A Self-Indexed Register File for Efficient Arithmetical Computing Hardware

Lei Yang
Institution of Automation, Chinese Academy of Sciences,
University of Chinese Academy of Sciences
95 Zhongguancun East Road, 100190, Beijing, China
yanglei2013@ia.ac.cn

Shaolin Xie ,Zijun Liu, Xueliang Du and Donglin Wang
Institution of Automation, Chinese Academy of Sciences,
95 Zhongguancun East Road, 100190, Beijing, China
{shaolin.xie & zijun.liu & xueliang.du & donglin.wang}@ia.ac.cn

Abstract—**This paper presents a novel register file with self-indexed features, targeting the DSP/media algorithm with massive data locality. The self-indexed register file (SIRF) contains 128 high-speed registers, 4 input ports and 4 output ports. It can be accessed with the double circular window mode, or simply with the immediate index mode. SIRF can eliminate write after write (WAW) dependence without register renaming in hardware or redundant register allocation in compilers, and it can also reduce the address computation if the accessing pattern satisfies the circular window mode. The SIRF was implemented in a high performance mathematical processor(MaPU). Two detailed application examples, finite impulse response (FIR) filtering and image interpolating, are demonstrated to show how SIRF can accelerate DSP/media algorithms. Evaluation shows that SIRF can dramatically reduce memory access. A 2.88x speed-up and a 20% overall power reduction are observed with the evaluated algorithms.**

***Keywords-** register file; arithmetical computing; energy-efficient*

I. Introduction

Computation intensive digital signal processing (DSP) applications usually require high computing performance with constrained energy efficiency. For example, for applications like cellular phones and mobile communicators, ultralow power consumption and high DSP capabilities are both required. To maximize the performance with limited hardware resource for certain computing task like stereo vision algorithms [1] and convolutional neural networks [2], specially designed architecture and carefully optimized hardware components are often used, and they are often quite different from general purpose processors [3].

Among these specially designed components, memory system design is a quite important issue for DSP/media applications. Applications in multimedia, video processing, wireless communication require efficient memory system to exchange intensive data. Different from most general purpose processors' choice of a Von Neumann architecture [4], many digital processing processors and accelerators adapt custom memory structure to accelerate their computing tasks, such as Harvard architecture and super Harvard architecture [5]. That's because traditional memory system is designed to fit the feasibility of general purpose computing, and it would fail in dealing with the heavy computing burden with strict power requirement.

Especially, for a mathematical processor that adapts pipeline structure and parallelization to accelerate arithmetical algorithms, traditional register file would have certain problems that slow down the whole performance. For example, the frequently register access brings massive address calculation, and it actually could be saved for certain arithmetical algorithms that have fixed register access pattern. Another problem is the rapid increasing of assembly instruction to deal with the register access when the computing kernel require huge amount of register operations. What's more, to take the most advantage of data locality is also a main concern of on-chip memory design. Traditional register file along with cache is quite complex and requires extra on-chip resource (usually more than 50% of the total chip area [6]), and it hardly reaches arithmetical processor's strict requirement of performance and limited resource limitation.

This paper brings up a new register file design to accelerate mathematical computing. We design a self-indexed register file (SIRF), which consists of a group of registers and a set of self-indexed input/output ports. Besides the normal method to directly access registers by register ID, it offers a new way to generate register's index through double circular windows, so as to automatically inputting/outputting data between main memory and the computing core inside processor. SIRF supports the data access pattern of many computation intensive algorithm, such as 2-dimension filtering, FIR, matrix multiplication, FFT, etc.

The paper is organized as follows. Chapter 2 introduces some similar works as SIRF. Chapter 3 describes the structure of SIRF and the self-indexer. Chapter 4 demonstrates 2 typical algorithm implementation examples with SIRF: FIR and polyphase image interpolation, to show how SIRF works to accelerate mathematical computing. The acceleration and power consumption reduction brought by SIRF are evaluated and analyzed in chapter 5. Chapter 6 gives conclusion and future work.

