Scientific Computing Beyond CPUs: FPGA implementations of common scientific kernels

Melissa C. Smith
Oak Ridge National Laboratory
Oak Ridge, TN, USA 37831
smithmc@ornl.gov

Jeffery S. Vetter
Oak Ridge National Laboratory
Oak Ridge, TN, USA 37831
vetterjs@ornl.gov

Sadaf R. Alam
Oak Ridge National Laboratory
Oak Ridge, TN, USA 37831
alamsr@ornl.gov

Abstract

Reconfigurable Computing architectures consisting of FPGAs or similar programmable devices have been used in embedded systems for a number of years where integer and fixed point arithmetic are prevalent. Recently the logic capacity of these programmable devices has grown dramatically making 64b floating-point operations feasible and thus their use in scientific and high performance computing intriguing. We are using the SRC MAPstation to explore reconfigurable computing for computationally demanding scientific workloads at Oak Ridge National Laboratory. Our approach uses a high-level programming interface to the FPGAs, namely C and FORTRAN, which allows for straightforward porting of legacy code to the reconfigurable computing platform. In this paper, we present our results for some common Basic Linear Algebra Subroutine (BLAS) kernels running on the SRC MAPstation. These routines have very little or no data reuse, hence their performance on standard cache based systems is poor for large problem sizes. Due to the inherent parallelism of FPGAs and tightly coupled memory and computational units, the MAPstation performance is sustainable even for large problem sizes. We describe our algorithm analysis, programming paradigm, code migration techniques, and performance results. We also present our plans for a function library of commonly used scientific kernel implementations (i.e. BLAS, sparse matrix operations, FFTs, etc.) suitable for use on the SRC MAPstation and in the future, other reconfigurable computing platforms.

1. Introduction

Traditional computing paradigms are struggling to keep pace with analysis needs and are rapidly reaching limits on computing speed due to I/O bandwidth limitations and the impending clock wall. Furthermore, managing heat dissipation as devices devour more power is becoming increasingly difficult.

Reconfigurable computing (RC) with FPGAs potentially offers faster execution and lower power consumption all with slower clock speeds. Figure 1.1 shows the performance trend for FPGA devices far exceeds that of general microprocessors over the past decade. FPGAs have the ability to exploit inherent parallelism and match computation with application data flow (i.e. Data Flow Graph Theory). With gate densities now suitable for double precision operations, FPGAs can offer the "hardware-like" speed previously only found in custom ASICs with the "software-like" flexibility that can adapt to the changing needs of the application or suite of applications.

Many scientific applications at Oak Ridge National Laboratory (ORNL) and elsewhere depend on double precision floating-point operations. Furthermore, many of these applications rely on common computations or functions that limit performance. We have identified several of these functions as being computationally intensive and as candidates for implementation in FPGAs. Efforts to interface to legacy code have also emphasized the need for FORTRAN and C language tools for FPGA development. Moreover, we are interested in using multi-paradigm computing in our scientific applications. To do so across a broad spectrum of applications, high level language programming tools will be necessary to bring these devices and their capabilities to the users of our computing systems.

The remainder of the paper is organized as follows. In Section 2, we cover some related material on the RC/FPGA implementation of various kernels of interest. In Section 3, we give a brief overview of the RC system used in these studies. In Section 4, we discuss the details of a BLAS routine implementation. In Section 0, we look at the development of function libraries and their impact on programming for RC and multi-paradigm computing systems. Finally in Section 6, we offer some conclusions and a look at future research.

