A MODEL FOR MATRIX MULTIPLICATION PERFORMANCE ON FPGAS

Colin Yu Lin[*], *Hayden Kwok-Hay So*

Electrical and Electronic Engineering
The University of Hong Kong
Hong Kong
email: {linyu, hso}@eee.hku.hk

Philip H.W. Leong

Electrical and Information Engineering
The University of Sydney
Sydney, Australia
email: philip.leong@sydney.edu.au

ABSTRACT

Computations involving matrices form the kernel of a large spectrum of computationally demanding applications for which FPGAs have been utilized as accelerators. Their performance is related to their underlying architectural and system parameters such as computational resources, memory and I/O bandwidth. A simple analytic model that gives an estimate of the performance of FPGA-based sparse matrix-vector and matrix-matrix multiplication is presented, dense matrix multiplication being a special case. The efficiency of existing implementations are compared to the model and performance trends for future technologies examined.

1. INTRODUCTION

Dense and sparse linear algebra routines are extensively used in scientific computing and form the computational kernel of many applications in domains as diverse as computational fluid dynamics, optimization, circuit simulation, financial modelling, graph theory and acoustics.

Due to the importance of this problem, field programmable gate arrays (FPGAs) as compute accelerators for matrix operations have been studied. These designs have been able to combine parallel floating point operations with high memory bandwidth in order to achieve higher performance than microprocessors.

The performance of such accelerators, in turn, depends on a number of factors: from high-level issues such as the architecture of the computing device, memory organization, the schedule of operations and data I/O; to low-level details such as I/O bandwidth, memory bandwidth and integrated circuit technology. Given such complexity, performance metrics are most accurately obtained through the measurement of actual implementations on real hardware platforms.

In spite of the large amount of work involved, this approach only gives a single point in the design space. The implementation quality can therefore be quite difficult to judge, especially in cases where there are few prior results to compare. Equally importantly, it is often difficult to extrapolate the results to different architectures and technologies.

In an attempt to address the issues above, we present a performance model for two key sparse computational kernels: matrix-vector and matrix-matrix multiplications on FPGAs. The upper bounds on performance are expressed in terms of the amount of available compute resources, on-chip memory, and off-chip I/O bandwidth. Dense matrices are a special case of the model. Previously published results are also compared. Based on the model, trends for future FPGA matrix computation performance are extrapolated.

The main contributions of this work can be summarized as follows:

- A model of the upper bounds for performance of matrix-vector multiplication and matrix-matrix multiplication for dense and sparse matrices;
- Identification and prediction of performance bottlenecks of the matrix primitives in current and future technologies;
- Application of the model to explore the effects of technological improvements on matrix performance.

2. MATRIX OPERATIONS ON FPGAS

This section summarizes some key prior works on computation and I/O schemes for FPGA-based matrix operations. These will form the foundation of our model which is described in the next section.

2.1. Dense Matrix Multiplications

Dense matrix multiplications on FPGAs have been studied for over a decade. In [1], a set of multiply-accumulators (MACCs) was used for matrix-matrix multiplication. The result matrix was divided into columns, and each MACC was responsible for multiple columns of the results. Reference [2] presented a detailed architecture design using the

[*]This work was supported in part by the Research Grants Council of Hong Kong project GRF 717009.

same computation scheme. A scheme to minimize I/O operation was also presented to support such computation.

A study of the performance of dot product, matrix-vector and matrix-matrix operations for dense matrices was presented in 2004 [3]. This study was similar to our work in that different bottlenecks were identified, past trends analyzed and the performance of future systems extrapolated.

As pointed out in references [3] and [4], the FPGA on-chip memory was becoming the limiting factor for floating point matrix-matrix multiplication because on-chip memory was no longer able to store all data required and generated for the whole multiplication. Block decomposition was therefore required to divide the multiplication into smaller block matrix-matrix multiplications. In [5], a square block decomposition method with efficient on-chip memory utilization and minimum I/O operations was presented.

2.2. Sparse Matrix-Vector Multiplication

Most previous works focused on accelerating sparse matrix-vector multiplications as given by $y = Ax$, where x and y are dense vectors, and A is a sparse matrix.

In one of the first reported implementations [6], each result element in y was computed by summing the partial products via a reduction circuit. Since the parallel multipliers were limited to producing partial results of the same result element in y, the multipliers were under-utilized in certain cases leading to sub-optimal compute efficiency.

