

## Dynamically Programmable Cache for Multimedia Applications

Mouna Nakkar<sup>1</sup> and Paul Franzon<sup>2</sup>

<sup>1</sup>*Dept. Electrical/Electronics & Computer Eng,  
University of Sharjah,  
Sharjah, UAE, P.O. Box 27272  
Email: m\_nakkar@sharjah.ac.ae*

<sup>2</sup>*Dept. Electrical & Compute Eng.  
North Carolina State University  
Raleigh, NC, USA  
Email: paulf@ncsu.edu*

### Abstract

The demand for higher computational power is ever increasing, and hence, the computational resources on chip. To keep us with this increase, the size of the on-chip cache memory has also been consistently increasing. While some applications utilize the silicon real-state on chip, most application, especially multimedia and signal processing applications do not utilize the on-chip cache memory efficiently. This is because multimedia and signal processing applications data are not highly referenced. This paper shows novel cache architecture that utilizes the silicon real-state and increase performance of multimedia and signal processing applications. The main idea is to interleave Filed Programmable Cache Arrays (FPGAs) between data cache lines and use these added computational cells to execute some subroutines in the cache. Thus instead of juggling the data in and out of the cache, DPC allows algorithms to execute data while in the D-cache. The added FPGA will utilize the cache more, reduce memory bottleneck, and increase performance. The new architecture is called Dynamically Programmable Cache (DPC). DPC can function as a conventional Data cache, algorithm accelerator, or a combination of the two. This paper shows DPC machine having 5X the speedup over Altera FLEX10K and 2X the speed over high speed General Purpose Processors such as ULTRA SPARC II for some multimedia application.

**Keywords:** Reconfigurable Computing, DPC, FPGA, Multimedia and signal processing applications.

## **Introduction**

Multimedia and signal processing applications demand higher computing power. The widely known Adhmel's law [1] suggests enhancing the performance of the common case to increase microprocessors performance. Also, another common rule called the 90-10 rule which states that 90 percent of execution time is spent on computation and 10 percent is consumed on inner loops. Spatial structures excel in the execution of such computation intensive functions as compared to temporal structures [2]. To apply both these rules, it is better to have the computation power transfer to memory. Thus spatial compute engine is customized such that it will execute subroutines while data in the cache rather than juggling data in and out of memory to registers. Furthermore, these spatial structures are easily reconfigured to compute other applications providing flexibility to the system. These spatial structures are based on reconfigurable hardware, Filed Programmable Gate Arrays (FPGAs).

Reconfigurable Hardware which mostly is based on FPGAs has been used in many applications and systems to gain flexibility and increased performance. Impressive speedups have been reported for different applications such as RSA encryption [3], scanning application [4], and DNA sequence matching [5].

Despite the performance gain, researches argue that reconfigurable hardware has so many shortcomings which make it less ideal for general-purpose computing. Some of the obstacles in using FPGA are:

- 1 FPGA machines are small and cannot fit very large applications. Large applications may need to be divided across several FPGA chips.
- 2 Configuration time is high and expensive for the FPGA to be used once and discarded.
- 3 Usually real program do not have subroutines repeated enough worth loading into the FPGA machines.
- 4 It is always a faster approach to build an Applied Specified Integrated Circuit rather than configuring FPGA chips. Standard functions like multipliers and floating point adders are very slow in an FPGA when compared to most of microprocessors.
- 5 FPGA chips are limited in storing data. Usually, extra RAM support is needed to gain comparable performance.

To address some of these problems, researchers have proposed building reconfigurable machines which tightly couples reconfigurable hardware with microprocessors [6, 7, and 8]. Their approach is basically using FPGA fabric to act as a coprocessor accelerator for some applications. The MMX extension to Intel's processors adds more flexibility and performance to general-purpose processors [6]. Garp Architecture on the other hand proposes a stand alone processor based on MIPS architecture deriving a reconfigurable array which adds speedups of two or more for some applications [7]. IRAM architecture is based on adding hardware to regular memory [8]. DPC approach on the other hand adds FPGA arrays into the Data Cache. The idea is to process the data while in the cache instead of transferring them to the processor. The advantages to this approach include: more added flexibility to general-purpose processors, reduction in memory-to-processor bottleneck, acceleration of execution time of some applications, and elimination of configuration time by storing

the configuration in the cache.

Unlike other proposed architectures such as the IRAM [8], DPC is designed to fit in an ordinary processing environment which includes structural programs, libraries, context switching, virtual memories, and multiple users. The main thread of the program is managed by the CPU. In fact programs may never need to use this additional FPGA hardware for ordinary application. It is only for special subroutines and repeated algorithms. The processor may use DPC as a conventional D-Cache, coprocessor accelerator, or combination of both. As a coprocessor accelerator, CPU assigns simple repetitive subroutines to the DPC; otherwise, it acts as a conventional cache. This reverses the conventional method of executing all subroutines in the main processor. DPC increases performance by a factor of five for some multimedia applications [9, 10, and 11].

