On Implementability of Polymorphic Register Files

Cătălin Ciobanu*†, Georgi Kuzmanov*, Georgi Gaydadjiev*†
*Computer Engineering Laboratory, EEMCS,
Delft University of Technology,
The Netherlands
{c.b.ciobanu, g.k.kuzmanov, g.n.gaydadjiev}@tudelft.nl
† Department of Computer Science and Engineering,
Chalmers University of Technology,
Sweden
{catalin, georgig}@chalmers.se

***Abstract*—This paper studies the implementability of performance efficient multi-lane Polymorphic Register Files (PRFs). Our PRF implementation uses a 2D array of $p \times q$ linearly addressable memory banks, with customized addressing functions to avoid address routing circuits. We target one single-view and a set of four non redundant multi-view parallel memory schemes that cover all widely used access patterns in scientific and multimedia applications: 1) $p \times q$ rectangle, $p \cdot q$ row, $p \cdot q$ main and secondary diagonals; 2) $p \times q$ rectangle, $p \cdot q$ column, $p \cdot q$ main and secondary diagonals; 3) $p \cdot q$ row, $p \cdot q$ column, aligned $p \times q$ rectangle; 4) $p \times q$, $q \times p$ rectangles (transposition). Reconfigurable hardware was chosen for the implementation due to its potential in enhancing the PRF runtime adaptability. For a proof of concept, we prototyped a 2 read, 1 write ports PRF on a Virtex-7 XC7VX1140T-2 FPGA. We consider four sizes for the 16 lanes PRFs - 16×16, 32×32, 64×64 and 128×128 and three multi-lane configurations, 8, 16 and 32, for the 128×128 PRF. Synthesis results suggest clock frequencies between 111 MHz and 326 MHz while utilizing less than 10% of the available LUTs. By using customized addressing functions, the LUT usage is reduced by up to 29% and the clock frequency is up to 77% higher compared to a straight-forward implementation.**

I. Introduction

The number of transistors in current semiconductor devices has increased at a sustained rate, and it is expected to continue growing in the future [1]. However, new challenges such as power and thermal constraints make it infeasible to further increase the processors clock frequency. Therefore, in recent years the industry turned to Chip Multiprocessor (CMP) designs and to accelerators targeting specific workloads (e.g., Single Instructions Multiple Data (SIMD) extensions and hardware support for encryption algorithms [2]). When designing a new processor, predefined scenarios are used in order to anticipate the requirements of the target workloads and meet expectations. However offering a single best configuration is often impossible since new workloads will emerge in the future. The growing diversity of the computational tasks drives the need for higher adaptability of future computer systems which utilize technologies such as reconfigurable hardware and runtime partial reconfiguration.

When targeting vector architectures, such as IBM 370 [3] in the past, and more recent GPPs featuring SIMD extensions, e.g., Altivec [9], or the Synergistic Processor Units in the Cell BE [10], programmers are expected to optimize their code according to the number and width of the Vector Registers. As the available storage was divided in a fixed number of registers of equal sizes and shapes, any change in the size or the number of SIMD registers breaks programming compatibility with the existing software. As part of the Scalable computer ARChitecture (SARC) architecture [16], we have proposed a Polymorphic Register File (PRF) [5], designed to adapt to various data structures and to assist programming of high performance vector algorithms. The goal is to enable the programmers focus on the desired functionality of their code instead of describing complex data operations and transfers. The PRF is able to dynamically divide the available register storage into multidimensional registers of arbitrary shapes and sizes during runtime. Previous studies ([5], [16]) have shown that such PRFs are suitable for computationally intensive workloads such as Floyd, the Conjugate Gradient (CG) Method and dense matrix multiplication. It was also suggested that PRFs can improve the performance efficiency in state of the art many-core computers, potentially saving area and power [6]. More specifically, the potential benefits from using a 2D PRF are: i) improved storage efficiency, as the number of registers, their dimensions and sizes are customized to the workload requirements, and ii) performance gain, as the committed instructions number is greatly reduced.

The key to high performance allowed by the PRF lies in the capability of the latter to deliver aligned data elements to the computational units at high rates, allowing multiple vector lanes to operate in parallel and efficiently utilize the available bandwidth. Accessing rectangular blocks of data is widely used in linear algebra, as well as in multimedia and scientific applications. Reconfigurable hardware was chosen for the implementation due to its potential in enhancing the runtime adaptability of the PRF. This paper provides an implementability study of multi-module, multi-lane PRFs targeting state of the art FPGAs. More specifically, the main contributions of this paper are:

- A generic design of multi-lane, multi-port PRFs suitable for hardware implementations in FPGAs and ASICs;
- Proof of concept demonstration of a practical FPGA

