# Scalability Study of Polymorphic Register Files

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

***Abstract*—We study the scalability of multi-lane 2D Polymorphic Register Files (PRFs) in terms of clock cycle time, chip area and power consumption. We assume an implementation which stores data in a 2D array of linearly addressable memory banks, and consider one single-view and four suitable multi-view parallel access schemes which cover all basic access patterns commonly used in scientific and multimedia applications. The PRF design features 2 read and 1 write ports, targeting the TSMC 90nm ASIC technology. We consider three PRF sizes - 32KB, 128KB and 512KB and four multi-lane configurations - 8 / 16 / 32 and 64 lanes. Synthesis results suggest that the clock frequency varies between 500MHz for a 512KB PRF with 64 vector lanes and 970Mhz for a 32KB / 8-lanes case. Estimated power consumption ranges from less than 300mW (dynamic) and 10mW (leakage) for our 8-lane, 32KB PRF up to 8.7W (dynamic) and 276mW (leakage) for a 512KB with 64 lanes. We also show the correlation among the storage capacity, the number of lanes, and the chip overall area. Furthermore, we also investigated customized addressing functions. Our experimental results suggest up to 21% increase of the clock frequency, and up to 39% combinational hardware area reduction (nearly 10% of the total area) compared to our straightforward implementations. Concerning power, we reduce dynamic power with up to 31% and leakage with nearly 24%.***

## I. Introduction

Processor designs have reached a point where increasing the clock frequency is no longer feasible due to power and thermal constraints of the silicon technology. As more transistors are available with each semiconductor generation, two major trends are used to improve performance: i) Chip Multiprocessor (CMP) designs, relying on multithreading and ii) hardware acceleration of specific workloads. Examples of specialized extensions of General Purpose Processors (GPPs) include Single Instruction Multiple data (SIMD) facilities, exploiting Data Level Parallelism, but also custom hardware support for, e.g., encryption algorithms such as the Advanced Encryption Standard (AES) [1]. This shift in recent processor architectures also influenced the way maximum performance is achieved: the designers balance between adequate single threaded performance and multi-processor scalability, often complicating the programming paradigms.

The Polymorphic Register File (PRF) [4] was proposed to provide a relaxed view for the programmer of high performance vector applications. Unlike in existing vector architectures, such as IBM 370 [3] and the Synergistic Processor Units in the Cell BE [9], the PRF storage is not divided in a fixed number of equally sized registers. The PRF registers are multidimensional, with arbitrary sizes and can be created / resized at runtime. Previous studies ([4], [14]) have demonstrated that PRFs suit computationally intensive workloads such as Floyd, the Conjugate Gradient (CG) method and dense matrix multiplication. Moreover, PRFs could improve performance and efficiency in state of the art many-core computers, potentially saving area and power as shown in [5]. The benefits of two-dimensional (2D) PRFs are: i) improved storage efficiency, as the number of registers and their dimensions / sizes are dynamically following the workload requirements; and ii) performance gain, due to the reduced number of committed instructions. Previous work, however, did not study the PRF designs scalability in terms of different storage sizes.

This paper provides a scalability study of multi-module, multi-lane PRFs featuring 2 read and 1 write ports, targeting the TSMC 90nm ASIC technology. Three storage capacities are considered: 32KB, 128KB and 512KB along with four multi-lane configurations: 8 / 16 / 32 and 64 lanes. More specifically, the main contributions of this work are:

- Cycle time analysis, suggesting that the maximum clock frequency varies between 500MHz for a 512KB PRF with 64 vector lanes and 970Mhz for a 32KB, 8-lanes PRF;
- Relative combinational hardware and total area analysis indicate that for the 32KB and 128KB capacities, the combinational area grows exponentially when increasing the number of vector lanes. For 512KB PRF, the combinational area difference between the 8 and 16 lanes versions is of approximately 50%;
- Study of the dynamic and leakage power trends. Dynamic power varies between approximately 300mW for an 8-lane, 32KB PRF and 8.7W for 512KB, 64 lanes. Leakage power is between 10mW for a 32KB, 8 lanes PRF and 276mW for a 512KB, 64-lane PRF;
- Analysis of customized module addressing functions. Experimental results suggest clock frequency and combinational area improvements of up to 21% and 39%, with 10% decrease of total chip area. Furthermore, we can reduce dynamic power with 31% and leakage with nearly 24% compared to our straightforward implementations.

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

## II. Background and Related Work

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 [4]. Fig. 1 provides an example of a 2D PRF with a physical register size of 8 by 8 elements, containing 5 registers, defined using the Special Purpose Registers (SPR). Currently, only 1D and 2D operands are supported, but the PRF can be extended for any number of dimensions. Potential performance gains are due to multi-axis vectorization, efficient register storage utilization, higher

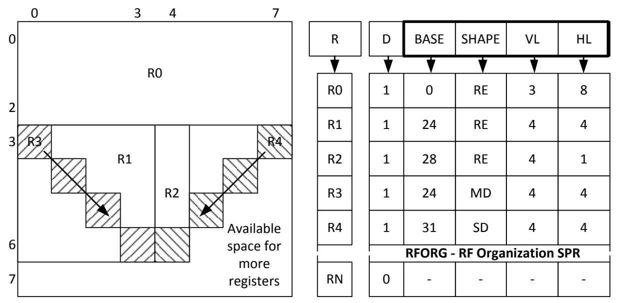


Fig. 1. The Polymorphic Register File, N = 8, M = 8

bandwidth utilization, customizable number of registers and reduction of the program code size.

