Closing the gap: CPU and FPGA Trends in sustainable floating-point BLAS performance

Keith D. Underwood K. Scott Hemmert

Sandia National Laboratories*
P.O. Box 5800 MS-1110
Albuquerque, NM 87185-1110
E-mail: {kdunder,kshemme}@sandia.gov

Abstract

Field programmable gate arrays (FPGAs) have long been an attractive alternative to microprocessors for computing tasks — as long as floating-point arithmetic is not required. Fueled by the advance of Moore's Law, FPGAs are rapidly reaching sufficient densities to enhance peak floating-point performance as well. The question, however, is how much of this peak performance can be sustained. This paper examines three of the basic linear algebra subroutine (BLAS) functions: vector dot product, matrix-vector multiply, and matrix multiply. A comparison of microprocessors, FPGAs, and Reconfigurable Computing platforms is performed for each operation. The analysis highlights the amount of memory bandwidth and internal storage needed to sustain peak performance with FPGAs. This analysis considers the historical context of the last six years and is extrapolated for the next six years.

KEYWORDS: IEEE floating point, arithmetic, FPGA, reconfigurable computing

1. Introduction

The current trends in FPGA performance produced by the advances in semiconductor technology according to Moore's Law will yield FPGA devices with a factor of three to eight more peak floating-point performance than comparable microprocessors by 2009[21]. Yet, a significant factor in the ability of FPGAs to achieve this advantage is the fact that CPU designers choose not to dedicate silicon area to floating-point units (FPUs) that would go unutilized. Even with careful hand optimization of dense matrix operations, most microprocessors achieve less than 90% of peak performance. These are the "best case" scenarios with many real applications achieving less than 50% of peak and some as low as 5% of peak. In light of this, the question is: will FPGAs be able to leverage their higher peak performance to produce higher sustained floating-point performance?

Traditionally, FPGAs have been shown to sustain much greater performance than microprocessors for a number of integer and fixed-point applications — particularly digital signal processing (DSP) applications. This hails from the ability of FPGAs to sustain a high fraction of peak performance rather than the sheer magnitude of their peak performance. Widely recognized FPGA friendly applications lend themselves to dataflow implementations and require relatively little medium term (longer than the depth of the dataflow pipeline) storage. Unfortunately, scientific applications that use double precision floating-point generally do not have the same characteristics as DSP applications. For example, scientific applications tend to have long inner loops, require a large average number of bytes from memory for each operation, exhibit poor caching behavior, and perform irregular, indirect addressing.

This paper presents a study of FPGA performance on double precision, floating-point, dense matrix operations. The three operations considered are vector dot product (`DDOT`), matrix-vector multiply (`DGEMV`), and matrix multiply (`DGEMM`). Each operation is evaluated to determine the sustainable performance over a range of FPGA devices and the Reconfigurable Computing (RC) platforms that use them. Both DDOT and DGEMV are traditionally memory limited and so this paper considers both what can be implemented (RC platforms) as well as the theoretical maximum (assuming all pins are dedicated to memory). For DGEMM, caching can alleviate memory bandwidth constraints; thus, a parameter study is performed to consider the trade-off between internal storage and memory bandwidth requirements to sustain peak performance. Historical, current, and projected FPGA floating-point performance are covered. A

*Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04)
0-7695-2230-0/04 $ 20.00 IEEE

prototype implementation on an Osiris reconfigurable computing card with a Xilinx Virtex II 6000 FPGA is used as a reference point for validation.

As part of this study, this paper uses sustainable performance on BLAS operations as a metric to compare performance trends between FPGAs and microprocessors. The results show that FPGAs can sustain a much higher percentage of peak than microprocessors for memory bound dense matrix operations by supporting higher memory bandwidth. In addition, for non-memory bound dense matrix operations, FPGAs are now competitive with microprocessors and are poised to overtake them. In the remainder of the paper, related work and platforms evaluated are presented (Sections 2 and 3), each of the three dense matrix operations are analyzed (Sections 4, 5, and 6), and conclusions and future work are presented (Sections 7 and 8 respectively).

2. Related Work

There has been extensive research into floating-point arithmetic on FPGAs. Several efforts[19, 3, 5, 8, 13] have investigated the use of custom floating-point formats in FPGAs. Others have studied translating floating-point to fixed point[12] or automatically optimizing the bit widths of floating-point formats[9] as an alternative. Compared to IEEE standards[11], these formats require significantly less area and run significantly faster. Customized formats enable significant speedups for certain applications, but many scientific applications depend on the dynamic range and high precision of IEEE double-precision floating-point to maintain numerical stability. Thus, this work focuses on the IEEE standard. Indeed, some application developers within the DOE labs are beginning to discuss the need for greater precision than the standard IEEE formats, and such formats may be an aspect of future work.

Early work on IEEE floating-point[7] found that single precision implementations were feasible, but extremely slow. Performance significantly improved[15], but was still uncompetitive. Eventually, FPGAs achieved competitive, sustained, IEEE standard floating-point performance[14]. Since then, a variety of work[16, 3, 13, 22] has demonstrated the growing feasibility of IEEE compliant, single precision floating-point arithmetic and other similarly complex floating-point formats. Indeed, some work[20] suggests that a collection of FPGAs can provide dramatically higher performance than a commodity processor.

A number of works have focused on optimizing the format and the operators to maximize FPGA performance. In [16] a delayed addition technique is used to achieve impressive clock rates. In other work, [3, 13], the details of the floating-point format are varied to optimize performance. The specific issues of implementing floating-point division in FPGAs has been studied[22] as well as mechanisms to leverage new FPGA features to improve general floating-point performance[17], but neither cover double precision. Indeed, only [10] and [21] have covered the performance of IEEE double precision floating-point.