II. Related work

As the closest part to processor in the entire memory hierarchy, register file directly determines the top performance and working rate of the processor. There are many works in register file design and optimization.

A 4R/2W register file for dynamic voltage and frequency scaling processor is introduced in [7]. It uses full-*N* separated read ports and also reconfigurable write scheme to achieve power-efficiency. It has similar goal as SIRF, and both of them adapt configurable structure to improve performance.

An energy-efficient unified register file design for mobile graphic processing unit (GPU) is presented in [8]. The design re-arranges register file structure and uses fast indexing and allocation mechanism to reduce both dynamic and leakage energy of register file.

The optimization in [9] explores various trade-offs for the register file hierarchy and gives new schedule plan, and [10] gives new register file implementations with novel refresh solutions using bank bubble and bank walk-through for general purpose graphic processing unit (GPGPU). All these designs try to improve register file structure to achieve better performance and energy-efficiency of the whole processor.

III. SIRF structure

Rather than general purpose register file, the self-indexed register file contains 128 registers and a set of self-indexed input/output ports, as shown in figure 1. The 4 input ports supply the ability to write 4 registers concurrently, and the 4 output ports could send the value of 4 registers at most at the same time. Each port is equipped with a configurable indexer. The configuration of an indexer defines an accessible window of registers for its corresponding port, containing a pair of start and end pointers and a stride value.

There are 2 methods to access SIRF: directly access and self-indexed access. The former method just likes most general purpose register file, and the registers could be accessed by their register ID. For example, the expression "ALU.R0=R[2]" reads the value of #2 register to the register R0 of ALU; and expression "R[127]=ALU.R1" assigns the value of ALU's register R1 to the #127 register of SIRF.

The self-indexer has double circular window structure, as shown in figure 2. It could sequentially and circularly access the registers inside the configured window through a pointer hold by the indexer. More window configuration can be assigned to an indexer to form double circular data windows, which allow an inner window to slide inside the outer window. We use the word "location" to represent the register ID selected by the indexer's pointer. There are 4 main configurable parameters in this structure:

- *ISTART*: The start location of the outer circular window, ranged from 0 to 127.
- *MSIZE*: The size of the outer circular window (the number of registers in the outer circular window).
- *ISIZE*: The size of the inner circular window (the number of registers in the inner circular window).
- *STEPSIZE*: The stride of inner circular window. The number of current register's ID grows by *STEPSIZE*.

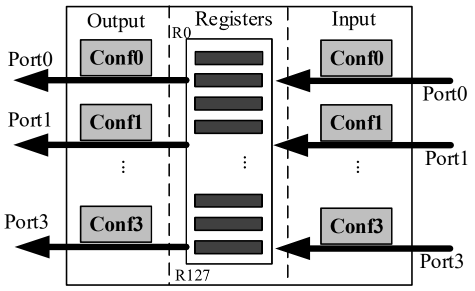

Figure 1. Structure of self-indexed register file: 4 output ports and 4 input ports. Each port is companied with a set of configurations, which can be configured independently as double circular window indexer.

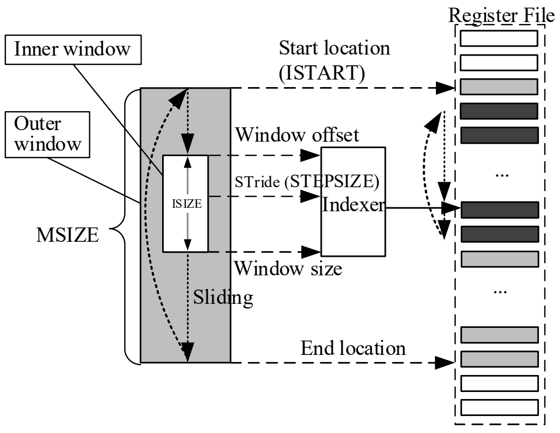

Figure 2. Buffer configuration for double circular windows: left is the logic representation of the double circular buffer, and right is the register file mapping. The 2 windows slide circularly under their pre-set constrains