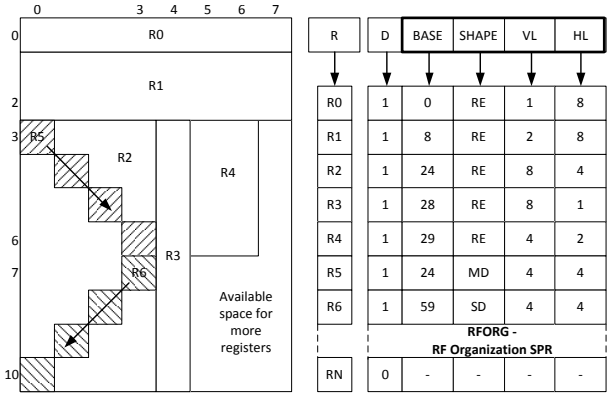

Fig. 1. The Polymorphic Register File, N = 11, M = 8

TABLE I
THE MODULE ASSIGNMENT FUNCTIONS

Scheme	m_v	m_h
rectangle only	$i\%p$	j % q
rect&row	$\left(i+\left\lfloor\frac{j}{q}\right\rfloor\right)\%p$	$j\%q$
rect&col	$i\%p$	$\left(\left\lfloor\frac{i}{p}\right\rfloor+j\right)\%q$
row&col	$\left(i+\left\lfloor\frac{j}{q}\right\rfloor\right)\%p$	$\left(\left\lfloor\frac{i}{p}\right\rfloor+j\right)\%q$
rect&rect, $p<q$	$i\%p$	$(i-i\%p+j)\%q$
rect&rect, $q<p$	$(i+j-j\%q)\%p$	$j\%q$

TABLE II
CUSTOMIZED ADDRESSING FUNCTION - **RECT&ROW** SCHEME

Acc. Type	Customized addressing function
$p\times q$	$c_{i,rect\&row,p\times q}=\left\lfloor\frac{i\%p+\left(k-\left\lfloor\frac{j}{q}\right\rfloor\%p-c_{j,rect\&row,p\times q}-i\%p\right)\%p}{p}\right\rfloor$
	$c_{j,rect\&row,p\times q}=\begin{cases}1, l<j\%q\\0, otherwise\end{cases}$
$1\times p\cdot q$	$c_{i,rect\&row,1\times p\cdot q}=0$
	$c_{j1}=\begin{cases}1, l<j\%q\\0, \text{ otherwise}\end{cases}$
	$c_{j,rect\&row,1\times p\cdot q}=c_{j1}+\left(k-i\%p-\left\lfloor\frac{j}{q}\right\rfloor\%p-c_{j1}\right)\%p$
Main Diag.	$c_{j1}=\begin{cases}1, l<j\%q\\0, \text{ otherwise}\end{cases}$
	$\delta_{m.d.}=k-i\%p-((l-j\%q)\%q)\%p-c_{j1}-\left\lfloor\frac{j}{q}\right\rfloor\%p$
	$c_{j2}=((\delta_{m.d.}\%p)\cdot\omega_{q+1})\%p$
	$c_{i,rect\&row,md}=\left\lfloor\frac{i\%p+(l-j\%q)\%q+q\cdot c_{j2}}{p}\right\rfloor$
	$c_{j,rect\&row,md}=c_{j1}+c_{j2}$
Sec. Diag.	$c_{j1}=\begin{cases}-1, l>j\%q\\0, otherwise\end{cases}$
	$\delta_{s.d.}=k-i\%p-((j\%q-l)\%q)\%p-(c_{j1}\%p)-\left\lfloor\frac{j}{q}\right\rfloor\%p$
	$c_{j2}=((\delta_{s.d.}\%p)\cdot\omega_{q-1})\%p$
	$c_{i,rect\&row,sd}=\left\lfloor\frac{i\%p+(j\%q-l)\%q+q\cdot c_{j2}}{p}\right\rfloor$
	$c_{j,rect\&row,sd}=c_{j1}-c_{j2}$

design. Synthesis results for a prototype with 2 read and 1 write ports with 64-bit data path using the selected Module Assignment Functions (MAFs) targeting a Virtex-7 XC7VX1140T-2 FPGA. We consider four sizes for the 16 lanes PRFs and three multi-lane configurations for the 128×128 PRF. Results suggest a maximum clock frequency between 111 MHz and 326 MHz while only using less than 10% of the available LUTs;

- Customized addressing functions and their evaluation. The LUT usage is reduced by up to 29% and the maximum clock frequency is increased by up to 77% compared to the baseline, suboptimal implementation.

The remainder of this paper is organized as follows: the background information and related work are presented in Section II. The theoretical basis are defined in Section III. Section IV describes the evaluation methodology, and evaluates the results. Finally, the paper is concluded in Section V.

II. BACKGROUND AND RELATED WORK

When designing a processor, the requirements of the target applications have to be anticipated in order to meet expectations, using predefined scenarios. However, the system requirements may change as new workloads become relevant, making it impossible to forecast all requirements of all workloads, especially in the age of ubiquitous computing [18]. Therefore, offering a single best configuration is often impossible. The growing diversity of tasks which need to be accomplished by the processors drives the need for higher flexibility of future computer systems. To a certain extent, software may mask the low level architectural details, but in domains such as high performance computing or embedded systems, hardware support is preferable to meet the performance and efficiency constraints.