Only a few researchers have studied matrix operations on FPGAs. Specifically, [14] studies several aspects of single precision floating-point matrix multiplication and compares it to a microprocessor. More recently, [10] considers both performance and power issues for double precision floating-point matrix multiplication. To our knowledge, no other work has studied other dense matrix operations (such as `DDOT` or `DGEMV`), trends in sustained FPGA performance, or the memory bandwidth and internal storage that will be needed to maintain those trends.

3. Platforms and Devices

Five FPGAs, three RC platforms, and three microprocessors are evaluated here. These devices and systems were selected to represent the "best available" over the course of several years to enable the extrapolation of trend lines. The devices, platforms, and CPUs are shown in Table 1. The choice of FPGAs and CPUs is explained in [21]. The platforms are chosen to provide real design points that are representative of boards that could be built with the FPGAs available so that system level trends could be considered.[1] In particular, the FPGA chosen for 1997 was not available on the representative platform chosen for 1997, but it is the FPGA that is most representative of FPGAs of the time that could still be placed and routed with the tools available to the authors. A second important point about the 1997 data points in this paper is that they fall well below the FPGA trend lines. This is because one double precision floating-point multiply accumulate is $1.17\times$ the size of one device. Splitting the multiply accumulate over two devices underutilizes the space and yields an artificially low number.

The Annapolis Microsystems WildForce board[2] consists of four Xilinx 4000 series compute FPGAs connected with nearest neighbor links and connected through a global crossbar. Each FPGA has access to a dedicated, four byte wide, 40 MHz SRAM memory. The SLAAC-1V[18] is a research board designed by USC ISI-East with two Xilinx Virtex 1000 compute FPGAs. Each FPGA has four banks of four byte wide, 65 MHZ SRAM memory. The Osiris board[4] is a research board designed by USC ISI-East. It has one Xilinx Virtex-2 FPGA with 10 banks of four byte wide 200 MHz SRAM and one 8 byte wide bank of 133 MHz SDRAM.

The Pentium-II processor is interfaced to PC-66 memory (66 MHz SDRAM). Data for this processor was either estimated (based on the relative memory performance to the

[1] The set may not be ideal, but it was chosen based on author experience and should be adequate for the purpose.

Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04)
0-7695-2230-0/04 $ 20.00 IEEE

Year	Device	Platform	CPU
1997	Xilinx XC4085XLA-09	Annapolis Microsystems WildForce	Pentium-II 266 MHz
1999	Virtex 1000-5	SLAAC-1V	
2000	Virtex-E 3200-7		Athlon 1.0 GHz
2001	Virtex-II 6000-5	Osiris	
2003	Virtex-II Pro 100-6		Pentium-4 3.06 GHz

Table 1. FPGAs, Reconfigurable Computing platforms and CPUs analyzed

Athlon for the vector dot product and matrix-vector multiply) or taken from the web[23] (for matrix multiplication). The 1 GHz Athlon performance was measured using the ATLAS 3.4.1 implementation of the BLAS libraries on a system with PC-133 (133 MHz SDRAM) running Linux 2.2 (on a cluster computing optimized distribution from Scyld corporation). The Pentium-4 measurements used a 533 MHz front side bus and dual channel PC2100 (266 MHz DDR-SDRAM) memory. It was measured with ATLAS 3.4.1 under the RedHat 9 operating system. Neither the Athlon nor the Pentium-4 used represent the absolute fastest version of these processors during the years they represent, but they are within a very small constant factor of it.

Since memory bandwidth is a critical factor in system performance, memory bandwidth trends are graphed in Figure 1. Three sets of data are shown. The first is the trend in CPU memory bandwidth extrapolated out to 2009. The second is the trend in RC platform bandwidth extrapolated out to 2009, which considers actual configurations of FPGA based systems that are being built. Admittedly, these platforms are quite expensive. The final set of data extrapolates the maximum achievable memory bandwidth of a single FPGA. For a single FPGA, it is assumed that the fastest memories available at the time are connected to all of the pins of the device with a 30% control overhead for the period from 1997 to 2000 assuming SRAM devices are used and a 40% control overhead after that (assuming SDRAM devices are used). These estimates attempt to be as conservative as reasonable, however, the projection of maximum achievable memory bandwidth for an FPGA clearly overshoots what was achievable in 2003. These projections may be optimistic given the concern throughout the computing industry that memory performance is not tracking processor performance. However, the projections are well within the pin counts and pin bandwidths projected by the ITRS[1].

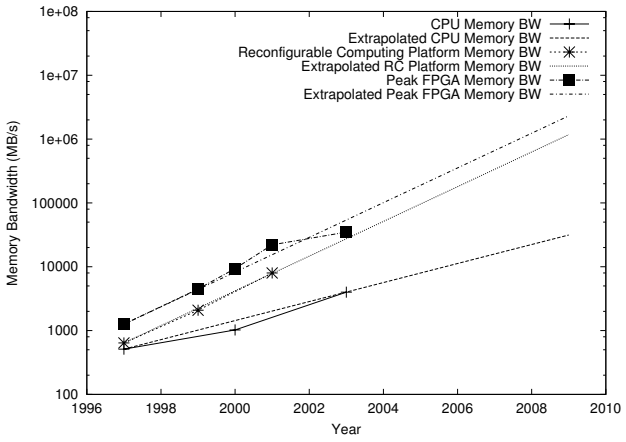


Figure 1. Memory bandwidth of the platforms and devices analyzed

4. Vector Dot Product

The standard vector dot product (the DDOT BLAS routine) is the sum of the pairwise products of two vectors, or:

$$p = \sum_{i=0}^{N-1} \mathbf{x}_i \mathbf{y}_i \tag{1}$$

which requires $2N$ memory accesses to perform $2N$ floating-point operations. Thus, for each data item, one floating-point operation is performed. In modern processors, DDOT is memory bound. That is, although the processor floating-point unit can perform several gigaflops per second, the memory cannot provide data to support it. The maximum sustainable floating-point rate is:

$$FLOPs = \frac{BW}{8} \tag{2}$$

where BW is the memory bandwidth in bytes per second and 8 bytes are required to store a floating-point number. This is graphed in Figure 2 on a log-log graph.

