On-chip Adaptive Circuits for Fast Media Processing

Rama Sangireddy, *Member, IEEE* and Arun K. Somani, *Fellow, IEEE*

***Abstract*— Applications depending on their nature demand either higher computing capacity or larger data storage capacity or both. Hence, providing on-chip memory and computing resources that are fixed in nature is expensive and does not enable an efficient utilization of on-chip silicon real estate. In this paper, we design the circuit of an *Adaptive Register File Computing (ARC)* unit, a novel on-chip *dual-role* circuit with a minimal area overhead of 0.233mm^2 at 0.18μ technology. It supplements the conventional register bank to provide larger register storage capacity, or acts as a specialized computing unit to provide higher on-chip computing capacity, depending on the requirement of a specific application. The paper discusses the circuit level details for the implementation of the dual-role ARC unit, its integration in a wide-issue processor pipeline and the corresponding performance enhancement in various multimedia applications.**

I. Introduction

Most current day applications like multimedia and digital signal processing applications demand higher computing power. The widely known 90-10 rule predicates that 90% of the execution time is expended by about 10% of the application code which is compute-intensive. Spatial structures (specialized computing unit designed for a specific function) excel in the execution of such compute-intensive functions as compared to temporal structures (such as a general purpose processor). However, allocation of additional budget of silicon real-estate on the chip to increase the computing power or for an increase in the on-chip data storage capacity, irrespective of the characteristics of an application being executed will not always enhance the performance. Instead, the additional silicon resources on the chip can be utilized more adaptively for enhancing either computing power or the storage capacity, as the application demands.

Based on the above observations, we have designed an *Adaptive Register file Computing* (ARC) unit at the circuit level. The unit strives towards achieving greater performance by providing a larger register storage capacity or higher on-chip computing power, depending on the requirements of an application. One of the significant contributions of the paper is the design of a dual-role on-chip component that can be a register bank or a specialized computing unit that accelerates matrix operations, and the investigation of the hypothesis that different applications need different amounts of on-chip memory capacity and computing power based on their characteristics. The adaptive unit can adapt to either role based on the needs of the application, and enhances the performance of various multimedia applications.

A. Related Research

Genov $et\ al$ [1] had presented a digital architecture for parallel vector-matrix multiplication with a die size overhead of $9mm^2$ at 0.5μ CMOS technology. Shen-Fu$et\ al$ [2] had proposed recursive algorithms for DFT computation to reduce the number of multipliers during matrix multiplication implementation, with a core area overhead of 3791x3828μm^2 for a 64-pt radix-4 DFT processor. On the other hand, our proposed design implements a dual-role component with only an area of 0.233mm^2 at 0.18μ, for enhancing performance of various multimedia applications.

R. Sangireddy is with the department of Electrical Engineering, University of Texas at Dallas, USA.

A. K. Somani is with the department of Electrical and Computer Engineering, Iowa State University, Ames, IA, USA.

Earlier, the first known attempts to use the memory elements for computation are that of by Kautz [3] and Stone [4]. During the past decade, attention has been drawn towards the significance of designing a reconfigurable coprocessor coupled with the general purpose processor [5]. Ye et al [6] developed *Chimaera*, a micro-architecture that integrates a reconfigurable functional unit (RFU) into the pipeline of a dynamically scheduled superscalar processor. Razdan $et\ al$ [7] explored ways to incorporate hardware-programmable resources and described compilation/synthesis system that automatically exploits the resources to improve the performance of general purpose applications. The Garp architecture [8] combines reconfigurable hardware with a standard MIPS processor on the same die to exploit the better features of both. A *Reconfigurable Computing Cache* (RFC) based architectures were proposed for low-power media processing in [9].

The proposed architecture in this paper is different from these reconfigurable computing architectures in that, all these architectures focus on the design of reconfigurable computing elements that can process a multiple set of functions according to the needs of applications. In this paper we have proposed and designed at circuit-level, a dual-role on-chip component that can be a register bank or a specialized computing unit depending on the needs of the application. Further, the register file architecture and its organization has been widely researched to enhance the performance of wide-issue processors. However, this paper is the first work to the best of our knowledge, that proposes a dual-role register organization that can enhance the on-chip register storage capacity or provide higher on-chip computing bandwidth, as demanded by an an application.