The presence of FPGA in the D-Cache has several advantages: reduces communication bottleneck between memory and CPU, increases FPGA configuration storage, increases computation per memory access, and hence increases performance. DPC machine allows the data to be executed while in the D-Cache rather than transferring the data back and forth from CPU to D-cache. This reduces communication bottleneck between CPU and D-Cache. The reduced communication increases performance and reduces execution time. In addition, the FPGA fabric will have huge memory bandwidth to store configuration; it is known that FPGA's main problem is memory bandwidth.

Multimedia algorithms and subroutines are best suited for DPC. These algorithms usually carry simple and highly repetitive simple instructions such as multiply-accumulate, add and shift instructions. In addition, multimedia datum is represented by small binary sets; meaning one line of cache can carry several multimedia data sets. With DPC, these algorithms can be easily executed inside the D-cache instead of transferring them into the processor.

This paper shows a detailed description of the DPC architecture as well as its performance compared to conventional FPGA chips. The organization of this paper is as follow: Section II describes the DPC architecture, Section III briefly presents the DPC simulator, Section IV presents three different multimedia algorithms: Convolution, Motion Estimation, and Run Length Coding, Section V shows DPC performance compared to other approaches, and finally Section VI offers concluding remarks and future work.

## **DPC Architecture**

### **DPC Top View**

DPC unit fits into an ordinary microprocessor environment where the CPU manages the main thread of a program. The CPU uses DPC unit as a coprocessor accelerator only when the algorithm being executed requires partial acceleration. It is not necessary for the DPC to operate as a coprocessor accelerator for each thread in the CPU but only for specialized subroutines. Figure 1 shows a block diagram of the DPC fitting in an ordinary processor environment. Figure 1 shows the CPU communicates with Main Memory through the Memory Management Unit (MMU) and DPC

replaces D-Cache. The system environment is exactly like any General Purpose Processor except DPC replaces D-Cache. The FPGA fabric can be placed on any cache levels, but in this design it is placed only on level-1 D-Cache for simplicity.

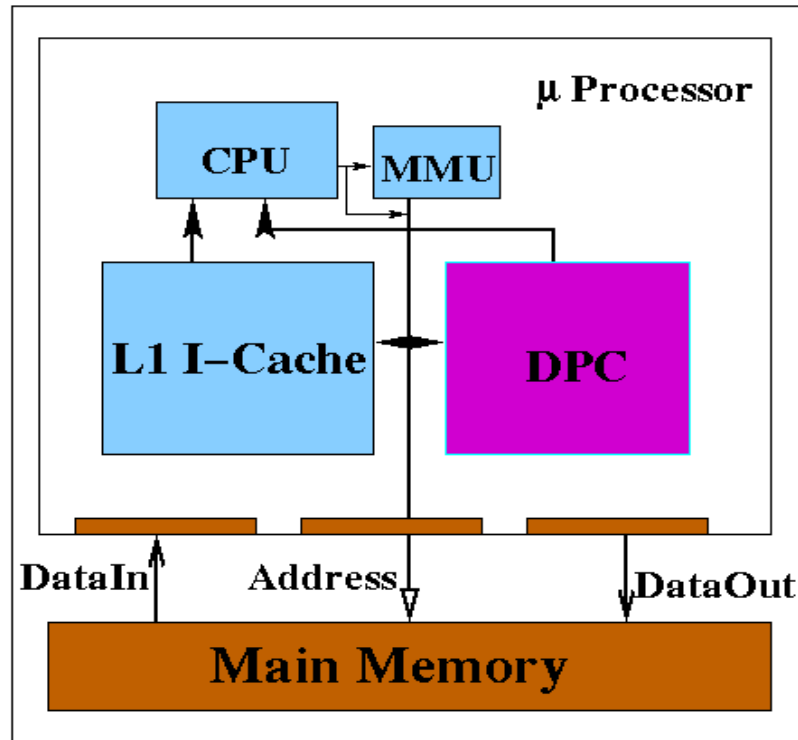


Figure 1: DPC replaces D-Cache in an Ordinary Processor Environment.