**Previous works** indicate possible reductions of the number of executed instructions by three orders of magnitude due to PRF [4]. 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 [4]. 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 [14]. One of the objectives of the PRF, as part of the Scalable ARChitecture (SARC) project [14], is multi-core scalability. 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 when using up to 256 PRF instances, with 32 vector lanes each. Moreover, a similar performance level could be achieved by fewer PRF instances than the cores needed in a Cell BE-based system, potentially saving area and power [5]. An FPGA implementation, prototyped in [6], can adjust additional PRF parameters during runtime (e.g., total storage size, number of lanes and ports), at the expense of lower clock frequency compared to the ASIC version presented here.

In all previous studies, up to 32 vector lanes were considered. All related studies indicate that the ability of the PRF to provide data to multiple parallel vector lanes at high rates is a key enabler of high performance computing. The main goal of the work presented hereafter is to analyze the scalability of the PRFs and identify the possible bottlenecks of the considered implementation. In order to determine if even higher performance PRFs than studied before are practically feasible, we examine configurations with up to 64 parallel lanes, and evaluate the clock cycle time, area and power.

We assume a PRF implementation containing $N \times M$ data elements, stored using $p \times q$ memory modules, which enables efficient use of up to $p \cdot q$ parallel vector lanes [2], each containing one or more vector functional units. Throughout this paper, we will use "$\times$" to refer to a 2D matrix, and "$\cdot$" to denote multiplication.

We consider five parallel access schemes suitable for the implementation of the PRF: the single-view Rectangle Only (**ReO**) scheme, which supports conflict free accesses shaped as $p \times q$ rectangles, suggested in [11], and a set of four multi-view schemes, supporting conflict free access to the most common vector operations for scientific and multimedia applications, also used in [6]: 1) Rectangle Row (**ReRo**): $p \times q$ 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) Rectangle Column (**ReCo**): $p \times q$ 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 Column (**RoCo**): $p \cdot q$ row, $p \cdot q$ column, aligned ($i\%p = 0$ or $j\%q = 0$) $p \times q$ rectangle; 4) Rectangle Transposed Rectangle (**ReTr**): $p \times q$, $q \times p$ rectangles (transposition) if $p\%q = 0$ or $q\%p = 0$.

TABLE I
THE MODULE ASSIGNMENT FUNCTIONS

| Scheme | $m_v$ | $m_h$ |
|---|---|---|
| **ReO** | $i\%p$ | j % q |
| **ReRo** | $\left(i + \left\lfloor \frac{j}{q} \right\rfloor\right)\%p$ | $j\%q$ |
| **ReCo** | $i\%p$ | $\left(\left\lfloor \frac{i}{p} \right\rfloor + j\right)\%q$ |
| **RoCo** | $\left(i + \left\lfloor \frac{j}{q} \right\rfloor\right)\%p$ | $\left(\left\lfloor \frac{i}{p} \right\rfloor + j\right)\%q$ |
| **ReTr**, $p < q$ | $i\%p$ | $(i - i\%p + j)\%q$ |

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. The row and column of the memory module are computed by the Module Assignment Functions (MAFs) $m_v()$ and $m_h()$, and the intra-module position by the addressing function $A()$. MAFs for the five considered schemes are shown in Table I.

For all the schemes considered, the standard linear address assignment function [6] 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:

$$A(i,j) = \left(\left\lfloor \frac{i}{p} \right\rfloor + c_i\right) \cdot \left(\frac{M}{q}\right) + \left\lfloor \frac{j}{q} \right\rfloor + c_j \qquad (1)$$

The $c_i$ and $c_j$ coefficients are unique for each memory scheme and access shape. For the standard, non-customized addressing function, $c_i = c_j = 0$. Their complete mathematical definitions are provided in [11] and in [6].

**Related Work**: The efficient processing of multidimensional matrices 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) [10]. 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) [8] approach did not make use of data registers. CSI allows the processing of 2D data streams of arbitrary length, but requires data caches to benefit from data locality. Our approach suggests the register file as a cost-effective alternative of high speed data caches.

The Vector Register Windows (VRW) [12] concept allows 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

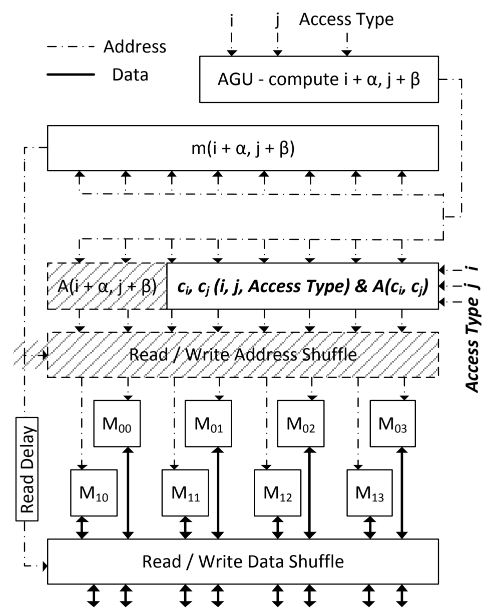


Fig. 2. PRF block diagram, p=2, q=4

shapes of the two dimensional registers. A Modified MMX (MMMX) [15] supports 8 multimedia registers, each 96 bits wide, with matrix operations limited to only loads and stores.

