

Accelerating BLAS and LAPACK via Efficient Floating Point Architecture Design

Farhad Merchant, Anupam Chattopadhyay, *Senior Member, IEEE*, Soumyendu Raha,
S K Nandy, *Senior Member, IEEE*, and Ranjani Narayan

Abstract—Basic Linear Algebra Subprograms (BLAS) and Linear Algebra Package (LAPACK) form basic building blocks for several High Performance Computing (HPC) applications and hence dictate performance of the HPC applications. Performance in such tuned packages is attained through tuning of several algorithmic and architectural parameters such as number of parallel operations in the Directed Acyclic Graph of the BLAS/LAPACK routines, sizes of the memories in the memory hierarchy of the underlying platform, bandwidth of the memory, and structure of the compute resources in the underlying platform. In this paper, we closely investigate the impact of the Floating Point Unit (FPU) micro-architecture for performance tuning of BLAS and LAPACK. We present theoretical analysis for pipeline depth of different floating point operations like multiplier, adder, square root, and divider followed by characterization of BLAS and LAPACK to determine several parameters required in the theoretical framework for deciding optimum pipeline depth of the floating operations. A simple design of a Processing Element (PE) is presented and shown that the PE outperforms the most recent custom realizations of BLAS and LAPACK by 1.1X to 1.5X in GFlops/W, and 1.9X to 2.1X in Gflops/mm².

Index Terms—Parallel computing; instruction level parallelism; power-performance trade-offs; high performance computing; floating point unit

1 INTRODUCTION

Domain specific computing platforms have gained immense popularity in the last decade. For domain specific computing, custom architectures are developed for efficient realization of several algorithms/computations pertaining to the domain of interest. Several architectural parameter such as size of the memory, bandwidth in the memory subsystem, and compute resource choices are chosen that are specific to the domain of interest. For domain specific computing, accelerators are preferred as an ideal underlying platform due to their better power performance over general purpose computers [1][2][3]. While accelerators like General Purpose Graphic Processing Units (GPGPUs) dissipate more power than desired, there are several domain specific accelerators designed to overcome this shortcoming of GPGPUs [2][4].

Domain customized platforms and/or accelerators are gaining popularity due to their area and power performance [5][6][7][8]. Performance in these accelerators is achieved by setting several architectural parameters that are well suited for computations pertaining to the domain. Parameters such as size of the memory at different levels and bandwidth of the memory that is nearest to the compute resources is well experimented in the literature [2]. Through pipelining of the processor and memory subsystem it is ensured that the processor is able to operate at the highest possible speed with lowest power penalty for the technology node [9][10]. Several design space exploration techniques are developed to

arrive at an optimum architectural parameters for optimal performance in the domain. These techniques are computer architecture simulator based techniques and allow tweaking of parameters such as memory size and memory bandwidth.

Basic Linear Algebra Subprograms (BLAS) and Linear Algebra Package (LAPACK) and/or their platform dependent variants are the basic building block for several high level software packages like Intel’s DAAL, Spark’s MLlib, Berkeley’s CAFFE, UTK’s PLASMA, and MAGMA packages [11][12]. Performance of BLAS and LAPACK eventually decides performance of these packages. Hence, it is important to have a high performance realization of these packages. Efficient realization of BLAS and LAPACK on different contemporary platforms has been ever researched topic [1][2][13]. All these efforts of efficient realizations are through software optimizations and efficient exploitation of memory hierarchy [14][15]. Major reason for centralization of efforts toward software optimizations and efficient exploitation of memory hierarchy is mainly due to several architectural parameters that are not in the control of programmer [16]. For example, the depth of the pipeline (pipeline stages) in the underlying platform [17]. In this paper, we present a theoretical framework that assists in establishing a relation between pipeline depth of different floating point operations with size and type of the workload. Major contributions in this paper are as follows:

- We present a comprehensive theoretical framework that allows us to predict processor performance based on pipeline depths of different floating point operations like multiplier, adder, square root, and divider for BLAS and LAPACK
- Characterization of BLAS and LAPACK is presented where we try to determine several parameters to be fitted in our theoretical framework to arrive at optimum number of pipeline stages for floating point operations

• Farhad Merchant and Anupam Chattopadhyay are with School of Computer Science and Engineering, Nanyang Technological University, Singapore
E-mail: {mamirali,anupam}@ntu.edu.sg

• Soumyendu Raha and S K nandy are with Indian Institute of Science, Bangalore

• Ranjani Narayan is with Morphing Machines Pvt. LTD.

