Achieving Efficient Realization of Kalman Filter on CGRA through Algorithm-Architecture Co-design

Farhad Merchant[1], Tarun Vatwani[2] Anupam Chattopadhyay[2], Soumyendu Raha[3], S K Nandy[4], and Ranjani Narayan[5]

[1] Institute for Communication Technologies and Embedded Systems, RWTH Aachen University, Germany
[2] Hardware and Embedded Systems Lab, Nanyang Technological University, Singapore
[3] Scientific Computation Laboratory, Indian Institute of Science, Bangalore
[4] Computer Aided Design Laboratory, Indian Institute of Science, Bangalore
[5] Morphing Machines Pvt. Ltd.

Abstract. In this paper, we present efficient realization of Kalman Filter (KF) that can achieve up to 65% of the theoretical peak performance of underlying architecture platform. KF is realized using Modified Faddeeva Algorithm (MFA) as a basic building block due to its versatility and REDEFINE Coarse Grained Reconfigurable Architecture (CGRA) is used as a platform for experiments since REDEFINE is capable of supporting realization of a set algorithmic compute structures at run-time on a Reconfigurable Data-path (RDP). We perform several hardware and software based optimizations in the realization of KF to achieve 116% improvement in terms of Gflops over the first realization of KF. Overall, with the presented approach for KF, 4-105x performance improvement in terms of Gflops/watt over several academically and commercially available realizations of KF is attained. In REDEFINE, we show that our implementation is scalable and the performance attained is commensurate with the underlying hardware resources[6].

Keywords: kalman filter; reconfigurable architectures; computation; parallelism

1 Introduction

Coarse Grained Reconfigurable Architectures (CGRAs) have been active topic of research due to their power performance and flexibility [9]. CGRAs are capable of domain customization and they are targeted to achieve performance of Application Specific Integrated Circuits (ASICs) and flexibility of Field Programmable Gate Arrays (FPGAs) through presence of ASIC-like structures [14][8]. Typically, CGRAs occupy middle ground between ASICs and FPGAs [16][17]. Furthermore, CGRAs are preferred as a platform in the application domains like signal processing and automotive that are embedded in nature [9]. Acceleration of scientific code on CGRA with high precision floating point arithmetic is yet to be fully explored. There exist very few attempts in the literature where computations like double precision General Matrix Multiplication

[6] The paper has been accepted in ARC 2018

(dgemm), QR factorization (dgeqrf), and LU factorization (dgetrf) are accelerated on CGRAs. Scientific application like Kalman Filter (KF) is rich in these matrix operations and has wide range of applications from trajectory optimization, and navigation to econometric [1]. Here we see an opportunity to accelerate KF using a highly efficient Dense Linear Algebra (DLA) accelerator. Since, KF can be transformed as a series of matrix operations, we use Modified Faddeeva Algorithm (MFA) as a basic building block for our realization of KF [15]. We take an approach of algorithm-architecture co-design where we identify several macro operations in the routines required for MFA and realize them on a specialized data path that leads to significant performance improvement in KF. Major Contributions in this paper are as follows:

- We adopt a library based approach and realize KF using MFA where MFA is realized using dgemm, dgetrf, and dgeqrf routines. The first realization is capable of achieving 30% of the theoretical peak performance of the underlying platform. We call this implementation as a base realization of KF
- A set of macro operations identified in dgemm, dgeqrf, and dgetrf are realized on a tightly coupled Reconfigurable Data-path (RDP) and the implementation results in 66% of performance improvement over base realization at minimal area and energy cost. We call this implementation as hardware optimized KF
- A scheduling optimization is presented for KF that results in 30% performance improvement over the hardware optimized realization and 116% improvement over base realization. We also show that the final implementation is able to achieve 4-105x performance improvement over several academic and commercial realizations of KF. We call this implementation as software optimized KF
- Parallel realization of KF is presented on REDEFINE and it is shown that REDEFINE scales well with increasing size of co-variance matrix

For our experiments, we use Processing Element (PE) design presented in [9]. We optimize Floating Point Unit (FPU) design presented in [13] with recommendations presented in [12] for optimum Instructions Per Cycle (IPC). The paper is organized as follows: In section 2, MFA, KF and REDEFINE are discussed. In section 3, we present case studies on multicore and General Purpose Graphics Processing Unit (GPGPU) and show that the performance attained on these platforms even with highly tuned software packages is not satisfactory. KF implementation is discussed in section 4. Parallel realization of KF and results are discussed in section 5. We conclude our work in section 6.

