2009 22nd International Conference on VLSI Design

FPGA based High Performance Double-precision Matrix Multiplication

Vinay BY. Kumar Siddharth Joshi Sachin B. Patkar H. Narayanan
vinayby@iitb.ac.in {s_joshi, patkar, hn}@ee.iitb.ac.in
Department of Electrical Engineering
Indian Institute of Technology, Bombay
India, Mumbai - 400076

***Abstract*—We present two designs (I and II) for IEEE 754 double precision floating point matrix multiplication, an important kernel in many tile-based BLAS algorithms, optimized for implementation on high-end FPGAs. The designs, both based on the rank-1 update scheme, can handle arbitrary matrix sizes, and are able to sustain their peak performance except during an initial latency period. Through these designs, the trade-offs involved in terms of local-memory and bandwidth for an FPGA implementation are demonstrated and an analysis is presented for the optimal choice of design parameters. The designs, implemented on a Virtex-5 SX240T FPGA, scale gracefully from 1 to 40 processing elements(PEs) with a less than 1% degradation in the design frequency of 373 MHz. With 40 PEs and a design speed of 373 MHz, a sustained performance of 29.8 GFLOPS is possible with a bandwidth requirement of 750 MB/s for design-II and 5.9 GB/s for design-I.**

I. Introduction

Field Programmable Gate Arrays (FPGAs) are increasingly being seen as a promising avenue for High Performance Computing (HPC), especially with the introduction of high-end FPGAs like Virtex-4/5. These FPGAs are a very attractive choice due to their abundant local memory, embedded high-speed resources like DSP blocks, PCIe endpoints etc., and their reconfigurability and lower power consumption compared to general purpose hardware.

An inspection of Level-3 BLAS routines shows that matrix multiplication (*dgemm*) and triangular equation solution (*dtrsm*) form the building blocks for many important linear algebra algorithms, in fact, the *dtrsm* itself can be expressed in terms of *dgemm*. MEMOCODE-07 hosted a challenge problem requiring a hw/sw codesign based acceleration of complex integer matrix-multiply. Underwood [1] chose matrix multiplication as one of the three main routines for FPGA acceleration in order for HPC. Matrix-Multiplication, therefore, presents as an important candidate for hardware acceleration.

In this paper, we present designs for double precision floating point matrix multiplication, based on the rank-1 update algorithm, targeted at the Virtex-5 SX240T, a high-end Xilinx FPGA. As compared to others this algorithm enables better re-use of data from the input matrices. The processing elements (PEs) use off-the-shelf floating point operators from Xilinx Coregen, unlike other related work, resulting in advantages such as the choice of custom-precision, short-design time, portability across devices, better IEEE 754 compliance, etc. The PEs are designed so as to scale linearly in terms of resources with negligible (<1%) degradation in speed. Some of the recent work [2] reports 35% speed reduction associated with scaling, which is typically due to increased routing complexity. The proposed design(II) is tolerant to burst-like input which suits well with a high performance I/O bus like PCIe – allowing it to scale seamlessly across multiple FPGAs.

The under utilisation of device primitives and the over-dependence on the distributed memory available in FPGAs results in lower performance with respect to scaling. The designs presented in this work have evolved by careful use of the high-performing resources on modern FPGAs. Special care has been taken to address issues related to scaling for large FPGAs, setting this work apart from related art.

The following sections are organised as follows — Section II discusses related work with a quick background on the subject, Section III discusses the underlying algorithm, Section IV elaborately discusses both the designs, Section V presents an evaluation of the design, Section VI presents an analysis on design parameters, Section VII critically compares our design with the best among the related work, and finally Section VIII concludes the paper.

II. Background

The rank-1 update algorithm used in this paper is an elementary idea, variations of which have also been applied to cache-aware computing on general purpose processors [3], though not as aptly. This was chosen to be implemented on an FPGA since it is particularly suitable for the task as verified by both Dou and Prasanna.

Much of the related work are designs targeted and optimised for Virtex II Pro, which is an entry level device for HPC that made floating point computation feasible for the first time on FPGAs.

A. Related Work

The two most recent significant designs are those by Dou [4] and Prasanna [2]. They propose linear array based processing elements which are able to sustain their performance using a technique called memory switching.