Two parameters '*I*' and '*S*' are used to control the position of the two windows. *S* stands for the offset between outer circular window and inner circular window, and *I* stands for the offset between the current pointer's position and the start location of inner circular window. The current register pointer's location *current_index* could be calculated as:

$$current\_index = ISTART + (S + I)\ \%\ MSIZE \qquad (1)$$

The instruction option "I++" and "S++" along with the 4 configurable parameters control the exact behavior of SIRF. Corresponding to the definition of the parameters, the instruction option "I++" increases the offset between the current pointer's position and the start location of inner circular window, and each time "I++" modifies the value of I as:

$$I = (I + STEPSIZE)\ \%\ ISIZE \qquad (2)$$

Similar to I++, instruction option "S++" increases the offset between outer circular window and inner circular window. Each time "S++" is used, it resets the value of *I* to *I=0*, and the value of *S* is modified as:

$$S = \left(S + STEPSIZE\right) \% \ MSIZE \tag{3}$$

The usage of SIRF's self-indexer includes 2 typical scenarios: single window indexing and double window indexing. Here we briefly demonstrate the 2 usage scenarios:

A. *Single window indexing:*

Single window indexing mode is reached by simply configuring the size of inner window and outer window to be equal, i.e. *ISIZE=MSIZE*. In this circumstance the inner indexing window couldn't slide anymore, and the instruction "S++" is invalid. Figure 3 shows an example where *ISIZE=MSIZE*=3. The size of inner window and outer window are both 3, and the whole SIRF acts as a circular buffer with 3 registers.

B. *Double window indexing:*

By setting *ISIZE<MSIZE*, SIRF's inner circular window could slide inside the outer window. In this scenario, the instruction option "I++" increases the current register index ID in the range of inner window, which is the same as single window mode. The option "S++" moves the position of inner window and keeps it inside the outer window. The current index number points to the start position of the inner window. Figure 4 shows an example where *ISIZE*=2, and *MSIZE*=3. The inner window slides inside the outer windows with option "S++", keeping the register index grows under the double circular window range.

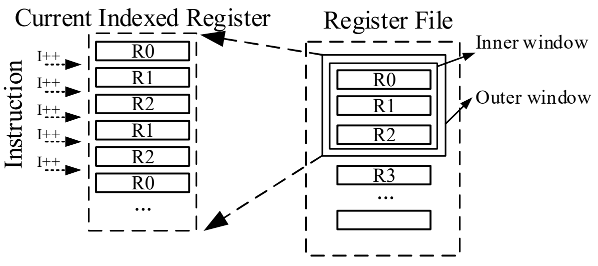

Figure 3. Example of single window indexing (*ISIZE*=3, *MSIZE*=3)

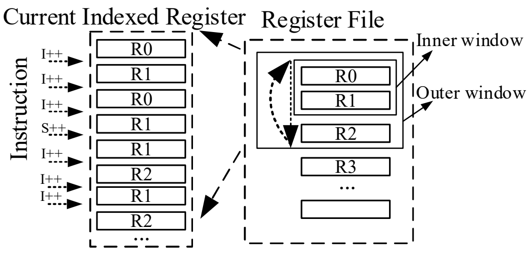

Figure 4. Example of double window indexing (*ISIZE*=2, *MSIZE*=3)

IV. EXAMPLES OF SIRF IN DSP/MEDIA APPLICATION

A. *FIR*

Finite impulse response (FIR) filters are used in many digital signal processing applications, as they are known to have some very desirable properties like guaranteed stability and linear phase. Here we consider a causal discrete-time FIR filter of order N:

$$\begin{aligned} y[n] &= b_0 x[n] + b_1 x[n-1] + \cdots + b_N x[n-N] \\ &= \sum_{i=0}^{N} b_i \cdot x[n-i] \end{aligned} \tag{4}$$