2 Background and Related Work

In this section, we first discuss MFA and KF briefly and show connection between MFA and KF. In section 2.1, we introduce REDEFINE and discuss unique features of REDEFINE CGRA. We focus on some of the recent realizations of KF in section 2.2.

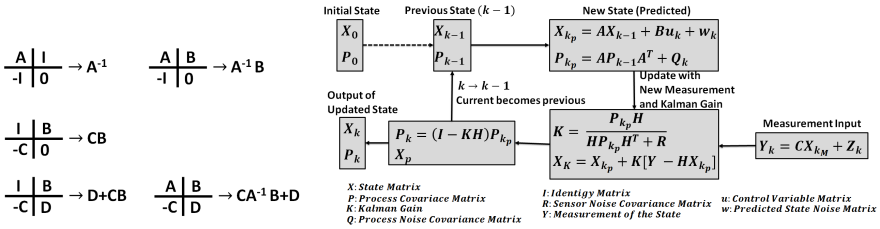
2.1 Background

Modified Faddeeva Algorithm and Kalman Filter MFA was originally presented in [15] that is an enhancement over the original Faddeeva algorithm. QR factorization was incorporated instead of Gaussian elimination process to improve stability of Faddeeva algorithm. For rectangular matrices $A_{m\times n}, B_{m\times p}, C_{k\times n}$, and $D_{k\times p}$, compound matrix is shown in equation 1.

$$M = \begin{bmatrix} A & B \\ -C & D \end{bmatrix} \tag{1}$$

Applying MFA on compound matrix M incorporates following two steps.

- **Step 1:** Upper triangularization of matrix A using QR factorization and update matrix B
- **Step 2:** Annihilate matrix C using diagonal elements of upper traingularized matrix A



(a) Several Operations that can be Performed Using MFA based on Different Initial Values of Matrices A, B, C, and D

(b) Multi Dimensional Kalman Filter

Fig. 1. MFA and KF

At the end of an MFA operation, the matrix D is of the form $D + CA^{-1}B$ that is also known as the Schur complement. Versatility of MFA is described in figure 1(a) where it is shown that based on different initial input matrices, several operations like matrix multiplication, matrix addition, and linear system solution can be obtained. Due to versatility in MFA, it is desirable to write applications in terms of matrix operation that can translate into series of calls of MFA. One such application is KF. A simplistic multi-dimensional KF is shown in the figure 1(b).

An iteration of KF requires complex matrix operations ranging from matrix multiplication to computation of numerical stable matrix inverse. One such classical GR based numerical stable approach is presented in [20]. From figure 1(b), matrix inverse being the most complex operation in KF, and computation of inverse using QR factorization being a numerical stable process, proposed library based approach is the most suitable for such applications.

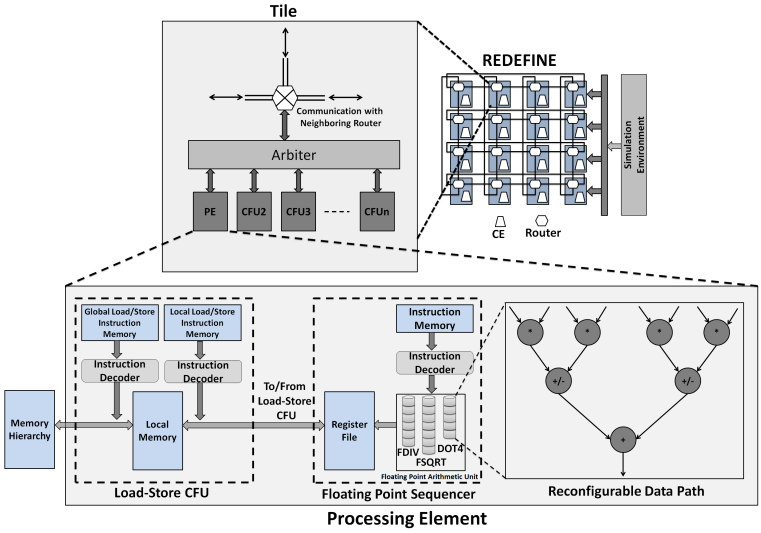


Fig. 2. REDEFINE CGRA and Design of PE (with RDP)