In [7] and [8], a different computation scheme was used. Each result element was assigned to a different MACC. When a MACC finished a data element, the next unassigned one was allocated, this being repeated until all the results were obtained. The overall computation efficiency was therefore improved as none of the MACCs were idle during the computation. Moreover, use of a MACC rather than a multiplier allowed the final summation step to be eliminated.

In terms of data I/O, all the above designs initially stored the sparse matrix A in off-chip memory and the dense vector y on-chip. Non-zero data elements of A were then read in during the course of computation. In [9], however, it was assumed that both A and y were initially stored on-chip. Similar to previous designs, MACCs were used to carry out the compute tasks, but in contrast to the previous cases, a high-level synthesis technique was developed to assign operations to the MACCs. As a result, the computation of a result element could end up being carried out on different MACCs, and extra on-chip communications become necessary.

2.3. Sparse Matrix-Matrix Multiplication

A sparse matrix-matrix multiplication is given by $C = AB$ where A is sparse and B is dense. Unlike sparse matrix-vector multiplication, very limited work has been performed to accelerate sparse matrix-matrix multiplication on FPGAs.

We are only aware one previous work addressing this problem [10]. This implementation used a similar computation, I/O and block decomposition scheme to the dense multiplier design in [5]. Extra circuits were incorporated to handle the index information of non-zero data elements from the sparse matrix and to control on-chip memory access.

3. PERFORMANCE MODEL

Although the performance of matrix computations on FPGAs depend on numerous factors such as data I/O scheduling, there is fundamental limit on its performance that depends on architectural parameters such as I/O bandwidth and resource availability. A model that relates theoretical peak performance to these parameters is developed in this section. Performance is measured by its throughput, which is expressed as the number of basic compute operations completed per second, denoted **OPS**. The data type is not specified as it does not affect the theoretical performance model.

We assume that all input data are initially stored off-chip and must be loaded onto the FPGA fabric during the operation. Similarly, the results of the computation are written to off-chip memory. An aggregated bandwidth of b is available between off-chip memories and the FPGA. Once transferred to the FPGA, m words of on-chip memory are available for holding temporary results. For the sake of peak performance modeling, on-chip memories are assumed to have infinite bandwidth. Therefore, any compute element may take data from any memory location at any time.

MACCs are used exclusively as compute elements. Each MACC is able to complete one multiply-accumulate operation per cycle, and operates at a frequency of f Hz. To construct a MACC on the target FPGA, r_{macc} units of resource must be used. Assuming a total of r units are available, the number of MACC units available will be $k = \left\lfloor \frac{r}{r_{macc}} \right\rfloor$.

Without loss of generality, matrices are assumed to be $n \times n$ in size. We denote the density of non-zero elements in a sparse matrix α, i.e. if the number of non-zero elements is $\overline{N}$ and the total number of elements is N, then $\alpha = \frac{\overline{N}}{N}$. For sparse matrices, $0 < \alpha \ll 1$, Dense matrices are handled by $\alpha = 1$ and are a special case of our sparse model.

Finally, since we are mainly concerned with high performance computations on large matrices, we assume that $n, m \gg k > 1$.

3.1. Compute and I/O-Memory Performance Bounds

The performance of an FPGA matrix computation depends on two important sets of architectural parameters: compute resources and I/O bandwidth. The amount of compute resources available clearly limits the maximum possible number of operations committed each cycle. We call this the computational bound, $\mathbf{OPS}_{comp}$. Since each MACC generates one result each cycle by performing 1 add and 1 multiply operation, the peak performance is:

$$\mathbf{OPS}_{comp} = 2kf. \tag{1}$$

Since input data must be read from off-chip memories before any computation can be performed, overall performance may be also be limited by data I/O bandwidth. Even assuming perfect data I/O scheduling, limited on-chip memory resources dictate the amount of data I/O that can be performed. We call this the I/O-memory bound:

$$\mathbf{OPS}_{io} = \frac{Q_{op}}{Q_{io}} \cdot b \tag{2}$$

where Q_{op} denotes the number of compute operations required and Q_{io} denotes the number of I/O operations required. Since the FPGA performance is limited by either the compute or the I/O-memory bound, (1) and (2) give an upper bound on performance:

$$\mathbf{OPS}_{max} = \min\left\{2kf,\ \frac{bQ_{op}}{Q_{io}}\right\}. \tag{3}$$