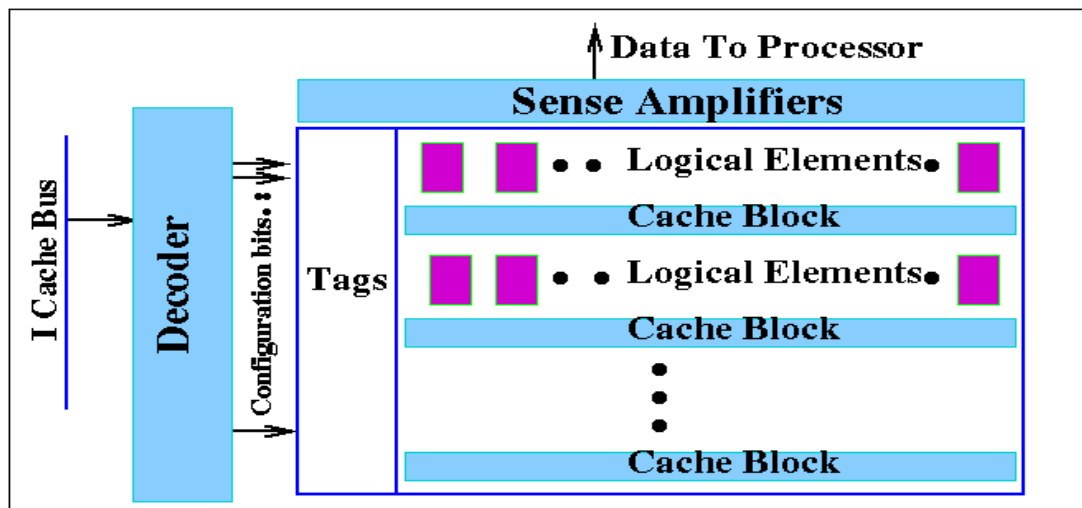
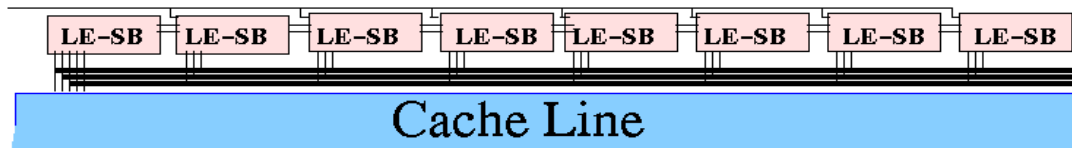


Figure 2: FPGA Fabric and Data Cache Merged in the DPC Unit.

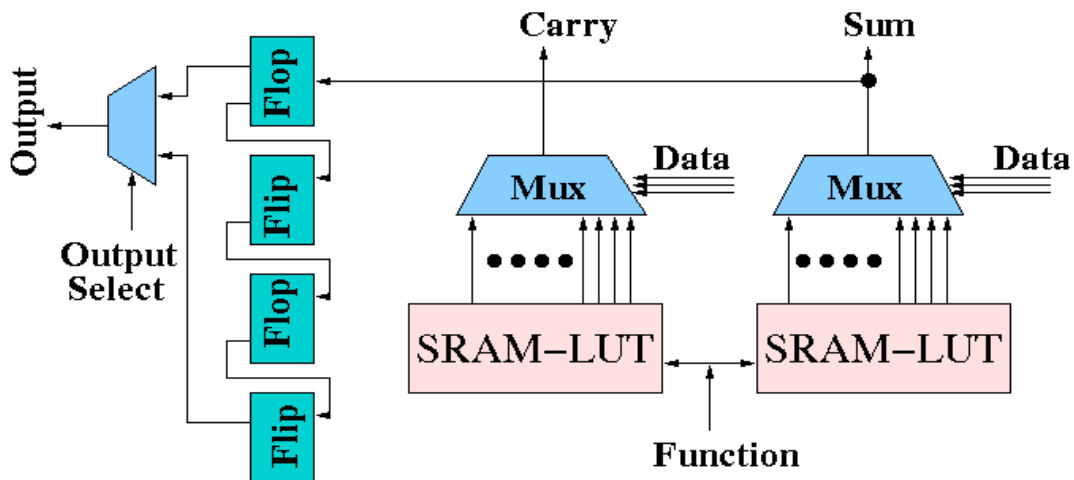
**Detailed Structure of the DPC Unit**

The DPC architecture is similar to any conventional D-Cache with added FPGA fabric. This FPGA fabric is actually an array of 8X16 FPGA rows meshed into the rows of D-cache as shown in Figure 2. The number of FPGA array can be scaled according to the type of application. In this project, 8x16 FPGA row is sufficient for the small sample of multimedia applications. Figure 2 shows FPGA fabric interleaved with cache lines (cache blocks) where several cache lines can be associated with one FPGA row. In this project, the FPGA row-to-cache lines ratio is 1 to 16; meaning one FPGA row is associated with 16 cache lines. These cache lines are for storing configurations and data. The FPGA Fabric can receive its data and configurations from 16 different cache lines; configurations are the command lines to program the FPGA fabric.

Each FPGA row contains several Logical Elements (LE) and Switching Box (SB) as shown in Figure 3. A Logical Element is the basic building cell of any FPGA fabric and a Switching Box is the routing which connects these LEs with each other. Figure 3 shows one FPGA row of 8 LEs and SBs. The logical element can be programmed to implement several functions, such as “and”, “or”, “xor”, “add”. Etc. The logical element used in this project is similar to the logical element used in the Xilinx and Altera chips [12] [13]. The conventional Logical Element is basically an SRAM Look-Up Table (SRAM-LUT) shown in Figure 4. Figure 4 shows a basic LE and 4 D-Flip Flops. The D-Flip Flops are used to store the results of up to 4 cycles for data forwarding.