In equation (4), x[n] is the input signal and y[n] is the output signal. *N* is the filter order and b_i is the value of the impulse response at the *i*th instant for $0 \le i \le N$ of an *N*th-order FIR filter. FIR filters with exactly linear phase can easily be designed, however, they are data-dense and actually require considerably huge amount of arithmetical operations and hardware components like multipliers and adders [11]. What's more, the computing kernel usually exchanges huge input/output data, and proper memory system design is essential for reducing delay caused by exchanging data.

More than simple circular buffer, we configure SIRF as a double data window to accelerate the algorithm to fit an ASIC-like implementation based on a vector processor. In our implementation, the basic operation data length is *VS*, and for each vector dot production we need a filter and an input vector with length of *M*. For the vector machine, the *VS* concurrent dot production need *M-1+VS* input elements. So we use *M-1+VS* registers in SIRF as the buffer for the core FIR computation.

The whole implementation includes 2 procedures according to the buffer behavior implemented by SIRF: *initializing buffer* and *updating buffer*. For the input vector X(n), *initializing buffer* loads the first *M-1+VS* elements from X(n) into SIRF, and does the first *VS* dot production, as shown in figure 5.

Step 1 Load *K* sets of vector (vector length is *VS*) and load all the filter coefficients.

Step 2 Dot product the *i*th element of coefficients and the [*i, VS+i-1*]th vector in the SIRF buffer to generate *M* results. Then accumulate the *M* output results to get the first filter results.

Figure 5. Algorithm flow for initializing buffer

After loading the first *M-1+VS* elements, the second procedure updates the buffer with 1 vector each time (*VS* elements) and finishes the rest of the algorithm, as shown in figure 6.

Step 1 Load *VS* elements from X(n) to SIRF to update the *VS* registers.

Step 2 Dot product the *i*th element of coefficients and the [*i, VS+i-1*]th vector in the SIRF buffer to generate *M* results. Then accumulate the *M* output data to get the first filtering results.

Step 3 Repeat step 2 until all the elements of X(n) are updated.

Figure 6. Algorithm flow for updating buffer

The data flow in *updating buffer* is demonstrated in figure 7, where a set of vectors consisted of *VS* source data are generated by SIRF and are multiplied and accumulated to generated filtering results. In our implementation, 5 SIRF registers are used as buffer. The configuration of SIRF is: *ISIZE=4*, *MSIZE=5*. In the first procedure, 5 vectors are loaded into SIRF, then in the following procedure only 1 vector is loaded to update SIRF each time. The register updating is demonstrated in figure 8. In the figure, 4 registers within inner data window are accessed circularly with the instruction "I++". When the instruction "S++" is sent, the inner window grows by 1 register and the new registers keep the updating for the algorithm. Here, the SIRF usage is a typical double-circular circular window configuration for DSP algorithm acceleration.

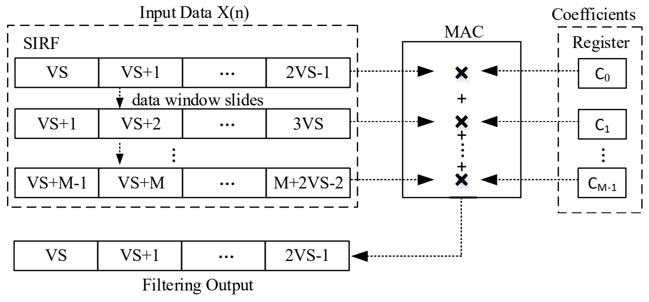

Figure 7. Demonstration of SIRF usage in FIR. Several sets of *VS* elements are provided by SIRF to do dot production to generate one FIR result.

