2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID)

A Systematic Approach for Acceleration of Matrix-Vector Operations in CGRA through Algorithm-Architecture Co-design

Farhad Merchant*, Tarun Vatwani‡, Anupam Chattopadhyay§, Soumyendu Raha¶,
S K Nandy†, Ranjani Narayan‖ and Rainer Leupers*
*Institute for Communication Technologies and Embedded Systems, RWTH Aachen University, Germany
{farhad.merchant, leupers}@ice.rwth-aachen.de
†CADLab, Indian Institute of Science, Bangalore, India 560012
{{tarun}@cadl, {raha,nandy}@cds}.iisc.ac.in
‡Indian Institute of Technology, Jodhpur
§School of Computer Engineering, Nanyang Technological University, Singapore
anupam@ntu.edu.sg
¶Scientific Computation Laboratory, Indian Institute of Science, Bangalore, India 560012
‖Morphing Machines Pvt. Ltd.
ranjani.narayan@morphingmachines.com

***Abstract*—Matrix-vector operations play pivotal role in engineering and scientific applications ranging from machine learning to computational finance. Matrix-vector operations have time complexity of $O(n^2)$ and they are challenging to accelerate since these operations are memory bound operations where ratio of the arithmetic operations to the data movement is $O(1)$. In this paper, we present a systematic methodology of algorithm-architecture co-design to accelerate matrix-vector operations where we emphasize on the matrix-vector multiplication (gemv) and the vector transpose-matrix multiplication (vtm). In our methodology, we perform a detailed analysis of directed acyclic graphs of the routines and identify macro operations that can be realized on a reconfigurable data-path that is tightly coupled to the pipeline of a processing element. It is shown that the PE clearly outperforms state-of-the-art realizations of gemv and vtm attaining 135% performance improvement over multicore and 200% over general purpose graphics processing units. In the parallel realization on REDEFINE coarse-grained reconfigurable architecture, it is shown that the solution is scalable.**

***Index Terms*—dense linear algebra, matrix-vector operations, instruction level parallelism, scalability**

I. Introduction

Matrix-vector operations have been a key kernel in several application domains. For example, radial basis function neural network (RBFNN) based face recognition system requires updating operations on QR and/or LU factorization routines. These updating operations are of the form of matrix-vector multiply (gemv) operations or vector transpose-matrix multiply (vtm) operations and exhibit time complexity of $O(n^2)$. A similar phenomena is observed in applications like computational finance and kalman filter (KF) where the entire factorization need not be recomputed but only a small region of the matrix is updated and corresponding column vectors in the factorization need to be recomputed [1][2]. Typically, gemv and vtm operations are challenging to acceleration since these operations are memory bound operations where ratio of computation to data movement is $O(1)$.

Recently, there have been several attempts in accelerating memory bound operations through customized computing. Majority of these attempts have been on field programmable gate arrays (FPGAs) [3][4]. Typically, coarse-grained reconfigurable architectures (CGRAs) exhibit performance characteristics where the flexibility and the performance occupies middle ground between the fine-grained reconfigurable/reprogrammable devices like FPGAs/programmable devices like multicores and the application specific integrated circuits (ASICs) [5]. Furthermore, CGRA like REDEFINE has a unique feature where it can be customized to support several application domains even post-silicon. Application acceleration for the domains of signal processing, machine learning and dense linear algebra (DLA) is well demonstrated in the literature. It is shown in [6] and [7] that with a reconfigurable data-path (RDP) that is tightly coupled to the pipeline of a processing element (PE) can attain up to 74% of the theoretical peak in the PE in double precision general matrix-matrix multiply (dgemm). In this paper, we focus on dgemv and dvtm (where "d" is for double precision arithmetic) operations for accelerations through algorithm-architecture co-design where we identify macro operations in the directed acyclic graphs (DAGs) of dgemv and dvtm operations and realize them on RDP resulting in 50% of the theoretical peak performance of the PE. Major contributions in the paper are as follows:

- We analyze dgemv and dvtm routines and identify macro operations in the DAGs of dgemv and dvtm
- The macro operations are realized on the RDP that is tightly coupled to the PE. The PE is able to attain up to 50% of the theoretical peak in gemv and vtm
- For parallel realization of dgemv and dvtm, we attach

the PE as a custom function unit (CFU) in the compute elements (CEs) of REDEFINE CGRA and show that the solution is scalable

For our implementation, we have used floating point unit presented in [8] with recommendations given in [9]. In future, we plan to incorporate posit arithmetic units generated from [10]. Organization of the paper is as follows: Details of implementations of dgemv and dvtm in the literature are discussed in section II along with preliminary details of REDEFINE CGRA. In section III, analysis of dgemv and dvtm is presented where macro operations are identified. Implementation details and results are discussed in section IV. Parallel realization of dgemv and dvtm is also discussed in the section IV. We conclude our work in section V.