4.1. FPGA Implementation

Implementing vector dot product on an FPGA carries unique challenges. Like many BLAS routines, DDOT is based on multiply accumulate (MACC) operations. On mi-

Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04)
0-7695-2230-0/04 $ 20.00 IEEE

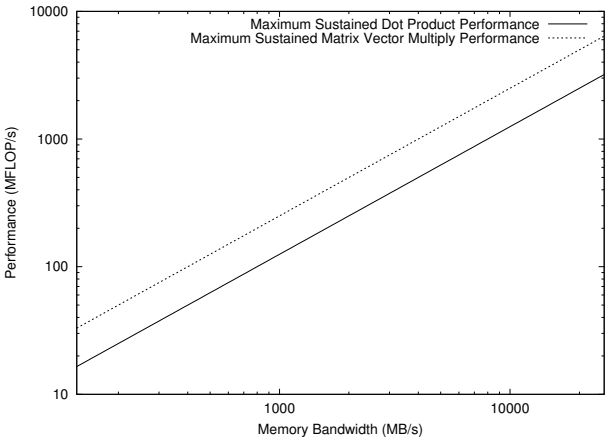


Figure 2. Maximum achievable performance versus memory bandwidth

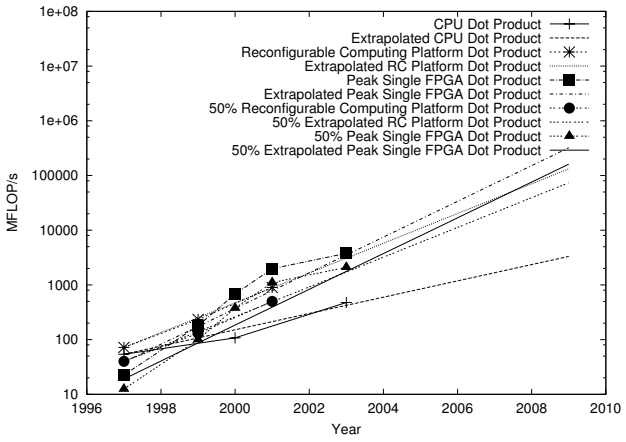


Figure 4. A comparison of double precision floating-point dot product performance on CPUs, FPGAs, and RC platforms

croprocessors, a dot product uses multiple concurrent multiply accumulates to hide the pipeline latency of the floating-point unit. The results from the partial accumulations are summed at the end of the operation. The final summation causes multiple pipeline stalls; however, multi-gigahertz clock rates produce small actual time penalties.

Three challenges face implementors of DDOT on FPGAs:

- the adder pipeline is deeper,
- multiple MACC units are required to fully utilize high bandwidth memory, and
- the clock rate is lower.

These factors require additional control logic inside and outside of the multiply-add and MACC units to efficiently use FPGA resources. First, a multiplier bypass multiplexor (labeled MB) is required in the multiply-add (Figure 3(b)) to reuse the adder for portions of the final summation. Second, the MACC must perform 13 concurrent operations to hide the adder latency. This requires a second feedback path (with associated control) through the FP multiplexor in the MACC (Figure 3(c)) to sum the 13 results. The added logic is shown with dashed lines in Figure 3.

Unfortunately, the long pipeline and low clock rate of floating-point operations in FPGAs cause a performance penalty for smaller vectors. This is generally not a problem since memory latency is a dominant factor for small vector performance. Longer vectors will hide the penalty.

4.2. Performance Comparisons

The performance of DDOT on an FPGA is based on the memory bandwidth, but can also be limited by the floating-point performance of the FPGA. Figure 4 compares the performance of commodity CPUs with that of FPGA platforms and the peak performance possible with a single FPGA that dedicates all pins to memory. CPU performance was measured for the 2000 and 2003 data points with vectors of 1000 elements[2] and estimated for the 1997 data point based on the memory bandwidth and architecture.[3] Note that each of the CPU data points correspond very closely to the memory bandwidth of the processor over 8. When extrapolated, this trend line grows at a rate of $2.8\times$ every 3 years.

Peak DDOT performance is estimated for each FPGA based on the maximum multiply accumulate performance for the FPGA (from [21]). This assumes that all of the FPGA's pins are dedicated to high speed memory (see Figure 1) and assumes that 90% of the floating-point performance can be achieved if adequate memory bandwidth is available. This assumption requires vectors of 7500 elements or more because of the overhead of the final summation through the parallel units. The line for 50% of achievable performance is also plotted and only requires vectors of 1000 elements. This trend extrapolates to $4.5\times$ every two years (see [21]) because the pin bandwidth of the FPGA never becomes the limiting factor.

The other pair of lines presents dot product performance on reconfigurable computing platforms. These platforms often use multiple FPGAs and provide a "realistic" design point with a "realistic" memory bandwidth.[4] Most of these points are estimated, but an implementation has been

[2] Performance was constant across a wide range of vector lengths.

[3] This is a best case estimate assuming the older processor sustained as much of the peak memory bandwidth as newer processors.

[4] Realistic in that people buy it, but not realistic in a cost comparison with traditional processors.

Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04)
0-7695-2230-0/04 $ 20.00 IEEE

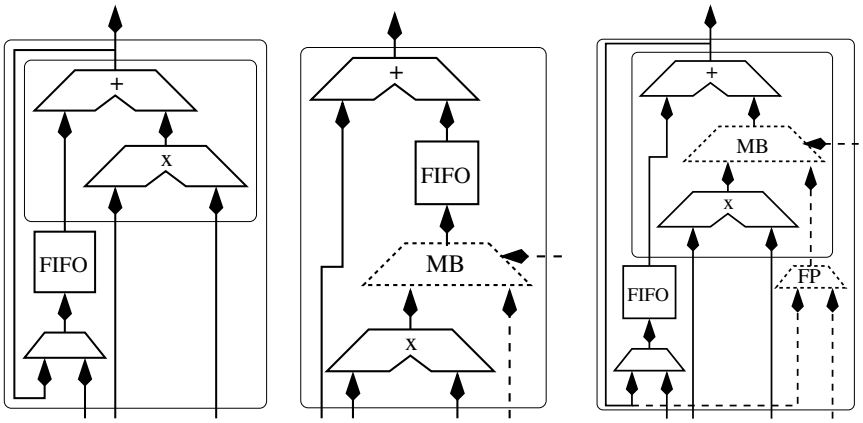


Figure 3. (a) A standard multiply-accumulate; **(b)** A modified multiply-add for dot product; **(c)** A modified multiply-accumulate for dot product

tested on the Osiris board for validation. It is assumed that 90% efficiency can be achieved with the memory accesses, since these platforms use predominantly SRAM. This curve grows at a much slower pace ($3.5\times$ every two years) than FPGA performance because it starts at a point using multiple FPGAs and slowly reduces the total number of FPGAs per board. For each platform, the limitation is the amount of memory bandwidth provided; however, the memory bandwidth is much higher than the competitive CPU giving the RC platform a significant edge in total performance.

5. Matrix-Vector Multiply

The DGEMV BLAS routine is defined as:

$$\mathbf{w}_i = \sum_{j=0}^{N-1} \mathbf{A}_{ij}\mathbf{y}_j + \mathbf{z}_j \tag{3}$$

The lower bound on memory accesses is $N^2 + 3N$[5] and performs $2N^2$ floating-point operations. Achieving that requires that the vector be cached in the processor. In the limit, two floating-point operations are performed for each element retrieved from memory. DGEMV is also a memory limited operation on microprocessors. The maximum sustainable floating-point rate is:

$$FLOPs = \frac{2 \times BW}{8} \tag{4}$$

where BW is the memory bandwidth in bytes per second and 8 bytes are required to store a floating-point number. This is graphed in Figure 2 on a log-log graph.

[5]It must retrieve a matrix and two vectors and store one vector.

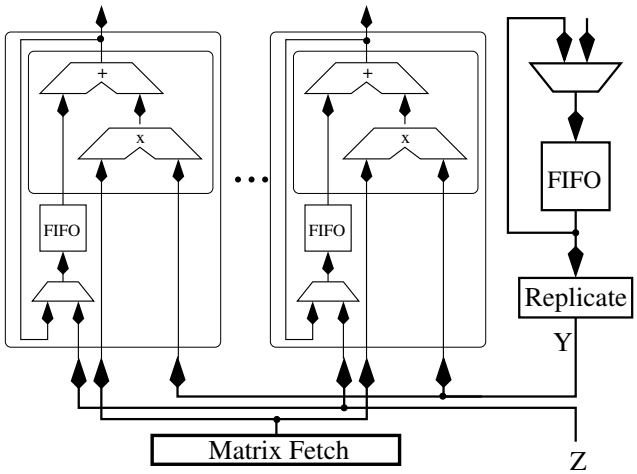


Figure 5. Matrix vector multiplication implementation

5.1. FPGA Implementation

Matrix-vector multiplication has much more inherent parallelism than vector dot product. When the matrix dimension is greater than the depth of the addition pipeline, it is possible to use all of the floating-point capabilities of the FPGA without making special adaptations to the multiply accumulate unit. Instead, the standard MACC of Figure 3(a) is used with additional storage and control outside of it as shown in Figure 5.

The design in Figure 5 assumes that the matrix, **A**, is stored in one logical memory with a bandwidth that will support some number of MACC units, m. Thus, the vectors

Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04)
0-7695-2230-0/04 $ 20.00 IEEE

y and **z** are loaded and broadcast to all of the MACC units while the matrix is fetched from memory and distributed to the MACC units. Before broadcasting to the MACC units, each element of **y** is replicated k times (where k is the pipeline depth of the adder). Effectively, there are k rows of the matrix being multiplied with the vector in each MACC unit. The vector **z** is used to initialize the summation unit. Thus, one MACC unit is fed the j^{th} element of k rows to match k copies of the j^{th} element of **y**. The matrix data can be fetched directly in this order from SRAM memories or can be fetched in larger pieces from SDRAM with re-ordering performed in the matrix fetch unit. Note that, once loaded, **y** is reused (through the feedback path) $\frac{N}{k \times m}$ times. This design does not apply the scalar multiples to **y** and **z** that are indicated in the standard, but such a change would be relatively simple.

The limitation of this approach is the need to store a vector for large values of N. The alternative is to divide the vector into parts of length L and perform the equivalent of $\frac{N}{L}$ smaller matrix-vector multiplies. The first would add the vector **z** to the result (as in Equation 3) and subsequent portions of the operation would add the intermediate result in place of **z**. This would increase the number of memory accesses to $N^2 + \frac{3N^2}{L}$. For FPGAs from 1999 and forward, an L value of 2000 is achievable. Thus, the N^2 term for the matrix access still dominates. This issue would only serve to decrease performance estimates for earlier FPGAs (thus skewing the trend line) and will not be considered further.