The rest of the paper is organized as follows. Section II presents the design and implementation of the ARC unit. Section III presents the performance analysis of a wide-issue processor supplemented with an ARC unit. Section IV concludes the discussion.

II. Design of ARC Unit

The development of the proposed architecture first involved the design of a dual bank register file and suitable schemes for logical to physical register mapping in such design. In the dual register bank organization, shown in Figure 1, the RF1 register bank acts as a conventional register file bank that supplies the operand values to the functional units. The RF2 bank is designed as an ARC unit, to act as an additional bank of registers, or as a specialized computing unit. When the RF2 acts as a register bank in tandem with the RF1 bank, the physical registers in RF1 are used for logical to physical register mapping at dispatch stage. Results from the functional units are always written to registers in RF1. The register values in RF1 are written to RF2 whenever RF2 has free registers and thus simultaneously freeing the corresponding registers in RF1. Thus, the register in RF1 is freed up much earlier than that is done in a conventional monolithic register file. Subsequently the freed register in RF1 joins the free pool of registers used to map to subsequent logical destination registers. This leverages the processor to process a larger number of in-flight instructions to draw higher instruction level parallelism (ILP). A register in RF2 is freed according to the conditions followed in the case of a conventional monolithic register file. That is, for a current logical to physical register mapping, the physical register in RF2 is freed when a subsequent instruction with same logical destination commits. The freed register in RF2 is again ready to assume another register value from RF1. The mode of operation of the architecture when RF2 acts as a register bank in tandem with the RF1 bank is

further discussed in detail in [10]. In this paper, we focus on the next step that involves the identification and design of a suitable compute-intensive function to fit into the ARC unit.

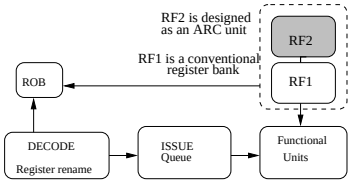

Fig. 1. ARC unit placement in the processor pipeline.

A. ARC as a Computing Unit

The *Adaptive Register file Computing* (ARC) unit is designed to spatially process a compute-intensive function to accelerate the application processing. In order for such an architecture to be effective, both the compute-intensive function and the ARC unit must possess certain desirable characteristics as follows.

- The compute-intensive function should be generic and widely used in most applications. Else, the on-chip hardware resources allocated for computing such a function will be under-utilized.
- Design and implementation of such a function should provide reasonable speedups over a general purpose processor.
- The time to load the configuration (configuration overhead) prior to the computations in the ARC unit should be minimal for it not to become a bottleneck.
- The overall minimal hardware complexity in the circuit should be such that the access time of the ARC unit is low.

For the above reasons, the circuit-level design and implementation of the ARC as a computing unit is a challenging task.

Areas such as signal processing and imaging require enormous computing power, and thus more on-chip computing resources. A dissection of the algorithms used in these, and related applications, reveal that many of the fundamental actions involve matrix operations. Most of these operations are matrix multiplications, which are frequently occurring operations in a wide variety of real world algorithms. The Discrete Cosine Transform (DCT), the Discrete Fourier Transform (DFT), and Singular Values Decomposition (SVD), used in digital image/signal processing including compression and beamforming applications are some of those applications [11], [12]. The multiplication of two matrices of size NxN each, in a general purpose processor consumes $\mathcal{O}(N^3)$ operations, requiring $\mathcal{O}(N^3)$ addition and $\mathcal{O}(N^3)$ integer multiplication operations (considering the elements in the matrix to be integers), and hence becomes a bottleneck to the performance of the processor. Therefore, larger number of on-chip processing elements that compute in parallel are required.

Subsequently, we choose the integer matrix multiplication function as a compute-intensive function to be implemented in the ARC unit. Consider two NxN matrices A = $[A_{ij}]$ and B = $[B_{ij}]$. The product C = $[C_{ij}]$ of the two matrices is given by

$$C = \quad A * B \tag{1}$$

such that

$$C_{ij} = \sum_{k=0}^{N-1} A_{ik} B_{kj} \tag{2}$$

To reduce the complexity of the matrix multiplication operations, we design a 3-LUT based computation, wherein the time to configure the LUTs would be a maximum of eight clock cycles. The design of the ARC unit is shown in Figure 2. It consists of eleven 3-LUTs addressed by the 32-bit integer multiplicand (the sign bit is extended by one bit to make it a 33-bit integer). The RF2 register bank has 128 registers, same as the RF1 bank. But, in this case only 88 of those 128 registers in RF2 are used for computing as shown in the Figure 2. The addition of the partial products (PPs) obtained from the LUTs is performed by multiple stage carry save adders (CSA).

