that are less power hungry than the baseline (2- and 4-issue $\rho$-VEX), when the CGRA is active in the CJPEG application, only a few resources are used. The combination of these two conditions results in a much low power environment than the $\rho$-VEX3 processor.

### 5.3 Energy and Performance Trade-Off

Figure 4 shows the normalized Energy-Delay Product (EDP) for the benchmarks in all the tested systems. As in the energy chart, bars above 1 represent higher

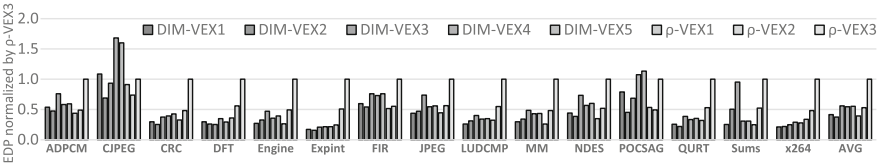


**Fig. 4.** EDP for each benchmark normalized by the baseline ($\rho$-VEX3). Bars under 1 represent worst EDP than baseline, while bars above 1 represent better EDP.

EDP and bars under 1 lower EDP than the baseline. The EDP is the product of the energy spent by the system and the execution time of an application. It is used to measure the trade-off between energy and performance (if a system fails to deliver performance, it can still show good results in energy). As can be seen in the Fig. 4, almost all of the benchmarks show better EDP than the baseline. The cases that the baseline is better are when the performance is too highly affected (CJPEG in DIM-VEX1, DIM-VEX4 and DIM-VEX5), or when both performance and energy cannot reach acceptable levels (POCSAG in DIM-VEX4 and DIM-VEX5). On average, the best trade-off between systems comes with DIM-VEX2, reducing EDP by 62.6%, closely followed by $\rho$-VEX1 (60.8%) and DIM-VEX1 (58.6%).

### 5.4 Area Analysis

Finally, we analyze the impact in area that is added by the CGRA on our systems. In the Fig. 5, the total area of each of the evaluated systems is presented. In the DIM-VEX configurations, the area is divided between the $\rho$-VEX processor area and the CGRA. It is clear that all the extra resources in the CGRA can occupy a high amount of space that may be prohibitive in some environments. However, if one considers the configuration DIM-VEX2, it is possible to reach better energy consumption and better performance in all, but one (Sums), of the benchmarks, at the price of an extra 84% in area.

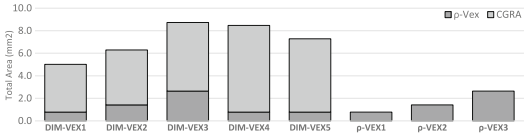


**Fig. 5.** Total area occupied by the evaluated systems. Bars in DIM-VEX configurations are split in the area occupied by the $\rho$-VEX processor and the CGRA.

## 6 Conclusions and Future Work

In this work we have proposed a design time configurable system that is also reconfigurable at runtime (DIM-VEX). By designing a set of configurable processors coupled with reconfigurable logic, we have shown that it is possible to

further expand the ILP capabilities of multiple issue processors. Our system is also able to save energy in many scenarios, while keeping the superior performance. All these advantages come at the price of extra area.

**Acknowledgments.** This work was produced under grant from the Brazilian agencies FAPERGS, CAPES and CNPQ and the European Network HiPEAC.

## References

1. Beck, A.C.S., Lang Lisbôa, C.A., Carro, L.: Adaptable Embedded Systems, 1st edn. Springer, New York (2013). https://doi.org/10.1007/978-1-4614-1746-0
2. Beck, A.C.S., Rutzig, M.B., Carro, L.: A transparent and adaptive reconfigurable system. Microprocess. Microsyst. **38**(5), 509–524 (2014)
3. Compton, K., Hauck, S.: Reconfigurable computing: a survey of systems and software. ACM Comput. Surv. **34**(2), 171–210 (2002)
4. Gonzalez, A., Tubella, J., Molina, C.: Trace-level reuse. In: Proceedings of the 1999 International Conference on Parallel Processing, pp. 30–37. IEEE Computer Society (1999)
5. Govindaraju, V., Ho, C.H., Nowatzki, T., Chhugani, J., Satish, N., Sankaralingam, K., Kim, C.: DySER: unifying functionality and parallelism specialization for energy-efficient computing. IEEE Micro **32**(5), 38–51 (2012)
6. Gschwind, M., Altman, E., Sathaye, S., Ledak, P., Appenzeller, D.: Dynamic and transparent binary translation. Computer **33**(3), 54–59 (2000)
7. Gustafsson, J., Betts, A., Ermedahl, A., Lisper, B.: The Mälardalen WCET benchmarks: past, present and future. In: WCET, vol. 15, pp. 136–146 (2010)
8. Koenig, R., Bauer, L., Stripf, T., Shafique, M., Ahmed, W., Becker, J., Henkel, J.: KAHRISMA: a novel hypermorphic reconfigurable-instruction-set multi-grained-array architecture. In: 2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010), pp. 819–824. IEEE, March 2010
9. Lysecky, R., Stitt, G., Vahid, F.: Warp processors. ACM Trans. Des. Autom. Electron. Syst. **11**(3), 659–681 (2006)
10. Sartor, A.L., Becker, P., Hoozemans, J., Wong, S., Beck, A.C.S.: Dynamic trade-off among fault tolerance, energy consumption, and performance on a multiple-issue VLIW processor. IEEE Trans. Multi-Scale Comput. Syst. (2017)
11. Scott, J., Lee, L.H., Arends, J., Moyer, B.: Designing the low-power M•CORE™ architecture. In: Power Driven Microarchitecture Workshop, pp. 145–150 (1998)
12. Souza, J.D., Carro, L., Rutzig, M.B., Beck, A.C.S.: A reconfigurable heterogeneous multicore with a homogeneous ISA. In: Proceedings of the 2016 Conference on Design, Automation & Test in Europe, DATE 2016, pp. 1598–1603 (2016)
13. Watkins, M.A., Nowatzki, T., Carno, A.: Software transparent dynamic binary translation for coarse-grain reconfigurable architectures. In: 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 138–150. IEEE, March 2016
14. Wong, S., van As, T., Brown, G.: $\rho$-VEX: a reconfigurable and extensible softcore VLIW processor. In: 2008 International Conference on Field-Programmable Technology, pp. 369–372. IEEE, December 2008