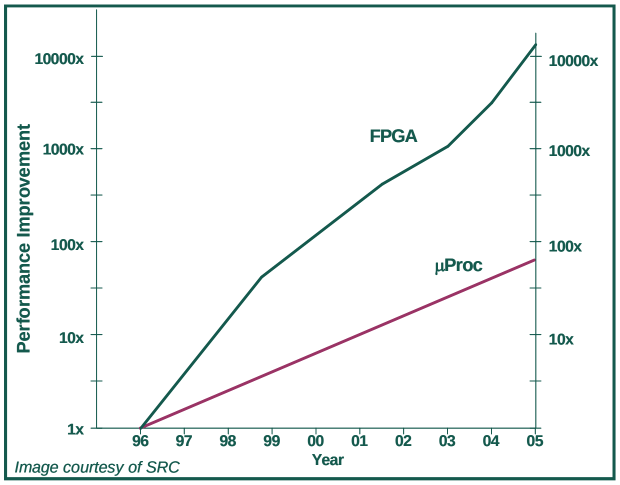


Figure 1.1 Performance trends for FPGAs and microprocessors

2. Related Work

Recent advances in FPGA logic and capacity now provide the means to effectively implement floating point applications. It has been shown that current FPGAs with abundant on-chip memory and I/O pin resources provide peak floating-point performance surpassing that of microprocessors [1, 15]. In [2, 3] Prasanna et al. implemented floating-point cores and dense matrix-vector multiplication on FPGA devices and compared the performance with that of general purpose microprocessors. Dou et al. [16] implement a 64-b floating-point block matrix multiplication algorithm for arbitrary matrix sizes. In both of these implementations, a Hardware Description Language (HDL) is used to target the FPGA devices. Additionally, with the exception of [1], these implementations are not fully compatible with the BLAS DGEMM function call.

Thus far, attempts to parallelize floating-point FFTs in reconfigurable hardware further exhibit the competitiveness between CPUs and FPGAs [4]. The FPGA FFT implementation, reported in [5], is designed to specifically target semi-empirical Car-Parrinello MD calculations. When FPGAs are used, bandwidth limitations—indicative of the parallel input format of an FFT—force more effective implementations using FIFO-based producer-consumer models. Examples of such implementations introduce a serial collapsed version of the FFT's butterflies [6]. Finally, the implementation in [7] is optimized for continuous data FFTs in FPGAs and ASICs. This approach extensively uses a corner turn module that was studied and benchmarked in [8].

In contrast, there has been limited work in the field of floating-point Sparse Matrix-Vector (SpMatVec) multiplication using application-specific designs in FPGAs. In [9], the authors present an optimized SpMatVec kernel implementation on a Virtex-II Pro FPGA device and compare its performance to that of an Itanium® 2 processor. In [10], a multi-FPGA based implementation is explored and its performance is compared to a general purpose multiprocessor system. While the authors show that FPGAs can be utilized to achieve high performance for the SpMatVec multiplication kernel, they do not discuss issues of implementing the kernel on a FPGA-based high performance computing system.

3. SRC-6 Overview

The SRC-6 MAPstation [11] is a paring of general-purpose microprocessors with a Multi-Adaptive Processor or MAP®. The host is powered by dual 2.8 GHz Xeon® microprocessors (µPs) running the Linux operating system. The MAP® board(s) is connected via SNAP® cards which plug into the DIMM slot on the microprocessor motherboard or via a high-bar switch for multi-MAP configurations. Each MAP®, as shown in Figure 3.1, consists of two user-configurable Xilinx® Virtex II XC2V6000 devices [12] running at 100 MHz, a control processor (FPGA, pre-configured), and six 4MB SRAM banks referred to as On Board Memory (OBM). Code for the µPs is written in standard C or FORTRAN. Code for the MAP® hardware is also written in C or FORTRAN and compiled by an SRC-proprietary compiler that targets the MAP® components. Calls to execute on the MAP® are function/subroutine calls from the standard C or FORTRAN modules. The CARTE® environment [11] builds and links a unified executable that binds the application and configuration(s) for the MAP® hardware.

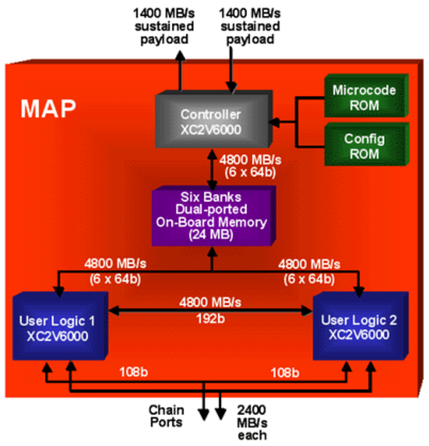


Figure 3.1 Hardware Architecture of the SRC MAP Processor [11]

4. BLAS Kernel Implementation

FPGAs have long been used to improve the performance of applications using integer and fixed-point operations—particularly DSP applications. Unfortunately, most scientific applications do not have these same characteristics. However, the high degree of configurability and dataflow nature of FPGAs can be potentially used to overcome some of the issues that plague CPUs with respect to dense floating point operations.