3.2. Matrix-Vector Product

Independent of the data I/O and computation schedule, the number of compute operations is given by:

$$Q_{op} = 2\alpha n^2. \tag{4}$$

In contrast, Q_{io} depends on n, as well as the data I/O schedule, which in turn, is a function of on-chip memory size m. For large matrices, block decomposition is usually employed so that the current set of data can fit within the on-chip memory. In this work, we assume that A is decomposed into $\frac{n^2}{\mu_r\mu_c}$ sub-matrices of size $\mu_r \times \mu_c$. Correspondingly, x and y are decomposed into $\mu_c \times 1$ and $\mu_r \times 1$ blocks respectively for each block matrix-vector multiplication.

In a block matrix-vector product, the number of non-zero data elements read from sub-block $A_{r,c}$ is $\alpha\mu_r\mu_c$. The corresponding sub-vector of x, x_c, must also be read from off-chip memory. Note that this is unnecessary if all the data elements in the i-th column of $A_{r,c}$ are zero. The probability that at least one non-zero element exists in a column of $A_{r,c}$ is given by $\beta_x = 1-(1-\alpha)^{\mu_r}$. Consequently, the expected number of input data elements needed from x_c is $\beta_x\mu_c$.

In addition, the vector y must be stored to off-chip memory and this consumes data bandwidth. Assuming $m \geq \mu_r$, it is possible to schedule the compute operations so that the sub-vector y_c is only stored once at the end of the computation. Again, this is unnecessary if all data elements in the i-th row of $A_{r,c}$ are zero, and the probability that at least one non-zero element exists is given by $\beta_y = 1-(1-\alpha)^n$. The number of output I/O is thus $\beta_y n$, yielding:

$$Q_{io} = (\alpha\mu_r\mu_c + \beta_x\mu_c)\cdot\frac{n^2}{\mu_r\mu_c} + \beta_y n = \alpha n^2 + \frac{\beta_x n^2}{\mu_r} + \beta_y n. \tag{5}$$

To minimize Q_{io}, μ_r should be maximized by setting $\mu_r = m$. As a result, from (3), (4) and (5),

$$\mathbf{OPS}_{max} = \min\left\{2kf,\ \frac{2b}{c_{mv}}\right\} \tag{6}$$

where $c_{mv} = 1 + \frac{1}{\alpha}\left(\frac{\beta_x}{m} + \frac{\beta_y}{n}\right)$.

3.3. Matrix-Matrix Product

The sparse matrix-matrix product is given by $C = AB$, where A is an $n \times n$ sparse matrix with a density of non-zero data elements equals to α. B and C are usually dense $n \times l$ matrices where $1 < l \ll n$.

Similar to the case of sparse matrix-vector multiplication, independent of data I/O and compute schedule, the total number compute operations is given by:

$$Q_{op} = 2\alpha n^2 l. \tag{7}$$

The number of data I/O is similarly dependent on the block decomposition scheme and I/O schedule. Assume that matrices A, B and C are decomposed into $\mu_{Ar} \times \mu_{Ac}$, $\mu_{Br} \times \mu_{Bc}$ and $\mu_{Ar} \times \mu_{Bc}$ sub-matrices respectively. It follows that $\mu_{Ac} = \mu_{Br} = \mu_s$. We assume the on-chip memory is large enough to hold partial results of one block of matrix C during the computation, i.e.,

$$m \geq \mu_{Ar}\mu_{Bc}\ . \tag{8}$$

Q_{in} is the amount of data that needs to be transferred from A and B. For each sub-matrix of A, $\alpha\mu_{Ar}\mu_s$ elements of data must be read. If all the data in the i-th column of A are zero, it is unnecessary to read in the i-th row in the corresponding blocks from matrix B. If the probability of a non-zero column in the sparse block A is $\beta_B = 1-(1-\alpha)^{\mu_{Ar}}$, then the expected number of input data elements from the corresponding block of matrix B is $\beta_B\mu_s\mu_{Bc}$. The total number of block matrix-matrix products is $\frac{n^2 l}{\mu_{Ar}\mu_s\mu_{Bc}}$, so $Q_{in} = (\alpha\mu_{Ar}\mu_s + \beta_B\mu_s\mu_{Bc})\frac{n^2 l}{\mu_{Ar}\mu_s\mu_{Bc}}$.