Manuscript received October 20, 2016;

- Extensive simulations are carried out to arrive at an optimum pipeline depth of multiplier, adder, square root, and divider for BLAS and LAPACK in a Processing Element (PE). It is shown that our theoretical curves corroborate to our simulations. Finally with synthesis results it is shown that our PE outperforms recently presented custom linear algebra accelerator

We choose a scalar processor for our initial theoretical framework and then extend framework for superscalar processor. The paper is organized as follows: In section 2, we discuss some of the works in the literature focusing on optimum pipeline depth of the processor. In section 3, we focus on theoretical framework and derive expression for optimum pipeline depth for several operations encountered in BLAS and LAPACK. Characterization of BLAS and LAPACK is presented in section 4. We present a Processing Element (PE) design in section 4 for experimental setup that is to validate our theoretical framework and discuss results in section 5. In section 6, we conclude our work.

2 RELATED WORK

There is a significant theoretical and experimental work done in the recent past that establishes relation between pipeline depth of a microprocessor and cache size [9][10].

In [9], authors have presented interesting work that focuses on improving processor performance by having deeper pipeline considering Intel Pentium 4 as a baseline case. Relation between processor performance, pipeline depth, and cache size is established for several benchmarks. The paper presents simulator based experimental results. It is concluded that with 100% increase in the performance in the Pentium 4 like processors, performance improvement of 35-90% can be attained. A major shortcoming of the work presented in [9] is that the work presents interesting empirical results and does not establish succinct theory for predicting performance by varying pipeline depth and cache size.

In [10], authors have presented an analytical model that derives optimal pipeline depth as a function of power and performance for a superscalar processor. The model is validated using a cycle accurate simulator of a contemporary superscalar processor. Authors in [10] build on the base case presented in [18] where it is shown that for s_i pipeline stages, if t_i is the latch free time to complete the operation in pipe i , then in the scenario where all the pipe stages operate at same frequency, $\frac{t_i}{s_i} = \frac{t_j}{s_j}, \forall i, j$. If c_i is latch overhead in i^{th} pipeline stage than time per stage of pipe i is $T_i = \frac{t_i}{s_i} + c_i, \forall i$. In case of absence of pipeline stalls, throughput of such a machine would be $G = \sum_{i=1}^k (\frac{1}{T_i})$, where k is number of pipe stages in the pipeline. In [10], authors have extended this baseline model to incorporate pipeline stalls. The work presented in [10] becomes one of the starting point for the work presented in this paper.

In [19], authors have analyzed trade-off between greater throughput in deeper pipeline and penalty due to hazards in deeper pipeline. Sensitivity in Cycles-per-Instruction and cycle time are considered as parameters to arrive at optimum pipeline depth. It is shown that the total time can be modeled as a sum busy and non busy time of the pipeline considering pipeline hazards as a parameter. Simulation is performed for 35 different types of workloads and it is clearly shown that the optimum pipeline depth varies between 13 to 35 for these workloads. Such a revelation gives us motivation to work further on a class of workloads for the

workload specific (or domain specific) accelerator. The theoretical framework presented in [19] forms foundation of our theoretical framework and the framework presented in [19] is revisited in the prelude of section 3.

Theoretical framework presented in [20] is continuation of the theoretical framework presented in [19]. In [20], authors have optimized pipeline for power and performance considering 55 workloads. The problem of optimum pipeline depth is well studied by considering parameters like dynamic power increase, clock gating, and leakage power in [20].

In [21], authors have presented several floating point unit architecture extensions to accelerate matrix factorizations. The work presented in [21] is interesting and through several extension to the floating point unit architecture, significant performance improvement over baseline accelerator is achieved. The limitation of the work presented in [21] is lack of theoretical framework that helps to decide the architectural parameters. The work presented in [21] serves as a major benchmark for the work presented in this paper.

In this paper, we have considered several theoretical and experimental framework as a motivation and/or baseline for our theoretical framework. We dwell on the idea of arriving at optimum pipeline depth for the domain customized accelerator. We perform analysis of the workload which is BLAS and LAPACK in this case and based on that we arrive at optimum pipeline depth of multiplier, adder, square root, and divider for the accelerator. Number of independent operations in the Directed Acyclic Graphs (DAGs) of the several routines BLAS and LAPACK are considered as parameters for floating point unit co-design for domain specific accelerator.