A PRF is a parameterizable register file, logically reorganized under software control (by the system / application programmer or by the runtime system) to support multiple register dimensions and sizes simultaneously [5]. Fig. 1 provides an example of a two-dimensional PRF with a physical register size of 11 by 8 basic elements. In this example, the available storage has been divided into 7 logical registers, each with different location and dimensions, defined using the Register File Organization (RFORG) Special Purpose Registers (SPRs). For each logical register it is required to specify the location of the upper left corner (Base), the shape (REctangular, Main or Secondary Diagonals), and the dimensions (Horizontal and Vertical Lengths).

The additional flexibility of the PRF is achieved by adding one level of indirection (the RFORG SPRs) when accessing the vector registers. The benefits of a 2D PRF are:

- Potential performance gain, as the number of elements processed with a single instruction is increased due to multi-axis vectorization, greatly reducing the number of committed instructions;
- Improved storage efficiency, as the number of vector registers, their dimensions and sizes are dynamically set during runtime follow the workload requirements;
- Reduced static code footprint, as the target algorithm may be expressed with fewer, higher level instructions. The same binary instructions may be used regardless of the shapes, dimensions and data types of the operands. The

TABLE III
CUSTOMIZED ADDRESSING FUNCTION - **RECT&COL** SCHEME

Acc. Type	Customized addressing function
$p \times q$	$c_{i,rect\&col,p\times q} = \begin{cases} 1, k < i\%p \\ 0, \text{ otherwise} \end{cases}$
	$c_{j,rect\&col,p\times q} = \left\lfloor \frac{j\%q+\left(l-j\%q-\left\lfloor \frac{i}{p} \right\rfloor \%q-c_{i,rect\&col,p\times q}\right)\%q}{q} \right\rfloor$
$p \cdot q \times 1$	$c_{i1} = \begin{cases} 1, k < i\%p \\ 0, \text{ otherwise} \end{cases}$
	$c_{i,rect\&col,p\cdot q\times 1} = c_{i1} + (l - \left\lfloor \frac{i}{p} \right\rfloor \%q - j\%q - c_{i1})\%q$
	$c_{j,rect\&col,p\cdot q\times 1} = 0$
Main Diag.	$c_{i1} = \begin{cases} 1, k < i\%p \\ 0, \text{ otherwise} \end{cases}$
	$\delta_{m.d} = l - j\%q - ((k - i\%p)\%p)\%q - c_{i1} - \left\lfloor \frac{i}{p} \right\rfloor \%q$
	$c_{i2} = ((\delta_{m.d}\%q) \cdot \omega_{p+1})\%q$
	$c_{i,rect\&row,md} = c_{i1} + c_{i2}$
	$c_{j,rect\&col,md} = \left\lfloor \frac{j\%q+(k-i\%p)\%p+p\cdot c_{i2}}{q} \right\rfloor$
Sec. Diag.	$c_{i1} = \begin{cases} 1, k < i\%p \\ 0, \text{ otherwise} \end{cases}$
	$\delta_{s.d.} = c_{i1} + \left\lfloor \frac{i}{p} \right\rfloor \%q + j\%q - ((k - i\%p)\%p)\%q - l$
	$c_{i2} = ((\delta_{s.d.}\%q) \cdot \omega_{p-1})\%q$
	$c_{i,rect\&col,sd} = c_{i1} + c_{i2}$
	$c_{j,rect\&col,sd} = \left\lfloor \frac{j\%q-(k-i\%p)\%p-p\cdot c_{i2}}{q} \right\rfloor$

TABLE IV
CUSTOMIZED ADDRESSING FUNCTION - **ROW&COL** SCHEME

Acc. Type	Customized addressing function
$p \cdot q \times 1$	$c_{i,row\&col,1\times p\cdot q} = 0$
	$c_{j1} = \left\lfloor \frac{j\%q+\left(l-\left\lfloor \frac{i}{p} \right\rfloor \%q-j\%q\right)\%q}{q} \right\rfloor$
	$c_{j2} = \left(k - i\%p - \left\lfloor \frac{j}{q} \right\rfloor \%p - c_{j1}\right)\%p$
	$c_{j,row\&col,1\times p\cdot q} = c_{j1} + c_{j2}$
$1 \times p \cdot q$	$c_{i1} = \left\lfloor \frac{i\%p+\left(k-\left\lfloor \frac{j}{q} \right\rfloor \%p-i\%p\right)\%p}{p} \right\rfloor$
	$c_{i2} = \left(l - j\%q - \left\lfloor \frac{i}{p} \right\rfloor \%q - c_{i1}\right)\%q$
	$c_{i,row\&col,p\cdot q\times 1} = c_{i1} + c_{i2}$
	$c_{j,row\&col,p\cdot q\times 1} = 0$
$p \times q$	$c_{i,row\&col,p\times q} = \left\lfloor \frac{i\%p+\left(k-i\%p-\left\lfloor \frac{j}{q} \right\rfloor \%p\right)\%p}{p} \right\rfloor$
	$c_{j,row\&col,p\times q} = \left\lfloor \frac{j\%q+\left(l-j\%q-\left\lfloor \frac{i}{p} \right\rfloor \%q\right)\%q}{q} \right\rfloor$

microarchitecture resolves the operands compatibility.