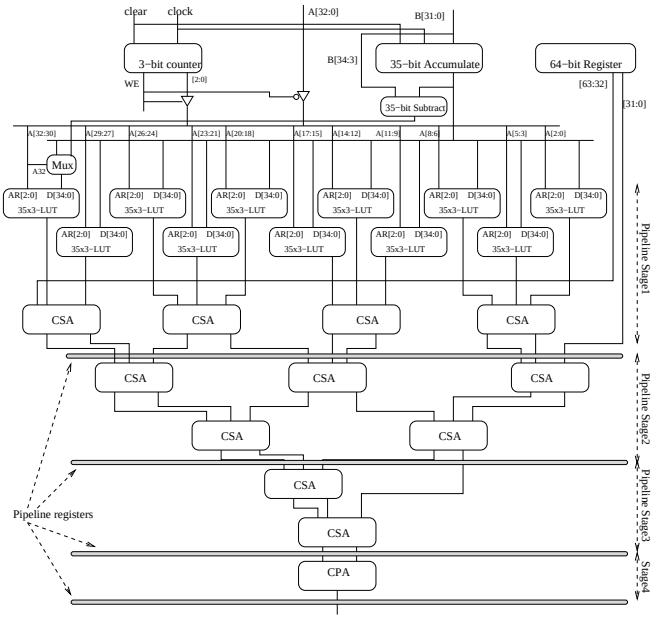

Fig. 2. 32-bit 8-cycle Reconfigurable Matrix Multiplier. The write enable (WE) and clock signals are connected (not shown in figure) to all the LUTs. The multi-stage addition of partial products using CSAs is pipelined as shown. The end result is computed in a carry propagate adder (CPA).

The matrix multiplication operation in the ARC unit consists of two stages - configuration and computation. For this purpose we introduce two new instructions - ARC-CONF with one operand A, and ARC-COMP with three operands C, B, and C ($dest$, $source$, $source$). The instruction (ARC-CONF A) when invoked, performs a write operation for eight clock cycles controlled by a 3-bit counter. The eight rows in each of the ten right-most 3-LUTs are loaded with the values of 0, A, $2A$, $3A$, $4A$, $5A$, $6A$, $7A$, respectively, as addressed by the output from the 3-bit counter. Simultaneously, the eight rows in the left-most 3-LUT are configured with 0, A, $2A$, $3A$, $-4A$, $-3A$, $-2A$, $-A$, respectively, to facilitate a signed integer multiplication operation. Since we assume a single LUT cannot be loaded in parallel, we partially load all the LUTs each cycle. Thus all the LUTs in the ARC unit are configured in parallel and hence the total configuration time for each element of matrix A is eight clock cycles.

When the configuration of the LUTs is completed, the instruction (ARC-COMP C, B, C) is invoked. The operation consumes two source operands, wherein the first source operand is used to lookup the contents of the LUTs, while the second source operand is an intermediate result obtained from the earlier ARC-COMP operations. The eleven partial products obtained from the current lookup operation are combined with the two 32-bit parts of the intermediate result (it can be seen in Figure 2 that a proper bit-alignment is made to perform the addition of the thirteen partial products, i.e., the lower 32-bit value of the intermediate result is added to the least significant partial product, while the higher 32-bit value of the intermediate result is added to the most significant partial product). Subsequently, the addition of the thirteen PPs is performed in five stages of addition using CSAs and the result obtained thereafter is stored in the designated destination operand.

The addition of PPs using CSAs and a carry propagate adder (CPA) is pipelined as shown in the Figure 2. The first stage constitutes the lookup operation and first level of CSA computation. Subsequently, four levels of CSA computations are placed in two stages (two CSA levels in each stage). Due to the higher computation latency in a CPA as compared to a CSA, one CPA computation is placed in a single

stage. The pseudo code for a multiplication of two NxN matrices and a sequence of instructions generated for the multiplication of two 2x2 matrices in the ARC unit is shown in Figure 3.