Dou has proposed the design of a matrix multiplier highly optimised for the Virtex II Pro series of FPGAs. The design included an optimized custom 12-stage pipelined floating point

1063-9667/09 $25.00 © 2009 IEEE
DOI 10.1109/VLSI.Design.2009.13

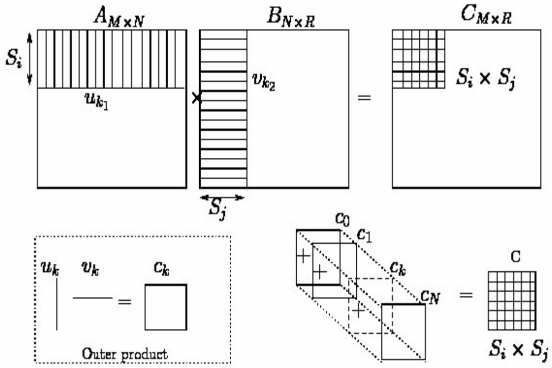

Fig. 1. The rank-1 update scheme

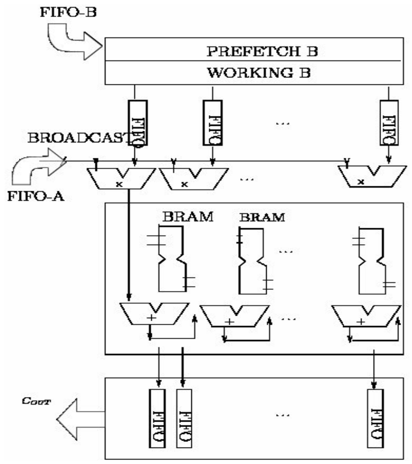

Fig. 2. Overview of Design-I

MAC, but with a few limitations like no support for zero and other denormal numbers. This design required the subblock dimensions to be powers of two. The bandwidth requirement was low at 400 MBps with 12.5 Mb of local memory utilization and they report a PE design with a synthesis frequency of 200 MHz and therefore estimated a 15.6 GFLOPS performance accommodating 39 PEs on a Virtex II pro XC2V125, a large theoretical device. As these are only synthesis results, the real frequency after placing and routing 39 PEs could be much less. Also, correcting the limitations of MAC would affect resource usage and speed.

Zhou and Prasanna have reported an improved version [2] of their design reported in [5]. The recent one reports 2.1 GFLOPS for 8 PEs running at 130 MHz on a cray XD1 with XC2VP50 FPGAs. About 35% speed degradation was observed when scaled from 2 to 20 PEs. In the earlier paper they presented a design with a peak performance of 8.3 GFLOPS for the Virtex II Pro XC2V125, where the clock degradation was 15% when the number of PEs increases from 2 to 24.

III. Algorithm

The rank-1 update scheme for matrix multiplication, illustrated in the Figure 1, has been described here for the convenience of the reader. The paper partly follows the notation introduced by Dou [4] as both are variations of the same algorithm. Consider A, B and C of dimensions M×N, N×R and M×R respectively. The objective is to compute $C = AB$. When a S_i×N panel of A (say, PA) and a N×S_j panel of B (say PB) are multiplied, the result is a subblock of the matrix C with dimensions $S_i \times S_j$. The outer product of the k^{th} column of the vector (u_k) from PA and the k^{th} row vector(v_k) from PB is an intermediate result and accumulation of such results with k ranging over the panel length (from 1 to N) is the required subblock of C.

For an outer product, each element of vector u_k multiplies each element of vector v_k. Thus, $n(v_k)$ or S_j elements are re-used $n(u_k)$ or S_i times with S_j multiplications each time. What also follows is that if one element of u_k and all the elements of v_k are available to the system then each of the products can be carried out independently. This design exploits these key ideas. With this, the focus for the design now shifts to effective utilization of resources like BRAMs, DSP48 blocks etc.

IV. Implementation

The designs have been optimised to effectively use the high performance primitives available on a modern high-end FPGA. This section describes two designs, I and II. The goals for the first design were full utilisation of PEs and overlapping I/O and computation and thereby sustaining peak performance. These were achieved at the cost of a high I/O bandwidth and sub-optimal use of available resources. The second design addresses all the limitations of the first design, and the section on design II describes its evolution in terms of more effective use of the previous data-path and resources.

Broadly, broadcasting elements of PA to all PEs and the streaming in of elements of PB to the prefetch registers is central to both the schemes and the relative rate at which they are streamed in, and the manner of their re-use is what essentially differentiates the two.