3 THEORETICAL FRAMEWORK

In the initial part of this section, we revisit theory presented in [9], [10], and [19]. Latter we extend theory for domain customized architectures by considering workload characterization. The total time T for the pipeline of the processor can be given by

$$T = T_{BZ} + T_{NBZ} \quad (1)$$

where T_{BZ} , and T_{NBZ} represent busy and non-busy time respectively. Typically, T_{BZ} is when pipeline is busy while T_{NBZ} is when pipeline is stalled due one of the hazards. From [19], ratio of total time T to the total number of instructions N_I is given by

$$\frac{T}{N_I} = (t_o + \frac{\gamma N_H t_p}{N_I}) + (\frac{t_p}{p}) + (\frac{\gamma N_H t_o p}{N_I}) \quad (2)$$

In equation 2, t_p is the total logic delay of the processor, p is the number of pipeline stages in the design, t_o is the latch overhead for the technology, N_I is total number of instructions, N_H is total number of pipeline hazards, and $\gamma = \frac{1}{N_H} \sum_{h=1}^{N_H} \beta_h$ where β_h is the fraction of the total pipeline delay encountered by each particular hazard.

In equation 2, the first term is independent of pipeline depth; the second term varies inversely with p ; and the last term varies linearly with p . To obtain minimum at particular value of p , equation 2 can be differentiated and equated to 0. That will give p_{opt}

$$p_{opt}^2 = \frac{N_I t_p}{\gamma N_H t_o} \quad (3)$$

Few observations about optimum pipeline depth can be made from equation 3. As t_o which is latch overhead decreases with lowering node of technology, optimum pipeline depth increases. Lower the hazards in the workload the pipeline depth increases. As γ which is fraction of the pipeline that hazards stall decreases, the optimum pipeline depth increases.

We extend this theory for BLAS and LAPACK through workload characterization where we consider characteristics of the specific workload to arrive at an optimum pipeline depth of different operations in encountered in the workload. To extend theoretical frame work, we consider analytical pipeline model presented in [10] that encompasses several pipes namely fixed point unit pipe, load-store pipe, and branch pipe. We extend the theoretical model presented in [10] and incorporate a floating point pipe as shown in figure 1.

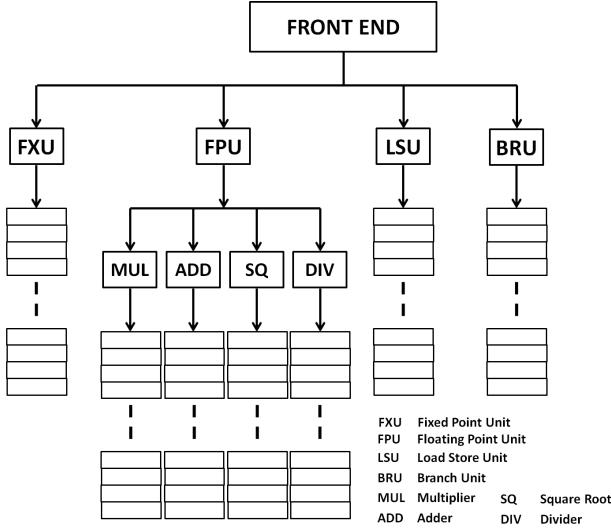


Fig. 1: Pipeline Model

As shown in the figure 1, the model has four pipes: fixed point, floating point, load store, and branch. Since, in BLAS and LAPACK, the operations are floating point in nature and the operations encountered are multiply, addition, division, and square root, we further divide floating point unit pipeline into multiplier pipe, adder pipe, divide pipe, and square root pipelines. Our objective is to arrive at an optimum pipeline depth of these floating point hardware units. The types of arithmetic instructions encountered in BLAS and LAPACK can be given by a set $K = \{M, A, S, D\}$ where M , A , S , and D are for multiplication, adder, square root and divider insturctions respectively. The total number of instructions in a routine of BLAS and/or LAPACK is given by