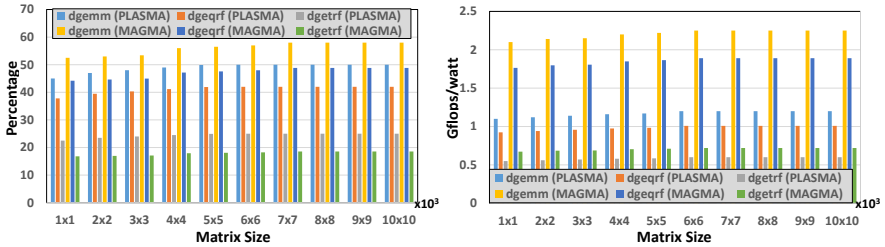
REDEFINE REDEFINE is a CGRA that can be domain customized for variety application domains [10].In REDEFINE, several Tiles are connected through a packet switched Network-on-Chip (NoC) [11][10]. Each Tile consists of a Compute Element (CE) and a Router. CEs in REDEFINE can be enhanced with a Custom Function Units (CFUs) that can be tailored for a desired application domain. For our implementation, we use a Processing Element (PE) that is highly customized for Dense Linear Algebra (DLA) computations [9]. REDEFINE micro-architecture along with PE is shown in figure 2. It can be observed in the figure 2 that the PE consists of a RDP as an arithmetic unit that can be reconfigured at run-time through instruction to realize identified macro operations in the Directed Acyclic Graph (DAG) of DLA computations. It is well established in the literature that such an approach yields significant improvement in energy efficiency in the overall system [3].

2.2 Related Work

Due to wide range of applications of KF, there are several implementations of KF in the literature. We present brief summary of some of the recent realizations of KF. Early realizations of KF are based on systolic array since systolic arrays are highly suitable for matrix computations [4]. One of the recent realization of KF presented in [22] is targeted for advanced driver assistance system. The implementation presented in [22] does not focus on the acceleration of matrix operations encountered in KF and hence leaves opportunities for further performance improvements. Similarly, KF implementation presented in [2] focuses on parallel implementation using OpenMP and FPGA based implementation presented in [18] focuses on optimizations in arithmetic operations in KF for performance improvement. MFA based KF presented in [7] focuses on

denoising of images with additive white Gaussian noise. Although the work presented in [7] realized KF using MFA, the implementation is not scalable and can operate only on the images of size 512×512. Furthermore, focus of the work presented in [7] is not on acceleration of KF through acceleration of matrix operations in MFA. In this paper, we focus on energy efficient fast realization of KF by accelerating gemm, QR factorization, and LU factorization.

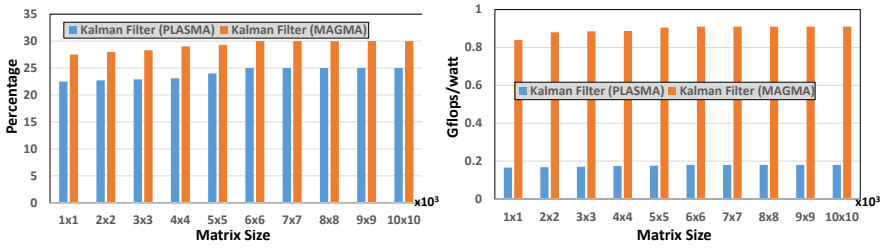
3 Case Studies



(a) Percentage of Theoretical Peak Performance Attained in dgemm, dgeqrf, and dgetrf in in Intel Core i7 (PLASMA) and Nvidia Tesla C2075 (MAGMA)

(b) Performance Attained in terms of Gflops/watt in dgemm, dgeqrf, and dgetrf in Intel Core i7 (PLASMA) and Nvidia Tesla C2075 (MAGMA)