II. Background and Related Work

We briefly discuss dgemv, dvtm, and REDEFINE CGRA in sections II-A1 and II-A2 respectively, and in section II-B, several recent realizations of dgemv and dvtm are discussed.

A. Background

1) dgemv and dvtm: For a matrix A of size $m \times n$ and a vector x of size $n \times 1$, matrix vector operation is given by

$$y = Ax \tag{1}$$

where y is a vector of size $n \times 1$. Similarly, a dvtm operation is given by

$$y^T = x^T A \tag{2}$$

Both the operations, dgemv and dvtm are encountered in numerous engineering and scientific applications. Performance of these routines on multicore and GPGPU is shown in the figure 11.

It can be observed in the figure 11(a) that the performance attained by dgemv and dvtm goes up to 15 and 19 Gflops in multicore and 8 and 9 Gflops in the GPGPUs respectively. This performance is 15 and 20% of the theoretical peak in multicore and 5 and 7% in GPGPU respectively. For experiments, we have used OpenBLAS for multicores and cuBLAS for GPGPU. Performance in-terms of Gflops/watt is depicted in figure 1(c). It can be observed in the figure 1(c) that the performance of dgemv and dvtm in multicore is as low as 0.3 and 0.12 Gflops/watt respectively, while performance in GPGPU is even lower at 0.11 Gflops/watt in dgemv and 0.04 Gflops/watt in dvtm.

2) REDEFINE CGRA: Figure 2(a) depicts REDEFINE CGRA where Tiles are connected through a packet switched NoC. Each Tile contains a CE and a Router [11][12]. CEs in REDEFINE can be enhanced with tailor-made CFUs for different application domains [13][14].

It has been shown in the literature that REDEFINE is the perfect candidate for DLA computations and significantly outperforms state-of-the-art software packages on multicore and GPGPUs [6][7]. While implementing general matrix-matrix multiply (dgemm), it is observed that REDEFINE PE is capable of attaining performance improvement of 3-81x over the state-of-the-art realizations of dgemm as shown in figure 2(b).

B. Related Work

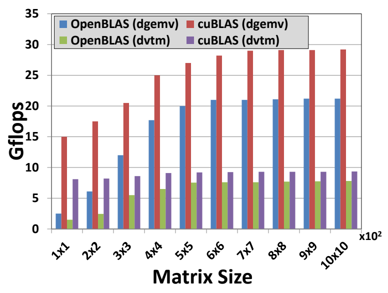
Due to wide range of applications of matrix computations, there have been several attempts in the literature to accelerate these computations on multicore, GPGPU, FPGA, and other platforms [15]. A magnificent amount of effort is spent in acceleration of compute bound operations like matrix-matrix multiply, QR factorization, LU factorization and Cholesky factorization [16][17]. Acceleration of basic linear algebra subprograms (BLAS) on CGRA is presented in [18] where the authors have presented extensive architectural exploration for selection of architectural parameters in implementation of the BLAS. A detailed evaluation of BLAS on FPGA is presented in [19] where the authors have compared performance of BLAS on multicores, FPGA and GPGPUs. The work presented in [19] has special emphasis on dgemv operation where authors have implemented a coarse-grained dot product unit to accelerate the matrix-vector operations. The work presented in [19] does not focus on run-time reconfigurability and hence supports limited matrix sizes. A universal matrix-vector multiplication architecture is presented in [20] that is capable of supporting dense as well as sparse matrix-vector operations. With the design presented in [20], it is observed that the sustained performance is 33% of the theoretical peak in 16 PEs while for 32 PEs the sustained performance is hardly 20% of the theoretical peak of the underlying platform that is emulated on an FPGA. Movidius Myriad presented in [21] and PEPSC presented in [22] are suitable for compute bound operations like matrix-matrix multiply. In this paper we present algorithm-architecture co-design for dgemv and dvtm to accelerate and attain energy efficient implementation of these routines first in a PE and then in REDEFINE CGRA where the PE acts as a CFU.