$$N_I = \sum_i^K N_{iI} \text{ where } i \in K \quad (4)$$

Similarly, total number of hazards are given by

$$N_H = \sum_i^K N_{iH} \text{ where } i \in K \quad (5)$$

To arrive at an optimum pipeline depth of the each individual pipes shown in the figure 1, we can replace N_I and N_H by corresponding pipe parameters. From equation 2, Time per Instruction (TPI) is given by

$$TPI = \sum_i^K \frac{T_i}{N_{iI}} \text{ where } i \in K \quad (6)$$

where $T_i = (t_o + \frac{\gamma N_{iH} t_p}{N_{iI}}) + (\frac{t_p}{p}) + (\frac{\gamma N_{iH} t_o p}{N_{iI}})$, $i \in K$. T_M , T_A , T_D , and T_S are the total execution times for multiplier, adder, divider, and square root pipelines for an instruction stream. Parameter t_o is technology dependent and not dependent on the type of the instruction. Equation 3 can be modified as

$$p_{opt_i}^2 = \frac{N_i t_{pi}}{\gamma_i N_{iH} t_o} \text{ where } i \in K \quad (7)$$

In equation 7, p_M , p_A , p_D , and p_S is the total number of pipeline stages in multiplier, adder, divider, and square root hardware unites respectively. Similarly, γ_M , γ_A , γ_D , and γ_S are the total pipeline delay for each pipeline averaged over total number of hazards for each pipe. From [19], $\gamma = \frac{1}{N_H} \sum H_h$ where β_h is fraction of total pipeline delay encountered by each particular hazard.

In general, in absence of workload characterization, we can vary different parameters like γ , N_I , N_H , and p in equation 2 and comment on effect of different parameters on the time T .

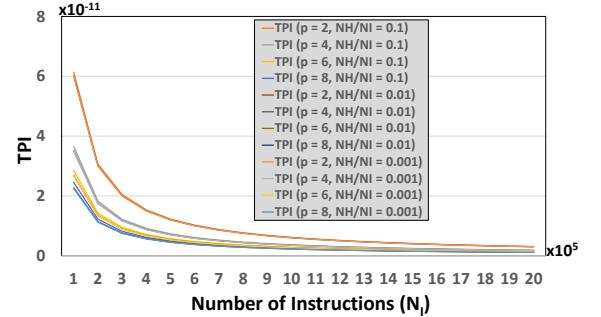


Fig. 2: TPI for Different Sizes of Workload for 2, 4, 6, and 8 (keeping $\frac{N_H}{N_I} = 0.1, 0.01$, and 0.001)

In figure 2, it can be observed that for a fixed number of pipeline stages p , as the problem size increases, the TPI saturates. For example, $p = 2$ and $\frac{N_H}{N_I} = 0.1, 0.01$, and 0.001 then TPI saturates at instruction count of 10×10^5 in the workload. This is mainly because smaller pipelines require large number of instruction to saturate and approach lower bound of TPI. It can also be observed in the figure 2 that for relatively larger pipelines (for $p = 4, 6$, and 8) attained TPI progressively increases. This is mainly because of increased operating frequency of the pipeline stages.

Effect on TPI of varying pipeline depth for a particular workload with varying hazards is shown in figure 3. It can be observed in the figure 3 that as we increase pipeline depth, TPI decreases and optimum is achieved. Beyond optimum, a linear increase in the TPI is observed. It can also be observed that the theoretical curve presented in 3 is fairly flat around optimum leaving considerable scope in choosing best design point for the optimum pipeline depth.

Effect of varying γ and pipeline stages p on TPI is shown in figure 4. It can be observed in the figure 4 that for a smaller values of γ , optimum achieved in the theoretical curve is around 4 and as we increase value of γ , a deeper pipeline becomes optimum

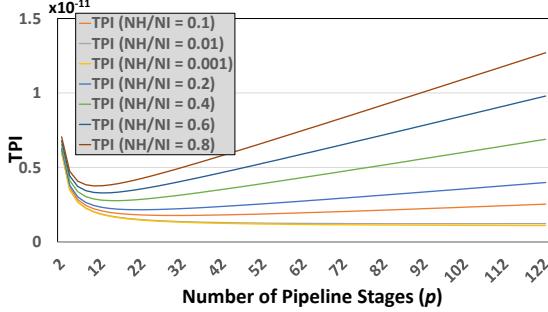


Fig. 3: TPI for Different Pipeline Stages p and Varying Workload (keeping $\frac{N_H}{N_I} = 0.1, 0.01, 0.001, 0.2, 0.4, 0.6$, and 0.8)