Data output is similar to the matrix-vector product case. The total number of data output $Q_{out} = \beta_C nl$, where $\beta_C = 1-(1-\alpha)^n$, and the total amount of data I/O is given by:

$$Q_{io} = Q_{in} + Q_{out} = \left(\frac{\alpha}{\mu_{Bc}} + \frac{\beta_B}{\mu_{Ar}}\right)n^2 l + \beta_C nl \tag{9}$$

From (8) and (9), we can see that μ_s has no effect on the FPGA on-chip memory utilization and compute performance. Therefore, $\mu_s = n$ should be satisfied to avoid unnecessary I/O. To minimize Q_{io} in (9) subject to (8), the

Table 1. Summary of matrix multiplication model

Operation	Condition	Limiting Factor	OPS$_{max}$
Matrix-Vector Product	$kf < \frac{b}{c_{mv}}$	computation	$\left\lfloor \frac{R}{r_{macc}} \right\rfloor \cdot 2f$
	$kf > \frac{b}{c_{mv}}$	on-chip memory and IO bandwidth	$\frac{2B}{c_{mv}}$
Matrix-Matrix Product	$2kf < \frac{\sqrt{mb}}{c_{mm}}$	computation	$\left\lfloor \frac{R}{r_{macc}} \right\rfloor \cdot 2f$
	$2kf > \frac{\sqrt{mb}}{c_{mm}}$	on-chip memory and IO bandwidth	$\frac{\sqrt{MB}}{c_{mm}}$

following block sizes should be chosen:

$$\mu_{Ar} = \sqrt{\frac{\beta_B m}{\alpha}}, \quad \text{and} \quad \mu_{Bc} = \sqrt{\frac{\alpha m}{\beta_B}}. \tag{10}$$

In the case when $\alpha = 1$, $\beta_B = \beta_C = 1$, and $\mu_{Ar} = \mu_{Bc} = \sqrt{m}$, showing that a square block decomposition of C gives the best results. This is consistent with previous works.

Finally, from (3), (7), (9) and (10), we have

$$\textbf{OPS}_{max} = \min\left\{2kf, \frac{\sqrt{mb}}{c_{mm}}\right\} \tag{11}$$

where $c_{mm} = \sqrt{\frac{\beta_B}{\alpha}} + \frac{\sqrt{m}\beta_C}{2\alpha n}$.

3.4. Summary

The performance model for sparse matrix-vector and matrix-matrix products are summarized in Table 1. This table allows one to design and evaluate matrix multiplication accelerators on FPGAs. As an example, consider the design of a dense matrix-matrix multiplier. If the system I/O bandwidth is a constraint of the design, to ensure maximum performance, the number of MACCs should be $k \geq \frac{\sqrt{mb}}{2f}$.

4. MODEL EVALUATION

To evaluate the validity of the model, previously published results on FPGA matrix accelerations are compared against the performance determined by the model under the same architectural constraints. The results are shown in Table 2.

Although matrix operation accelerations on FPGAs have been extensively studied, only a selected subset involving double precision floating-point are included in the table. Furthermore, to the best of our knowledge, there has only been one prior work on accelerating sparse matrix-matrix multiplications on FPGAs [10]. An improved version of that design using the optimal block decomposition scheme of equation (10) was used in this study. A total of 26 sparse matrices from the University of Florida Sparse Matrix Collection were used in the implementation and 3 of the representative results are shown in Table 2.

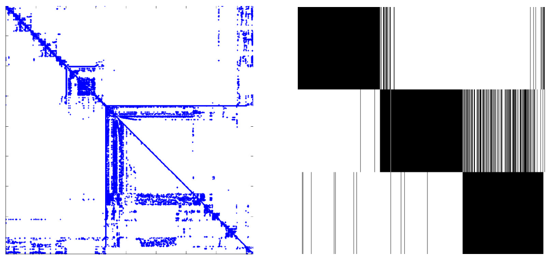

Fig. 1. Left: Sparsity pattern of 'rajat21'. Right: All-zero (white) and not all-zero (black) block columns of 'rajat21'

4.1. Model and Implementation Comparison

For dense matrix-vector and matrix-matrix multiplications, the reported performance lies within 95% of the model, indicating that the model has good predictability of actual performance. At the same time, the results also show that previously published accelerators have been performing close to optimality given the architectural constraints of the time.