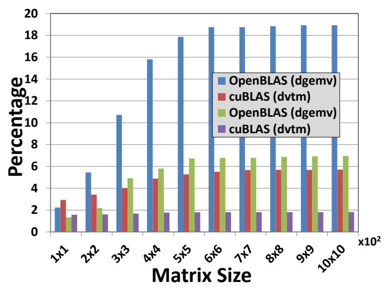
III. Analysis of dgemv and dvtm

We present DAG based analysis of dgemv and dvtm routines. From equation 1, for $n = 4$ the dgemv is given by

$$\begin{aligned} y &= Ax \\ \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} &= \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \\ &= \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 \\ a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 \end{bmatrix} \end{aligned} \tag{3}$$

It can be observed in equation 3 that the expressions to be computed for dgemv are regular in nature and dgemv can be realized as a series of inner products. Similar expressions can be arrived at for $n = 3$ and $n = 2$. DAGs for equation 3 are shown in figure 3 where DAGs for $n = 3$ and $n = 2$ are also depicted. The figure 3 shows computations of only one element in the result vector y since the DAGs for the other elements in the result vector y would be same as the one shown. It can be observed that the DAGs are very regular and dgemv can be realized as a series of inner products. From


(a) Performance of dgemv and dvtm in multicore and GPGPU In-terms of Gflops


(b) Performance of dgemv and dvtm in multicore and GPGPU In-terms of Percentage of Theoretical Peak Performance of the Underlying Platform

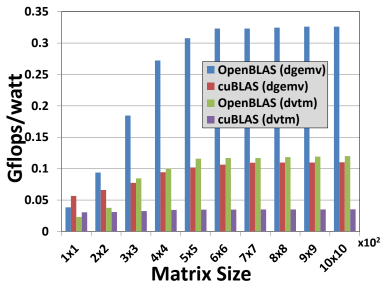
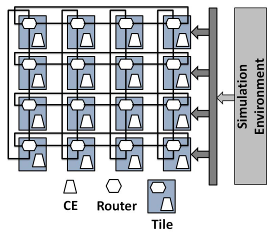

(c) Performance of dgemv and dvtm in multicore and GPGPU In-terms of Gflops/watt

Fig. 1. Performance of dgemv and dvtm in Multicores and GPGPU



(a) REDEFINE CGRA

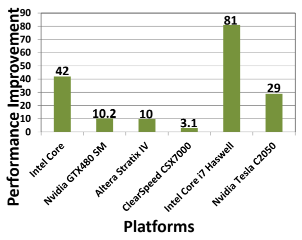

(b) Performance of REDEFINE PE in dgemm over multicore, GPGPU, FPGA and ClearSpeed Platforms

Fig. 2. Performance of dgemv and dvtm in Multicores and GPGPU

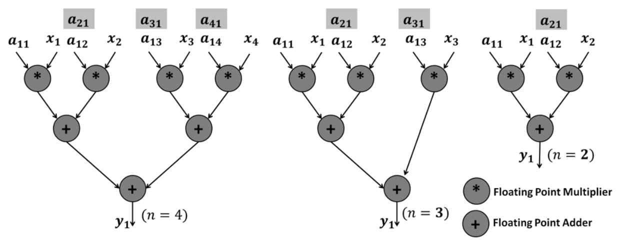


Fig. 3. DAGs in dgemv when Realized as a Series of Inner Products. Shaded Inputs are Applied While Implementing dvtm

equation 2, for $n = 4$, dvtm is given by

$$y^T = x^T A$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}^T = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11}x_1 + a_{21}x_2 + a_{31}x_3 + a_{41}x_4 \\ a_{12}x_1 + a_{22}x_2 + a_{32}x_3 + a_{42}x_4 \\ a_{13}x_1 + a_{23}x_2 + a_{33}x_3 + a_{43}x_4 \\ a_{14}x_1 + a_{24}x_2 + a_{34}x_3 + a_{44}x_4 \end{bmatrix} \quad (4)$$