A. Design-1

This design assumes $P = S_j$ and $S_i = S_j$, where P is the number of PEs. The $S_i > S_j$ case is also acceptable. Figure 2 gives an overview of this design. The following enumerated list will describe all the major labeled components, shown in Figure 3, of the PE. It will be clear shortly that this design requires 2 words per design clock.

Component Description:

1) **B Prefetch Unit**: This unit is used to prefetch S_j elements of the next row of PB while the current row is used. The input to the first of such units is a stream of elements from the matrix B, in a row major fashion. Each unit has one data input and two data outputs: a serial-shift-forward, which happens every clock cycle and a parallel-load-down which happens every P shifts

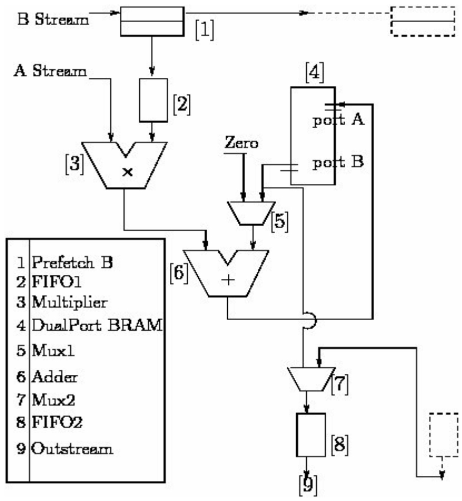

Fig. 3. 1 PE: Design-I

(or S_i clocks if $S_i > S_j$). These P words are available at the output for at most S_i clocks.

2) **FIFO1** : When the B Prefetch units were connected directly to the multipliers, a severe and unexpected drop in the design frequency was observed. This drop was inferred to be due to the routing overhead in bringing the data lines from a 64 bit register to the 13 DSP48 blocks which make up a double precision multiplier. A FIFO built out of BRAM was placed in the path in order to reduce the length of the routing path thus ensuring the expected design frequency. Due to the physical proximity of the BRAMs to the DSP blocks on Xilinx FPGAs, complicated routing is avoided and also, now, the 64 bits have to route to one BRAM instead of 13 DSP48s.
3) **Multiplier** : A standard double precision floating point multiplier IP(version v3.0) from Xilinx is used for this block. A latency of 19 cycles gives it a maximum clock speed of about 431 MHz using 13 DSP48 units. One input to the multiplier comes from the prefetch unit via the FIFO1 and the other input is the element from matrix A which is broadcast to all multipliers. The output of this multiplier is one of the inputs to the adder. The recent floating-point v4.0 is superior in terms of area resource(DSP48) usage and latency, especially for Virtex-5 series, but reduces the speed, hence is not used in the design.
4) **Dual-Port BRAM** : This dual port blockram is used as the storage space for the accumulation step of the algorithm. The *adder* writes back to the RAM using port A and reads from the RAM using Port B. The output of port B is duplicated as the input to Mux2 as well.
5) **Mux1** : This mux ensures '0' is added to the incoming product stream, whenever a new panel is read in, while the previous result in the BRAM is copied into FIFO2's.
6) **Adder** : A standard double precision floating point adder IP(version v3.0) from Xilinx is used for this block. It receives two inputs, one from the multiplier and one from the Mux1. It writes back to blockram at the appropriate location considering its own latency.
7) **Mux2** : The mux is used to switch connections between the BlockRAM (Result-backup mode), and the other instances of FIFO2 (Serial-dataout mode).
8) **FIFO2**: In order to ensure that there is no stalling in the pipeline the result needs to be backed up. Since both the ports of the result BRAM are busy, a separate memory unit is used for the back-up, in the form of FIFO2. The final updated data of the result sub-matrix 'C' (one column of 'C' when we talk about 1 PE) will be loaded into the corresponding FIFOs (Result-backup mode) of the PEs. When the result has been copied into the FIFO, input of the FIFO gets connected to the output of FIFO2 of the previous PE, thus allowing us to take the output in a streaming fashion (Serial-dataout mode).