One of the main goals of the PRF is to facilitate processor scalability. The customization process is shifted from the design time to runtime. The higher level instruction set offers binary instruction compatibility between software implementations with different data types or vector operand dimensions, potentially reducing the development costs. The PRF is dynamically adjustable during runtime, making it an Adaptable architecture. With proper compiler and runtime support, it can be easily integrated in Self-Adaptable and Autonomic Computing Systems [8].

Previous works show reductions of the number of executed instructions by three orders of magnitude due to PRF [5]. Furthermore, PRFs allow performance benefits when compared to the Cell processor for Floyd and the main kernel of the CG Method - sparse matrix vector multiplication [5]. The PRF programming interface allows high performance dense matrix multiplication with at least 35 times less instructions than a hand-crafted version for the Cell BE [16]. A CG case study evaluated the PRF based system scalability in a heterogeneous multi-core architecture and showed CG acceleration by two orders of magnitude using up to 256 PRF cores, with 32 vector lanes each. Moreover, a similar performance level could be achieved by fewer PRF cores compared to a Cell BE-based system, potentially saving area and power [6].

In all previous studies, up to 32 vector lanes were used, proving that the ability of the PRF to deliver data to multiple parallel vector lanes at high rates is the key to high performance. The main goal of our work hereafter is to study the implementability of the PRFs and identify the possible bottlenecks of the considered implementation. We examine configurations with up to 32 parallel lanes, and evaluate the clock frequency and FPGA resources utilization.

Related Work: The efficient processing of multidimensional arrays has been targeted by other architectures as well. One approach is to use a memory to memory architecture, such as the Burrows Scientific Processor (BSP) [12]. Being optimized for executing Fortran code, the ISA composed of high level vector instructions with a large number of parameters. The arithmetic units were equipped with 10 registers which are not directly accessible by the programmer. The Polymorphic Register File also creates the premises for a high level ISA, but can reuse data directly within the register file. The Complex Streamed Instructions (CSI) [11] approach also did not make use of any data registers. CSI allows the processing of two-dimensional data streams of arbitrary length, but requires data caches to benefit from data locality. Our approach can use the register file to avoid high speed data caches.

The Vector Register Windows (VRW) [14] concept allows the grouping of consecutive vector registers in a 2D window. However, one of the dimensions is fixed, contrary to our proposal. The Matrix Oriented Multimedia (MOM) [7] also uses a 2D register file, but with a fixed number of registers which used sub-word parallelism in order to store up to 16x8 elements. The Polymorphic Register File also supports sub-word level parallelism but doesn't restrict the number or shape of the two dimensional registers. Modified MMX [17] supports 8 multimedia registers, each 96 bits wide. However, the matrix operations are limited to only loads and stores.

The Register Pointer Architecture(RPA) [15] provides extra storage to a scalar processor by adding two additional register files - Dereferencible Register File (DRF) and the Register Pointers (RP). The DRF provides the extra storage space, while the RP provide indirect access to the DRF. The PRF also uses indirect accessing to a dedicated register file, but the RPA maps scalar registers, while in our proposal each indirection register maps to a matrix, being more suitable for

TABLE V
CUSTOMIZED ADDRESSING FUNCTION - **RECT&TRECT** SCHEME

Acc. Type	Customized addressing function
$p \times q$	$c_{i,rect\&trect,p<q,p\times q} = \begin{cases} 1, k < i\%p \\ 0, otherwise \end{cases}$
	$\delta_{p\times q} = l - (c_{i,rect\&trect,p<q,p\times q} \cdot p)\%q - \left(\left\lfloor \frac{i}{p} \right\rfloor \cdot p\right)\%q - j\%q$
	$c_{j,rect\&trect,p<q,p\times q} = \left\lfloor \frac{j\%q+\delta_{p\times q}\%q}{q} \right\rfloor$
$q \times p$	$c_{i1} = \begin{cases} 1, k < i\%p \\ 0, \text{ otherwise} \end{cases}$
	$c_{i2} = \frac{\left(l - c_{i1}\cdot p - \left(\left\lfloor \frac{i}{p} \right\rfloor \cdot p\right)\%q - j\%q - (l\%p - j\%p)\%p\right)\%q}{p}$
	$c_{i,rect\&trect,p<q,q\times p} = c_{i1} + c_{i2}$
	$c_{j,rect\&trect,p<q,q\times p} = \left\lfloor \frac{j\%q+(l\%p-j\%p)\%p}{q} \right\rfloor$

TABLE VI
THE ω CONSTANTS

p/q	2	4	8
2	1	1	1
4	3	1	1

(a) ω_{q+1}

p/q	2	4	8
2	1	1	1
4	1	3	3

(b) ω_{q-1}

p/q	2	4	8
2	1	3	3
4	1	1	5

(c) ω_{p+1}

p/q	2	4	8
2	1	1	1
4	1	3	3

(d) ω_{p-1}

vector processing.