It can be observed in the equations 3 and 4 that the dgemv and dvtm have same compute structure that is shown in the figure 3. It is also depicted in the figure 3 with shaded inputs that the compute structures required for realization of dgemv and dvtm are identical. Similar structures as shown in the figure 3 can be obtained for other values of n. The final value of n is limited by the architectural parameters such as size of the local register file and also the size of the memory as explained in [16] and [13]. For implementation of dgemv and dvtm in a PE, it is desirable to choose a hardware structure that matches with the structures shown in the figure 3 that is fully pipeline to exploit instruction level parallelism (ILP) in computation of resultant vector (or vector transpose) as well as ILP across the computation of the resultant vector (or vector transpose).

IV. Implementation and Results

For implementation of dgemv and dvtm, we choose a design that is depicted in figure 4. Operation of the PE shown in the figure 4 is given by following steps:

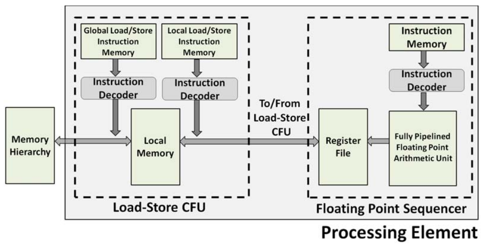


Fig. 4. PE Design

- **Step 1:** Load input data from the global memory (GM) to the local memory (LM). These operations are performed by the instruction in the Load-Store CFU. For matrix operations, the input data could be sub-matrices.
- **Step 2:** Load input data from LM to the local register file (RF) that is close to the floating point sequencer (FPS). As soon as the data is available in the RF, the computations begin.
- **Step 3:** While computations are performed, the final results are stored back to LM and from there the results are moved to the GM. For intermediate results, based on the re-usability of the intermediate results, the results are stored in the RF or LM and moved back and forth for computations.

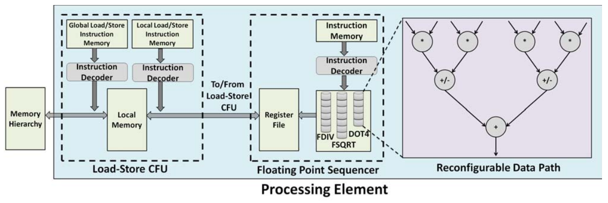


Fig. 5. PE Design Along with an RDP Tightly Coupled to the PE Pipeline

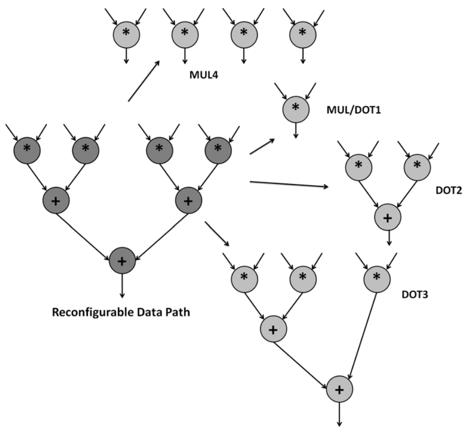


Fig. 6. Reconfigurations in the RDP to Support dgemv and dvtm Operations

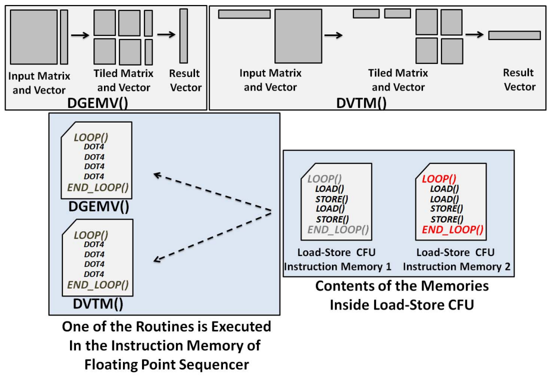


Fig. 7. Routines in PE

From above steps, it can be observed that the PE shown in the figure 4 is capable of attaining high performance through exploitation of ILP, re-usability of the data and overlap of computation and communication. We have applied micro-architectural enhancements in the PE that are suggested in [16]. Enhancement of PE with an RDP is shown in figure 5. For implementation of dgemv and dvtm we add support for two configurations in the RDP as shown in figure 6. As shown in the figure 5, the RDP is tightly coupled to the PE pipeline. Due to presence of RDP, there is no need for scalar multiplier or adder required in the PE as RDP can be configured to act as these scalar units. Different reconfigurations in RDP to support implementation of dgemv and dvtm are shown in the figure 6.