5.2. Performance Comparisons

Like vector dot product, matrix-vector multiplication is typically a memory bound operation; however, devices that provide enough memory bandwidth shift the limitation to the floating-point performance of the device. The performance of recent RC platforms is compared with recent commodity CPUs and the peak performance possible with a single FPGA in Figure 6. CPU performance was measured for the 2000 and 2003 data points with an N of 1000[6] and estimated for the 1997 data point based on the differences in memory bandwidth and architecture.[7] In all cases except the 2003 data point, this corresponds very closely to twice the dot product performance. It is unclear why the Pentium-4 failed to achieve a similar improvement, but this reduces the performance growth rate to just over $2.3\times$ every 3 years.

Like `DDOT`, peak `DGEMV` for the FPGAs is based on the maximum multiply accumulate performance of the FPGA. This is because the use of all of the FPGA pins for memory provides abundant bandwidth to achieve the peak performance. Unlike `DDOT`, `DGEMV` has sufficient independent parallelism to enable the FPGA to achieve its full peak performance on smaller matrix and vector sizes. Extrapolating this trend yields the same $4.5\times$ growth every two years as the dot product since the FPGA pin bandwidth never becomes the limiting factor.

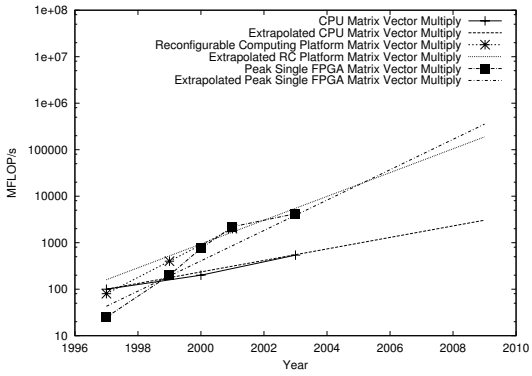


Figure 6. A comparison of double precision floating-point matrix-vector multiplication performance on CPUs, FPGAs, and RC platforms

Recent reconfigurable computing platforms are also included in this comparison. Most points are estimated, but an implementation on the Osiris board to validate the estimates has been simulated. Using the same assumptions used for the vector dot product yields a performance growth rate of $3.25\times$ every two years. This is lower than the FPGA growth rate because the number of FPGAs on RC platforms has been steadily decreasing. This growth rate may accelerate over the coming years (since less than one chip per board is unlikely). Nonetheless, since 1999, RC platforms have had both higher performance than commodity CPUs and a higher rate of growth. As a note, the performance for the 1997 RC platform data point is particularly low because the XC4000 series parts did not have sufficient internal storage to store a vector of any significant length. This limits the performance of that platform to slightly higher than the dot product performance. The slight improvement over dot product performance is achieved by having a higher inherent degree of parallelism that prevents the loss in efficiency seen with the dot product. This would give the trend line an artificially low starting point and artificially steep slope[8], but the trend is established based on the performance the 1997 part would achieve if it had internal storage.

The final point for discussion is the amount of storage needed to achieve this performance. As noted, the XC4000 series had insufficient internal storage to achieve the full potential of the 1997 RC platform. The Virtex 1000 part,

[6] Performance was constant across a wide range of values for N.

[7] This is a best case estimate assuming the older processor could sustain as much of the peak memory bandwidth as the newer processors.

[8] Although the embedded memories in newer FPGAs could be considered an architectural improvement (making this valid), similar improvements for matrix-vector multiplication are unlikely to occur.

Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04)
0-7695-2230-0/04 $ 20.00 IEEE

however, has enough storage for 1000 vector elements. The Virtex-2 6000 has enough storage for 10000 vector elements and the Virtex-2Pro series has even more storage. As long as the total storage in an FPGA is not decreased (an unlikely scenario), FPGAs already have abundant storage for matrix-vector multiplication.

6. Matrix Multiply

The standard matrix multiply (the `dgemm` BLAS routine) is defined as:

$$\mathbf{C}_{ij} = \sum_{k=0}^{N-1} \mathbf{A}_{ik}\mathbf{B}_{kj} + \mathbf{C}ij \tag{5}$$

In the best case, this requires $4N^2$ memory accesses[9] and performs $2N^3$ floating-point operations. This yields $\frac{N}{2}$ floating-point operations for each element retrieved from memory. Achieving the best case, however, imposes the unrealistic requirement that two matrices be cached in the processor. Fortunately, proper use of caching in modern processors allows them to sustain a high percentage of peak with relatively low memory bandwidth. If each matrix only had to be retrieved from memory once, the maximum sustainable floating-point rate would be:

$$FLOPs = \frac{\frac{N}{2} \times BW}{8} \tag{6}$$

where BW is the memory bandwidth in bytes per second, N is the dimension of the matrix, and 8 bytes are required to store a floating-point number. This is graphed in Figure 7 on a log-log graph.

Typically, however, the processor cannot store all of the matrices involved. Instead, some form of blocking is used to divide the matrix into smaller pieces[24]. These smaller pieces are loaded into the processor, the computations are performed on them, and the partial results stored. For example, for a 64×64 matrix multiply, each matrix might be broken into 64 regions that are 8×8. A row of these blocks would then be multiplied by a column of these blocks to create an 8×8 block of the result. In the process, the partial result (an 8×8 block) would be updated 8 times (although typically in local storage or cache). The ultimate result is that the matrices are fetched several times more than would otherwise be necessary. For blocks of dimension S, this yields a factor of $\frac{N}{S}$ increase in accesses to the **A** and **B** matrices, leading to $2N^2 + \frac{2N^3}{S}$ memory accesses. For large matrices, this approaches a floating-point rate of:

$$FLOPs = \frac{S \times BW}{8} \tag{7}$$

[9]This assumes square matrices and includes retrieving three matrices and storing one matrix.