Fig. 3. Performance of dgemm, dgeqrf, dgetrf, and KF in multicore and GPGPU



(a) Percentage of Theoretical Peak Performance Attained in KF in Intel Core i7 (PLASMA) and Nvidia Tesla C2075 (MAGMA)

(b) Performance Attained in terms of Gflops/watt in KF in Intel Core i7 (PLASMA) and Nvidia Tesla C2075 (MAGMA)

Fig. 4. Performance of dgemm, dgeqrf, dgetrf, and KF in multicore and GPGPU

In this section we present case studies on the operation encountered in MFA. Since, we implement KF using double precision FPU, we present case studies on dgeqrf,

dgetrf, and dgemm. We discuss performance of dgemm, dgeqrf, and dgetrf in highly optimized software libraries like Parallel Linear Algebra Software for Multicore Architectures (PLASMA) and Matrix Algebra for GPU and Multicore Architectures (MAGMA).

3.1 dgeqrf

Routine dgeqrf computes QR factorization of an input matrix A and returns uppter triangular matrix R and Q where $QQ^T = Q^TQ = I$. Householder Transform (HT) based QR factorization routine in Linear Algebra Package (LAPACK) is shown in algorithm 1.

Input: Matrix $A_{m \times n}$
Output: Upper triangle matrix R
for $i = 1$ *to* n **do**
  Compute Householder vector v
  Compute P where $P = I - 2vv^T$
  Update trailing matrix using $dgemv$
end

Algorithm 1: dgeqr2 in LAPACK (dgemv is double precision matrix-vector multiplication)

Input: Matrix $A_{m \times n}$
Output: Upper triangle matrix R
for $i = 1$ *to* n **do**
  Compute Householder vectors for block column $m \times k$
  Compute P where P is multiplication of Householder vectors
  Update trailing matrix using $dgemm$
end

Algorithm 2: dgeqrf in LAPACK

In algorithm 1, majority of the computations are performed in terms of double precision matrix vector multiplication (dgemv). Since dgemv is a memory bound operation, the realization of dgeqr2 yields hardly 10% of the theoretical peak performance in GPGPU and 2-3% of the theoretical peak performance in multicores. In the dense linear algebra software packages like LAPACK, PLASMA, and MGAMA, the software routines are tiled/blocked and written in terms of dgemm for efficient exploitation of memory hierarchy of the underlying platform where dgemm is compute bound operation [19]. dgeqrf routine that uses dgeqr2 and dgemm as subroutines is shown in algorithm 2.

3.2 dgetrf

dgetrf routine computes LU factorization of a general $m \times n$ matrix, so $A = PLU$ where L is unit lower triangular, U is upper triangular matrix, and P is permutation matrix. dgetrf2 routine is shown in algorithm 3.

Input: Matrix $A_{m \times n}$
Output: Unit lower triangular matrix L and upper triangular matrix U
for $i = 1$ *to* n **do**
    Find pivot
    Interchange pivot rows
    Compute elements of L matrix
    Update the trailing matrix using dgemv
end

Algorithm 3: dgetrf2 in LAPACK

Input: Matrix $A_{m \times n}$
Output: Unit lower triangular matrix L and upper triangular matrix U
for $i = 1$ *to* n **do**
    Find pivot
    Interchange pivot rows
    Compute elements of L matrix
    Update the trailing matrix using dgemm
end

Algorithm 4: dgetrf in LAPACK

A similar approach to QR factorization is adopted for LU factorization in LAPACK where a blocked routine shown in the algorithm 4 is developed to exploit the memory hierarchy of cache based platforms. For our implementation, since we do not require pivoting, we remove pivoting related routine from dgetrf2 and adopt the routine to support realization of MFA.

3.3 dgemm

Due to prevalent in the literature, we do not reproduce pseudo code of dgemm. Typically, dgemm is part of Level-3 Basic Linear Algebra Subprograms (BLAS) and used as a subroutine in the operations like LU and QR factorizations since dgemm is compute bound operation and theoretically dgemm is capable of attaining $O(n)$ computations to communication ratio for the matrices of size $n \times n$ [5].