The Register Pointer Architecture(RPA) [13] extends scalar processors by adding two additional register files - Dereferencible Register File (DRF) and the Register Pointers (RP). The DRF provides the 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 points to a matrix, being more suitable for vectors.

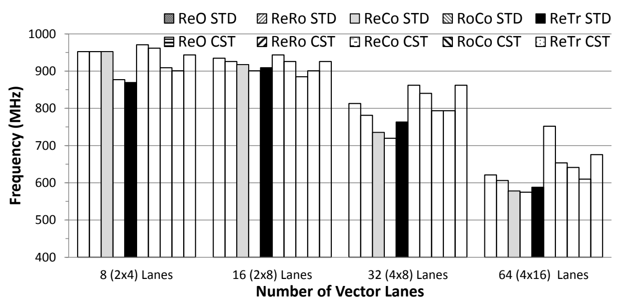
In order to adjust the number of registers and the total size of the physical register file in a VLIW, FPGA partial reconfiguration is used in [16]. Our approach assumes fixed physical register file size, but at a higher level logical view, offers variable fragmentation of the storage space, eliminating many overhead instructions, potentially improving performance. While partial reconfiguration is only available in FPGAs, the PRF does not rely on any specific hardware technology, therefore it can be successfully implemented in both ASICs and FPGAs.

## III. Design Under Study

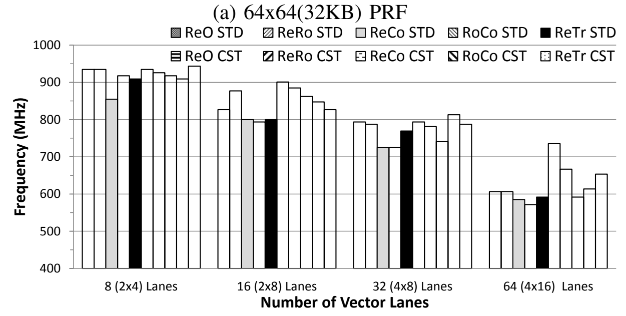
The block diagram of an 8 vector lanes PRF hardware implementation, first proposed in [6], is depicted in Fig. 2 (SPR not shown). The data of the PRF is distributed among $p \times q$ linearly accessible memory modules, organized in a 2D matrix with p rows. The Address Generation Unit (AGU) computes the addresses of all involved PRF elements, denoted as $i + \alpha, j + \beta$. The generated addresses are fed to the MAF $m()$, which controls the read and write shuffles.

In the standard case, the AGU provides the input to the regular addressing function, mapping each accessed element to the location in the corresponding memory module. However, the addresses need to be reordered according to the MAF before being sent to the memory modules.

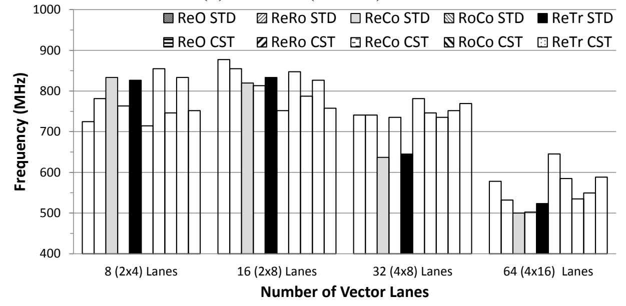
When using the customized addressing functions, the shaded blocks in Fig. 2 are replaced by the $c_i, c_j$ coefficients as well