Algorithm 1: **Matrix multiplication in ARC**

```
for i=0 to (N-1)
    for j=0 to (N-1)
    ARC-CONF A_ij
        for k=0 to (N-1)
        ARC-COMP C_ik, B_jk, C_ik

for N=2, the sequence of operations for execution in ARC:
    Initialize C_00 = C_10 = C_01 = C_11 = 0;
    ARC-CONF A_00
    ARC-COMP C_00, B_00, C_00
    ARC-COMP C_01, B_01, C_01
    ARC-CONF A_01
    ARC-COMP C_00, B_10, C_00
    ARC-COMP C_01, B_11, C_01
    ARC-CONF A_10
    ARC-COMP C_10, B_00, C_10
    ARC-COMP C_11, B_01, C_11
    ARC-CONF A_11
    ARC-COMP C_10, B_10, C_10
    ARC-COMP C_11, B_11, C_11
```

Fig. 3. Matrix multiplication in a ARC unit (shown excluding load/store and branch instructions).

Alternatively, the lookup operation can be performed using 4-LUTs. In that case, the addition of 10 PPs (eight from lookup and two from earlier intermediate result) still requires five stages of CSAs and hence results in same pipeline latency. However, the total number of memory elements (registers) in the ARC unit required for computing purposes would be 128, as compared to 88 in the 3-LUT based design, and the lookup time for a 4-LUT is slightly higher than that for the 3-LUT. Thus, the 4-LUT based design results in a larger area overhead and with a slightly higher computation latency. Hence we prefer a 3-LUT based design for the implementation. Similarly, the lookup operations can be performed using 5-LUTs wherein the addition of nine PPs (seven from lookup and two from earlier intermediate result) can be performed in a four-stage CSAs. However, the lookup time for the 5-LUT is much larger and offsets the advantage of reduced stages in addition. Further, the total number of registers required in RF2 for such operation is 224, which results in a very large area overhead on the chip.

The implementation of the ARC unit can further be modified to reduce the configuration time to 4 and 2 clock cycles, in line with the design of a *self configuring binary multiplier* proposed by Wojko and ElGindy [13]. The lookup operation with a configuration time of 4 cycles is shown in Figure 4. A 3-LUT is divided into two segments and the two segments of each 3-LUT are loaded with the respective contents in parallel. This is achieved with an additional 35-bit adder that adds a value of $4A$ to each of the consecutive outputs from the 34-bit accumulate. Thus it takes 4 clock cycles to load the values of 0, A, $2A$, and $3A$ into the rows in first segment, and simultaneously write the values of $4A$, $5A$, $6A$, and $7A$ into the rows of second segment. Among the three bits to perform the lookup in a 3-LUT, the two least significant bits (LSBs) are used for address lookup while the most significant bit (MSB) acts as a select signal for the multiplexer to choose either of the outputs from the two segments. Note that this is similar to a design where sixteen 2-LUTs are used for the lookup operation (with a total of 64 memory elements). However, in the 2-LUT implementation, the addition of 18 PPs (16 from lookup and 2 from earlier intermediate result) requires six stages of CSAs and hence results in a larger computation latency.

The lookup operation with a configuration time of 2 cycles is performed with the further segmentation of the 3-LUT, however with a larger area overhead due to the requirement of additional adders and decoders. The partial implementation of the design is shown in Figure 5. The total on-chip area consumed by the ARC unit, designed

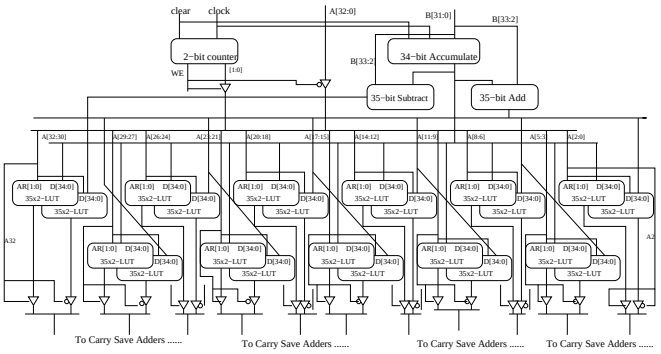

Fig. 4. 32-bit 4-cycle Reconfigurable Matrix Multiplier. The write enable (WE) and clock signals are connected (not shown in figure) to all the LUTs. The pipelined addition of partial products using CSAs is performed as shown in Figure 2.