**Figure 3:** 8 Logical Elements and Switching Boxes Connected to One Cache Line.



**Figure 4:** Logical Element for the FPGA Fabric in DPC

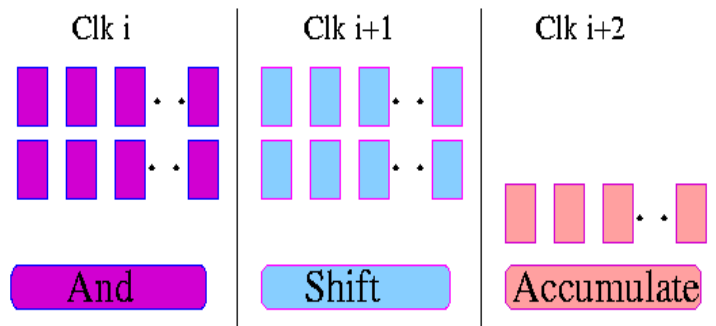
### Configuring the DPC Unit

When the DPC acts as a conventional D-Cache there is no need to configure the FPGA fabric. All access to the cache will be either read or write. However, when the DPC acts as coprocessor accelerator, the accesses to the cache will be one of the three options: data read, data write, or configuration write. As mentioned before, configurations are the commands to program the FPGA array.

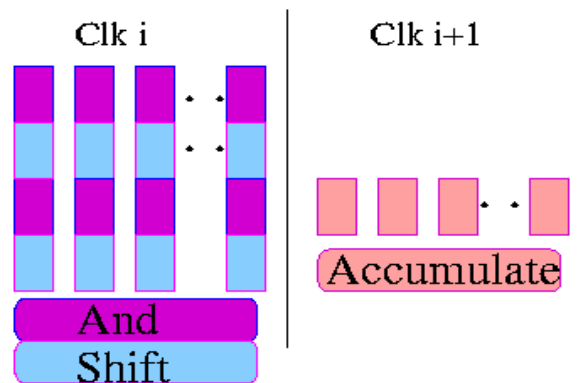
To configure DPC for a subroutine, the main program supplies configurations of that subroutine to the DPC. DPC stores these configurations in the cache. Configuring the DPC takes only one clock cycle where the DPC is ready for read/write use on the next cycle. One configuration access replaces 16 cache lines as mentioned above. The percentage of time the DPC is configured varies, depending on the applications.

### Virtualization Concept of the DPC Unit

When the DPC is used as a computational element, the decoder unit receives the instruction from the I-cache every clock cycle and executes it on the FPGA fabric. The configuration is issued once at the beginning of execution. Alternative configurations are stored in registers called Virtualization Registers (VRs), which are part of the cache lines and assigned for dynamic multi-context switching, i.e. virtualization. These registers have the capacity to hold up to three FPGA configurations as shown in Figure 5. Unlike other architectures [1] [2], the configurations stored in VRs are dynamic, meaning DPC allows the VRs to switch to different configurations each clock cycle. The VRs are used when the task is large and does not fit in the limited FPGA fabric. Virtualization is similar to partitioning the design across several chips, but instead configuration is stored in VRs. For example, suppose three instructions "And", "Shift" and "Accumulate" are configured in three consecutive cycles:  $i$ ,  $i+1$ , and  $i+2$ . Two of these instructions can be executed in one cycle if the "And" and "Shift" instruction are stored in VRs in cycle  $i$  and thus saving one clock cycle. Figure 6 shows a conceptual illustration of configuration switch. This process efficiently utilizes the limited FPGA area since large applications can be partitioned temporally. The VRs can be viewed as a library for repeated algorithms or instructions. Further, storing configurations in VRs can avoid decoder configuration delays.



**Figure 5:** Three Instruction Configurations in the FPGA array.



**Figure 6:** Using Virtualization Registers to hold Two Configurations in FPGA memory.

The data for the FPGA fabric is stored in the D-Cache and is allocated when the algorithm or subroutine is being executed. When the DPC is used as a computational device, both the configuration and the data are stored in the same cache lines. When the DPC executes algorithms, the result of one instruction may be dependent on another instruction. To avoid storing and loading these data into and from the cache, the LE contains sufficient registers to store the results of up to four previous states (4 D-Flip Flops) as shown in Figure 4.

### DPC Simulator

To evaluate the execution time of subroutines executed on DPC, a Verilog based simulator is designed using Cadence Design Tools. The size of the FPGA array used in this project is 8X16, and the cache line is 256 bits (32 bytes) with 20ns clock rate. The total cache size varies between 2Kbytes to 32Kbytes. It should be mentioned that cache size is irrelevant in evaluating the DPC as a coprocessor accelerator, because the FPGA area is not related to cache size instead a fixed number of arrays can be evaluated at the beginning of the design.

### DPC as Data Cache

As a Data Cache, DPC will allow only two accesses: Data Read and Data Write on one read/write cache port. The write policy of DPC is selected to be Write Back. The performance of the DPC as conventional cache matches any other D-Cache.