Data Flow - design 1: The inputs, elements from PA and PB are streamed in column major and row major order respectively. First, one of the rows from PB (v_k) shifts into the B prefetch unit. Once a complete row is shifted-in ($S_j = P$), the 'prefetch-line' registers are full and this data is loaded down to the 'working-line' registers. In the meanwhile the prefetch-line continues to shift-in the next row from PB (v_{k+1}). At this point, when working-line is available and connected to one of the inputs of the multiplier, elements from the corresponding column (u_k) from A are broadcast to the second input of all the multipliers. After a latency period of the multiplier (say, L_m) the first result of multiplication is available at the output along with the appropriate handshaking signals which are used to trigger accumulation of the outerproducts at the storage area.

Once the pipeline of the multiplier has been filled it shall not be stalled since no data dependencies exist between subsequent multiplications. This allows continuous feeding of the data at the maximum design frequency. The result of the addition, available after a latency period (say, L_a), is stored in the BRAM. We will see later that the pipeline may stall in one case. Zero is accumulated with the product stream during the first outer product of each new PA×PB, after which the accumulation happens with the appropriate location in the BRAMs. The FIFOs responsible for the output are loaded with the values parallely from the BRAMs once the final value of the result subblock $C_{S_i \times S_j}$ is ready. After the loading/backup is completed, these FIFOs switch modes allowing us to stream the data out in a serial fashion.

Merits and Summary: The design described above requires that $S_i \geq S_j = P$, implicitly assuming a bandwidth of 2-words per design clock cycle. PCIe is capable of such high bandwidths, and is the norm for today's large FPGAs. The merits and demerits of this design have been summarised in the following list, details about the performance and analysis are presented in a later section.

1) Overlaps I/O and computation completely. Therefore, except for the initial latency period, all the processing

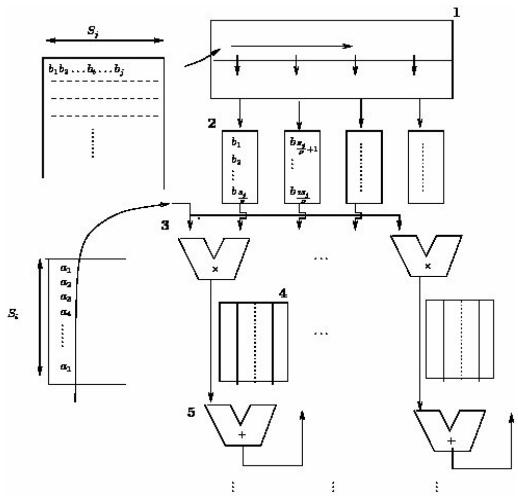

Fig. 4. 1 PE: Design-II

elements (both the floating point operators) are in use all the time, thus sustaining peak performance.

2) The design scales seamlessly ($< 1\%$ reduction in speed) as seen from Table I.
3) Uses off-the-shelf Coregen floating point adders and multipliers, allowing for portability across technologies and generations, custom-precision option, better IEEE compliance etc.

Drawbacks:

1) The design requires a sustained bandwidth of 2-words per cycle corresponding to the design frequency. For a design speed of, say, 350 MHz this translates to a 5.6 GBps bandwidth requirement, which can well be provided by today's standards, but is high nonetheless.
2) Little to almost no flexibility in the choice of S_i and S_j which might affect the overall runtime even with the sustained peak performance.
3) S_i cannot be less than the latency of the floating point adder as otherwise there would arise a data dependency. Practically this is not usually a problem due to the large sizes of matrices under question.

B. Design-II

Design-I assumed $S_i \geq S_j = P$ and so $S_j = P$ elements of B were being reused S_i times. The design-II ensures more re-use by allowing S_i and S_j to be greater than P, (i.e., $S_i, S_j \geq P$), however S_j needs to be a multiple of P.

As shown in the Figure 4 the data-path is similar to that of design-I. The design actually evolved from design-I in an attempt to find a better use of the existing components, especially the dummy FIFO1 used earlier. The following enumerated list describes the major modifications as

1) **B Prefetch Unit**: This is the same as described earlier in design-I, however two sets of registers are not necessary. 'BRAM Cache' is made part of the working-line by using it for storage.
2) **B Cache**: The design-I employed a dummy FIFO1 (BRAM based) to prevent the design frequency from falling drastically. This component now assumes a central role in design-II. Each BRAM now stores S_j/P consecutive elements of a row from the chosen panel of the matrix B, hence renamed B Cache. Configured in dual port mode, this BRAM can easily implement the working-line required for the design.
3) **Simple Dual-Port RAM**: To accommodate the larger sizes of S_i and S_j, the storage area has been increased in size and logically segmented into $S_j/n(PEs)$ zones each storing results corresponding to one S_j. Over all, the storage area accounts for the storage of $S_i \times S_j$ elements of the result block. Writing to the appropriate segments is handled by address generation and control.