3.4 Performance Evaluation of dgeqrf, dgetrf and dgemm on multicore and GPGPU

We evaluate performance of dgeqrf, dgetrf, and dgemm on multicore and GPGPU. Performance in terms of percentage of theoretical peak performance in Intel Core i7 for dgeqrf, dgetrf, and dgemm is shown in figure 3(a). It can be observed in the figure 3(a) that in Intel Core i7, performance attained by dgeqrf is 42% (21 Gflops), performance attained by dgetrf is 25% (12 Gflops) and performance attained by dgemm is 50% (24 Gflops). Similarly, percentage of theoretical peak performance attained in GPGPU for dgeqrf is 48% (247.2 Gflops), for dgemm it is 58% (298 Gflops), and for dgetrf it is 20% (103 Gflops). We ensure to compile PLASMA and MAGMA with OpenBLAS [21]. Performance attained in terms of Gflops/watt ranges between 0.6 to 2.1 for GPGPU and 0.6 to 1.1 for multicore as shown in figure 3(b). It can be observed that the performance attained by dgeqrf is 84.2% of the performance attained by dgemm and performance attained by dgetrf is 50% of the performance attained by dgemm. This is mainly due to presence of non-parallelizable computations in dgeqrf and dgetrf that limits the performance considering Amdahl's law [6].

Similarly, for KF the performance attained in terms of percentage of theoretical peak performance is 25% for multicore and 30% for GPGPU as shown in figure 4(a). Performance attained in terms of Gflops/watt for KF is 0.15 in multicore and 0.9 in GPGPU. Based on our case studies in dgemm, dgeqrf, and dgetrf it can be inferred that the performance attained in the latest multicore and GPGPU is hardly 50-52% even for highly parallel operations like dgemm. Due to this shortcoming of multicore and GPGPU, we choose a customizable platform for our implementation presented in [9] that is capable of achieving up to 74% of the theoretical peak in dgemm [9][14].

4 Kalman Filter Realization in Processing Element

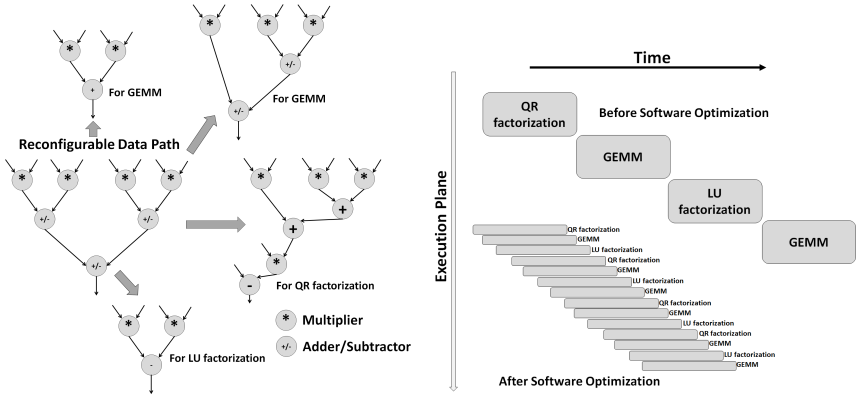
We present realization of KF on PE depicted in the figure 2. Three different implementations of KF are presented here: 1) base implementation of KF, 2) hardware optimized KF, and 3) software optimized KF.

4.1 Base Implementation of KF

In base implementation of KF, we realize matrix operation of in the MFA using scalar multiplier and adder in the PE. In base realization of KF, we are able to achieve up to 30% of the theoretical peak performance of the PE. Here, since we are using only one multiplier and adder for our implementation, the theoretical peak performance of the PE is 1.4 Gflops at 700 MHz.

4.2 Hardware Optimized KF

In hardware optimized KF, we revisit the basic operations required to be performed in MFA like dgemm, dgeqrf, and dgetrf and identify several macro operations in these basic operations. We realize these macro operations in RDP depicted in the figure 2.