### DPC as Coprocessor Accelerator

As a coprocessor accelerator, DPC will allow three different accesses to the cache: Data Read, Data Write, and Configuration Write. Configurations will replace up to 16 cache lines as mentioned before, and may replace valuable data. Configuring the FPGA fabric will last one memory access. Therefore, DPC will be ready for data read and write on the next clock cycle.

### DPC as Combination of the Two

The use of DPC can be alternated between regular D-Cache or FPGA accelerator. For example, DPC can be used as a coprocessor accelerator in a given memory access and on the next access writes configuration.

### Multimedia and DSP Algorithms

DSP algorithms are suitable application to DPC machines. Below is a review of some DSP algorithms: Convolution, Motion estimation, and Run Length Coding.

#### Convolution

The basic operation used to smooth images in image processing (image filtering) is Convolution [14]. The equation for convolution is given below:

$$g(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x - \varepsilon, y - \eta)h(\varepsilon, \eta)d\varepsilon d\eta \quad (1)$$

The number of multiplication in the above equation increases linearly with the increase of the boundary limits. As it can be seen most of the operations are simple. Further, the number of bits per pixel is maximum of 8. Black and white images have 256 shades which can easily be represented in an 8 bit number and high precision is not needed.

#### Motion Estimation

Motion Estimation is widely used in DSP and image compression algorithms. The matching criterion of this algorithm is given in the following equation:

$$MAE(i, j) = \frac{1}{M \cdot N} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} \left| C(x+k, y+l) - R(x+i+k, y+j+l) \right| \quad (2)$$

Where  $M \times N$  is the Macroblock size,  $(x, y)$  is the location of the first Macroblock on the object image, and  $(i, j)$  is the location on the reference image [14].

#### Run Length Coding

Run Length Coding is a lossless image comparison algorithm [14]. For this algorithm, the most common operations are Compares and Accumulates. The dominant instructions of the above algorithm are: multiply accumulate, add, and subtract. These operations may not need the big circuitry of a microprocessor integer unit.

This paper shows performance evaluation of the above algorithms: Convolution, Motion Estimation, and Run Length Coding. The performance evaluation includes a comparison of the three different algorithms implemented with DPC and a conventional FPGA chip. Also, a comparison of DPC and a General Purpose Processor (ULTRA Sparc4) is presented.

### DPC Performance as Coprocessor Accelerator

Convolution algorithm is implemented on four different systems: 1) Non Virtualized

DPC, 2) Virtualized DPC, 3) Altera FLEX 10K, and 4) Altera FLEX10K with external SRAM as shown in Figure 7. The clock rate for all the above designs is 20ns. The Non-Virtualization is included in the implementations to show that the DPC performance is similar to the Altera FLEX10K performance. This is true because the Altera FLEX10K LE is similar to the DPC LE except for the additional memory cells in the DPC LE. Figure 7 shows clearly that FLEX10K and DPC Non Virtualization match in performance. However, DPC Virtualization has up to four times the performance gain over ALTFLEX10K and two times the performance gain over FLEX10K with added SRAM [10]. The speed up is consistent with the increase of the number of parallel multipliers. A comparison of Convolution execution time run on Ultra Sparc4 machine is shown in Figure 8.

Figure 8 shows three different implementations made for a fair comparison: first implementation is DPC execution time vs. the number of parallel multipliers, second implementation is Sparc's execution time, and the third implementation is DPC forced to have one multiplier per memory access execution time. It can be shown that DPC has the least execution times, because the number of parallel multipliers will reduce execution time. DPC shows 10 times the performance of Sparc4 machines.