Design Merits:

1) Inherits all the merits from design I – as enumerated earlier.
2) Addresses all the identified drawbacks of design I, *viz* drastically reduces the bandwidth requirement, more flexibility in the choice of S_i and S_j and relaxes the constraint on S_i w.r.t the latency – the details of which are described in the following sections.

Data Flow: Data stream from a row of PB is fed to the prefetch unit as before, but the sequence of data is such that the i^{th} consecutive $\frac{S_j}{P}$ elements from a row with S_j elements, are loaded into the B-Cache storage corresponding to the i^{th} PE. Thus the following sequence is observed, assuming $b_1,\ b_2,\ b_3 \ldots$ are the contents of PB in row major fashion:

$$b_1, b_{\frac{S_j}{P}+1}, b_{\frac{2S_j}{P}+1}, \ldots b_2, b_{\frac{S_j}{P}+2}, \ldots$$

One element from A is used for $\frac{S_j}{P}$ cycles, where it multiplies all S_j elements of a row. During the first outerproduct computation, the multiplier result is accumulated with 0 and stored in the BRAM. Thus, one outerproduct computation takes $\frac{S_i \times S_j}{P}$ cycles to complete, after which the elements from the next column of A are required. As a result of this, the restriction of $S_i \geq L_a$ is relaxed to $\frac{S_i \times S_j}{P} \geq L_a$. But the most important consequence of the new design is that the bandwidth requirement is considerably reduced as a result of a trade-off with local memory usage/data re-use.

Illustrative Example: Consider the product of two square matrices A and B each with dimensions 800×800. With a design speed of 350 MHz, we consider the following two cases.

Case I : $[S_i = S_j = P;\ P = 50]$

In this case, the bandwidth required is 2 words per cycle which with 350 MHz means 5.6 GB/s ($= 2 \times 8 \times 350$). One outerproduct computation in this case takes S_i cycles and therefore one $S_i \times S_j$ subblock computation of the result takes $S_i \times 800$ cycles. For the entire matrix multiplication of $A \times B$, there are 16×16 such subblocks. Therefore, the total number of cycles for $A \times B$ computation is S_j $+ S_i \times 800 \times 16 \times 16 = 10240050$.

Case II : $[S_i = S_j = 400;\ P = 50]$

As $\frac{S_j}{P} = 8$, we see that one word of A is required every 8 clock cycles. So, a bandwidth which gives us 2 words for every 8 cycles, or 0.25 words per cycle or 700 MB/s($= .25 \times 8 \times 350$) will be sufficient. As for the total computation time, one can see that an $S_i \times S_j$ result subblock computation requires $\frac{S_i \times S_j}{P}$ and there are 4 such blocks here. Therefore, the total number of cycles for $A \times B$ computation is $S_j + \frac{S_i \times S_j}{P} \times 800 \times 4 = 10240400$.

Thus, design II solves the problem using significantly lower bandwidth than the first design. **The increase in the cycles required for computations is because of the increased setup time**.

V. Design Evaluation

Xilinx ISE 10.1sp1 and ModelSim 6.2e was used for implementation and simulation of the design respectively.

The most significant aspect of the design, from the Table I, appears to be the negligible variation of the speed despite scaling up to 40 PEs, an explanation for which is offered in the comparison section. **The drastic reduction in speed, from 373 to 201 MHz, on SX95T is attributed to the expected poor routing when the resource utilization reached $>$ 95%, therefore considered a corner case.**

As shown in Table II, due to abundance of resources their liberal use is justified. Appropriate pipeling, not shown in the figures, has been done in order to break the critical paths. It can also be seen from the resource usage at 40 PEs that a few more PEs can be accommodated in the SX240T.

The design was ported to the Virtex 2 Pro XC2VP100 for the sake of comparison and as shown in Table III, about 20 PEs can be fit with a frequency of about 134 MHz as opposed to 31 PEs and 200 MHz(synthesis) respectively by [4] (In a later usage of the same PE by one of the co-authors of [4], the actual implementation frequency was about 100 MHz [6])

TABLE I
Timing Information (post PAR)

No. PEs	SX240T(-2)[MHz]	SX95T (-3)[MHz]
1 PE	374	377
4 PEs	373	374
8 PEs	344	373
16 PE	-	373
19 PEs	-	373
20 PEs	372.8	201
40 PEs	371.7	-

VI. Performance Analysis

We present an analysis of the design parameters listed in Table IV studying their effect on performance and the constraints they impose. All the analysis is with respect to design-II.