according to the above described three different configuration schemes and implemented at 0.18μ technology, is shown in Table I. The three designs have been implemented using Verilog and the hardware synthesis to measure the area is performed using the standard design analyzer tools from Synopsys [14]. The amount of area overhead due to the integration of ARC unit in a conventional processor is around 0.1% of the total chip area (Intel Pentium 4 die size of $217mm^2$ or AMD Opteron die size of $193mm^2$ in 2004), and it will be around 0.2% in a MIPS32 Intrinsity FastMATH embedded processor (die size of $122mm^2$). To make an evenhanded comparison of our design with the architectures already implementing a large register file (with same number of registers as the combination of main register bank and the ARC unit), we measure the overhead of the area only due to additional logic (CSAs, Add/Sub units and routing) required to perform matrix multiplication computation. The corresponding results are shown in the third column of the table, which on average is $0.03206mm^2$. The table shows that the area overhead we incur, due to the added logic to the register file for performing matrix multiplication, is significantly less. On the other hand, to have a separate matrix multiplier on the chip in addition to the full register file capacity, it consumes considerable amount of area overhead depending on the implementation [1], [2].

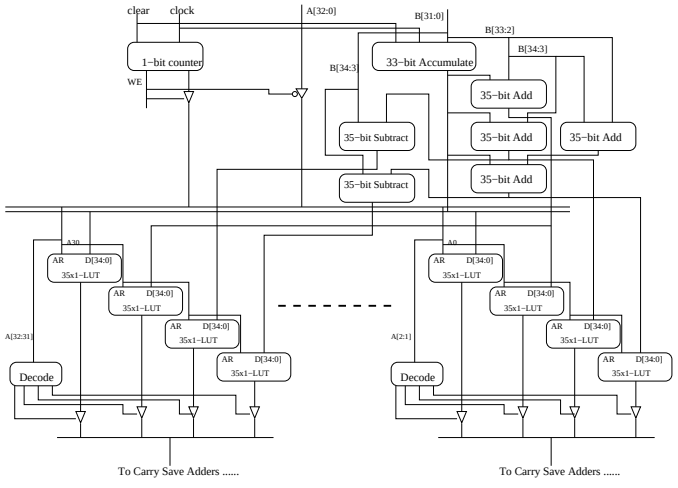

Fig. 5. 32-bit 2-cycle Reconfigurable Matrix Multiplier (partially shown). The write enable (WE) and clock signals are connected (not shown in figure) to all the LUTs. The pipelined addition of partial products using CSAs is performed as shown in Figure 2.

B. ARC unit as a register bank

A lookup-table (LUT) is a segment of SRAM, e.g. a 3-LUT is an SRAM logic with a bitline width of eight cells. The width of the

TABLE I

ON-CHIP AREA FOR ARC DESIGN USING 3-LUTS, IMPLEMENTED AT 0.18μ TECHNOLOGY. AREA OVERHEAD IS ARC UNIT AREA EXCLUDING LUTS (REGISTERS).

$Config.\ time$	$ARC\ area$ (mm^2)	$Area\ overhead$ (mm^2)
$2\ cycles$	0.233	0.04241
$4\ cycles$	0.199	0.03206
$8\ cycles$	0.178	0.02622

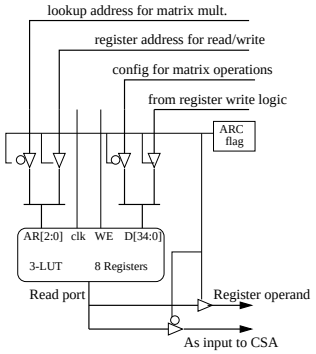

Fig. 6. A register sub-bank in the ARC unit.

wordline can be designed according to the functional requirements. A wordline in an LUT is read or written by selecting the wordline using a decoder logic, similar to the implementation of read and write accesses in on-chip memories. When the ARC unit acts a register bank, it can be considered to be consisting of eleven register sub-banks (corresponding to eleven 3-LUTs), each sub-bank comprising of eight registers. Though each sub-bank consists of a read and a write port, the ports can be concatenated to form the required number of read and write ports for the RF2 bank.

The design of a register sub-bank is shown in Figure 6. The flag sets the ARC unit into the register file mode or the computing mode. According to the flag setting, the address and data values appropriate for the current operation are selected. The access time for a register, i.e., to read a value from one memory bank, for each of the designs is shown in Table II. The access time values have been computed at 0.18μ technology, using a register file access time model derived from CACTI [15]. It can be observed that for an implementation with maximum possible configuration cycle time (for a 3-LUT based design it is 8 cycles, for a 4-LUT based design it is 16 cycles, for a 5-LUT based design it is 32 cycles), the register access time is slightly less than a corresponding design with reduced configuration cycle time. This is mainly due to the absence of a decoder and multiplexing logic to select one of the outputs from the segments of an LUT.