(a) 64x64(32KB) PRF



(b) 128x128(128KB) PRF



(c) 256x256(512KB) PRF

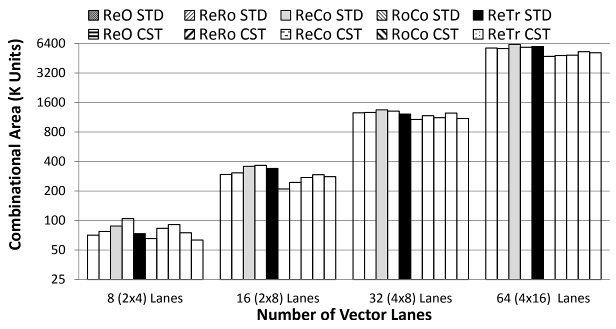
| Size | 64x64(32KB) | | | | 128x128(128KB) | | | | 256x256(512KB) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Scheme / Lanes | 8 | 16 | 32 | 64 | 8 | 16 | 32 | 64 | 8 | 16 | 32 | 64 |
| **ReO** | 1.9 | 0.9 | 6.0 | 21.1 | 0.0 | 9.0 | 0.0 | 21.3 | -1.4 | -14.3 | 5.5 | 11.6 |
| **ReRo** | 1.0 | 0.0 | 7.6 | 7.8 | -0.9 | 0.9 | -0.8 | 10 | 9.4 | -0.8 | 0.7 | 9.9 |
| **ReCo** | -4.5 | -3.5 | 7.9 | 10.9 | 7.3 | 7.8 | 2.2 | 1.2 | -10.4 | -3.9 | 15.4 | 7.0 |
| **RoCo** | 2.7 | 0.0 | 10.3 | 6.1 | -0.9 | 6.8 | 12.2 | 7.4 | 9.2 | 1.7 | 2.3 | 9.3 |
| **ReTr** | 8.5 | 1.9 | 12.9 | 14.9 | 3.8 | 3.3 | 2.4 | 10.5 | -9 | -9.1 | 19.2 | 12.4 |

(d) Maximum Frequency Difference, CST vs. STD (%, $-$ denotes decrease)

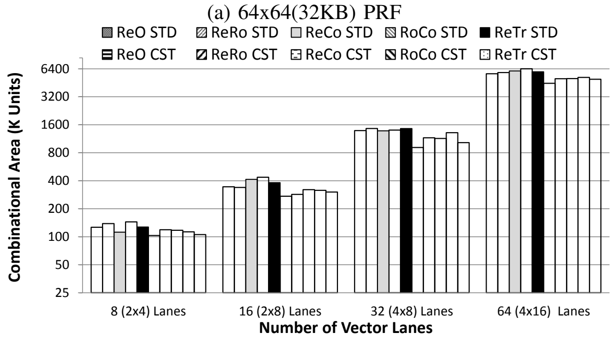
Fig. 3. Clock Frequency

as the customized addressing function (Eq. 1), eliminating the need to shuffle the read and write intra-module addresses.

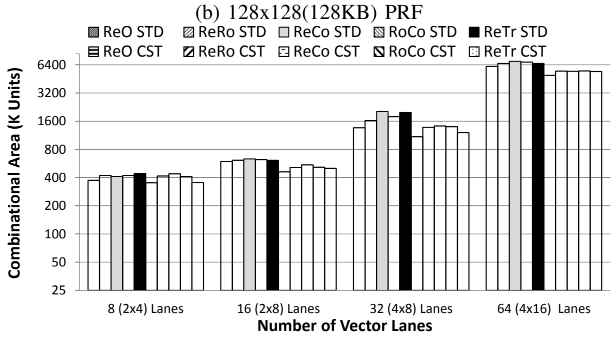
We implemented a PRF prototype design with 2 read and 1 write ports with 64-bit data path using SystemVerilog. For synthesis, Synopsys Design Compiler Ultra version F-2011.09-SP3 in topographical mode was used, which accurately predicts both leakage and dynamic power. As no particular workload was considered in this study, we did not use switching activity traces for power estimations. In all experiments, the tool was programmed to optimize for best timing, targeting the TSMC 90nm technology. In all considered designs, the shuffle networks have been implemented using full crossbars. Using the Artisan memory compiler, a 1GHz 256x64-bit dual-port SRAM register file was generated and used as the atomic storage element for our PRF. When the required capacity of the memory modules exceeded the available maximum of 256, several SRAM modules were connected together. We coupled two dual-port SRAMs and duplicated their data in order to