In order to adjust the number of registers in a VLIW, in [19] FPGA partial reconfiguration is used to adjust the size of the physical register file. Our approach assumes fixed physical register file, but offers a higher level view of the available storage space, eliminating many overhead instructions, possibly improving performance.

III. THEORETICAL BASIS

We propose a generic PRF implementation containing $N \times M$ data elements, stored using $p \times q$ memory modules, organized in a 2D matrix with p rows. Throughout this paper, we will use "$\times$" to refer to a 2D matrix and "$\cdot$" to denote scalar multiplication. We also assume that $N\%p = M\%q = 0$. Depending on the parallel memory scheme employed, such an organization allows the efficient use of up to $p \cdot q$ vector lanes.

We consider five parallel access schemes suitable for the implementation of the PRF: a single-view scheme which supports conflict free accesses shaped as to $p \times q$ rectangles, suggested in [13], and referred hereafter as the **rectangle only** scheme, and a set of four well selected, non redundant multi-view schemes, supporting conflict free access to the most common vector operations for scientific and multimedia applications, which we proposed in [4]. The multi-view parallel access schemes are denoted as **rect&row**, **rect&col**, **row&col** and **rect&trect**, each of them supporting at least two conflict-free access patterns: 1) **rect&row**: $p \times q$ rectangle, $p \cdot q$ row, $p \cdot q$ main diagonals if p and $q+1$ and co-prime, $p \cdot q$ secondary diagonals if p and $q-1$ are co-prime; 2) **rect&col**: $p \times q$ rectangle, $p \cdot q$ column, $p \cdot q$ main diagonals if $p+1$ and q are co-prime, $p \cdot q$ secondary diagonals if $p-1$ and q are co-prime; 3) **row&col**: $p \cdot q$ row, $p \cdot q$ column, aligned ($i\%p = 0$ or $j\%q = 0$) $p \times q$ rectangle; 4) **rect&trect**: $p \times q$, $q \times p$ rectangles (transposition) if $p\%q = 0$ or $q\%p$=0.

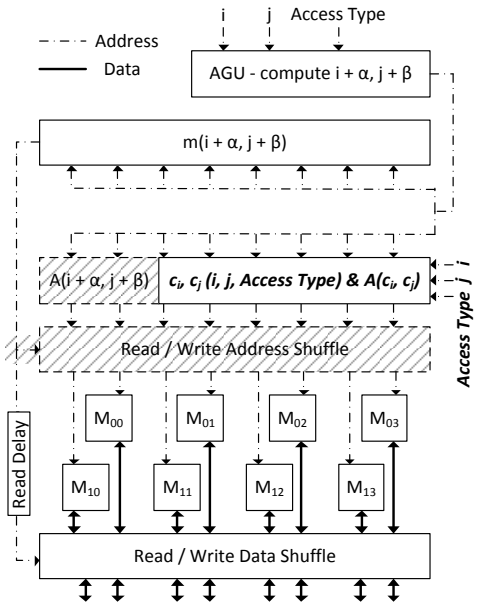

Fig. 2. PRF block diagram, p=2, q=4. Special Purpose Registers are not shown.

The mathematical operators used later in this section are:

Definition 1: The floor function (round to $-\infty$):

$$\lfloor x \rfloor = max\{z \in \mathbb{Z} | z \leq x\} \tag{1}$$

and has the following property:

$$\lfloor x + z \rfloor = \lfloor x \rfloor + z, \forall z \in \mathbb{Z} \tag{2}$$

Definition 2: The "%" modulo operator is defined as:

$$x\%n = x - n \cdot \left\lfloor \frac{x}{n} \right\rfloor, n \in \mathbb{N}, x\%n < n \tag{3}$$

The parallel access scheme assigns (i, j), the address of an element stored in the 2D PRF, to a position in one of the memory modules. This one-to-one mapping is performed by the module assignment functions $m_v()$ and $m_h()$, and the intra-module linear addressing function $A()$. The row of the memory module is computed by $m_v()$ and the column by $m_h()$. The module assignment functions for the five considered schemes are presented in Table I.

In practical designs, the dimensions of the PRF as well as the number of memory modules are powers of 2, therefore the multiplications, divisions and modulo operations required by the memory schemes are simplified to shifts or bit select operations, and the conditions imposed on p and q for the **rect&trect** scheme, as well as regarding the diagonal accesses for **rect&row** and **rect&col** schemes are satisfied. Without loss of generality, we assume $p < q$ when further referring to the **rect&trect** scheme.