III. PERFORMANCE ANALYSIS

We used Simplescalar-3.0 [16] to simulate a dynamically scheduled wide-issue processor with the simulation parameters summarized in Table III. The configurations for three different processor organizations used for the analysis purposes are as shown in Table IV. Configuration C1 is a base processor without an ARC unit. Configurations C2 and C3 are the base processor with an embedded ARC unit. The performance of the processor with ARC unit acting as RF2 register bank is analyzed in the configuration C2. Similarly, performance of the processor with ARC unit acting as a matrix multiplication unit is analyzed in the configuration C3. For the C1 and C2 configurations, matrix multiplication operation is performed in a conventional wide-issue processor with varied register file sizes and register access times as shown in Table IV. The register access time for C2 configuration is taken to be one cycle more as compared to the base processor, due to the larger register file. The pipelining of the register file is not considered for analysis purposes here, though the multiple-cycled pipelined register file designs have been proposed in the recent past.

TABLE II

REGISTER ACCESS TIME IN THE ARC UNIT AT 0.18μ TECHNOLOGY.

$ARC\ config\ time$	$Register\ access\ time\ (ns)$		
	$3-LUT$	$4-LUT$	$5-LUT$
$2\ cycles$	0.4484	0.4975	0.5466
$4\ cycles$	0.4524	0.5017	0.5510
$8\ cycles$	0.3628	0.5098	0.5594
$16\ cycles$	-	0.3831	0.5756
$32\ cycles$	-	-	0.4028

TABLE III

PROCESSOR SIMULATION PARAMETERS.

$Parameter$	$Value$
$Instruction\ cache$	32KB, 2-way, 1 cycle
$Data\ cache$	32KB, 4-way, 1 cycle
$Branch\ predictor$ $-mis-prediction\ latency$	bimodal, 2K table size 7 cycles
$Instruction\ issue\ queue\ size$	128
$Load/store\ queue\ (LSQ)\ size$	64
$Pipeline\ width$	4 and 8
$Functional\ units$ $-Integer\ arithmetic$ $-Integer\ multiplier$ $-floating\ point\ arithmetic$ $-floating\ point\ multiplier$	4 and 8 2 and 4 2 and 4 2 and 4
$L2\ unified\ cache$ $-latency$	256KB, 4-way, 64B line 6 cycles
$Memory$ $-latency\ first, next$ $-bus\ width$	70, 2 cycles 8B

During the computation in the ARC unit, the time to configure the LUTs for each element of A is taken to be 4 cycles. A slightly higher speedup in computation can be obtained if a 2 cycle configuration time is used, while an insignificantly lesser speedup is obtained if an 8-cycle configuration is used. In C3 configuration, even when the register file size is same as that of the base processor, the access of each register involves an additional delay in the multiplexer placed in the dual bank register file. This multiplexer delay is not seen in the C1 configuration. Hence, in the C3 case the register access time is taken to be 2 cycles. For a comparative evaluation between the configurations C2 and C3, where C2 has larger register storage capacity while C3 has higher computing capacity, the multiplication of matrices of size 256x256 is also performed according to the *blocking algorithm* [17], [18]. Blocking is used to achieve locality in the on-chip memory capacity available. To perform matrix multiplication by blocking, we divided the 256x256 matrix into four 128x128 blocks.

The speedups obtained by computing the matrix multiplication product in the processor with an ARC unit as compared to a base processor are shown in Figures 7 and 8. For each matrix size, the total number of cycles taken to execute the function in the processors with configurations C2 and C3 is normalized with the total cycles for execution in the base processor (configuration C1). In the case of matrix multiplication by blocking, the execution times obtained for configurations C1, C2 and C3 are normalized with the execution time obtained with matrix multiplication in a conventional way in processor configuration C1. As seen in Figure 8, for multiplication of matrices with smaller sizes, the C2 and C3 configurations perform similarly and better than the base configuration.