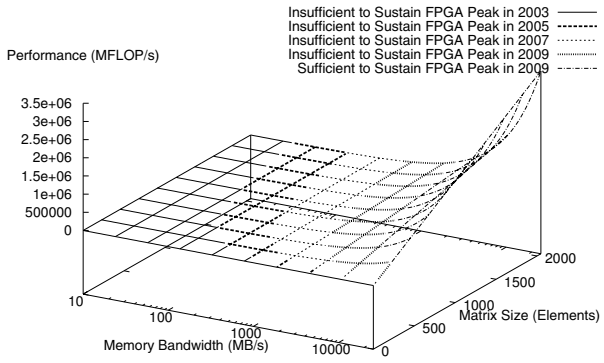


Figure 7. Maximum achievable performance versus memory bandwidth and matrix size

6.1. FPGA Implementation

One way to view matrix multiplication is as a collection of N matrix-vector multiplications. As such, it exhibits significantly more parallelism. Unfortunately, it is impractical to store all of a matrix in most modern devices (with the possible exception of the Itanium chips with the largest caches) for reasonable matrix sizes. Thus, the implementation chosen (shown in Figure 8) resembles a collection of matrix-vector multiplications, but blocks the matrix to reduce total storage requirements.

To perform the matrix multiplication, a $S \times \frac{S}{m}$ section of a block of the matrix **B** in column major order is loaded into each of m MACC units. Simultaneously, the matching block of the matrix **C** is also loaded into the MACC unit in column major order. This requires a total of $6S^2$ elements of storage at 8 bytes per element. The corresponding block of **A** is loaded into a FIFO that is used to broadcast it to all MACC units $\frac{S}{m}$ times (also in column major order). Each MACC unit creates S replicas of each element of **B** to match the number of rows of **A** that will be multiplied with it. This provides the concurrency needed to hide all of the latency of the adder. As each element of **B** and **C** is used, it is discarded. When the MACC unit finishes, it produces an intermediate version of the result **C**. This intermediate is fed back to the input to be added to the multiplication of the next pair of blocks from **A** and **B**. When the final version of a block of **C** is produced, it is stored. Overall, this requires no more than $6S^2$ elements of storage at 8 bytes per element. This includes 2 copies of each matrix block — one to operate on and one to change it from row major to column major order.

Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04)
0-7695-2230-0/04 $ 20.00 IEEE

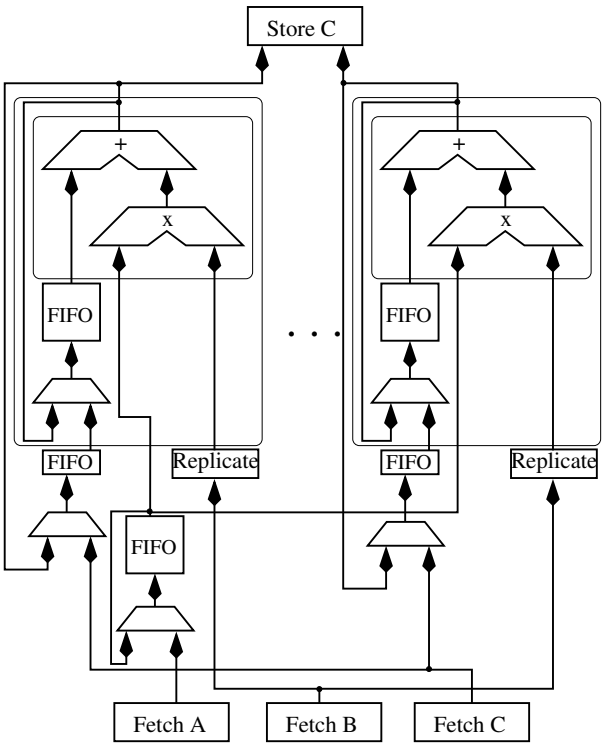


Figure 8. Matrix multiply implementation

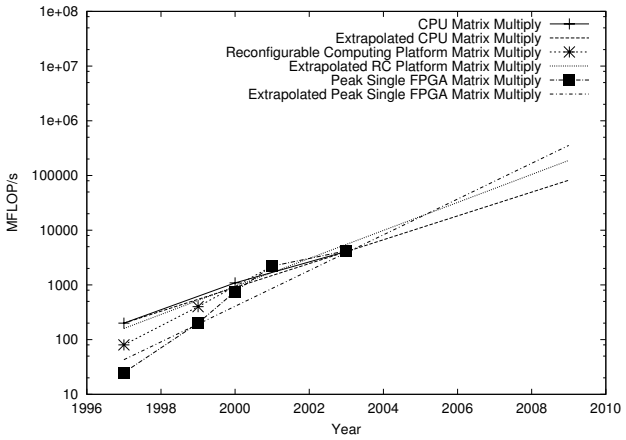


Figure 9. A comparison of double precision floating-point matrix-vector multiplication performance on CPUs, FPGAs, and RC platforms

6.2. Performance Comparisons

Unlike vector dot product and matrix-vector multiplication, matrix multiply is not typically memory bound. With relatively little caching, modern microprocessors can achieve a high percentage of peak performance with commodity memory attached. As seen in Figure 9, this enables microprocessors to maintain an edge over FPGAs and RC platforms up through 2003. For this graph, data points for the CPUs in 2000 and 2003 were measured and the data point for the CPU in 1997 was taken from [23]. The extrapolated trend line projects a $4.5\times$ growth in performance every 3 years.

FPGA performance on matrix multiply has been equivalent to matrix-vector multiply because FPGAs can provide sufficient bandwidth to maintain peak performance on matrix-vector multiply. As such, the rate of growth ($4.5\times$ every two years) is also the same as for matrix-vector multiply and, indeed, the same as for the multiply-add performance projected in [21].