For all the schemes considered, the standard linear address assignment function [4] is defined as:

$$A_{standard}(i, j) = \left\lfloor \frac{i}{p} \right\rfloor \cdot \left(\frac{M}{q}\right) + \left\lfloor \frac{j}{q} \right\rfloor, j < M \tag{4}$$

PRF Size	Vector Lanes	rect only			rect&row			rect&col			row&col			rect&trect		
		std.	cust.	Δ[%]	std.	cust.	Δ[%]	std.	cust.	Δ[%]	std.	cust.	Δ[%]	std.	cust.	Δ[%]
16×16	2×8	191.6	205.0	**+7.0**	183.6	189.7	**+3.3**	167.1	167.4	**+0.2**	163.4	168.3	**+3.0**	177.1	170.3	**-3.8**
32×32	2×8	190.4	194.5	**+2.2**	182.6	185.4	**+1.5**	164.8	163.5	**-0.8**	153.2	169.3	**+10.5**	174.1	178.1	**+2.3**
64×64	2×8	186.5	184.5	**-1.1**	176.4	187.8	**+6.5**	166.3	174.0	**+4.6**	155.9	169.0	**+8.4**	167.0	183.2	**+9.7**
128×128	2×8	153.6	195.0	**+27.0**	146.3	177.1	**+21.1**	148.0	170.8	**+15.4**	147.8	165.4	**+11.9**	151.6	178.4	**+17.7**
128×128	2×4	183.9	326.3	**+77.4**	186.1	246.0	**+32.2**	193.9	205.4	**+5.9**	175.0	194.0	**+10.9**	183.4	239.8	**+30.8**
128×128	4×8	111.5	141.3	**+26.7**	124.0	116.8	**-5.8**	114.9	122.5	**+6.6**	111.6	118.9	**+6.5**	113.0	141.7	**+25.4**
Avg. Δ	2×8	-	-	**+8.8**	-	-	**+8.1**	-	-	**+4.9**	-	-	**+8.5**	-	-	**+6.5**
128×128	**Avg. Δ**	-	-	**+43.7**	-	-	**+15.8**	-	-	**+9.3**	-	-	**+9.8**	-	-	**+24.6**

(a) Maximum Clock Frequency [MHz]

PRF Size	Vector Lanes	rect only			rect&row			rect&col			row&col			rect&trect		
		std.	cust.	Δ[%]	std.	cust.	Δ[%]	std.	cust.	Δ[%]	std.	cust.	Δ[%]	std.	cust.	Δ[%]
16×16	2×8	18034	15846	**-12.1**	18438	16350	**-11.3**	19585	18840	**-3.8**	20059	18227	**-9.1**	19308	17588	**-8.9**
32×32	2×8	18941	16075	**-15.1**	19326	16601	**-14.1**	20830	18755	**-10.0**	20791	18216	**-12.4**	20807	18042	**-13.3**
64×64	2×8	20008	15860	**-20.7**	20308	16714	**-17.7**	21860	19078	**-12.7**	21698	17808	**-17.9**	21807	17924	**-17.8**
128×128	2×8	21129	15965	**-24.4**	20873	16893	**-19.1**	22307	19670	**-11.8**	22459	18207	**-18.9**	22236	17868	**-19.6**
128×128	2×4	5538	6169	**+11.4**	5154	4243	**-17.7**	5644	5152	**-8.7**	5934	4699	**-20.8**	5979	4730	**-20.9**
128×128	4×8	64526	45719	**-29.1**	66811	49622	**-25.7**	92952	78811	**-15.2**	92529	76406	**-17.4**	93971	74511	**-20.7**
Avg. Δ	2×8	-	-	**-18.1**	-	-	**-15.6**	-	-	**-9.6**	-	-	**-14.6**	-	-	**-14.9**
128×128	**Avg. Δ**	-	-	**-14.0**	-	-	**-20.8**	-	-	**-11.9**	-	-	**-19.0**	-	-	**-20.4**

(b) Total Area [LUTs]

TABLE VII
VIRTEX-7 XC7VX1140T-2 SYNTHESIS RESULTS FOR 2 READ PORTS, 1 WRITE PORT, 64 BIT DATA WIDTH

The linear address function can be customized for accessing blocks of $p \cdot q$ elements. In this case, the coordinates (i, j) refer to the upper left corner of the accessed block, and (k, l) to the coordinates of the memory module in the $p \times q$ memory module array:

$$A_{customized}(i,j) = \left(\left\lfloor \frac{i}{p} \right\rfloor + c_i\right) \cdot \left(\frac{M}{q}\right) + \left\lfloor \frac{j}{q} \right\rfloor + c_j \quad (5)$$

The c_i and c_j coefficients are unique for each memory scheme and parallel access shape, and for the multi-view schemes [4] they are defined in Tables II, III, IV and V. The coefficients for the **rectangle only** scheme [13] are defined as:

$$c_{i,rect.only} = \begin{cases} 1, k < i\%p \\ 0, \text{otherwise} \end{cases} \qquad c_{j,rect.only} = \begin{cases} 1, l < j\%q \\ 0, \text{otherwise} \end{cases}$$

In order to compute the coefficients for the main and secondary diagonals, the ω constants are to be computed, which represent the multiplicative inverses of the pairs $(q+1; p), (q-1; p), (p+1; q)$ and $(p-1, q)$. Table VI contains the ω constants for $p = 2 \ldots 4$ and $q = 2 \ldots 8$.