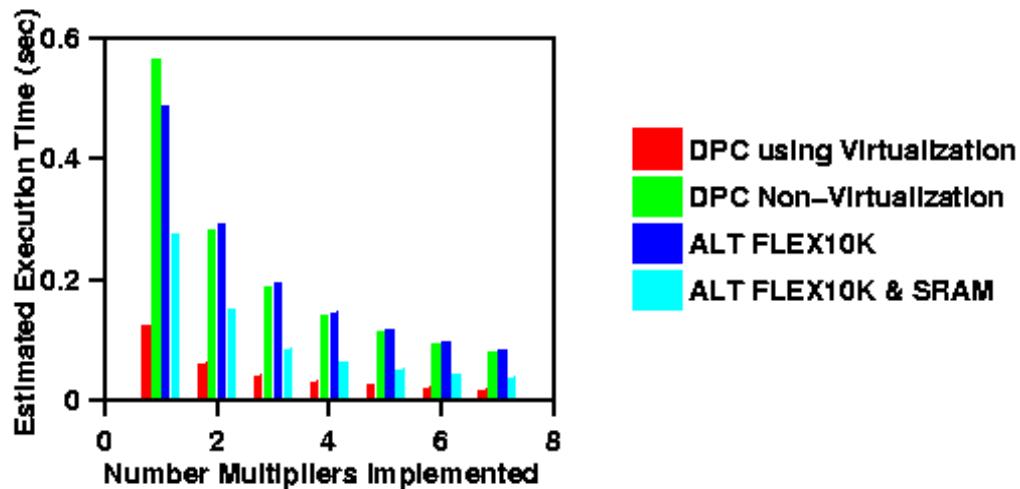
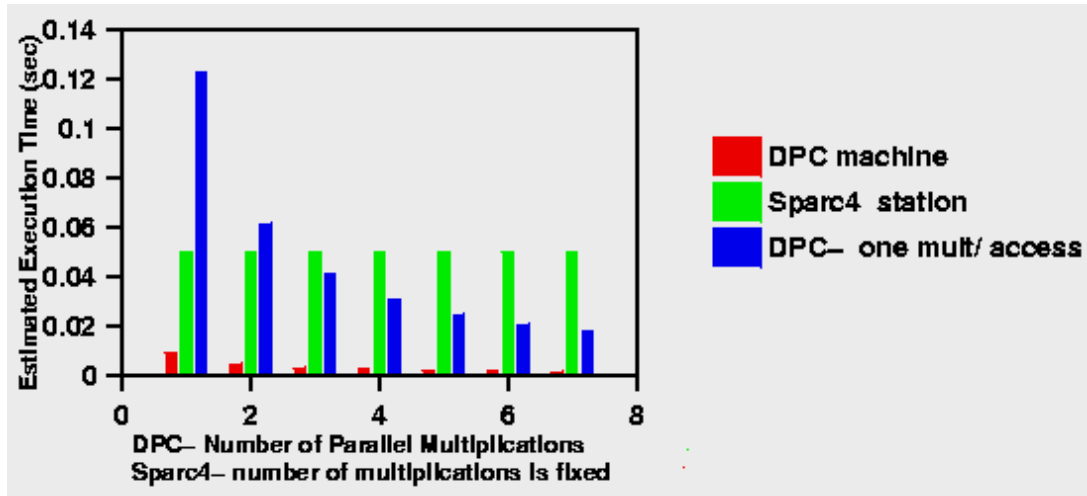


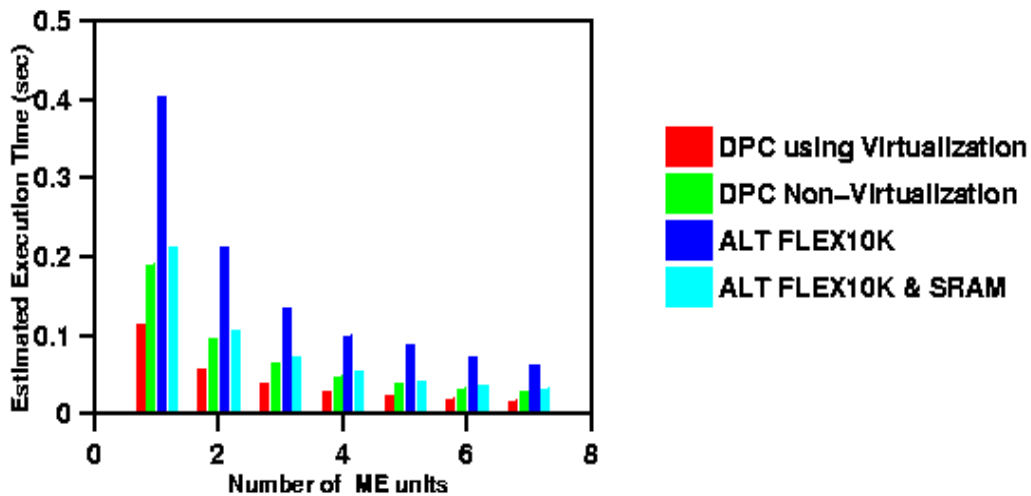
Figure 7: Execution Time for Convolution Algorithm.

Another application is Motion Estimation which is implemented with the DPC using Virtualization and Non-Virtualization schemes and compared to the FLEX10K and FLEX10K with the external SRAM. The algorithm is run for an image size of 526x438 as shown in Figure 9. The simulation shows that DPC is two times faster than Altera FLEX10K with SRAM [9] and four times faster than FLEX10K alone. Similar to Convolution algorithm's results, the performance is consistent for more parallel multipliers. The comparison with Sparc4 is shown in Figure 10. It is seen that Motion Estimation run on DPC machine has ten times the performance gains over Sparc4 machine.