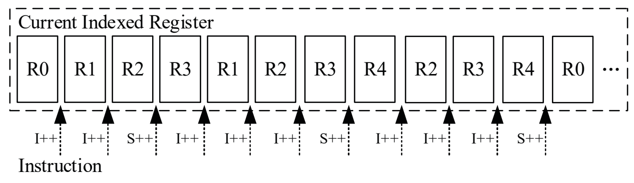

Figure 8. The updating of SIRF index for FIR

Now we analyze the number of memory access operation in our implementation. In the initializing buffer step, the memory access operation exists in three parts: *(M+VS-1)/VS* times of loading input data, *M/VS* times of loading filter coefficient, and 1 time of writing back result. In the updating buffer procedure, there are *(N-VS)/VS* times of loading input data and *(N-VS)/VS* times of writing back result. So the overall memory access operation counts as:

$$(M-1)/VS+M/VS+2+2*(N-VS)/VS \approx 2(M+N)/VS \tag{5}$$

The original algorithm requires a set of coefficient and an input data vector for each dot production, the length of which are both *M*. This requires *2(M/VS)* times of memory access operation. For input vector X(n) with length of *N*, the total counts are *2N(M/VS)*. The writing back result requires *(N/VS)* times of writing memory operations. So the overall memory access counts as:

$$2N(M/VS)+ \left(N/VS\right) \tag{6}$$

So, the memory access count complexity's speed-up is evaluated by equation (5) times equation (6), and it is about O(*MN/(M+N)*). According to this result, the speed-up grows rapidly with the increasing of computing scale.

B. Image Polyphase Filtering Interpolation

Image interpolation is a procedure used in expanding and contracting digital images [12]. It relates to the method of placing new pixel values into a regularly sampled grid given a discrete subset of points taken from a smaller grid. Polyphase interpolation method has variable filters for different interpolation requirements so as to offering better image resizing quality, and it is commonly used in industry [13].

A full image polyphase interpolation could be divided into 2 steps: interpolating in vertical direction and in horizontal direction. The filter adapts to Lanczos-n function [14]:

$$f(x)=\sin c(x)\cdot \sin c(x/n) \tag{7}$$

For the pixel *X(i,j)* in the source image, we mark the vertical interpolating pixel *Y(i,j)*, and the final target pixel *Z(i,j)*. First, the image interpolates in vertical direction. The corresponding position of target pixel *Y(x,j)* in the original image is *(i+u, j)*, which is determined by the vertical zoom ration of target image *Y* and original image *X*. The phase *u* is determined by the vertical filter, where $\lfloor x \rfloor$ is the integral part of *x*. Then the vertical interpolating target image *Y(x,j)* is calculated as:

$$Y(x,j)=f(X, Vfilter(uPhase, Vtap))$$
$$=\sum_{index=0}^{Vtap-1} Vfilter(uPhase, index)*X(i-Vtap/2+index+1, j) \tag{8}$$

Here, to simplify the demonstration, we use a simple 1:2 image scaling example to show the SIRF accelerating mechanism. We use a 4 tap filter, and for each target pixel, there involves the multiplication and accumulation of 4 sets of source pixels and coefficients:

$$for\left(i=0; i<RowNum;\ i++\right)\ \{nY[i]=\sum_{m=0}^{3} C_m * nX_m[i]\}$$

The input data updating window is shown in figure 9. In this demonstration of 4 tap filtering, for each target interpolating pixel there are 4 source pixels required, and the source data update after generating 2 target pixels. The updating pattern still has a FIR-circular window style, and 1 pixel is replaced into the temporary buffer to form new data window for the interpolation. The figure shows the updating of source data and all target data are automatically prepared with the SIRF writing/reading windows.