In this section, we will analyze the BLAS routines *DGEMM* and *SGEMM* and implement them on the SRC MAPstation.

4.1. Definition and Analysis

The BLAS Level 3 matrix-multiply routine SGEMM/DGEMM is defined as:

$$\mathbf{C}_{ij} = \sum_{k=0}^{N-1} \alpha \cdot \mathbf{A}_{ik}\mathbf{B}_{kj} + \beta \cdot \mathbf{C}_{ij} \qquad \textbf{EQ. 1}$$

where α and β are scalars, and *A*, *B*, and *C* are matrices (*A* is an *m* x *k*, *B* is an *k* x *n*, and *C* is an *m* x *n* matrix).

The BLAS routines are divided into three levels, each of which offers increased scope for exploiting parallelism. This subdivision corresponds to three different types of basic linear algebra operations:

- Level 1 BLAS [18] : for vector operations, such as y <– αx + y,
- Level 2 BLAS [19] : for matrix-vector operations, such as y <– αAx + βy,
- Level 3 BLAS [20]: for matrix-matrix operations, such as C <– αAB + βC.

Here, *A*, *B*, and *C* are matrices, *x* and *y* are vectors, and α and β are scalars.

The performance potential of the three levels of BLAS is strongly related to the ratio of floating-point operations to memory references, as well as to the reuse of data when it is stored in the higher levels of the memory hierarchy. Consequently, the Level 1 BLAS cannot achieve high efficiency on most modern supercomputers. The Level 2 BLAS can achieve near-peak performance on many vector processors; on RISC microprocessors, however, their performance is limited by the memory access bandwidth bottleneck. The greatest scope for exploiting the highest levels of the memory hierarchy as well as other forms of parallelism is offered by the Level 3 BLAS [21].

Thus, for dense matrices, the Level 3 BLAS require $O(N^2)$ memory accesses and $O(N^3)$ floating-point operations where N is the order of the largest matrix operand.

These dense matrix operations are an interesting component in the performance analysis of computing systems, because they exercise the memory hierarchy and push the computation capabilities of the hardware. In our FPGA implementation, we will attempt to exploit data reuse and inherent data flow of the operations to best utilize the data flow architecture of the FPGA. We will exploit fine grain parallelism internal to the FPGA and operate on multiple rows of the matrices concurrently. Finally, we will exploit coarse grain parallelism utilizing both user FPGAs available on the SRC MAPstation to operate on multiple blocks of the matrices concurrently.

4.2. Implementation Strategy

Our FPGA implementation of the DGEMM BLAS routine will target the SRC MAPstation. We will attempt to fully utilize both XC2V6000 user FPGAs.

The strategy is to divide the matrices into blocks, as shown in Figure 4.1, and perform the matrix multiplication in blocks. The number of blocks is determined by the number of on-board-memory banks available to the FPGAs; in this case, six total banks which we will divide into four input banks (two per FPGA) and two output banks (one per FPGA), as shown in Figure 4.2(a). The sub-block size is determined by the number of floating point multiply-accumulate (MAC) units that will fit per FPGA (sub-block denotes the block size or rows that will be computed in parallel per FPGA). The design will handle arbitrary size matrices up to 1024x1024 (larger dimensions are possible but not addressed in this design). The methodology exploits the data flow architecture of the FPGA and achieves the maximum data reuse.

The computations are conducted in two stages as shown in Figure 4.2. In the first stage, Figure 4.2(a), FPGA0 is paired with blocks *A00* and *A01* of matrix *A* and *B00* and *B10* of matrix *B,* which are used to partially compute blocks *C00* and *C01* of matrix *C*. Similarly, FPGA1 is paired with blocks *A10* and *A11* of matrix *A* and *B01* and *B11* of matrix *B,* which are used to partially compute blocks *C10* and *C11* of matrix *C*. Once all of the sub-blocks have been computed in stage one, ownership of matrix *B* is exchanged in stage two as shown in Figure 4.2(b). The pseudo code representation of the computations and ownership exchange is given in Figure 4.3.

To take advantage of the data flow architecture and fine grain parallelism of the FPGA, the number of floating point computations (alternate FLOPS) is higher than the theoretical optimum.