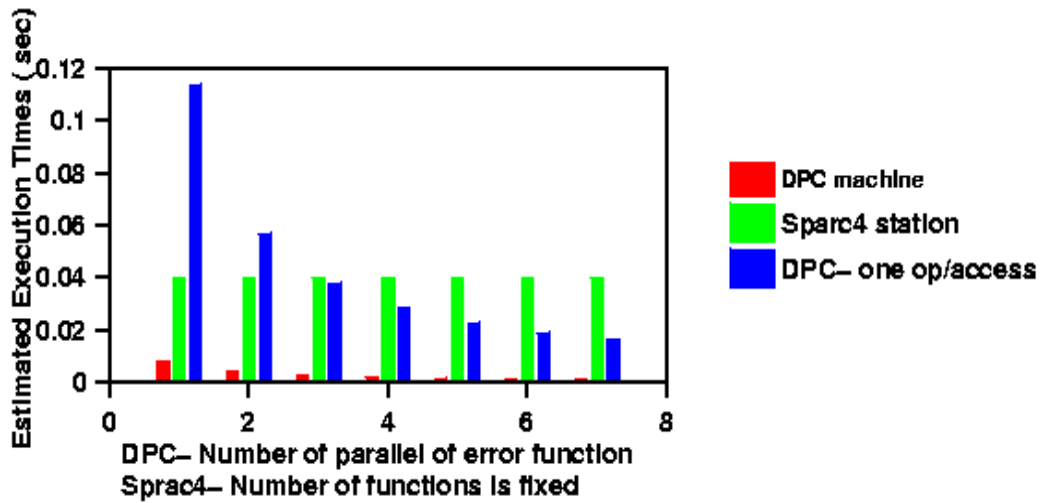


**Figure 8:** Execution Time for Convolution Algorithm with Comparison to Ultra Sparc4

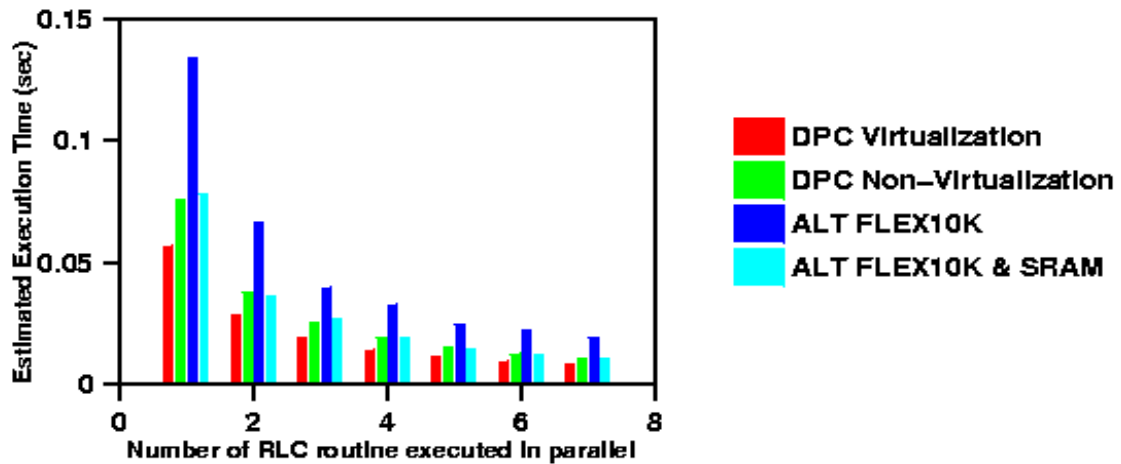
Another algorithm is Run Length Coding algorithm which is also implemented with Virtualized DPC and Non-Virtualized DPC schemes and compared to the FLEX10K and FLEX10K with external SRAM. The result of the comparison is shown in Figure 11 for an image size of 526x438. This is similar to the previous simulations for Motion Estimation and Convolution. Figure 11 shows DPC-Virtualization is two times faster than Altera FLEX10K and one and half times faster than Altera FLEX10K with additional SRAM. Figure 12 shows performance of DPC machine compared to Sparc4 station. It can be seen that DPC machine has twelve times the performance over Sparc4 machine.



**Figure 9:** Execution Time for Motion Estimation Algorithm.



**Figure 10:** Execution Time for Motion Estimation Algorithm with Comparison to Ultra Sparc4.



**Figure 11:** Execution Time for RLC Algorithm.

When comparing DPC to an FPGA chip (Altera FLEK10K), it can be concluded that Non Virtualized DPC's performance matches that of Altera FLEX10K that is because their FPGA fabric is based on the same Logical Element design. However, the performance of DPC Virtualization is higher than FLEX10K even if we add more memory to FLEX10K. The advantage of DPC over FLEX10K is in Virtualization that is because Virtualization allows configurations to be stored in memory and hence save configuration cycles and gain performance.

When comparing DPC to conventional GPP (Ultra Sparc4), the speedup in execution times is due to the fact that DPC reduces memory to CPU traffic.