Reconfigurable computing platforms also demonstrate the same characteristics (peak performance and performance growth) as matrix-vector multiply. Again, all but one of these points are currently estimated, but an implementation has validated one data point on an Osiris board. As with matrix-vector multiply and dot product, the growth rate has been lower than what might be expected in the future since the number of devices per board is unlikely to keep shrinking; however, even the current trends indicate that RC platforms will outperform microprocessors in the near future.

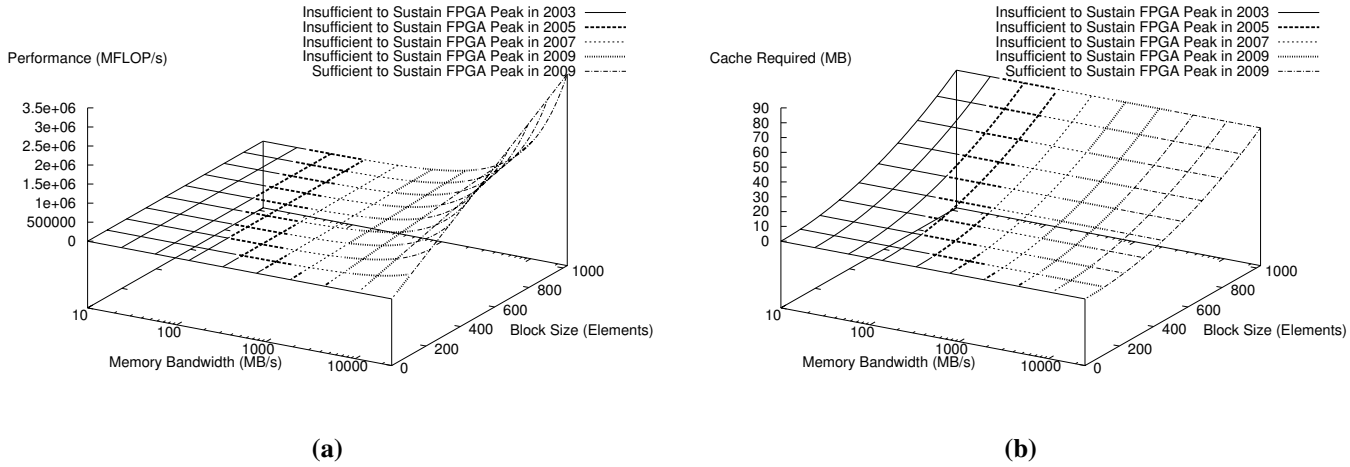
6.3. Memory Requirements

The key feature that distinguishes matrix multiplication from matrix-vector multiplication is that it requires much less memory bandwidth to maintain peak performance. This is an important factor since supercomputing applications of FPGAs are unlikely to provide the level of memory bandwidth that many RC platforms provide (due to the additional cost and negative reliability implications of several extra banks of memory). As such, it is important to examine the storage needs of FPGAs relative to their peak bandwidth and peak performance.

Figure 10(a) illustrates the high levels of performance achievable with relatively little memory bandwidth and sufficient internal storage. For example, the 4 GFLOP/s of peak floating-point capability available in 2003 can be sustained with only 512 MB/s of memory bandwidth and only 192 KB of internal storage in the FPGA. This easily falls within the limits of even the Virtex-II family. The important observation here is that internal memory can be traded for external memory bandwidth.

The flip side of this analysis is to consider the amount of internal memory needed for various memory bandwidth and block size combinations. Figure 10(b) compares the size of the internal cache needed to the amount of memory bandwidth needed and the block size. By 2009, FPGA platforms

Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04)
0-7695-2230-0/04 $ 20.00 IEEE



(a) **(b)**

Figure 10. (a) Maximum achievable performance versus memory bandwidth and block size; (b) Memory needed in an FPGA to maintain peak performance versus memory bandwidth and block size

	CPU	FPGA	RC Platform
ddot	Memory	FLOPS	Memory
dgemv	Memory	FLOPS	FLOPS
dgemm	FLOPS	FLOPS	FLOPS

Table 2. Performance constraint

will need to provide several gigabytes per second of memory bandwidth or several megabytes of internal storage. For this algorithm, however, both the memory bandwidth and internal storage needed to maintain peak performance appear to be easily achievable.

An interesting revelation from this analysis is the significant gains in efficiency that FPGAs achieve. These gains appear to arise from the ability to explicitly manage internal storage and the high aggregate internal memory bandwidth. Commodity processors typically achieve only 80-90% of their peak performance on dense matrix multiplies while requiring hundreds of kilobytes of cache and gigabytes per second of memory bandwidth. By contrast, the FPGA only requires 192 KB of storage and 512 MB/s of external memory bandwidth. If this trend holds for other algorithms, it will be a significant advantage for FPGAs in the realm of high performance computing.

7. Conclusions

In summary, FPGAs and Reconfigurable Computing platforms are able to significantly outperform modern microprocessors on memory bandwidth sensitive double precision floating-point operations, including vector dot product and matrix-vector multiplication (the `ddot` and `dgemv` BLAS operations). Indeed, unlike CPUs, FPGAs are often limited by peak FLOPs rather than by memory bandwidth (see Table 2). The trends indicate that this has been the case for several years and that the performance gap will widen. For matrix multiplication (the `dgemm` BLAS operation), commodity processors still hold a slight edge thanks to their significantly higher peak performance; however, FPGAs can achieve a higher percentage of peak with less memory bandwidth and less internal storage. When combined with trends that indicate that the peak double precision floating-point performance of FPGAs will soon outstrip commodity CPUs, this offers great promise for the future of FPGAs in scientific computing. This is an initial implication that FPGAs will not hit the memory wall as soon as commodity processors giving them an edge in the future of high performance computing.