We implement instruction RECONFG that reconfigures the RDP where the reconfigured data-path serves as a data-path required for realization of dgemv/dvtm. The argument in the instruction can re-coinfigure data-path to implement MUL4, DOT1, DOT2, or DOT4 instructions. Data-paths for all these instructions are shown in the figure 6. We decide to realize these selected data-paths for implementation of dgemv and dvtm as these data-paths are the essential compute structures (macro operations) encountered in the DAGs of dgemv and dvtm routines. These macro operations are shown in the figure 3. We restrict the reconfigurable data-path to 4 multipliers and 3 adders as our aim is to accelerate matrix computations in general through hardware realization of macro operations encountered in these computations. The RDP can be extended in dimensions and functionality as explained in [23].

Memory contents of different memories in the PE as we realize dgemv and dvtm in the PE is shown in figure 7. Computation regimes for dgemv and dvtm are also shown in the figure 7. It can be observed in the figure 7 that while realizing dgemm or dvtm routines, the steps are:

- **Step 1:** Divide input matrix and input vector to appropriate tiles sizes of $b \times b$ and $b \times 1$. The size of the tile depends on the size of RF and the number of inputs to RDP. Since, in this case, the number of inputs that can be supplied to the RDP are 8, we choose $b = 4$
- **Step 2:** Move blocks of the matrix to the LM from the GM. Here, we ensure that the blocks of the matrices that are immediately required for computing the final results are moved as early as possible
- **Step 3:** Move input matrix blocks and vector elements to the local RF. As soon as the inputs to the RDP are available, the operation to be performed by the RDP starts the execution. Since, dgemv and dvtm are accumulation process, we ensure that the intermediate results are kept close to the compute units (RDP in this case) in the RF as long as possible until the final result is calculated
- **Step 4:** Once the final vector is calculated, the result is moved back to the GM. Mostly, the entire result vector is in LM since, it is not possible to accommodate the whole result vector in the RF

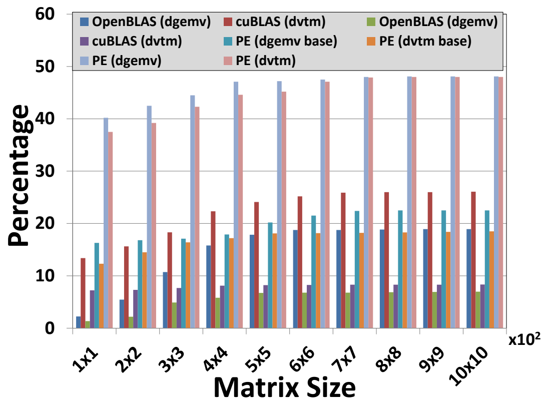


Fig. 8. Performance of dgemv and dvtm in mutlicore, GPGPU and PE

Performance improvement in dgemv and dvtm after employing RDP is shown in figure 8. It can be observed that the performance improvement is 2.2x compared to the base realization