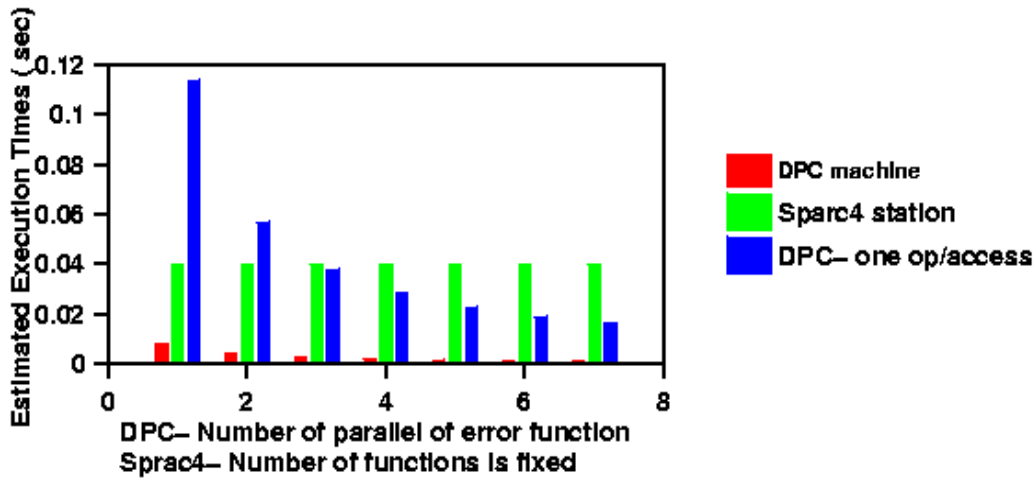


Figure 12: Execution Time for RLC with Comparison to Ultra Sparc4.

### Concluding Remarks

This paper shows an analysis of DPC performance. DPC is analyzed as a coprocessor accelerator, a conventional D-Cache, and a combination of the two. As a coprocessor accelerator, three different algorithms are run on DPC: Convolution, Motion Estimation, and Run Length Coding. The simulation results for all different algorithms are consistent and show that Non-Virtualized DPC's performance matches that of a conventional FPGA chip, Altera FLEX10K. That is because they both have the same Logical Element design. However, Virtualized DPC increases performance by a factor of four over FLEX10K. Even if we support FLEX10K with additional memory, the performance of DPC will still be higher by a factor of two. Virtualized DPC allows configurations to be stored in the cache, taking advantage of cache memory to increase FPGA memory bandwidth. DPC has higher FPGA memory bandwidth over conventional FPGA chips. When comparing DPC with a conventional General Purpose Processors such as Ultra Sparc4, DPC has improved performance for all the above mentioned algorithms by at least a factor of ten. The reason is because the memory traffic is eliminated and hence performance is increased.

DPC is ideal cache architecture for multimedia and signal processing applications where it is known that their algorithms have heavy spatial structures rather than temporal structures. DPC offers higher performance for applications of this type because it provides lesser communications time between registers and memory. Furthermore, DPC offers high silicon real-state utilization.

### References

- [1] J.L. Hennessy & D.A. Patterson, "Computer Architecture a Quantitative

- Approach 3<sup>rd</sup> edition," Morgan Kaufmann Publishers, USA, 2003.
- [2] A DeHon, "The Density Advantage of Configurable Computing," *Computer*, vol. 33, no. 4, pp 41-49, Apr. 2000.
  - [3] J.E. Vuillemin, et all, "Programmable active memories: Reconfigurable systems come of age," *IEEE Transactions on VLSI*, vol. 4, no. 1, Mar. 1996.
  - [4] B. Gunther and Geroge Milne and et. all, "Assessing document relevance with runtime reconfigurable machines," in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, Apr. 1996, pp 10-17.
  - [5] Eric Lemoine and David Merceron, "Runtime reconfiguration of FPGA for scanning genomic database," in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, Apr. 1995, pp. 90-98.
  - [6] M. Mittal, A. Peleg, and U. Weiser, "*MMX Technology Architecture Overview*," *Intel Technology Journal*, pp 1-12, Q3, 1997.
  - [7] J. Hauser and J. Wawrzynek, "Garp: A MIPS Processor with a Reconfigurable Coprocessor," in *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '97, April 16-18, 1997)*.
  - [8] J. Gebis, S. William, C. Kozyrakis, D. Patterson, "VIRAM1: A Media-Oriented Vector Processor with Embedded DRAM", in the 41st Design Automation Student Design Contest, San Diego, CA, June 2004.
  - [9] M. Nakkar and P.D. Franzon and et. all, "Dynamically Programmable Cache," in the 1998 SPIE Configurable Computing: Technology and Applications, Boston MA, Vol. 3526, pp. 218-226, 1998.
  - [10] M. Nakkar and P.D. Franzon and et all, "Dynamically Programmable Cache Evaluations and Virtualization," in the 1999 seventh ACM International symposium on Field Programmable Gate Arrays, Monterey CA, Feb. 1999.
  - [11] M. Nakkar and P.D. Franzon, "Feasibility Study of Dynamic Programmable Cache (DPC)," in the 2002 International Arab Conference on Information Technology, Doha Qatar, December 2002.
  - [12] Xilinx, Xilinx, *The Programmable Logic Data Book*, Xilinx Inc., 1994.
  - [13] Algotronix, *Algotronix Data Book*, Edinburgh, UK CAL1024 Preliminary Data Sheet, 1988.
  - [14] V Bhaskaran and K. Konstantinides, *Image and Video Compression Standards Algorithms and Architectures*, Kluwer Academic Publishers, 101 Philip Drive, Assinippi Park, 1996.