Each element of A is used $\frac{S_j}{P}$ times, in an outerproduct and therefore the entire computation of the outerproduct takes $\frac{S_i \times S_j}{P}$ cycles. In order to overlap I/O and computation, the algorithm requires that we prefetch S_j elements of B for the next outerproduct. We have therefore

$$S_i + S_j \leq \frac{Si \times S_j}{P} \times \beta \tag{1}$$

We also see that the $S_i \times S_j$ needs to be maximized here. The constraint on memory gives us Eq 2 which on approximation gives Eq 3

$$2S_i \times S_j + 2S_j = 2(S_i + 1) \times S_j \leq m \tag{2}$$

$$2(S_i) \times S_j \leq m \tag{3}$$

To maximize $f(S_i, S_j) = S_i \times S_j$, under the constraints Eq 1 and Eq 3 we use the Lagrangian constrained optimization method

$L(S_i, S_j, \lambda, \mu) =$
$S_i \times S_j + \lambda \left(\beta \frac{S_i \times S_j}{P} - (S_i + S_j)\right) + \mu \left(m/2 - S_i \times S_j\right)$

$$\frac{\partial L}{\partial S_i} = S_j - \lambda + \lambda\beta\frac{S_j}{p} - \mu S_j = 0 \tag{4}$$

$$\frac{\partial L}{\partial S_j} = S_i - \lambda + \lambda\beta\frac{S_i}{p} - \mu S_i = 0 \tag{5}$$

Equations 4 and 5 suggest $S_i = S_j$. If we substitute $S_i = S_j = S$, we get

$$\text{Maximize} \quad S \tag{6}$$

$$S \geq \frac{2P}{\beta} \tag{7}$$

$$S \leq \sqrt{\frac{m}{2}} \tag{8}$$

TABLE II
Resource utilization for SX95T and SX240T devices (post PAR)

No PEs	DSP48E	FIFO	BRAM	Slice Reg	Slice LUT
1 PE	16	1	2	2511	1374
4 PE	64	4	8	10377	5451
8 PE	128	8	16	20865	10886
16 PE	256	16	32	41841	21750
20 PE	320	20	40	52329	27176
40 PE(sx240)	640	40	80	69%	36%
RESOURCES PER DEVICE					
Device					
SX240T	1056	516	516	149760	149760
SX95T	640	244	244	58880	58880

TABLE III
Resource utilization for Virtex II Pro XC2VP100 (post PAR)

	$\text{Tot}_{\mathbf{xc2vp100}}$	$\text{U}_{15\ PE}$	$\text{U}_{20\ PE}$
MULT18x18s	444	240	304
RAMB16s	444	90	114
Slices	44096	30218 (68%)	37023 (83%)
Speed		133.94 MHz	133.79 MHz

TABLE IV
List of parameters

parameters	meaning
β	bandwidth in terms of the no. of words per design clock
x_a, x_b	such that $x_a + x_b \leq \beta$
m	total amount of local memory
n	num. of columns of a (or rows of b)

The following analysis for the minimum required bandwidth demonstrates the burst-friendly nature of the design. We know that S_i words of A are required for S_j words of B within $\frac{S_j \times Si}{p}$ cycles. Thus we get the values for min(x_a) and max(x_b) for a constant bandwidth of β. Thus we get the values for min(x_a) and max(x_b) for a constant bandwidth of β.

$$\min(x_a) = \frac{P}{S_j} \tag{9}$$

$$\max(x_b) = \beta - \frac{P}{S_j} \tag{10}$$

For the case where $S_i = S_j$ equal distribution of bandwidth is the best approach, for other cases a similar analysis results in the appropriate distribution. The availability of more than the minimum amount of bandwidth means that the excess bandwidth can be used to transfer as much A as required in one go – further trading bandwidth with local storage. This caching also creates time during which the bandwidth can be used for other I/O, allowing for sharing the same bandwidth across multiple FPGA boards.

VII. Comparison

The following compares a few aspects of ours designs with the recent related work. In particular we compare with Dou [4] and Prasanna [2], [5], the former of which was identified superior to other related work by Craven-2007 [7].