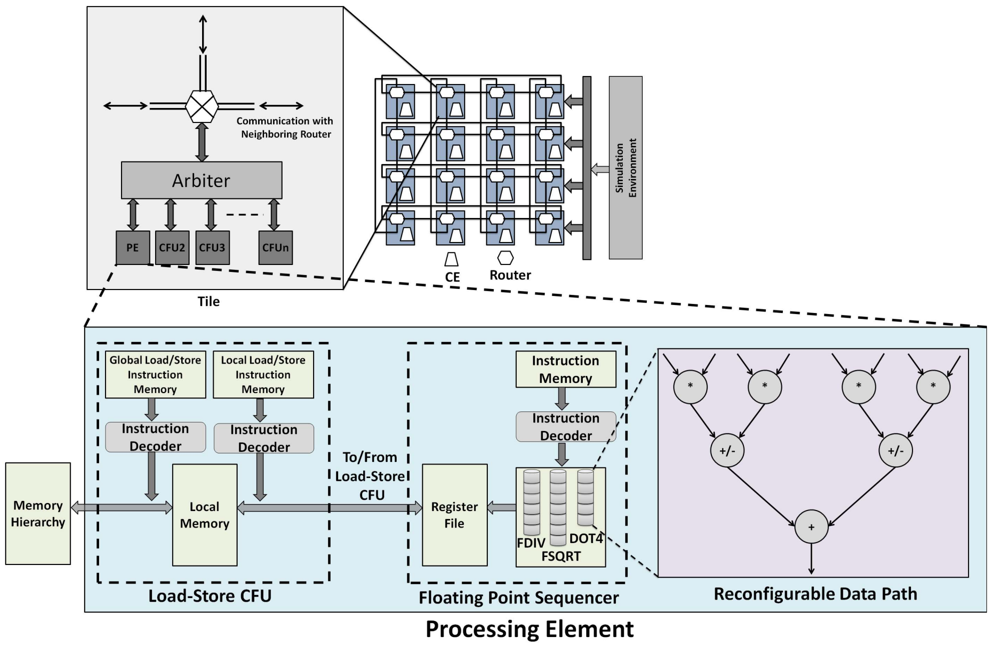


Fig. 9. PE as a CFU in REDEFINE

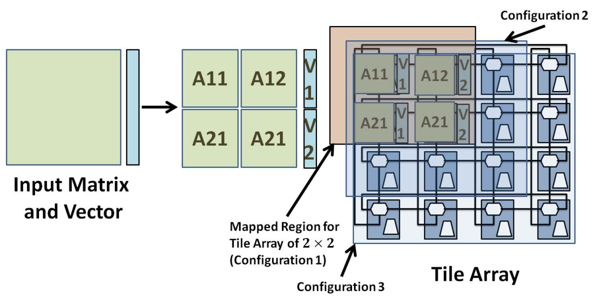


Fig. 10. Mapping of dgemv in REDEFINE Tile Array

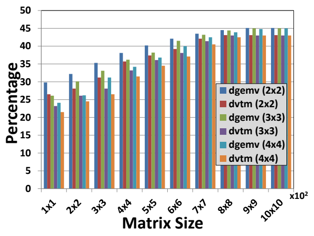
where base realization is the implementation without RDP. It can also be observed that the performance attained in memory bound routines dgemv and dvtm is close to 50% of the theoretical peak of the PE unlike multicore and GPGPU where the performance is ranging between 5-20% of the theoretical peak performance. It can also be observed in the figure 8 that the performance in base realization of dgemv and dvtm in the PE is similar to that of multicore and GPGPU. We report results in-terms of percentage of theoretical peak performance due to varying theoretical peak of different platforms.

A. Parallel Realization in REDEFINE CGRA

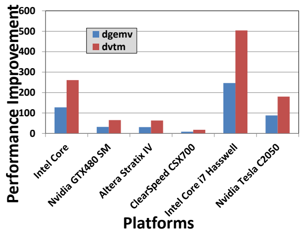
With promising results attained in the PE, we envision the PE as CFU in REDEFINE CEs. The logical place of the PE as CFU in REFEFINE is shown in figure 9.

An abstract view of a Tile is shown in the figure 9 where the PE is attached as a CFU. A unique feature of REDEFINE is that it can operate with multiple CFUs simultaneously. In our experiments, since we aim to accelerate memory bound DLA computations, we attach only PE as a CFU while detach rest of the CFUs. For parallel realization in REDEFINE CGRA with Tile array of size $T \times T$ and input matrix of size $n \times n$ following are the steps:

- Divide input matrix of size $n \times n$ into $T \times T$ blocks resulting in the block matrix of size $N \times N$ where $N = \frac{n}{T}$
- Similarly, divide vector of size $n \times 1$ to $N \times 1$ vector where $N = \frac{n}{T}$
- Depending on the size of the Tile array to be used, there has to be copies of the input vectors created and distributed among the Tiles for the operation in case of dgemv
- In case of dvtm, we choose to create copies of input vector instead of the matrix to save the memory space



(a) Performance of dgemv and dvtm in REDEFINE CGRA In-terms of Theoretical Peak Performance