As the matrix size is increased to 128x128, the demand for higher computing power takes precedence and hence the C3 configuration performs better. However, as the matrix size is further increased to

TABLE IV

VARIOUS PROCESSOR CONFIGURATIONS SIMULATED. C2 ARCHITECTURE IS SIMULATED AS DISCUSSED IN SECTION IV

$Index$	$Configuration$	Number of registers available	Computing bandwidth available
C1	Base processor without ARC unit	128 registers (access time 1 cycle)	8 Integer ALU and 4 Integer Multipliers
C2	Base processor with ARC unit as RF2 register bank	128+128 registers (access time 2 cycles)	same as base processor
C3	Base processor with ARC unit as computing unit	128 registers (access time 2 cycles)	ARC unit, 8 Integer ALU, and 4 Integer Multipliers

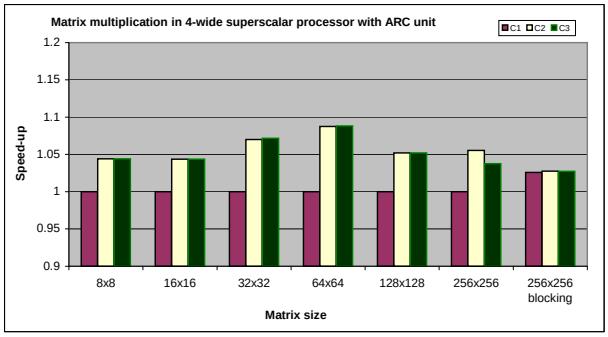

Fig. 7. Matrix multiplication in 4-wide processor without and with ARC unit

256x256, the demand for larger register capacity is more than the demand for higher computing power. Thus, in this case though C3 performs better than C1, processor configuration C2 performs even better. In the case of multiplication of matrices by blocking, the configuration C3 performs better than C2 and C2 performs better than C1. This is due to the fact that, blocking helps in utilizing the register capacity efficiently, and thus the demand for computing bandwidth is more than the demand for the register capacity.

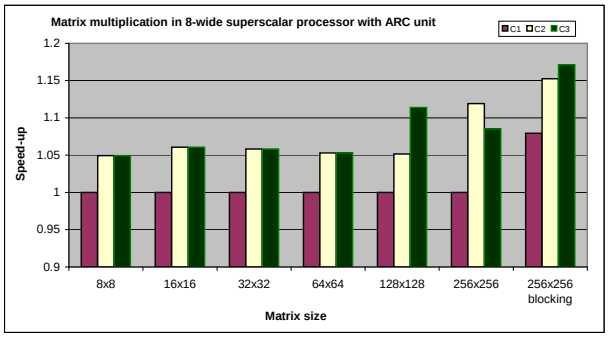

Fig. 8. Matrix multiplication in 8-wide processor without and with ARC unit.

From the above results, and especially analyzing the performance of various configurations of an 8-wide superscalar processor in executing multiplication of matrices of sizes 128x128, and 256x256 with and without blocking, it can be concluded that applications depending on their nature, demand either higher computing capacity or larger data storage capacity or both. Hence, providing on-chip memory and computing resources that are fixed in nature is expensive and does not enable an efficient utilization of on-chip silicon real estate. Instead, it is more beneficial to design a portion of on-chip resources to be reconfigurable, so that it can be used either as a memory element or a computing unit, as the situation demands.

The performance enhancement with execution of MPEG decode, JPEG compression and decompression applications in both 4-wide and 8-wide processors is shown in Figure 9. For each application, the execution time in the processor with the ARC unit is shown relative to that in the processor without the ARC unit. For each benchmark, the four columns indicate the total execution time in the base processor, time spent on core function in the base processor, execution time in the proposed architecture, and time spent on core function in the proposed architecture. The applications are run with an image of 574,000 pixels. The MPEG decode application has DCT as the compute-intensive function, the JPEG compression has FDCT as the compute-intensive function, and the JPEG decompression has IDCT as the compute-intensive function. Each of these functions contain matrix operations as the core computations that are accelerated using the ARC unit.