Instead of fetching 4 pixels from main memory, only 1 pixel is needed for the updating of SIRF, and the indexes for each source operator and destination operator are all updated properly by the self-index system. As a result, the complexity of memory fetching operation becomes O(*N*) because of SIRF, instead of O($4\times N$) generated from the original algorithm process. As for the interpolation with more taps (like 8 tap and 16 tap), SIRF brings bigger acceleration with the increasing of algorithm complexity. In the task where image is interpolated from $Nh\times Nv$ to $Lh\times Lv$ with N tap filter, the memory access speed up for one interpolating direction is $(Lh\times N\times LV) / (Lh\times Lv) = N$. As for the full size interpolating that contains the same 2 procedures, the overall speed-up is $N+N=2N$. That means the interpolation with SIRF has no relationship with algorithm's tap number, and it accesses memory with fixed number for certain image resolution, so as to bringing advantage for ASIC-like image processing hardware design.

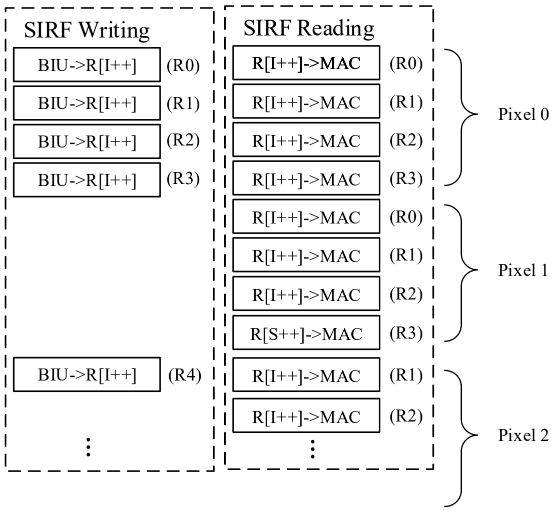

Figure 9. SIRF usage demonstration in 4 tap image interpolation, where 1 register is fetched each time to automatically generate source data consisted of 4 registers.

V. Evaluation

A. *Effect on overall implementation performance*

This section discusses SIRF's effect on overall performance. Besides FIR and image interpolation mentioned before, we also analyze another two typical DSP algorithms: floating point matrix multiplication (FMM) and FFT (including single-point complex FFT and fixed-point complex FFT). Each algorithm uses 3 groups of input size, as listed in table 1, and average results are obtained for performance comparison. All these algorithms are implemented on the mathematical processor (MaPU) and are described in the original MaPU benchmarks in [15]. To get precise instruction statistics, the memory access related instruction counts are generated from a cycle accurate simulator [16], and the correctness of these benchmarks has been verified in hardware experiments.

The memory access instruction ratio (average result from every 3 tests with different input size) with/without SIRF is shown in figure 10. The cycle accurate simulator gives the overall instruction number and the memory-related instruction number fired in the whole algorithm execution stage. In the cases without SIRF, the memory access operations related to SIRF's register are replaced by ordinary load/store operations that interact directly with main memory. The results of all the algorithms show reduction of memory access instruction, as shown in figure 11. As for the 2 FFT groups the memory access instructions are reduced to about half as before. Another 3 groups show similar or better reduction effect. Especially, the image interpolation experiment reaches about 0.77 memory accessing reduction, as image's pixels are frequently reused in the multiple-tap interpolating process, and SIRF takes good advantage of that to reduce frequently memory access.

TABLE I. Test Case List

Algorithm	Size 1	Size 2	Size 3
FIR	N=1024, M=64	N=2048, M=128	N=4096, M=128
Image Interpolation	64×64, 4 tap	240×135, 4 tap	240×135, 16 tap
FMM	64×64	128×128	256×256
Cplx SP FFT	256 points, 3 epochs	1024 points, 4 epochs	4096 points, 4 epochs
Cplx FP FFT	256 points, 3 epochs	1024 points, 4 epochs	4096 points, 4 epochs

Here we give a glimpse of the acceleration of SIRF on the speed-up on the overall performance of tested benchmarks. We measure the sustained performance by Giga Operations Per Second (GOPS). As shown in figure 11, speed-ups are observed in all the tested benchmarks, ranging from 1.29 to 2.88. As a matter of fact, all the algorithms are implemented with great optimization and having quite good efficiency considering the hardware resources of MaPU. The usage of SIRF brings in additional acceleration and it acts as an important design principle in final high-performance implementations.