(b) Performance Improvement In-terms of Gflops/watt in dgemv and dvtm in the PE over Other Platforms

Fig. 11. Performance of dgemv and dvtm in Multicores and GPGPU

Mapping of dgemv in REDEFINE with different configurations is shown in figure 10. Similar mapping can be applied for dvtm routine as well. Performance attained in dgemv and dvtm in-terms of theoretical peak performance of REDEFINE is shown in figure 11(a). It can be observed in the figure 11(a) that in REDEFINE, the dgemv and dvtm routines are able to attain performance of 45% and 43% of the theoretical peak respectively. Performance improvement in PE compared to multicores, GPGPU, and several other platforms is shown in figure 11(b). It can be observed that the PE is capable of attaining 3-200x performance improvement over the state-of-the-art realizations of dgemv and dvtm.

V. Conclusion

We presented a systematic approach for acceleration of memory bound operations like dgemv and dvtm through algorithm architecture co-design. A detailed directed acyclic graph based analysis of dgemv and dvtm was presented where several macro operations in these routines were identified. These macro operations are realized on a specialized reconfigurable data-path that can adapt at run-time for realization of macro operations. It was shown that in a processing element, memory bound operations are able to attain performance that is 50% of the theoretical peak of the processing element while in multicore and general purpose graphics processing units, these performance is ranging between 5-20%. In parallel realization in REDEFINE coarse-grained reconfigurable architecture, these routines are able to attain up to 45% of the theoretical peak. In REDEFINE with different configurations, it is shown that the realization is scalable.

References

[1] F. Merchant, T. Vatwani, A. Chattopadhyay, S. Raha, S. K. Nandy, and R. Narayan, "Achieving efficient realization of kalman filter on cgra through algorithm-architecture co-design," in *Applied Reconfigurable Computing. Architectures, Tools, and Applications*, N. Voros, M. Huebner, G. Keramidas, D. Goehringer, C. Antonopoulos, and P. C. Diniz, Eds. Cham: Springer International Publishing, 2018, pp. 119–131.

[2] E. Agullo, C. Coti, J. Dongarra, T. Hrault, and J. Langem, "Qr factorization of tall and skinny matrices in a grid computing environment," in *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, April 2010, pp. 1–11.

[3] Z. E. Rkossy, F. Merchant, A. Acosta-Aponte, S. K. Nandy, and A. Chattopadhyay, "Scalable and energy-efficient reconfigurable accelerator for column-wise givens rotation," in *2014 22nd International Conference on Very Large Scale Integration (VLSI-SoC)*, Oct 2014, pp. 1–6.

[4] F. A. Merchant, T. Vatwani, A. Chattopadhyay, S. Raha, S. K. Nandy, and R. Narayan, "Efficient realization of householder transform through algorithm-architecture co-design for acceleration of qr factorization," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–1, 2018.

[5] F. A. Merchant, T. Vatwani, A. Chattopadhyay, S. Raha, R. N. S. K. Nandy, and R. Leupers, "Applying modified householder transform to kalman filter," in *2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID)*, (in press) VLSI Design 2019.

[6] F. Merchant, T. Vatwani, A. Chattopadhyay, S. Raha, S. K. Nandy, and R. Narayan, "Achieving efficient qr factorization by algorithm-architecture co-design of householder transformation," in *2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID)*, Jan 2016, pp. 98–103.

[7] F. Merchant, A. Chattopadhyay, G. Garga, S. K. Nandy, R. Narayan, and N. Gopalan, "Efficient qr decomposition using low complexity column-wise givens rotation (cgr)," in *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*, Jan 2014, pp. 258–263.

[8] F. Merchant, N. Choudhary, S. K. Nandy, and R. Narayan, "Efficient realization of table look-up based double precision floating point arithmetic," in *2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID)*, Jan 2016, pp. 415–420.

[9] F. Merchant, A. Chattopadhyay, S. Raha, S. K. Nandy, and R. Narayan, "Accelerating blas and lapack via efficient floating point architecture design," *Parallel Processing Letters*, vol. 27, no. 03n04, p. 1750006, 2017. [Online]. Available: https://www.worldscientific.com/doi/abs/10.1142/S0129626417500062

[10] R. Chaurasiya, G. John, R. Shrestha, N. Jonathan, N. Sangeeth, N. Kaustav, F. Merchant, and R. Leupers, "Parameterized posit arithmetic hardware generator," in *2018 IEEE International Conference on Computer Design (ICCD)*, (in press) ICCD 2018.