(a) 64x64(32KB) PRF



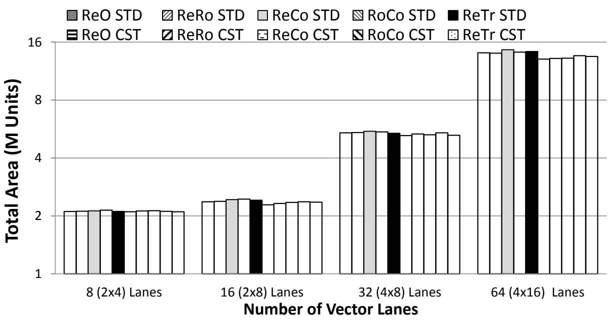
(b) 128x128(128KB) PRF



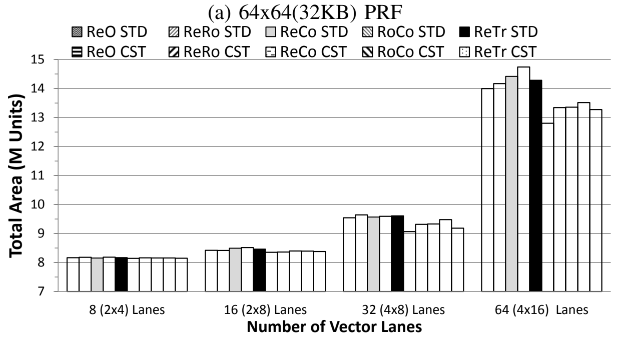
(c) 256x256(512KB) PRF

| Size | 64x64(32KB) | | | | 128x128(128KB) | | | | 256x256(512KB) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Scheme / Lanes | 8 | 16 | 32 | 64 | 8 | 16 | 32 | 64 | 8 | 16 | 32 | 64 |
| **ReO** | -8 | -29 | -14 | -18 | -18 | -21 | -34 | -21 | -6 | -23 | -20 | -20 |
| **ReRo** | 8 | -20 | -7 | -15 | -14 | -16 | -20 | -14 | -1 | -17 | -15 | -17 |
| **ReCo** | 4 | -23 | -17 | -22 | 5 | -22 | -17 | -18 | 6 | -14 | -30 | -22 |
| **RoCo** | -28 | -20 | -5 | -10 | -22 | -28 | -6 | -19 | -3 | -17 | -22 | -20 |
| **ReTr** | -14 | -18 | -10 | -14 | -17 | -21 | -29 | -17 | -20 | -18 | -39 | -18 |

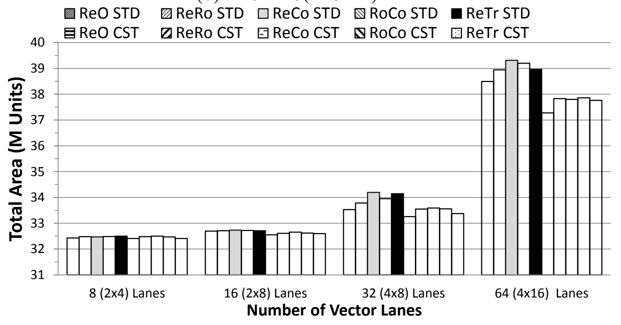
(d) Combinational Area Difference, CST vs. STD (%, − denotes decrease)
Fig. 4. Combinational Area



(a) 64x64(32KB) PRF



(b) 128x128(128KB) PRF



(c) 256x256(512KB) PRF

| Size | 64x64(32KB) | | | | 128x128(128KB) | | | | 256x256(512KB) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Scheme / Lanes | 8 | 16 | 32 | 64 | 8 | 16 | 32 | 64 | 8 | 16 | 32 | 64 |
| **ReO** | -0.2 | -3.6 | -3.3 | -7.3 | -0.3 | -0.8 | -5 | -8.5 | -0.1 | -0.4 | -0.8 | -3.2 |
| **ReRo** | 0.3 | -2.5 | -1.7 | -6.2 | -0.2 | -0.6 | -3.4 | -5.9 | 0 | -0.3 | -0.7 | -2.9 |
| **ReCo** | 0.2 | -3.3 | -4.1 | -9.6 | 0.1 | -1.1 | -2.5 | -7.4 | 0.1 | -0.2 | -1.8 | -3.9 |
| **RoCo** | -1.4 | -3 | -1.1 | -4.2 | -0.4 | -1.5 | -1.2 | -8.3 | 0 | -0.3 | -1.2 | -3.4 |
| **ReTr** | -0.5 | -2.4 | -2.4 | -5.8 | -0.3 | -1 | -4.5 | -7.1 | -0.3 | -0.3 | -2.3 | -3.1 |

(d) Total Area Difference, CST vs. STD (%, − denotes decrease)
Fig. 5. Total Area

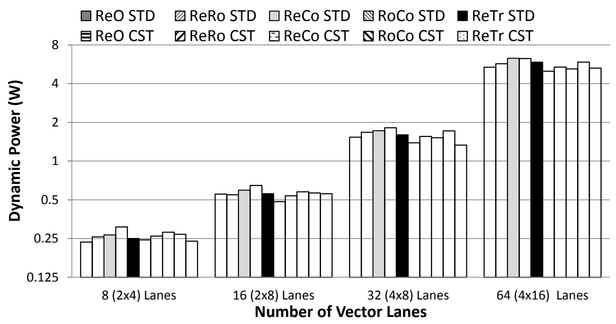
obtain 2 read ports. For all considered configurations, we label the straightforward designs which use the regular addressing function as **STD**, and the ones using the custom addressing function and the $c_i$ and $c_j$ coefficients as **CST**.

## IV. Evaluation Results

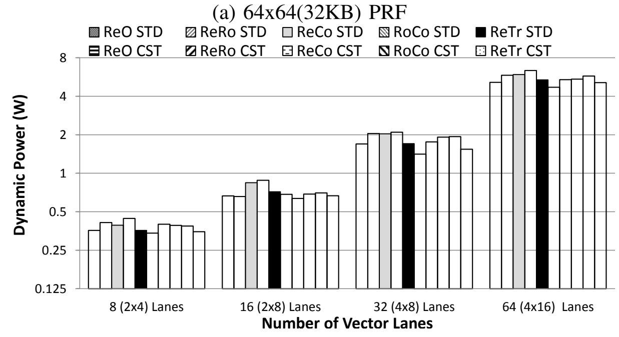
We study clock frequency, chip area and dissipated power.