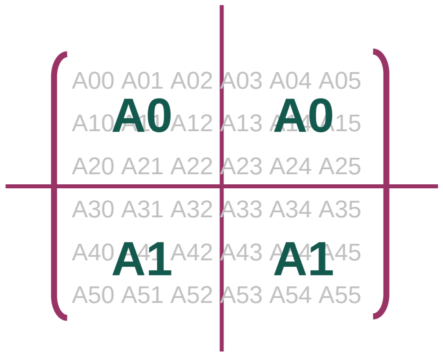
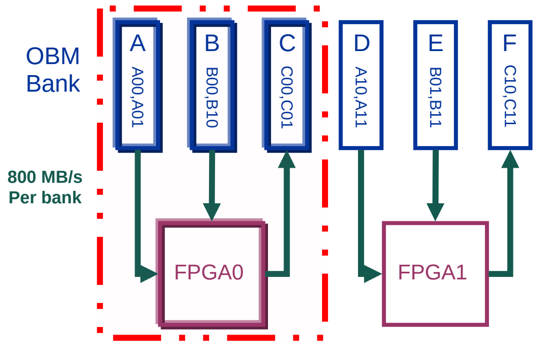
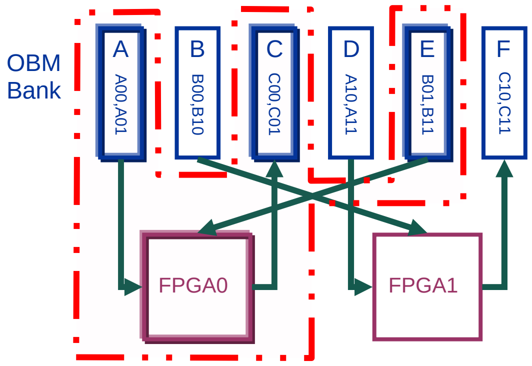


Figure 4.1 Block decomposition of the Matrix



(a) Stage One



(b) Stage Two

Figure 4.2 FPGA to OBM parings

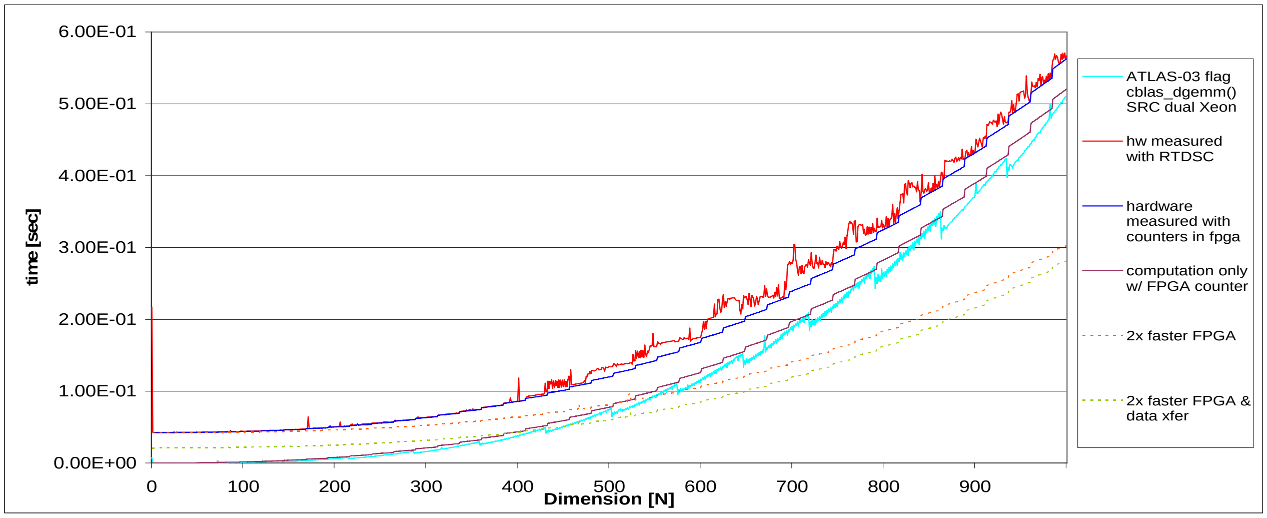
```
DMA matrices A, B, C
Set Stage0 Bank permissions
for stage=0; stage<2
    for i=0; i<1/2 N; i+= p
        cache p rows of A in BRAM
        if (stage==0) {start=0;stop=1/2 N}
        else {start=1/2 N; stop=N}
        for j=start, j<stop
            for k=0; k=N
                AEp = kth element of row p of A
                BE1 = column k, jth element of B
                BE1* = alpha
                sum_p = MAC(AEp, BE1)
            }
            CEp = sum_p + beta*CEp
        }
    }
    Sync user chips
    Set Stage1 Bank permissions
}

Where:
p = number of rows processed in parallel
N = dimension of matrices
```