[11] M. Alle, K. Varadarajan, A. Fell, R. R. C., N. Joseph, S. Das, P. Biswas, J. Chetia, A. Rao, S. K. Nandy, and R. Narayan, "Redefine: Runtime reconfigurable polymorphic asic," *ACM Trans. Embed. Comput. Syst.*, vol. 9, no. 2, pp. 11:1–11:48, Oct. 2009. [Online]. Available: http://doi.acm.org/10.1145/1596543.1596545

[12] J. Nimmy, C. R. Reddy, K. Varadarajan, M. Alle, A. Fell, S. K. Nandy, and R. Narayan, "Reconnect: A noc for polymorphic asics using a low overhead single cycle router," in *2008 International Conference on Application-Specific Systems, Architectures and Processors*, July 2008, pp. 251–256.

[13] M. Mahadurkar, F. Merchant, A. Maity, K. Vatwani, I. Munje, N. Gopalan, S. K. Nandy, and R. Narayan, "Co-exploration of nla kernels and specification of compute elements in distributed memory cgras," in *2014 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV)*, July 2014, pp. 225–232.

[14] N. T. Prashank, M. Prasadarao, A. Dutta, K. Varadarajan, M. Alle, S. K. Nandy, and R. Narayan, "Enhancements for variable n-point streaming fft/ifft on redefine, a runtime reconfigurable architecture," in *2010 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*, July 2010, pp. 178–184.

[15] A. Haidar, T. Dong, P. Luszczek, S. Tomov, and J. Dongarra, "Batched matrix computations on hardware accelerators based on gpus," *The International Journal of High Performance Computing Applications*, vol. 29, no. 2, pp. 193–208, 2015. [Online]. Available: https://doi.org/10.1177/1094342014567546

[16] F. Merchant, A. Maity, M. Mahadurkar, K. Vatwani, I. Munje, M. Krishna, S. Nalesh, N. Gopalan, S. Raha, S. K. Nandy, and R. Narayan, "Micro-architectural enhancements in distributed memory cgras for lu and qr factorizations," in *2015 28th International Conference on VLSI Design*, Jan 2015, pp. 153–158.

[17] Z. E. Rkossy, F. Merchant, A. Acosta-Aponte, S. K. Nandy, and A. Chattopadhyay, "Efficient and scalable cgra-based implementation of column-wise givens rotation," in *2014 IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors*, June 2014, pp. 188–189.

[18] Z. E. Rákossy, D. Stengele, A. Acosta-Aponte, S. Chafekar, P. Bientinesi, and A. Chattopadhyay, "Scalable and efficient linear algebra kernel mapping for low energy consumption on the layers cgra," in *Applied Reconfigurable Computing*, K. Sano, D. Soudris, M. Hübner, and P. C. Diniz, Eds. Cham: Springer International Publishing, 2015, pp. 301–310.

[19] S. Kestur, J. D. Davis, and O. Williams, "Blas comparison on fpga, cpu and gpu," in *2010 IEEE Computer Society Annual Symposium on VLSI*, July 2010, pp. 288–293.

[20] S. Kestur, J. D. Davis, and E. S. Chung, "Towards a universal fpga matrix-vector multiplication architecture," in *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, April 2012, pp. 9–16.

[21] M. H. Ionica and D. Gregg, "The movidius myriad architecture's potential for scientific computing," *IEEE Micro*, vol. 35, no. 1, pp. 6–14, Jan 2015.

[22] A. Sethia, G. Dasika, T. Mudge, and S. Mahlke, "A customized processor for energy efficient scientific computing," *IEEE Transactions on Computers*, vol. 61, no. 12, pp. 1711–1723, Dec 2012.

[23] S. Das, K. Madhu, M. Krishna, N. Sivanandan, F. Merchant, S. Natarajan, I. Biswas, A. Pulli, S. Nandy, and R. Narayan, "A framework for post-silicon realization of arbitrary instruction extensions on reconfigurable data-paths," *Journal of Systems Architecture*, vol. 60, no. 7, pp. 592 – 614, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1383762114000836