- **Scaling**: As reported previously [2], [5] frequency falls by about 35% and 15% respectively by scaling to 20 PEs. Our designs show negligible($<1\%$) degradation in frequency up to 40 PEs. Further, the low-bandwidth requirements and burst-friendly behaviour allows design-II to scale well across multiple FPGAs due to low bandwidth requirement per FPGA.
- **Flexibility**: Dou's design requires matrix subblock dimensions to be powers of 2. Prasanna supports square matrices of limited size in [5] and arbitrary size in [2]. Our designs support arbitrary matrix sizes as in [4] without placing extra constraints on $S_i,\ S_j$.
- **PE/MAC**: Dou's custom MAC [4] is highly optimized for Virtex-2 Pro and may not scale across families of FPGAs. The MAC doesn't support zero and denormal numbers. Our design uses floating-point units from core-generator making the design more flexible(portable, scalable, customizable) along with better IEEE compliance. It is to be noted that these are optimized for Virtex-5.

  We were able to place and route only 20 PEs on Virtex-2 Pro XC2VP100 as opposed to 31 PEs (synthesis-only) by [4] which was possible because of the custom designed MAC which use only 9 18x18 multipliers as opposed to 16 by core generator. But such custom MAC may not be appropriate in the context of, say, Virtex-5 SX240T where there are about 1200 DSP48s and the coregen floating point units are highly optimized to use them effectively.
- **I/O–Computation Overlap**: The designs use a variant of 'pipelining' or buffering for the purpose of overlapping I/O and computations as opposed to memory switching used in related works. This may be a factor in the better scaling of our designs as explained below. Memory switching requires two memory-banks to alternately feed the processing elements. This places constraints on the placement of the memory banks with respect to the processing units. In this implementation, one memory unit feeds another, except for those connected directly to the processing units. This takes advantage of their physical proximity on the device and the better routing between BRAMs and DSP48 blocks.

VIII. Conclusion

In this paper two designs for matrix multiplication are presented which vividly demonstrate the trade-off between memory and bandwidth. The simplicity of the designs and the use off-the-shelf floating point units from Xilinx Coregen offer easy reproduction of the design, portability across FPGA families and maintainability along with better IEEE compliance and options such as custom precision. The designs are able to sustain the peak performance, like a few other related work, achieved by use of a technique alternative to memory switching, which also has a favourable impact on routing. Our designs scale well with $<1\%$ degradation in speed and design-II further enables scaling across multiple FPGAs. For about 40 PEs, with a design frequency of 373 MHz on Virtex-5 SX240T FPGA, a sustained performance of 29.8 GFLOPS is possible with a bandwidth requirement of 750 MB/s for design-II and 5.9 GB/s for design-I. The design can be made available upon request. Future work includes porting it for use with the CRL-India's supercomputer.

Acknowledgment

The authors acknowledge Sunil Puranik and others from CRL-India for their insights on HPC; Rahul Badghare and Pragya Sharma for their help in the timing analysis.

References

[1] K. D. Underwood and K. S. Hemmert, "Closing the gap: Cpu and fpga trends in sustainable floating-point blas performance," in *FCCM*. IEEE Computer Society, 2004, pp. 219–228.

[2] L. Zhuo and V. K. Prasanna, "Scalable and modular algorithms for floating-point matrix multiplication on reconfigurable computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 4, pp. 433–448, 2007.

[3] K. Goto and R. van de Geijn, "High performance implementation of the level-3 BLAS," accepted: 28 October 2007.

[4] Y. Dou, S. Vassiliadis, G. K. Kuzmanov, and G. N. Gaydadjiev, "64-bit floating-point fpga matrix multiplication," in *FPGA '05: Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*. New York, NY, USA: ACM, 2005, pp. 86–95.

[5] L. Zhuo and V. K. Prasanna, "Scalable and modular algorithms for floating-point matrix multiplication on fpgas," *IPDPS*, vol. 01, p. 92a, 2004.

[6] G. Kuzmanov and W. van Oijen, "Floating-point matrix multiplication in a polymorphic processor," in *International Conference on Field Programmable Technology (ICFPT)*, December 2007, pp. 249–252.

[7] S. Craven and P. Athanas, "Examining the viability of fpga supercomputing," *EURASIP J. Embedded Syst.*, vol. 2007, no. 1, pp. 13–13, 2007.