Figure 4.3 MAP pseudo code for DGEMM

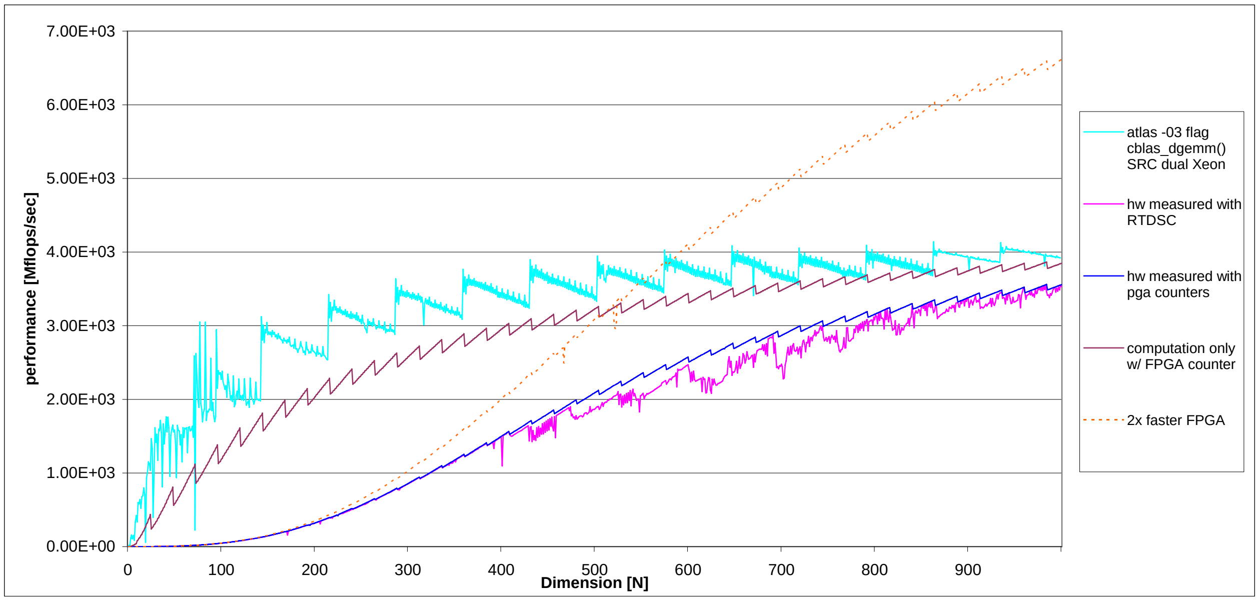
4.3. Results

The SRC MAPstation implementation resulted in full utilization of the two user FPGAs. We implemented both a single and double precision version. In the single precision version, SGEMM, we were able to place 25 single precision MAC units per FPGA. This enabled us to process up to 25 rows of matrix *A* per FPGA in parallel and achieve a modest FLOP rate for a 1000x1000 element matrix of 8.14 GFLOPS compared to 7.77 GFLOPS on a 3.06GHz dual-Xeon processor. In the double precision implementation, DGEMM, we were only able to place 12 double precision MAC units per FPGA. The increased size of the double precision macros reduced our number of parallel processing elements by slightly more than half. We were still able to process up to 12 rows of matrix *A* per FPGA in parallel and achieve a sustained FLOP rate for a 1000x1000 element matrix of 3.53 GFLOPS compared to 4.14 GFLOPS on the previously mentioned dual-Xeon processor (and 3.91 GFLOPS on the 1.8GHz dual-Xeon processor in the SRC MAPstation). The plots in Figure 4.4 show the DGEMM results.