8. Future Work

The BLAS routines are widely used by high performance computing applications. Indeed, they are at the core of the popular LINPACK benchmark[6]. Thus, they provide a good initial metric for comparing the sustained floating-point capabilities of FPGAs and CPUs; however, these dense matrix operations are not characteristic of the scientific computing workload within the Department of Energy (DOE) Advanced Simulation and Computing (ASCI) program. In the future, we will be considering multi-dimensional fast Fourier transforms (FFTs) and sparse matrix solvers. We will also be analyzing how future super-

Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04)
0-7695-2230-0/04 $ 20.00 IEEE

computers might leverage the capabilities of FPGAs.

References

[1] *International Technology Roadmap for Semiconductors*. December 2003.

[2] Annapolis Micro Systems, Inc. WILDFORCE reference manual, 1999.

[3] P. Belanovic and M. Leeser. A library of parameterized floating-point modules and their use. In *Proceedings of the International Conference on Field Programmable Logic and Applications*, 2002.

[4] P. Bellows, V. Bhaskaran, J. Flidr, T. Lehman, B. Schott, and K. D. Underwood. GRIP: A reconfigurable architecture for host-based gigabit-rate packet processing. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, April 2002.

[5] J. Dido, N. Geraudie, L. Loiseau, O. Payeur, Y. Savaria, and D. Poirier. A flexible floating-point format for optimizing data-paths and operators in fpga based dsps. In *Proceedings of the ACM International Symposium on Field Programmable Gate Arrays*, Monterrey, CA, February 2002.

[6] J. J. Dongarra. The linpack benchmark: An explanation. In *1st International Conference on Supercomputing*, pages 456–474, June 1987.

[7] B. Fagin and C. Renard. Field programmable gate arrays and floating point arithmetic. *IEEE Transactions on VLSI*, 2(3):365–367, 1994.

[8] A. A. Gaar, W. Luk, P. Y. Cheung, N. Shirazi, and J. Hwang. Automating customisation of floating-point designs. In *Proceedings of the International Conference on Field Programmable Logic and Applications*, 2002.

[9] A. A. Gaar, O. Mencer, W. Luk, P. Y. Cheung, and N. Shirazi. Floating point bitwidth analysis via automatic differentiation. In *Proceedings of the International Conference on Field Programmable Technology*, Hong Kong, 2002.

[10] G. Govindu, L. Zhuo, S. Choi, P. Gundala, and V. K. Prasanna. Area and power performance analysis of a floating-point based application on FPGAs. In *Proceedings of the Seventh Annual Workshop on High Performance Embedded Computing (HPEC 2003)*, September 2003.

[11] IEEE Standards Board. IEEE standard for binary floating-point arithmetic. Technical Report ANSI/IEEE Std. 754-1985, The Institute of Electrical and Electronics Engineers, New York, 1985.

[12] M. P. Leong, M. Y. Yeung, C. K. Yeung, C. W. Fu, P. A. Heng, and P. H. W. Leong. Automatic floating to fixed point translation and its application to post-rendering 3d warping. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 240–248, Napa Valley, CA, April 1999.

[13] J. Liang, R. Tessier, and O. Mencer. Floating point unit generation and evaluation for fpgas. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 185–194, Napa Valley, CA, April 2003.

[14] W. B. Ligon, S. P. McMillan, G. Monn, F. Stivers, K. Schoonover, and K. D. Underwood. A re-evaluation of the praticality of floating-point on FPGAs. In *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, pages 206–215, Napa Valley, CA, April 1998.

[15] L. Louca, T. A. Cook, and W. H. Johnson. Implementation of IEEE single precision floating point addition and multiplication on FPGAs. In *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, pages 107–116, 1996.

[16] Z. Luo and M. Martonosi. Accelerating pipelined integer and floating-point accumulations in configurable hardware with delayed addition techniques. *IEEE Transactions on Computers*, 49(3):208–218, 2000.

[17] E. Roesler and B. Nelson. Novel Optimizations for Hardware Floating-Point Units in a Modern FPGA Architecture. In *Proceedings of the 12th International Workshop on Field Programmable Logic and Applications (FPL'2002)*, pages 637–646, August 2002.

[18] B. Schott, P. Bellows, M. French, and R. Parker. Applications of adaptive computing systems for signal processing challenges. In *Proceedings of the Asia South Pacific Design Automation Conference*, 2003.

[19] N. Shirazi, A. Walters, and P. Athanas. Quantitative analysis of floating point arithmetic on FPGA based custom computing machines. In *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, pages 155–162, 1995.

[20] W. D. Smith and A. R. Schnore. Towards and RCC-based accelerator for computational fluid dynamics applications. In *Proceedings of The International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'03)*, pages 226–232, 2003.

[21] K. D. Underwood. FPGAs vs. CPUs: Trends in peak floating-point performance. In *Proceedings of the ACM International Symposium on Field Programmable Gate Arrays*, Monterrey, CA, February 2004.

[22] X. Wang and B. E. Nelson. Tradeoffs of designing floating-point division and square root on Virtex FPGAs. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 195–203, Napa Valley, CA, April 2003.

[23] R. C. Whaley. Automatically tuned linear algebra software (ATLAS), Jan. 2004. From webpage at http://www.linuxclustersinstitute.org/Linux-HPC-Revolution/Archive/PDF00/Whaley.pdf.

[24] R. C. Whaley, A. Petitet, and J. J. Dongarra. Automated empirical optimizations of software and the ATLAS project. *Parallel Computing*, 27(1–2):3–35, 2001.

Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04)
0-7695-2230-0/04 $ 20.00 IEEE