(a) Different Configurations of RDP Corresponding to Identified Macro Operations in dgemm, dgeqrf, and dgetrf

(b) Software Optimization in KF Resulting in Reduction in the Run-time (logical diagram)

Fig. 5. RDP Configurations for KF and Scheduling of Different Routines in KF

Configurations of RDP corresponding to the identified macro operations in dgemm, dgeqrf and dgetrf are shown in figure 5(a). With these configurations, we achieve up to 50% of the theoretical peak of the PE where theoretical peak of the PE is 4.9 Gflops at 700 MHz. Here theoretical peak of the PE is increased since we are using RDP that consists of 4 multipliers and 3 adders.

We perform a software optimization in MFA by analysis of the DAG of MFA. We overlap dgeqrf, dgemm, and dgetrf routines as shown in figure 5(b). Optimization diagram shown in the figure 5(b) is logical flow of computations post software optimization. To overlap these routines, we identify pipeline stalls in the PE while execution and insert independent instructions while also maintaining operation correctness. Overlapping dgeqrf, dgemm, and dgetrf results in significant reduction in the run-time of MFA that directly translates to performance improvement in KF. After software optimization, we are able to attain 65% of the theoretical peak in PE which is 30% improvement. Performance improvement after each optimization is shown in figure 6. In the figure 6, we have also incorporated the performance attained in multicore and GPGPU.

It can be observed in the figure 6 that the performance of KF in multicore and GPGPU is hardly 20-30% of the theoretical peak of these platforms, while we achieve up to 65% of the theoretical peak of PE in KF which is 2.15x higher.

5 Parallel Realization and Results

For parallel realization of KF, we use three different configurations of REDEFINE. Two configurations are shown in figure 7. In configuration 1 we use 2×2 Tile array, in configuration 2 we use 3×3 Tile array, and in configuration 3, we use 4×4 Tile array. In our simulations, for configuration 1 and configuration 2, we use last column of the Tile array as a memory where we attach memories as a PE.

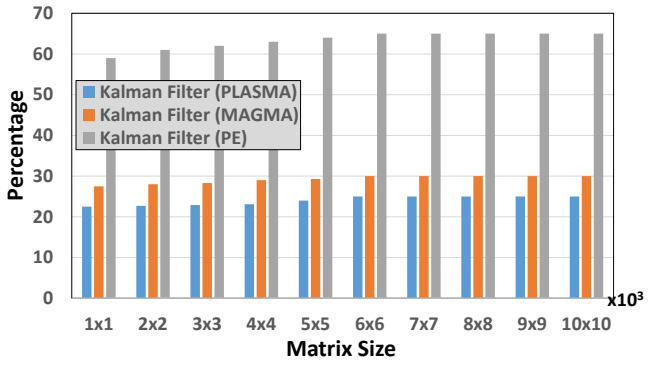


Fig. 6. Performance of KF in PE, multicore, and GPGPU

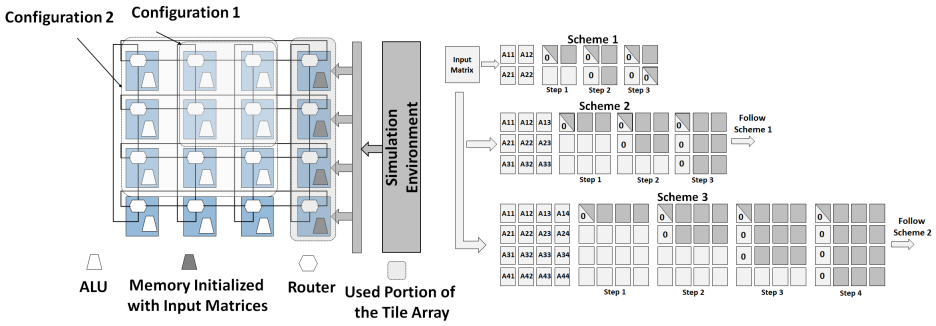
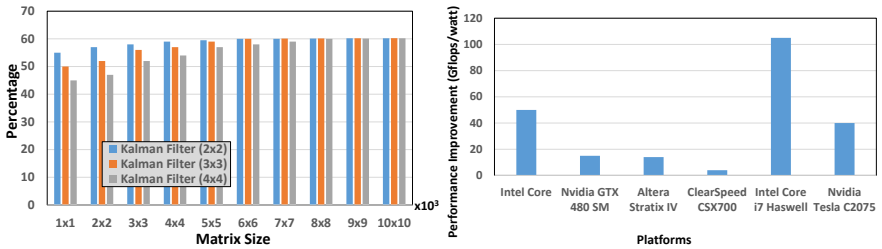