In Figure 4.4(a), if we compare the plots of 'computation only' and computation with data transfer, we see that the data transfer time in and out of hardware is significant and impacts the "time to solution". For this reason, we are interested in other data transfer (streaming DMA) and memory methodologies (as found in the SGI solution with RASC [16]) to hide or mitigate this impact on execution time and performance. Also note, from the dashed plots of Figure 4.4(a) & (b), that faster and/or denser FPGAs can significantly improve both performance and execution time. For comparison, plots of the ATLAS implementation of DGEMM running on the dual Xeons of the SRC system are also shown. ATLAS (Automatically Tuned Linear Algebra Software) is a popular package that includes BLAS and LAPACK API [17].



(a) Execution time



(b) Performance

Figure 4.4 DGEMM Graphs

Our resulting design was created with SRC's CARTEv1.8 development suite which allows the designer to generate hardware and software from a unified environment using high-level languages, namely C or FORTRAN. A similar design by Dou [13], which used traditional hardware language to target the FPGA, achieved a peak performance of 15.6 GFLOPS with a larger, faster device (XC2VP125-7) running at 200MHz (twice as fast). Their implementation contained 39 double precision MAC units (3.25x's our number). The significance here is that we were able to complete a competitive design (when device size and speed are accounted for) using high-level languages as opposed to the traditional hardware languages often used to target FPGAs. This path from high-level languages such as C and FORTRAN is significant for users which may be unfamiliar with hardware languages. The learning curve for hardware languages and hardware design is steep and while they make for higher performance and more area efficient designs when used by experienced designers, novice designers are much less productive when using these languages. The ability to conduct hardware/software co-design from a unified environment can be empowering for computer scientists and engineers.

5. Function Libraries

Determining the optimal granularity for function libraries has been a challenge for the scientific community for many years. The addition of alternate computing technologies such as FPGAs adds additional complexity to this conundrum.

Our approach has been to first identify candidate applications that we think are suitable for RC, analyze these applications to determine the computational bottlenecks, and finally, determine if FPGAs can be used to speed up these applications by accelerating these kernels.

We have identified several application areas at ORNL which are computationally challenging, significant to our science research, and potential candidates for acceleration with RC architectures: molecular dynamics, climate modeling, life sciences, nanoscience, and bioinformatics. During this process, we have noted that many of these scientific applications use several common or similar functions such as dense matrix-matrix and matrix-vector operations (e.g. DDOT, DAXPY, DGEMM), 2D and 3D FFTs, and sparse matrix operations. **Error! Reference source not found.** shows how these applications and kernels overlap.

The goal is to identify and assemble a library of user friendly and familiar functions that are pertinent to scientific applications and of the appropriate computational complexity for use in RC architectures.

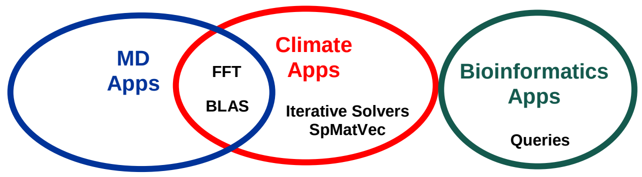


Figure 5.1 Candidate Kernels & Applications

5.1. Function Identification

Our analysis has identified several pertinent functions for consideration. The initial list includes: BLAS routines, sparse matrix vector operations, FFTs, and bioinformatics queries.

Earlier in the paper, we discussed the implementation of the BLAS DGEMM routine. We have also implemented several other BLAS routines such as DDOT, DGEMV, and DAXPY along with their single precision counterparts. These routines are widely used in many scientific applications and DGEMM is often used as a point of reference in performance profiles. As discussed in [1], commodity processors still hold a slight edge in peak performance but current trends indicate that the peak double precision floating-point performance of FPGAs will reverse this in the near future. Coupled with the memory efficiency that FPGAs achieve (can do more with less, requiring less storage relative to commodity CPUs) indicates that FPGAs will have an edge in performance computing in the near future.

Sparse matrix vector operations are commonly used in iterative solvers for linear systems. They are generally not efficient on general purpose processors due to the poor data locality and resulting high cache miss rate. They are also characterized by low utilization of the floating point unit(s) due to the high ratio of load/store to floating point operations (by comparison, dense matrix operations such as DGEMM typically have O(N/2) floating point operations for each memory access). The memory efficiency eluded to earlier allows FPGAs to avoid the penalty of cache misses and the high density of FPGAs allows for many processing elements to be populated across the device. Locally distributed memory banks keeps data vectors spatially near-by and fast host to FPGA communication keeps these banks populated with data.