**Clock Frequency:** The clock frequency evaluation results are presented in Fig. 3. The first five configurations use the regular addressing function, while the last five use customized ones. As suggested by Fig. 3(a), for a 32KB PRF, the highest clock frequency - 970MHz - is estimated for the 8 lanes **ReO** CST scheme. The clock frequency decreases as more vector lanes are added to the design, and the slowest 32KB configuration is **RoCo** STD, clocked at 574MHz. The difference between 8 and 16 vector lanes is less than 50MHz for all the configurations, increasing to around 100Mhz - 150MHz between 16 and 32 lanes. The largest drop in clock speed, of approximately 200 MHz is when 64 lanes are available. The same trend holds for the 128KB and 512KB configurations (Fig. 3(b) and 3(c)), with the slowest design being **ReCo** STD, which runs at 500MHz. For a 512KB PRF, the fastest STD scheme is **ReO**, which has the simplest MAF and custom addressing functions, with a minimum operating frequency of 578MHz, up to 15% faster than the multi-view designs. The results suggest that the number of vector lanes influences the clock frequency stronger than the storage size. Had larger SRAM modules been utilized, storage size influence would probably be further reduced.

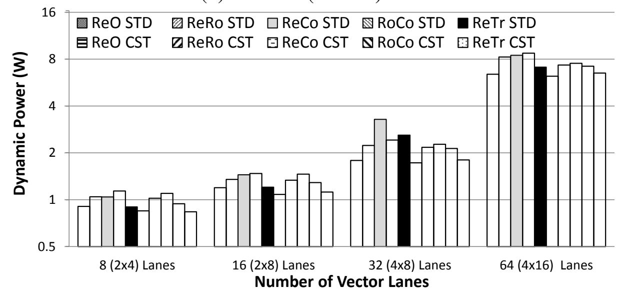
Fig. 3(d) depicts the relative difference when utilizing the customized addressing functions. Positive values indicate frequency increase, while negative - frequency decrease. If 8 or 16 lanes are used, customizing the addressing functions does



(a) 64x64(32KB) PRF



(b) 128x128(128KB) PRF

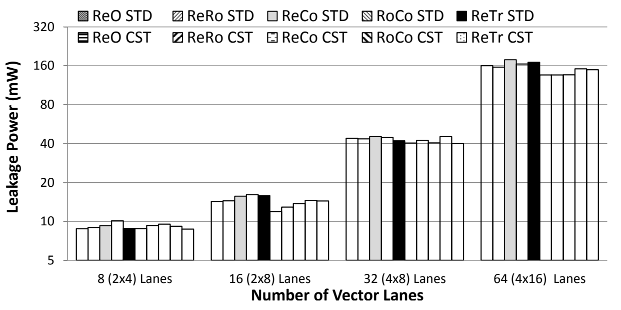


(c) 256x256(512KB) PRF

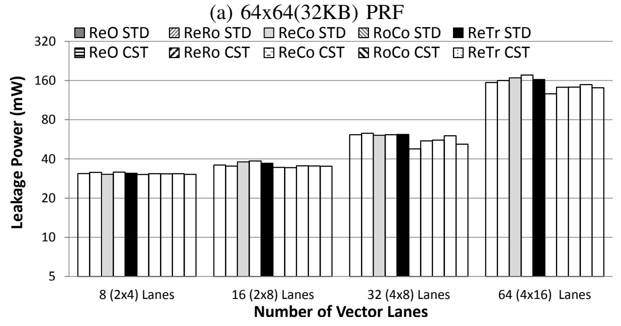
| Size | 64x64(32KB) | | | | 128x128(128KB) | | | | 256x256(512KB) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Scheme / Lanes | 8 | 16 | 32 | 64 | 8 | 16 | 32 | 64 | 8 | 16 | 32 | 64 |
| **ReO** | 4 | -12.7 | -9.4 | -7 | -4.8 | 2.9 | -16.7 | -8.6 | -6.4 | -9.4 | -3.5 | -2.9 |
| **ReRo** | 1.5 | -1.9 | -7.1 | -5.9 | -2.9 | -3.4 | -14.1 | -7.7 | -2.1 | -1.2 | -2.9 | -11.2 |
| **ReCo** | 4.8 | -2.8 | -11.9 | -17.3 | -0.3 | -18.6 | -5.6 | -7.9 | 5.4 | 1 | -30.9 | -11.1 |
| **RoCo** | -12.4 | -12.5 | -5.4 | -6.4 | -12.9 | -20.4 | -7.7 | -9.5 | -17.2 | -12.7 | -11.6 | -17.5 |
| **ReTr** | -4.7 | -0.2 | -17 | -10.2 | -2.3 | -6.9 | -9.5 | -4.9 | -7 | -6.9 | -30.7 | -8.4 |

(d) Dynamic Power Difference, CST vs. STD (%, − denotes decrease)