IV. Conclusions

Applications that demand either higher on-chip computing power or larger on-chip data storage capacity are continuously emerging. In this paper, we have proposed *Adaptive Register file* architecture,

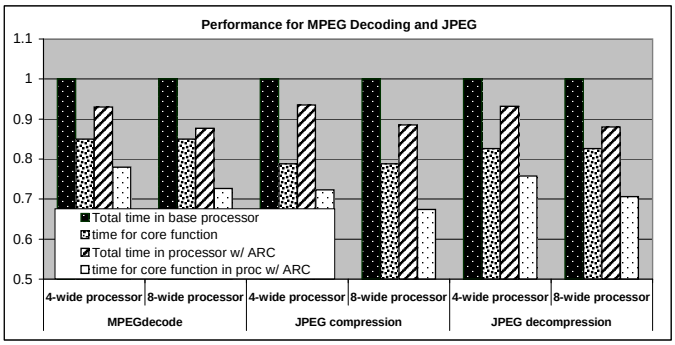

Fig. 9. Performance of MPEG decode, JPEG compression and decompression applications in 4-wide and 8-wide processors with and without ARC unit.

a novel circuit to provide a feasible solution and address the above problems concurrently. The *Adaptive Register file Computing* (ARC) unit provides a higher on-chip computing capacity by executing a compute-intensive function, and provides larger register file resources to meet the register storage capacity requirements. Results showed a considerable performance gain in various multimedia applications, when processed in a conventional wide-issue processor supplemented with the ARC unit to accelerate matrix operations.

References

[1] R. Genov and G. Cauwenberghs, "Charge-mode parallel architecture for vector-matrix multiplication", *IEEE Transactions on Circuits and Systems-II*, Volume: 48, Issue: 10, pp. 930-936, October 2001.

[2] H. Shen-Fu and S. Wei-Ren, "Design of low-cost and high-throughput linear arrays for DFT computations: algorithms, architectures, and implementations", *IEEE Transactions on Circuits and Systems-II*, Volume: 47, Issue: 11, pp. 1188-1203, November 2000.

[3] W. Kautz, "Cellular Logic-in-Memory Arrays", *IEEE Transactions on Computers*, Volume: C-18, Issue: 8 , pp. 719-727, August 1969.

[4] H. S. Stone, "A Logic-in-Memory Computer", *IEEE Transactions on Computers*, pp. 73-78, January 1970.

[5] A. DeHon, "DPGA-coupled microprocessors: commodity ICs for the early 21st Century", *Proc. IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 31-39, 1994.

[6] Z. A. Ye, A. Moshovos, S. Hauck, and P. Banerjee, "CHIMAERA: a high-performance architecture with a tightly-coupled reconfigurable functional unit", *Proc. 27th International Symposium on Computer Architecture*, pp. 225-235, 2000.

[7] R. Razdan and M. D. Smith, "A High-Performance Microarchitecture With Hardware-Programmable Functional Units", *Proc. 27th Annual International Symposium on Microarchitecture, MICRO-27*, pp. 172-180, 1994.

[8] T. J. Callahan, J. R. Hauser, and J. Wawrzynek, "The Garp Architecture and C Compiler", *IEEE Computer*, Volume: 33, Issue: 4, pp. 62-69, April 2000.

[9] R. Sangireddy, H. Kim, and A. K. Somani, "Low-power high-performance Reconfigurable Computing Cache Architecture", *IEEE Transactions on Computers*, Vol. 53, Issue: 10, pp. 1274-1290, October 2004.

[10] R. Sangireddy, "Register Organization for Enhanced On-chip Parallelism", *Proc. IEEE 15th International Conference on Application-specific Systems, Architectures and Processors, ASAP2004*, pp. 180-190, September 2004.

[11] Keshab K. Parhi, "VLSI Digital Signal Processing Systems Design and Implementation", Wiley, 1999.

[12] S. Y. Kung, "VLSI Array Processors", Prentice Hall, 1988.

[13] Mathew Wojko and Hossam ElGindy, "Self Configuring Binary Multiplier for LUT addressable FPGAs", *Proc. Australasian conference on Parallel and Real-Time Systems*, 1998.

[14] http://www.synopsys.com/

[15] P. Shivakumar and N. P. Jouppi, "CACTI3.0: An Integrated Cache Timing, Power, and Area Power Model", DEC WRL Research 2001/2, August 2001.

[16] Doug Burger and Todd M. Austin, "The SimpleScalar Tool Set, Version 2.0", Computer Sciences Department Technical report # 1342, University of Wisconsin-Madison, June 1997.

[17] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, 1989.

[18] Michael E. Wolf and Monica S. Lam, "A data locality optimizing algorithm", *Proc. ACM Conference on Programming Language Design and Implementation*, Volume 26, Issue 6, pp. 33-44, May 1991.