For sparse matrix-vector multiplication, however, the reported implementation performance reached only 77-80% of the maximum performance. Moreover, in the case of sparse matrix-matrix multiplication, the implemented results ranged from 86% to 125% of the predicted performance.

This unpredictability stems from the fact that non-zero data elements were assumed to be randomly distributed in the model, while in practice, data in sparse matrices usually exhibit relatively regular organizations. As an example, the input sparse matrix 'rajat21' is shown in Figure 1. On the left it shows the location of non-zero elements in the matrix. On the right hand side of Figure 1, the black areas represent the columns in the sparse block with non-zero data, and the white areas represent zero columns.

According to the assumption that non-zero elements are randomly distributed, the probability of a non-zero column in the sparse block is $\beta_B = 78\%$. However, as can be seen in Figure 1, in actuality it is approximately 40%. Such a regular sparsity pattern reduces I/O operations of matrix B by about 38% and as a result, the implementation performance is about 25% better than the model performance.

On the other hand, for input matrices such as 'appu' in the table, the non-zero input data are distributed relatively randomly. As a result, the implemented results are within 1% of the predicted performance by the model.

4.2. Performance Bounds

The model is useful as a guideline for implementation and to determine performance bottlenecks. For instance, consider the first row of Table 2. Although the FPGA device supported up to 21 MACCs, only 4 were used in the implementation. Yet, when the available I/O bandwidth in the

Table 2. Comparison of model against published results.

Work	Op.	FPGA	Sparse Matrix	Size n	Density α (%)	Performance (GFLOPS)			MACCs k		Bandwidth b (GWords/s)	
						Impl.	Model	ratio	Impl.	Avail	Req'd	Avail
[11]	DMxV	XC2VP50	-	-	-	1.355	1.4	96.8%	4	21	0.7	0.7
[12]	DMxM	XC2V6000	-	-	-	4.5	4.7	95%	16	16	0.06	-
[13]		XC2VP30	-	-	-	1.79	1.8	99.4%	9	9	0.07	-
[11]		XC2VP50	-	-	-	2.06	2.08	99%	8	8	0.06	2.1
[14]		XC2VP125	-	-	-	8.3	8.3	100%	24	24	0.17	-
		XC2VP125	-	-	-	6	6	100%	24	24	0.12	-
[4]		XC2VP125	-	-	-	15.6	15.6	100%	39	39	0.31	0.4
[15]		XC4VLX200	-	-	-	4.8	4.8	100%	12	12	0.12	-
[16]		XC5VSX240T	-	-	-	29.8	29.8	98.7%	40	40	0.44	0.75
[6]	SpMxV	XC2VP70	radfsky3	21,200	0.33	2.16	2.79	77%	8	8	1.44	1.8
[7]		Virtex II Pro	lhr11c	10,964	0.19	1.49	1.69	88%	-	-	1.05	1.06
		Virtex II Pro	garon2	13,535	0.2	1.64	1.91	86%	-	-	0.93	1.06
		Virtex II Pro	olafu	16,146	0.39	1.67	2.02	83%	-	-	0.53	1.06
[8]		XC4VLX200	radfsky3	21,200	0.33	1.55	1.94	80%	8	12	1	1
this	SpMxM	XC6VLX760	heart1	3,557	10.95	76.13	86.73	86%	147	147	0.55	0.8
		XC6VLX760	appu	14,000	0.95	37.79	37.85	99.85%	65	147	0.8	0.8
		XC6VLX760	rajat21	411,676	0.0011	1.63	1.3	125%	3	147	0.8	0.8

system is considered, it becomes apparent that performance is limited by a low bandwidth of 0.7 GWords/s.

Similarly, all the reported sparse matrix-vector multiplication implementations were I/O bound.

In the cases of dense matrix-matrix multiplication, all reported implementations were limited by compute resource availability. The maximum number of MACCs was employed in all cases as a result.

The performance of sparse matrix-matrix multiplication is dependent on the sparsity of the input matrix. Out of the 26 cases tested, only one input matrix, 'heart1', resulted in a design that is computation bound. The large value of $\alpha = 10.95\%$ caused the design to behave closer to a dense matrix multiplication. The implementation performance is about 86% of the model performance, with the overhead being mainly due to the time to store the result matrix C. The rest of the 26 matrices tested were on-chip memory and IO bound because of their low sparsity.