FFTs are used in a variety of applications such as climate models, molecular dynamics, and many others. When data is distributed across many nodes in a parallel programming paradigm, the communication costs become a significant impediment to performance. In FPGAs and RC systems, we can implement these FFTs in a single or closely coupled pool of FPGAs. This inherently reduces the overhead of communications since data does not need to be transferred to other processors in the system; the processing unit is often on the same chip or device (or worst case in a closely coupled neighbor device with a streamlined high-bandwidth communication channel between them). This provides for much faster computation and more efficient corner turn implementations.

Bioinformatics queries are inherently suitable for RC implementation: pattern matching and highly parallel. Recent work in the area has determined that FPGAs and RC architectures are a good fit for this application problem [14]. Our current work focuses on finding the most suitable problem structure and domain for candidate RC architectures. We are working to analyze data streams and memory capacities to determine the most appropriate problem domains and algorithms for implementation.

5.2. Implementation Issues

The main issue with forming a library of functions for RC architectures is finding the appropriate computational complexity to make the implementation a "win" for the architecture. While the bandwidth on/off the FPGA device is considered sufficient in most cases, it is still an overhead cost as is the time to configure the FPGA. Configuration time can often be hidden behind other computations, while the time to transfer data to and from the FPGA device must still be taken into account even when using streaming data. If the implementation does not provide a performance advantage that outweighs this cost, then it is not desirable. Techniques such as "function fusing" can be used to form more complex library functions and gain this necessary advantage. The ultimate goal is to perform the maximum amount of computations or processing on the data while it resides at the FPGA; not very different from the goal for commodity CPUs but the cost for not accomplishing this goal is more evident.

Finding appropriate functions to "fuse" is more application dependent and thus the hindrance to forming a vast library of familiar functions. While some of the functions we have mentioned are for large problems computationally dense, more work is needed, possibly in the form of performance models and analysis, to determine how best to manage these libraries and assist the user in determining when they should be used to replace the default library.

6. Conclusions and Future Work

We successfully used a High Level Language (HLL) tool (CARTEv1.8) to design for FPGAs. While implementation of the DGEMM routine on the SRC MAPstation still required some hardware knowledge to be efficient and take advantage of all the FPGA and RC architecture has to offer, the use of the HLL tool is more efficient for the programmer than using a Hardware Descriptive Language (HDL) tool and generates code that performs well compared to other HDL implementations. The programmer must be aware of memory limitations, FPGA limitations, and some of the 'tricks' to take advantage of the FPGA and architecture strengths.

The development of function libraries for FPGAs and RC architectures requires analysis to determine candidate functions for FPGA implementation. We determined in this process that the traditional breakout level of the library functions (e.g. BLAS and VSIPL) may not always be appropriate for RC implementation. Further analysis is needed in "function fusing" to determine if adding computational weight can provide a "win" for these functions. Also, the floating point performance of FPGAs is growing faster than that of commodity CPUs and may soon make this a moot point. Additionally, faster FPGAs and higher bandwidth RC architectures (to reduce data transfer costs dramatically) could also void this conclusion.

FPGA growth rates are exceeding commodity CPUs and future implementations will ultimately provide some interesting platforms for exploration of scientific codes. The use of HLL to target these platforms will be necessary for making them usable in the scientific community with its stockpile of legacy code. Other tools are needed to identify candidate codes or regions in the application for RC implementation. Additionally, tools are needed for resource management for multi-paradigm platforms.

Our future work will focus on some of these issues. We will continue to work towards the maximum utilization of the FPGA resources via these HLL programming tools and look toward additional function/kernel library development. We also plan to focus on the idea of resource management for devices existing in multi-paradigm platforms (FPGAs, GPUs, and other alternative devices). The decision on what accelerator to use for a given application can be complicated due to function availability, performance metrics, device availability, and other factors. Finally, tools to assist in kernel identification would significantly aid the programmer in determining where to focus for implementation. The ability to identify regions of the code which are appropriate for implementation on a particular RC platform given its memory capacity, memory architecture, FPGA size and density, and other factors would be a significant step towards more efficient use of these devices.