IV. EXPERIMENTAL SET-UP AND RESULTS

In this section, we propose the generic PRF design, describe the methodology used for the FPGA evaluation and present the synthesis results.

Generic PRF design: The block diagram of an 8 vector lanes PRF hardware implementation is shown in Fig. 2. The data of the PRF is distributed among $p \times q$ linearly accessible memory modules, organized in a 2D matrix with p rows. Depending on the parallel memory scheme employed, such an organization allows the efficient use of up to $p \cdot q$ lanes.

The input of the Address Generation Unit (AGU) consists of the upper left coordinates of the accessed vector (i and j) and the access type (e.g., rectangle, row, column or diagonal), and computes the addresses of all PRF elements which are accessed, denoted as $i + \alpha, j + \beta$. The generated addresses are fed to the module assignment function, which controls the read and write shuffles. Since accessing the memory modules introduces a delay of one cycle, it is also necessary to delay the control signals for the data shuffle block when performing a read operation.

In the standard case, the AGU provides the input to the regular addressing function (Eq. 4), mapping each accessed element to the location in the corresponding memory module. However, the addresses should be reordered according to the MAF before arriving at the memory modules.

When using the customized addressing functions, the shaded blocks in Fig. 2 are replaced by the c_i, c_j coefficients as well as the customized addressing function (Eq. 5), eliminating the need to shuffle the read and write intra-module addresses. This is possible because the c_i and c_j coefficients are computed only using the upper left coordinates of the accessed vector (i and j) and the coordinates of the memory module (k and l).

Experimental Set-Up: The PRF mitigates some of the restrictions imposed by previous vector architectures, regardless of the implementation technology used, allowing variable number of runtime adjustable vector registers of arbitrary shapes and sizes. Generally, ASIC implementation offers the highest performance, but by using reconfigurable hardware the runtime adaptability may be further enhanced. ASIC PRF will have the total storage size as well as its aspect ratio fixed. Using FPGAs, this limitation may be removed using runtime partial reconfiguration. Further enhancements such as runtime adjustment of the number of vector lanes, register file ports and data width may allow the PRF to scale with performance, power and thermal constraints.

As a proof of concept, we implemented a PRF prototype design (excluding the Special Purpose Registers) with 2 read

and 1 write ports with 64-bit data path, using Synplify Premier F-2011.09-1, and targeting a Virtex-7 XC7VX1140T-2 device. This prototype implementation uses full crossbars as read and write address shuffle blocks. We have coupled two dual-port BRAMs and duplicated the data in order to obtain 2 read and 1 write ports.

Experimental Results: The FPGA synthesis results are presented in Table VII. We consider four sizes for the 16 lanes PRFs - 16×16, 32×32, 64×64 and 128×128 and three multi-lane configurations for the 128×128 PRF - 8 / 16 / 32 lanes. We label the columns for the standard designs as *std.*, the custom ones as *cust.*, and the difference - by Δ *[%]*.

The highest clock frequency for the **rectangle only** scheme is 326 MHz, and 246 MHz for the multi-view schemes. The clock frequency and total area is mainly influenced by the number of lanes. By increasing the capacity from 16×16 up to 128×128, area increases by up to 3095 LUTs and the clock frequency is reduced by up to 38 MHz. For PRFs smaller than 64×64, storage capacity has little impact on the clock frequency. By doubling the number of lanes, the total area increases by a factor of around four, an exponential increase.

By customizing the addressing function, we can increase the clock frequency up to 77% for **rectangle only** and 32% for the multi-view schemes. On average, the frequency is increased by nearly 9% for 16-lanes PRFs and up to almost 25% for multi-view, 128×128 PRFs. The largest area savings are 29% for the 32-lane, 128×128 **rectangle only** PRF and nearly 26% for the 32-lane **rect&row** scheme.

The results show that even without optimized shuffle blocks, PRFs are implementable on FPGAs using the selected parallel access schemes, with a minimum clock frequency of 111 MHz and area usage of less than 94000 LUTs (10% of the available LUTs). By analyzing the detailed area and timing reports, we observed that the full crossbars consume a significant amount of the total area, and, being part of the design critical path they represent the main performance bottleneck in our PRF implementation. Therefore, the crossbars are the immediate candidates for further design optimizations.