Fig. 7. Different Configurations of REDEFINE for KF Realization and Scheduling



(a) Performance in Different Configurations of REDEFINE in KF Realization

(b) Power Performance Comparison of PE with Other Platforms for KF

Fig. 8. RDP Configurations for KF and Scheduling of Different Routines in KF

In configuration 3, we use entire Tile array for computations and hence we attach another memory PE to the Router along side the compute PE. Memory PE is divided into to segments and the second half of the segment in memory PE acts as a global memory that is accessible by all other Tiles while the first half of the memory segment

is private to the compute PE that is used for computations on local data. Typically, we have 256K bytes of memory per Tile and compute PE consists of 256 registers of width 64 bits. Scheduling technique for REDEFINE is shown in the right side of the figure 7. We use a technique where we divide input matrix into $k \times k$ blocks where k is size of row/column of the Tile array. These blocks are further divided into the sub-blocks where size of the sub-blocks depend on the number of local registers available and size of the local memory available to a PE. Blocks of the matrices are loaded and computation is performed and the result is stored to the global memory. Percentage of theoretical peak performance attained for each configuration is shown in figure 8(a). It can be observed in the figure 8(a) that the performance attained for each configuration saturates at 60% of the theoretical peak performance which is 2x higher than the performance attained in multicore and GPGPU.

We evaluate power performance of PE based on technique presented in [10]. We compare PE with other platform for KF as shown in figure 8(b). It can be observed that the PE is capable of achieving 4-105x higher performance improvement over platforms like ClearSpeed CSX700, Intel Core i7, and Nvidia GPGPU.

6 Conclusion

In this paper, we presented efficient realization of KF. We used versatile MFA as a tool for efficient realization of KF. Based on the case studies presented on dgemm, dgetrf, and dgeqrf, it was identified that the performance of these operations on multiore and GPGPU is not satisfactory even with hightly optimized software pachages like PLASMA and MAGMA. It was also shown that the performance attained in KF on multicore and GPGPU is 20-30% of the theoretical peak performance of underlying platform. To accelerate KF on customizable platform like REDEFINE, we identify macro operations in the routines of MFA and realized them on RDP. Our approach resulted in 67% improvement in the performance of KF. A software tuning in MFA was also presented that resulted in 30% performance improvement over the hardware optimized KF. Overall, our approach of algorithm-architecture co-design resulted in 116% of performance improvement over the base realization of KF. In terms of Gflops/watt, KF is 2-105x better than multicore and GPGPU platforms.

References

1. A. Baluni, F. Merchant, S. K. Nandy, and S. Balakrishnan. A fully pipelined modular multiple precision floating point multiplier with vector support. In *2011 International Symposium on Electronic System Design*, pages 45–50, Dec 2011.
2. Giuseppe Cerati, Peter Elmer, Steven Lantz, Ian MacNeill, Kevin McDermott, Dan Riley, Matev Tadel, Peter Wittich, Frank Wrthwein, and Avi Yagil. Traditional tracking with kalman filter on parallel architectures. *Journal of Physics: Conference Series*, 608(1):012057, 2015.
3. Saptarsi Das, Kavitha T. Madhu, Madhav Krishna, Nalesh Sivanandan, Farhad Merchant, Santhi Natarajan, Ipsita Biswas, Adithya Pulli, S. K. Nandy, and Ranjani Narayan. A framework for post-silicon realization of arbitrary instruction extensions on reconfigurable datapaths. *Journal of Systems Architecture - Embedded Systems Design*, 60(7):592–614, 2014.