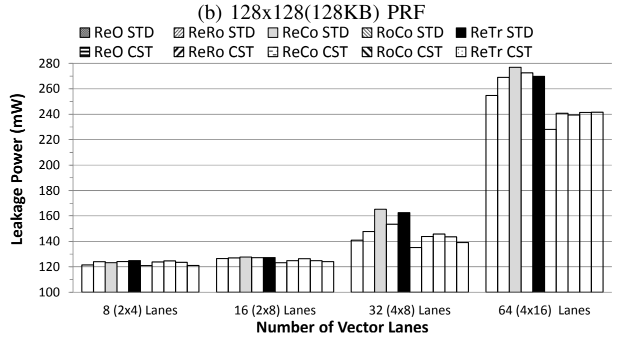
Fig. 6. Dynamic Power



(a) 64x64(32KB) PRF



(b) 128x128(128KB) PRF



(c) 256x256(512KB) PRF

| Size | 64x64(32KB) | | | | 128x128(128KB) | | | | 256x256(512KB) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Scheme / Lanes | 8 | 16 | 32 | 64 | 8 | 16 | 32 | 64 | 8 | 16 | 32 | 64 |
| **ReO** | 0.3 | -16.5 | -8.1 | -15.2 | -1.7 | -4.1 | -22.1 | -18 | -0.3 | -2.7 | -4 | -10.4 |
| **ReRo** | 3.4 | -10.6 | -2.3 | -13.1 | -2.2 | -2.8 | -12.5 | -11.1 | -0.2 | -1.7 | -2.6 | -10.4 |
| **ReCo** | 2.6 | -12.4 | -10.5 | -23.5 | 0.8 | -6.8 | -8.2 | -15.1 | 1.1 | -1 | -11.9 | -13.5 |
| **RoCo** | -9.3 | -9.4 | 1.5 | -8.1 | -2.9 | -8.4 | -1.8 | -15.7 | -0.5 | -1.8 | -6.5 | -11.4 |
| **ReTr** | -1.3 | -9.1 | -5.1 | -12.4 | -2.1 | -5.1 | -15.8 | -13.7 | -3.1 | -2.5 | -14.4 | -10.4 |

(d) Leakage Power Difference, CST vs. STD (%, − denotes decrease)

Fig. 7. Leakage Power

not necessarily enhance performance. However, for 32 or 64 lanes, the increase in clock frequency is as high as 21% for the 32KB **ReO** design, and 19.2% for the 512KB **ReTr** one. The shuffle networks are on the critical path of the design. For a small number of lanes, the increased complexity of the customized addressing functions cancels out the advantage of eliminating the address routing circuits. However, the complexity of the full crossbars increases drastically for 32 and 64 ports and the elimination of the address shuffles leads to reduced clock cycle times for 32 and 64 lanes PRFs.

**Silicon Chip Area:** Fig. 4 and 5 present the combinational and total area of the PRF. When the number of lanes is increased, the combinational area increases exponentially for the 32KB and 128KB configurations (Fig. 4 (a) and 4 (b)), at a rate of approximately 4 times for doubling the number of vector lanes. For the 512KB configuration (Fig. 4 (c)), the difference in terms of combinational area for the 8 and 16 lanes versions is of approximately 50%, significantly smaller than between 16 to 32 lanes. For the 32 and 64 lanes versions, the exponential trend is maintained.

In Fig. 4(d), the relative reduction of combinational area obtained by using customized addressing functions is shown. For the **ReTr** scheme, savings of up to 39% can be observed, while for the **ReO** scheme the highest combinational area savings amount to 34% for the 128KB configuration.

When also factoring in the non-combinational area, for the 32KB configuration (Fig. 5(a)), using 16 lanes consumes approximately 10% more total area than the 8 lanes version. However, when vector lanes are added an exponential increase in total area is observed. Compared to the 16 lanes configuration, total area more than doubles when using 32 lanes, and it increases by a factor of approximately 6 for the 64 lanes PRF.

If the storage capacity of the PRF is increased to 128KB (Fig. 5(b)), the differences between the different multi-lane configurations are reduced. The area overhead of the 16 lanes configuration compared to one with 8 lanes is less than 5%, while the 32 lanes configuration consumes less than 20% more. The total area of the 64 lanes version is approximately 70% larger than the 8 lanes baseline. When the storage grows to 512KB (Fig. 5(c)), the difference regarding total area between 64 lanes and 8 lanes is less than 22%. When correlating with the combinational and total area results, it is clear that the relative area of the SRAM blocks becomes dominant as the total size of the PRF increases. Therefore, our results suggest that it would be most efficient to design a small (e.g., 32KB) PRF with a small number of vector lanes (8 or 16). Moreover, if a large storage space is required, the relative cost of additional vector lanes is significantly reduced.

Fig. 5(d) indicates that the exponential growth of the combinational area when the number of lanes is increased translates into higher total area savings if using customized addressing functions: the largest total area savings are observed for the 64 lanes versions: 7.3% for the 32KB **ReO** scheme, and 9.6% for the 32KB **ReCo** one. As the total size of the PRF is increased and the proportion of the combinational circuits is reduced, the total area savings also decrease: 3.2% of the 512KB **ReO** scheme and 3.9% for the **ReCo** configuration.