V. Conclusions and Future Work

We proposed a design employing a 2D array of $p \times q$ memory modules and a parallel access memory scheme for a PRF implementation, connected to multiple vector lanes. We selected one single-view and a set of four complementary, non redundant multi-view Module Assignment Functions suitable for implementations of a PRF providing the following widely used conflict free patterns: 1) $p \times q$ rectangle, $p \cdot q$ row, $p \cdot q$ diagonals if $(p, q+1)$ and $(p, q-1)$ are co-prime; 2) $p \times q$ rectangle, $p \cdot q$ column, $p \cdot q$ diagonals if $(p+1, q)$ and $(p-1; q)$ are co-prime; 3) $p \cdot q$ row, $p \cdot q$ column, aligned $p \times q$ rectangle; 4) mutually transposed $p \times q$, $q \times p$ rectangles. Reconfigurable hardware was chosen for the implementation due to its ability to adapt the PRF at runtime. Synthesis results of a 2 read and 1 write ports PRF prototype for a Virtex-7 FPGA implementing the selected MAFs indicate feasible clock frequencies with trivial hardware area utilization. Using customized addressing functions further reduced the hardware area and the clock cycle time by avoiding address routing circuits. We have identified the full crossbar shuffle networks as the main bottleneck of our hardware implementation. Therefore, we will investigate the potential of employing custom solutions, or even dynamic runtime customization of the interconnect using partial reconfiguration. Furthermore, we plan to evaluate in more details the performance, energy consumption and hardware complexity of the proposed schemes.

References

[1] ITRS: International Technology Roadmap for Semiconductors. Online, 2011 Edition. http://www.itrs.net/.

[2] K. Akdemir et al. Breakthrough AES Performance with Intel AES New Instructions. White paper, June 2010. Available online (12 pages).

[3] W. Buchholz. The IBM System/370 vector architecture. *IBM Systems Journal*, pages 51–62, 1986.

[4] C. Ciobanu, G. K. Kuzmanov, and G. N. Gaydadjiev. Parallel Access Schemes for Polymorphic Register Files. *Under Review*, 2012.

[5] C. Ciobanu, G. K. Kuzmanov, A. Ramirez, and G. N. Gaydadjiev. A Polymorphic Register File for Matrix Operations. In *Proceedings of the 2010 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS 2010)*, pages 241–249, July 2010.

[6] CB Ciobanu, X. Martorell, G. Kuzmanov, A. Ramirez, and G. N. Gaydadjiev. Scalability Evaluation of a Polymorphic Register File: a CG Case Study. In *Proceedings of the 2011 Conference on Architecture of Computing Systems (ARCS 2011)*, pages 13–25, February 2011.

[7] Jesus Corbal, Roger Espasa, and Mateo Valero. MOM: a Matrix SIMD Instruction Set Architecture for Multimedia Applications. In *Proceedings of the ACM/IEEE SC99 Conference*, pages 1–12, 1999.

[8] W. Gentzsch, K. Iwano, D. Johnston-Watt, M.A. Minhas, and M. Yousif. Self-adaptable autonomic computing systems: An industry view. In *Database and Expert Systems Applications, 2005. Proceedings. Sixteenth International Workshop on*, pages 201 –205, aug. 2005.

[9] Linley Gwennap. AltiVec Vectorizes PowerPC. *Microprocessor Report*, 12(6):1–5, May 1998.

[10] IBM. *Cell BE Programming Handbook Including the PowerXCell 8i Processor*, 1.11 edition, May 2008.

[11] B.H.H. Juurlink, D. Cheresiz, S. Vassiliadis, and H. A. G. Wijshoff. Implementation and Evaluation of the Complex Streamed Instruction Set. *Int. Conf. on Parallel Architectures and Compilation Techniques (PACT)*, pages 73 – 82, 2001.

[12] D.J. Kuck and R.A. Stokes. The Burroughs Scientific Processor (BSP). *IEEE Transactions on Computers*, C-31(5):363–376, May 1982.

[13] G. Kuzmanov, G. N. Gaydadjiev, and S. Vassiliadis. Multimedia rectangularly addressable memory. *IEEE Transactions on Multimedia*, pages 315–322, April 2006.

[14] D.K. Panda and K. Hwang. Reconfigurable Vector Register Windows for Fast Matrix Computation on the Orthogonal Multiprocessor. In *Application Specific Array Processors, 1990. Proceedings of the International Conference on*, pages 202 –213, 5-7 1990.

[15] JongSoo Park, Sung-Boem Park, James D. Balfour, David Black-Schaffer, Christos Kozyrakis, and William J. Dally. Register Pointer Architecture for Efficient Embedded Processors. In *DATE '07: Proceedings of the conference on Design, Automation and Test in Europe*, pages 600–605, San Jose, CA, USA, 2007. EDA Consortium.

[16] A. Ramirez, F. Cabarcas, B. Juurlink, M. Alvarez Mesa, F. Sanchez, A. Azevedo, C. Meenderinck, C. Ciobanu, S. Isaza, and G. Gaydadjiev. The SARC Architecture. *IEEE Micro*, 30(5):16 –29, Sept.-Oct. 2010.

[17] A. Shahbahrami, B.H.H. Juurlink, and S. Vassiliadis. Matrix Register File and Extended Subwords: Two Techniques for Embedded Media Processors. In *Proceedings of the 2nd ACM Int. Conf. on Computing Frontiers*, pages 171–180, May 2005.

[18] Mark Weiser. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(3):3–11, July 1999.

[19] S. Wong, F. Anjam, and M.F. Nadeem. Dynamically Reconfigurable Register File for a Softcore VLIW Processor. In *Proceedings of the Design, Automation and Test in Europe Conference (DATE 2010)*, pages 969–972, March 2010.