4. W. M. Gentleman and H. T. Kung. Matrix triangularization by systolic arrays. In *SPIE Proceedings*, volume 298, pages 19–26, 1982.
5. Nicholas J. Higham. Exploiting fast matrix multiplication within the level 3 BLAS. *ACM Trans. Math. Softw.*, 16(4):352–368, 1990.
6. Mark D. Hill and Michael R. Marty. Amdahl's law in the multicore era. *Computer*, 41(7):33–38, July 2008.
7. B. Johnson, N. Thomas, and J. S. Rani. An fpga based high throughput discrete kalman filter architecture for real-time image denoising. In *2017 30th International Conference on VLSI Design and 2017 16th International Conference on Embedded Systems (VLSID)*, pages 55–60, Jan 2017.
8. Mahesh Mahadurkar, Farhad Merchant, Arka Maity, Kapil Vatwani, Ishan Munje, Nandhini Gopalan, S. K. Nandy, and Ranjani Narayan. Co-exploration of NLA kernels and specification of compute elements in distributed memory cgras. In *XIVth International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS 2014, Agios Konstantinos, Samos, Greece, July 14-17, 2014*, pages 225–232, 2014.
9. 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 *VLSI Design (VLSID), 2015 28th International Conference on*, pages 153–158, Jan 2015.
10. 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(99):1–1, 2018.
11. Farhad Merchant, Anupam Chattopadhyay, Ganesh Garga, S. K. Nandy, Ranjani Narayan, and Nandhini 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, Mumbai, India, January 5-9, 2014*, pages 258–263, 2014.
12. Farhad Merchant, Anupam Chattopadhyay, Soumyendu Raha, S. K. Nandy, and Ranjani Narayan. Accelerating BLAS and LAPACK via efficient floating point architecture design. *Parallel Processing Letters*, 27(3-4):1–17, 2017.
13. Farhad Merchant, Nimash Choudhary, S. K. Nandy, and Ranjani Narayan. Efficient realization of table look-up based double precision floating point arithmetic. In *29th International Conference on VLSI Design and 15th International Conference on Embedded Systems, VLSID 2016, Kolkata, India, January 4-8, 2016*, pages 415–420, 2016.
14. Farhad Merchant, Tarun Vatwani, Anupam Chattopadhyay, Soumyendu Raha, S. K. Nandy, and Ranjani Narayan. Achieving efficient QR factorization by algorithm-architecture co-design of householder transformation. In *29th International Conference on VLSI Design and 15th International Conference on Embedded Systems, VLSID 2016, Kolkata, India, January 4-8, 2016*, pages 98–103, 2016.
15. J. G. Nash and S. Hansen. Modified faddeeva algorithm for concurrent execution of linear algebraic operations. *IEEE Transactions on Computers*, 37(2):129–137, Feb 1988.
16. Zoltán Endre Rákossy, Farhad Merchant, Axel Acosta Aponte, S. K. Nandy, and Anupam Chattopadhyay. Efficient and scalable cgra-based implementation of column-wise givens rotation. In *ASAP*, pages 188–189, 2014.
17. Zoltán Endre Rákossy, Farhad Merchant, Axel Acosta Aponte, S. K. Nandy, and Anupam Chattopadhyay. Scalable and energy-efficient reconfigurable accelerator for column-wise givens rotation. In *22nd International Conference on Very Large Scale Integration, VLSI-SoC, Playa del Carmen, Mexico, October 6-8, 2014*, pages 1–6, 2014.

18. F. Sandhu, H. Selamat, S. E. Alavi, and V. Behtaji Siahkal Mahalleh. Fpga-based implementation of kalman filter for real-time estimation of tire velocity and acceleration. *IEEE Sensors Journal*, 17(17):5749–5758, Sept 2017.
19. Brian J. Smith. R package magma: Matrix algebra on gpu and multicore architectures, version 0.2.2, September 3, 2010. [On-line] http://cran.r-project.org/package=magma.
20. C.L. Thornton and G.J. Bierman. Givens transformation techniques for kalman filtering. *Acta Astronautica*, 4(78):847 – 863, 1977.
21. Qian Wang, Xianyi Zhang, Yunquan Zhang, and Qing Yi. Augem: Automatically generate high performance dense linear algebra kernels on x86 cpus. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 25:1–25:12, New York, NY, USA, 2013. ACM.
22. Guanwen Zhong, Smaïl Niar, Alok Prakash, and Tulika Mitra. Design of multiple-target tracking system on heterogeneous system-on-chip devices. *IEEE Trans. Vehicular Technology*, 65(6):4802–4812, 2016.