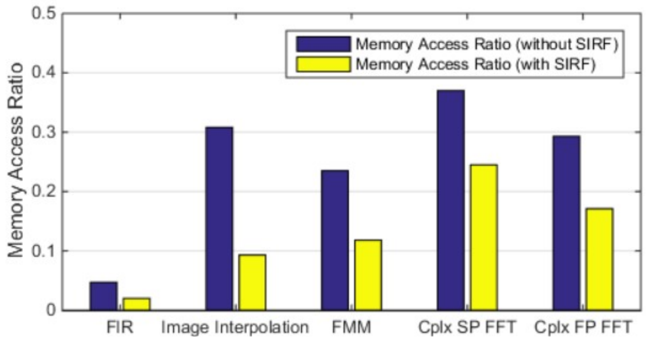

Figure 10. Average memory access ratio, caculated by the number of memory access instruction divide total instruction number

The main advantage is that SIRF's operation is much quicker than load/store operations, as SIRF is an inside-processor resources and the register operating has similar or higher speed comparing with other arithmetical units inside processor, while load/store operations are limited by the speed of main memory and the latency between main memory and processor. Furthermore, the self-indexer takes good use of mathematical algorithm's data pattern, and saves the heavy burden of register address's computing, giving performance increasing for the computing tasks.

B. *Power efficiency*

SIRF brings in another advantage: the reduction of power consumption, as it usually reduces the ratio of memory access operations (load/store operations), and the energy consumed by the register operating instructions in SIRF is much lower than that of memory access operations. The dynamic power consumption of each function units' instructions is shown in table 2 [15]. In the table, FALU, IALU, FMAC and IMAC are all different kinds of arithmetical units in the mathematical processor, and load/store instructions relate to memory access operations.

TABLE II. DYNAMIC ENERGY CONSUMED PER INSTRUCTION [15]

Instruction Type	Average Energy Per Instruction(Unit: pJ, 512bit)
Register R/W	133.25
Load/Store	609.2
FALU	345.65
IALU	335.18
FMAC	187.23
IMAC	788.77

The power consumption of SIRF operations are much lower than any other arithmetical operations, and compering with load/store operations, it consumes only about 1/5 power. The power reduction results are shown in figure 11. It shows 15%-20% power reduction brought by SIRF in all the five test cases. In these experiments, SIRF helps the processor to have better use of data locality and reduces the redundant memory access, so as to reducing the extra power consumption of load/store operations.

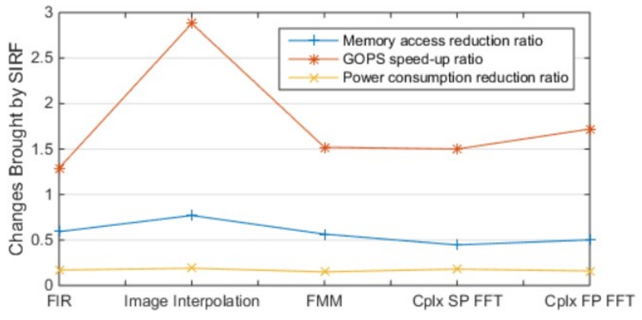

Figure 11. Changes brought by SIRF, including memory access reduction , overall performance speed-up and power consumption reduction

VI. CONCLUSION AND FUTURE WORK

This paper presents a novel self-indexed register file specially designed for DSP/media applications. The registers inside SIRF could be accessed via immediate register ID or auto-generated index. The indexer with two circular windows are designed and optimized based on the computing pattern of some frequently-used computing intensive algorithms, aiming to reduce memory access operations and increase functional units' usage ratio. Two application examples are demonstrated to show the design principle and how it can accelerate the applications. For the targeted DSP/media algorithms that have intensive computing requirements, SIRF can provide consecutive sets of data with high speed and low latency, thus improve the overall computing efficiency.