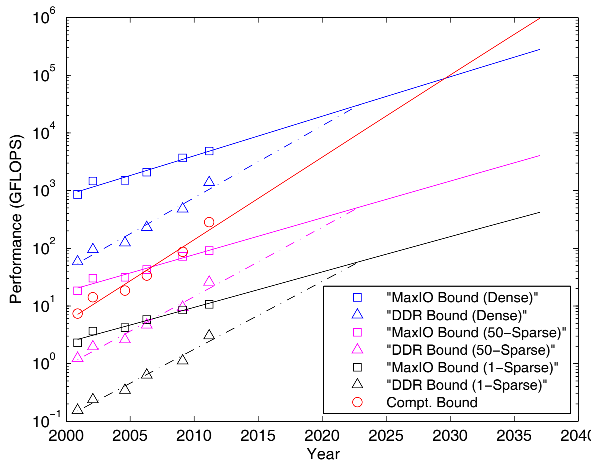

Fig. 2. Matrix-matrix multiplication performance bounds for Xilinx Virtex series FPGAs.

5. TECHNOLOGY TRENDS

Given the peak performance model presented in previous sections, we are now in the position to investigate the trends in technology and how they will affect the role of future FPGA-based matrix accelerators.

To provide a historical perspective of FPGA technology, the largest device from each generation of the Xilinx Virtex family were used in this study. The computation bound ($\mathbf{OPS}_{comp}$) and the I/O-memory bound ($\mathbf{OPS}_{io}$) for each device were calculated according to datasheets and implementation results. The theoretical matrix-matrix multiplication performance bounds were then computed according to our model, and the results presented in Figure 2. In the same figure, an exponential fit was made to extrapolate into the future.

To apply the model, k was defined as the maximum number of double-precision floating-point MACCs implementable with *both* logic and DSP resources on-chip, and f the maximum operating frequency of the MACC. m was the maximum number of double-precision floating-point words that can be stored using on-chip BlockRAMs. Two cases of I/O were considered. In the case of using memory as off-chip storage, the I/O bandwidth b was defined as the raw bandwidth of one DDR memory port using the *best* industrial DDR standard supported by the device at the time of its release. In the second case, we considered the aggregated raw I/O bandwidth of *all* user configurable I/O pins of the device when configured as differential pairs running at maximum frequency. Finally, we used a large matrix size of $n = 100,000$ in the result. Although this value is appropriate for a sparse matrix, it is too large for dense matrix operations. Fortunately, n has little effect on the dense matrix performance model bound.

5.1. Computation Resource vs. I/O-Memory Bound

As shown in Figure 2, whether a design is compute resource or I/O-memory bounded depends on the sparsity of the input matrix. This is because, according to (11), the computation bound, $\mathbf{OPS}_{comp} = 2kf$, is independent of α, while the I/O-memory bound, $\mathbf{OPS}_{io}$ is a strong function of α.

The performance of dense matrix-matrix multiplication ($\alpha = 1$) on FPGAs has been compute resource bound over the past decade. Moore's Law has meant that an exponential increase in k has been enjoyed. Beyond technology scaling, increasingly better floating point units that consume less logic resources and run at higher clock rates have also contributed to the continuous increase in compute performance.

With the amount of FPGA logic resources increasing at a much higher rate than I/O bandwidth, it is projected that performance will eventually become I/O-memory bound. In Figure 2, this occurs around the year 2030. However, the crossover point may be reached far earlier if, for instance, the compute units become faster or smaller by incorporating hard floating-point units in future FPGAs.

5.2. Sparsity vs. I/O-Memory Bound

An $n \times n$ matrix is *q-sparse* if the number of non-zero elements is qn. In the case of a "1-sparse" matrix (e.g. a diagonal one), it can be seen in Figure 2 that, in the past, performance has been limited by the I/O-memory bound. Based on the analysis in the previous subsection, the gap between $\mathbf{OPS}_{comp}$ and $\mathbf{OPS}_{io}$ will widen in the future.

The tradeoff between compute and I/O-memory bound approaches that of a dense matrix computation as q increases. Take the case of $q = 50$, i.e., $\alpha = 0.05\%$ as an example. The performance of sparse matrix multiplications has been computation resource bound until approximately 2005. Since 2006, it has become I/O-memory bound.