7. References

[1] K. D. Underwood, and K. S. Hemmert, "Closing the GAP:

CPU and FPGA Trends in Sustainable Floating-Point BLAS Performance," *In Proceedings of 2004 IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM04)*, 2004.

[2] G. Govindu, et al., "Analysis of High-Performance Floating-Point Arithmetic on FPGAs," *In Proceedings of the 11th Reconfigurable Architectures Workshop (RAW04)*, 2004.

[3] L. Zhuo, V. K. Prasanna, "Scalable and Modular Algorithms for Floating-Point Matrix Multiplication on FPGAs," *In Proceedings of the 18th International Parallel & Distributed Processing Symposium (IPDPS04)*, 2004.

[4] K. Scott Hemmert, and Keith D. Underwood, "An Analysis of the Double-Precision Floating-Point FFT on FPGAs," *IEEE Symposium on Field Programmable Custom Computing Machines (FCCM05)*, Apr. 2005.

[5] K. Araki, T. Sasaki, D. Mizoguchi, U. Nagashima, I. Miyoshi, and T. Tanahashi, "Development of a special purpose circuit for 3D-FFT using FPGA," *The Japanese Society of Fluid Mechanics: 18th Aeromechanics Symposium,* Dec. 2004.

[6] P. A. Jackson, C. P. Chan, J. E. Scalera, C. M. Reader, and M. M. Vai, "A systolic FFT architecture for real time FPGA systems," *High Performance Embedded Computing Conference (HPEC04)*, Sept. 2004.

[7] T. Dillon, "An efficient architecture for ultra long FFTs in FPGAs and ASICs," *High Performance Embedded Computing Conference (HPEC04)*, Sept. 2004.

[8] S. Akella, D. A. Buell, L. E. Cordova, J. Hammes, "The DARPA Data Transposition Benchmark on a Reconfigurable Computer," *8th International Conference on Military and Aerospace Programmable Logic Devices (MAPLD05)*, Sept. 2005.

[9] L. Zhuo, V. K. Prasanna, "Sparse Matrix-Vector Multiplication on FPGAs," In *Proceedings of the 13th International Symposium on Field Programmable Gate Arrays (FPGA05)*, 2005.

[10] M. deLorimier, A. DeHon, "Floating-point Sparse Matrix-Vector Multiply for FPGAs," *In Proceedings of the 13th International Symposium on Field Programmable Gate Arrays (FPGA05)*, 2005.

[11] SRC Computers, Inc., http://www.srccomp.com.

[12] Xilinx, Inc., Virtex-II Platform FPGAs: Complete Data Sheet, June 2004.

[13] Yong Dou, S. Vassiliadis, G.K. Kuzmanov, G.N. Gaydadjiev, "64-bit floating point FPGA matrix multiplication," *In Proceedings of the 13th International Symposium on Field Programmable Gate Arrays (FPGA05)*, 2005.

[14] Krishna Muriki, Keith D. Underwood, Ron Sass, "RC-BLAST: Towards a Portable, Cost-Effective Open Source Hardware Implementation," *19th International Parallel and Distributed Processing Symposium (IPDPS05)*, 2005.

[15] Keith Underwood, "FPGAs vs. CPUs: Trends in Peak Floating-Point Performance," *In Proceedings of the 12th International Symposium on Field Programmable Gate Arrays (FPGA04)*, 2004.

[16] SGI, http://www.sgi.com.

[17] ATLAS, http://www.netlib.org/atlas/.

[18] C. L. Lawson, R. J. Hanson, D. Kincaid, and F. T. Krogh, "Basic linear algebra subprograms for Fortran usage," *ACM Trans. Math. Soft*., 5 (1979), pp. 308-323.

[19] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson, "Algorithm 656: An extended set of FORTRAN Basic Linear Algebra Subrout*ines*," *ACM Trans. Math. Soft*., 14 (1988), pp. 18-32.

[20] J. J. Dongarra, J. Du Croz, I. S. Duff, and S. Hammarling, "Algorithm 679: A set of Level 3 Basic Linear Algebra Subprograms," *ACM Trans. Math. Soft*., 16 (1990), pp. 18-28.

[21] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, "LAPACK Users' Guide," *Society for Industrial and Applied Mathematics*, Philadelphia, PA, second ed., 1995.