The basic design principle of SIRF is to explore the data locality in register file and reduce main memory access. Thus it is applicable to any other DSP processors or application specific integrated circuits that targeting data intensive kernels, like convolution and matrix multiplication, etc. Applications that based on these kernels, like communication, image processing and neural network training, can leverage these highly optimized kernels to boost the performance with limited hardware and reduced power consumption.

Future work: Though SIRF demonstrates good performance in the evaluated applications, we believe it can be extended to other data intensive tasks, like matrix multiplication with different size, and some key computing kernels in 4G/5G communication. In addition, to improve the programmability, we are also investigating the complier support, which can automatically generate SIRF configuration from high level language for the targeted algorithm.

REFERENCES

[1] Tippetts, B., et al., Review of stereo vision algorithms and their suitability for resource-limited systems. Journal of Real-Time Image Processing, 2016. 11(1): p. 5-25.

[2] Moini, S., et al., A Resource-Limited Hardware Accelerator for Convolutional Neural Networks in Embedded Vision Applications. IEEE Transactions on Circuits and Systems II: Express Briefs, 2017.

[3] Eyre, J. and J. Bier, The evolution of DSP processors. IEEE Signal Processing Magazine, 2000. 17(2): p. 43-51.

[4] Backus, J., Can programming be liberated from the von Neumann style?: a functional style and its algebra of programs. Communications of the ACM, 1978. 21(8): p. 613-641.

[5] Smith, S.W., The scientist and engineer's guide to digital signal processing. 1997.

[6] Keitel-Schulz, D. and N. Wehn, Embedded DRAM development: Technology, physical design, and application issues. IEEE Design & Test of Computers, 2001. 18(3): p. 7-15.

[7] Chang, P.-Y., et al., A 4R/2W register file design for UDVS microprocessors in 65-nm CMOS. IEEE Transactions on Circuits and Systems II: Express Briefs, 2012. 59(12): p. 908-912.

[8] Chu, S.-L., C.-C. Hsiao, and C.-C. Hsieh. An energy-efficient unified register file for mobile GPUs. in Embedded and Ubiquitous Computing (EUC), 2011 IFIP 9th International Conference on. 2011. IEEE.

[9] Gebhart, M., et al., A hierarchical thread scheduler and register file for energy-efficient throughput processors. ACM Transactions on Computer Systems (TOCS), 2012. 30(2): p. 8.

[10] Jing, N., et al. An energy-efficient and scalable eDRAM-based register file architecture for GPGPU. in ACM SIGARCH Computer Architecture News. 2013. ACM.

[11] Liu, Y. and K.K. Parhi. Lattice FIR digital filter architectures using stochastic computing. in Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on. 2015. IEEE.

[12] Prashanth, H., H. Shashidhara, and B.M. KN. Image scaling comparison using universal image quality index. in Advances in Computing, Control, & Telecommunication Technologies, 2009. ACT'09. International Conference on. 2009. IEEE.

[13] Franzen, O., C. Tuschen, and H. Schröder. Intermediate image interpolation using polyphase weighted median filters. in Proc. SPIE. 2001.

[14] Rasti, P., et al. Improved interpolation kernels for super resolution algorithms. in Image Processing Theory Tools and Applications (IPTA), 2016 6th International Conference on. 2016. IEEE.

[15] Wang, D., et al. MaPU: A novel mathematical computing architecture. in High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on. 2016. IEEE.

[16] Yang, L., et al., An approach to build cycle accurate full system VLIW simulation platform. Simulation Modelling Practice and Theory, 2016. 67: p. 14-28.