Currently, I/O performance is the limiting factor for both matrix-vector and sparse matrix-matrix multiplication. Performance improvements may thus be achieved through the use of multiple DDR memory ports or high speed serial I/Os. Progress in I/O performance will also delay the crossover point for dense matrix-matrix multiplication.

6. CONCLUSION

An analytical model which relates FPGA-based dense and sparse matrix multiplication performance to architectural and system parameters was described and its output shown to closely match that of previously published designs.

Performance estimates for future technologies were extrapolated. It was shown that currently, very sparse problems are I/O-memory bound whereas dense problems are computation resource bound. In the future, however, it is likely that the performance of matrix-matrix multiplication will be bounded by I/O bandwidth and the amount of on-chip memory. Research to improve in this area is urgently needed so that increases in logic resources due to Moore's Law can be accompanied by comparable increases in I/O bandwidth and on-chip memory capacity.

In future work, we will extend the model to account for common sparse matrix types such as banded, triangular, Jordan normal form etc. We will also study higher level linear algebra operations such as eigenvalue solvers, iterative linear solvers and factorization-based linear solvers.

7. REFERENCES

[1] W. Ligon III, S. McMillan, G. Monn, K. Schoonover, F. Stivers, and K. Underwood, "A re-evaluation of the practicality of floating-point operations on FPGAs," in *Proc. FCCM, 1998*, pp. 206–215.

[2] J. Jang, S. Choi, and V. Prasanna, "Energy-Efficient Matrix Multiplication on FPGAs," in *Proc. FPL*, 2002, pp. 534–544.

[3] K. Underwood and K. Hemmert, "Closing the gap: CPU and FPGA trends in sustainable floating-point BLAS performance," in *Proc. FCCM, 2004*, pp. 219–228.

[4] Y. Dou, S. Vassiliadis, G. K. Kuzmanov, and G. N. Gaydadjiev, "64-bit floating-point FPGA matrix multiplication," in *Proc. FPGA*, 2005, pp. 86–95.

[5] L. Zhuo and V. Prasanna, "Design tradeoffs for BLAS operations on reconfigurable hardware," in *Proc. ICPP, 2005*.

[6] L. Zhuo and V. K. Prasanna, "Sparse matrix-vector multiplication on FPGAs," in *Proc. FPGA*, 2005, pp. 63–74.

[7] Y. Zhang, Y. H. Shalabi, R. Jain, K. K. Nagar, and J. D. Bakos, "FPGA vs. GPU for sparse matrix vector multiply," in *Proc. FPT, 2009*, pp. 255–262.

[8] G. Kuzmanov and M. Taouil, "Reconfigurable sparse/dense matrix-vector multiplier," in *Proc. FPT, 2009*, pp. 483–488.

[9] M. DeLorimier and A. DeHon, "Floating-point sparse matrix-vector multiply for FPGAs," in *Proc. FPGA*, 2005.

[10] C. Y. Lin, Z. Zhang, N. Wong, and H. K.-H. So, "Design space exploration for sparse matrix-matrix multiplication on FPGAs," in *Proc. FPT*, 2010, pp. 369–372.

[11] L. Zhuo and V. K. Prasanna, "High performance linear algebra operations on reconfigurable systems," in *Proc. SC*, 2005.

[12] S. G. Ziavras, "H-SIMD machine: configurable parallel computing for matrix multiplication," in *Proc. ICCD*, 2005.

[13] G. Kuzmanov and W. M. V. Oijen, "Floating-point matrix multiplication in a polymorphic processor," in *Proc. FPT, 2007*, pp. 249–252.

[14] L. Zhuo and V. K. Prasanna, "Scalable and modular algorithms for floating-point matrix multiplication on FPGAs," in *Proc. IPDPS*, 2004, p. 92.

[15] P. Russek and K. Wiatr, "Dedicated architecture for double precision matrix multiplication in supercomputing environment," in *Proc. DDECS*, 2007, pp. 1–4.

[16] V. B. Y. Kumar, S. Joshi, S. B. Patkar, and H. Narayanan, "FPGA Based High Performance Double-Precision Matrix Multiplication," in *Proc. VLSID*, 2009, pp. 341–346.