**Power:** Dynamic power evaluation results are shown in Fig. 6. For 32KB PRF (Fig. 6 (a)), the dynamic power increases exponentially from less than 0.3W for 8 lanes up to 6.3W for the **ReCo** STD scheme - an increase of more than 20 times. When the PRF capacity is increased to 128KB (Fig. 6 (a)), dynamic power consumption of 8 lane PRF increases by approximately 150mW, varying between 350mW to 443mW depending on the memory scheme. The highest power consumption for the 128KB, 64 lanes PRF remains 6.3W (**RoCo** STD). For the 512KB PRF, dynamic power consumption ranges from less than 1.2W for 8 lanes up to 8.7W for **RoCo** STD version, and Fig. 6 (c) suggests that the dynamic power increases by approximately 35% between 8 and 16 lane cases. The results show that for smaller PRFs (e.g., 32KB), dynamic power increases exponentially with the vector lanes, while for 512KB, the dynamic power penalty between 8 and 16 vector lanes is moderate.

Fig. 6 (d) shows that dynamic power can be reduced by customizing the addressing functions by up to 17.3% for the 32KB **ReCo** scheme with 64 lanes, 18.6% for the 128KB 16-lane **ReCo** scheme and 30.9% for 32-lane **ReCo**.

For the 32KB PRF, leakage power increases exponentially as vector lanes are added (Fig. 7 (a)), from less than 10mW for 8 vector lanes up to 178mW for 64 lanes. For larger capacity PRFs, the leakage from the SRAM blocks has a larger impact (Fig. 7 (b) and (c)). Therefore, there is a significant increase in leakage power for the 8 lanes PRF, which grows from less than 10mW for 32KB to approximately 30mW for 128KB and up to 125mW for the 512KB variant. However, the growth rate for the 64 lanes PRF is smaller: the highest leakage power for 32KB PRF is 178mW (STD **ReCo**), compared to 276mW for the 64-lane version. The results suggest a similar trend for the leakage as for the the dynamic power: the penalty of adding more lanes reduces as the capacity of the PRF increases.

Fig. 7 (d) reveals that using customized addressing functions can reduce leakage power by up to 23.5% for the 32KB PRFs with 64 lanes, 22.1% for the 128KB version with 32 lanes and 14.4% for the 512KB version with 32 vector lanes, the highest benefits being observed for the 32 and 64 lanes configurations.

## V. Conclusions and Future Work

We evaluated the scalability of 2D Polymorphic Register Files with 2 read and 1 write ports, using TSMC 90nm technology. We considered one single-view and four multi-view parallel memory schemes, three storage capacities: 32KB / 128KB / 512KB and four versions in terms of parallel vector lanes number: 8 / 16 / 32 and 64. Synthesis results suggest practically feasible clock frequencies, varying from 500MHz to 970MHz, and power consumption within implementable limits, ranging from less than 300mW dynamic and 10mW leakage up to 8.7W dynamic and 276mW leakage. Our results also indicate the correlation between the storage capacity, number of lanes and the combinational and total area. Using customized addressing functions further increases the maximum clock frequency by up to 21% and reduces by up to 39% the combinational area, and by nearly 10% the total chip size. We could reduce dynamic power with up to 31% and leakage with nearly 24% compared to our straightforward implementations. The evaluation results suggest that the PRF can be employed in both embedded and high performance computers. We have identified the full crossbar shuffle networks as the main implementation bottleneck. Therefore, in our future work, we will investigate customized interconnects.

## References

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

[2] K. Asanović. *Vector Microprocessors*. PhD thesis, University of California at Berkeley, 1998.

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

[4] C. Ciobanu et al. A Polymorphic Register File for Matrix Operations. In *Proceedings of SAMOS 2010*, pages 241–249, July 2010.

[5] C. Ciobanu et al. Scalability Evaluation of a Polymorphic Register File: a CG Case Study. In *Proceedings of ARCS '11*, pages 13–25, Feb. 2011.

[6] C. Ciobanu et al. On Implementability of Polymorphic Register Files. In *Proceedings of ReCoSoC 2012*, 2012.

[7] J. Corbal et al. MOM: a Matrix SIMD Instruction Set Architecture for Multimedia Applications. In *Proceedings of SC99*, pages 1–12, 1999.

[8] B. Juurlink et al. Implementation and Evaluation of the Complex Streamed Instruction Set. In *PACT '01*, pages 73 – 82, 2001.

[9] J. A. Kahle et al. Introduction to the Cell Multiprocessor. *IBM J. Res. Dev.*, 49(4/5):589–604, 2005.

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

[11] G. Kuzmanov et al. Multimedia rectangularly addressable memory. *IEEE Transactions on Multimedia*, pages 315–322, April 2006.

[12] D. Panda et al. Reconfigurable Vector Register Windows for Fast Matrix Computation on the Orthogonal Multiprocessor. In *Proceedings of Application Specific Array Processors*, pages 202–213, Sep 1990.

[13] J. Park et al. Register Pointer Architecture for Efficient Embedded Processors. In *DATE '07*, pages 600–605, 2007.

[14] A. Ramirez et al. The SARC Architecture. *Micro, IEEE*, 30(Issue:5):16 –29, Sept.-Oct. 2010.

[15] A. Shahbahrami et al. Matrix Register File and Extended Subwords: Two Techniques for Embedded Media Processors. In *Computing Frontiers '05*, pages 171–180, 2005.

[16] S. Wong et al. Dynamically Reconfigurable Register File for a Softcore VLIW Processor. In *DATE '10*, pages 969–972, March 2010.