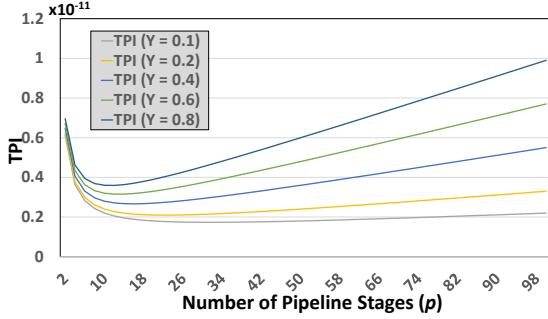


Fig. 4: TPI for Different Pipeline Stages p and $\gamma = 0.1, 0.2, 0.4, 0.6$, and 0.8

pipeline. From the figures 2, 3, and 4, we can make following remarks:

Remark 1: Pipeline will saturate as we increase the size of the workload. Higher the ratio $\frac{N_H}{N_I}$, worse TPI is attained for small size of workloads.

Remark 2: Higher the ratio $\frac{N_H}{N_I}$, shallow the optimum pipeline depth for the workload. It is better to have less number of pipeline stages if workload contains large number of hazards. For large number of hazards, if pipeline stages are higher than the optimum pipeline stages then the TPI attained deteriorates significantly as shown by red line (for $\frac{N_H}{N_I} = 0.8$) in the figure 3

Remark 3: Parameter γ that solely depends on the total number of hazards N_H and β_h which is fraction of the total pipeline delay encountered by each particular hazard plays an important role in determination of optimum pipeline depth. For large value of γ , if the pipeline stages are more than 20 and increased further, TPI deteriorates significantly as shown by blue line in the figure 4. For small value of γ , even if the number of pipeline stages are increased beyond optimum number, the increase in TPI is observed minimal

Based on the observations from the theoretical curves in the figures 2, 3, and 4, we can establish that it is important to characterize workloads of the domain of interest to arrive at an optimum pipeline depth of the different operations encountered in the computations pertaining to the domain.

4 BLAS AND LAPACK CHARACTERIZATION

Based on remarks in section 3 and theory presented in [9], and [10], we present detailed characterization of different routines in BLAS and LAPACK for determining several parameters that help

us arriving at optimum pipeline depths of multiplier adder, square root and divider for these packages.

4.1 Characterization of BLAS

For characterization of BLAS, we consider *inner product* (Level-1 BLAS), *matrix – vector multiplication* (Level-2 BLAS), and *general matrix – matrix multiplication* (Level-3 BLAS) as representative routines. These routines are known as *ddot*, *dgemv*, and *dgemm* respectively where 'd' is for double precision [22].

For vectors $x = [a_1 \ a_2 \ \dots \ a_n]$, and $y = [b_1 \ b_2 \ \dots \ b_n]$, inner product is given by

$$c = x^T y \\ = [a_1 \ a_2 \ a_3 \ a_4] \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \text{ for } n = 4 \quad (8)$$

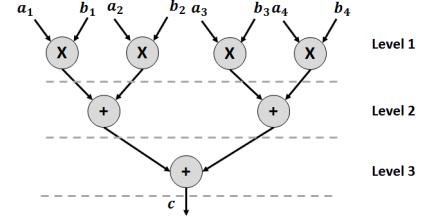


Fig. 5: 4-element Vector Inner Product

Directed Acyclic Graph (DAG) for $n = 4$ is shown in figure 5. It can be observed in the figure 5 that all the multiplications in the *inner product* of 4–element vector can be performed in parallel. In general for n –element vector there are n multiplications and all the multiplications can be executed in parallel. There are $n - 1$ additions in the *inner product*, and there is a dependency from the output of the multiplier for the first level of the addition as shown in the figure 5 and there are dependencies in the addition for each next level from the additions in the previous level. Considering, only dependency hazards, there will be no hazards in the multiplier pipeline. Associated parameters with multiplier, and adder pipelines shown in the figure 11 will be as follows:

$$N_I = N_{IM} + N_{IA} = n + n - 1 = 2n - 1$$

$$N_{HM} = 0 \text{ (considering only dependency hazards)}$$

$$\gamma_M = \infty$$

Determining γ_A is difficult as mentioned in [19]. Hence, we have to determine value of γ_A through a theoretical curve shown in figure 6. It can be observed that for large value of γ_A , a sharp rise in TPI is observed. For small value of γ_A , the curve becomes almost flat. Near optimum value in the curve, it is considerably flat allowing designer multiple choices for the number of pipeline stages. For figure 6, we have considered $\frac{N_H}{N_I} = 0.1$. Decreasing $\frac{N_H}{N_I}$ further gives a flat theoretical curve as observed in the figure 3. For multiplier, theoretical curve for TPI becomes a flat horizontal line as we increase the pipeline depth. This is mainly due to absence of dependency hazards in the multiplication.

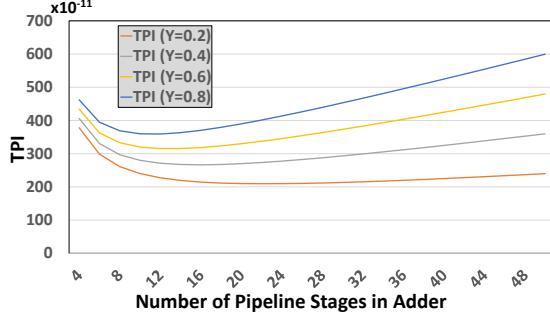


Fig. 6: TPI for Different Pipeline Stages p in Adder and $\gamma = 0.2, 0.4, 0.6$, and 0.8 for 1000– Element Vector Inner Product

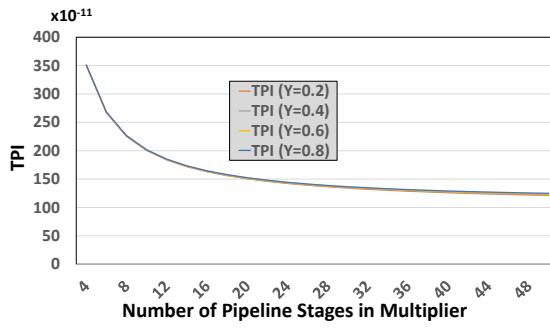


Fig. 7: TPI for Different Pipeline Stages p in Adder and $\gamma = 0.2, 0.4, 0.6$, and 0.8 for 1000– Element Vector Inner Product

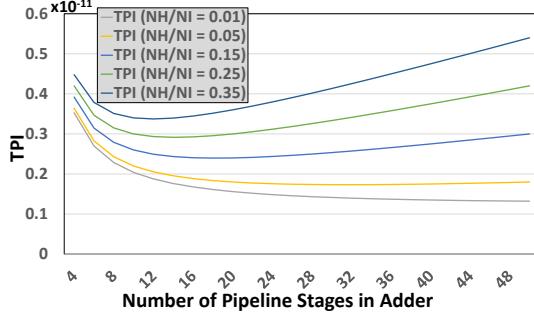


Fig. 8: TPI for Different Pipeline Stages p in Adder and $\frac{N_H}{N_I} = 0.01, 0.05, 0.15, 0.25$, and 0.35

For *matrix – vector*, and *matrix – matrix* multiplication,

$$y = Ax \quad (9)$$

$$C = AB \quad (10)$$

where y and x are vectors, and A , B , and C are matrices. Since, *matrix – vector* multiplication, and *matrix – matrix* multiplication can be viewed as a series of calls of *inner products*, the optimum number of pipeline stages for these routines for adder and multiplier are expected to be the same as what we achieved for *inner product*. It is well established that , in practical implementations of *matrix – vector* multiplication (DGEMV in BLAS) and *matrix – matrix* multiplication (DGEMM in BLAS), due to compiler optimizations the dependency hazards reduce [23]. This reduction in the hazards will lead to increase in the γ_A and decrease in the ratio $\frac{N_H}{N_I}$. In figure 8, TPI for different

ratio of $\frac{N_H}{N_I}$ is shown. It can be observed in the figure 8 that as the ratio $\frac{N_H}{N_I}$ increases, the growth in TPI is sharper.

4.2 Characterization of LAPACK

For LAPACK, we consider two most popular factorization routines namely DGEQRF (QR factorization), and DGETRF (LU factorization with partial pivoting) for characterization.

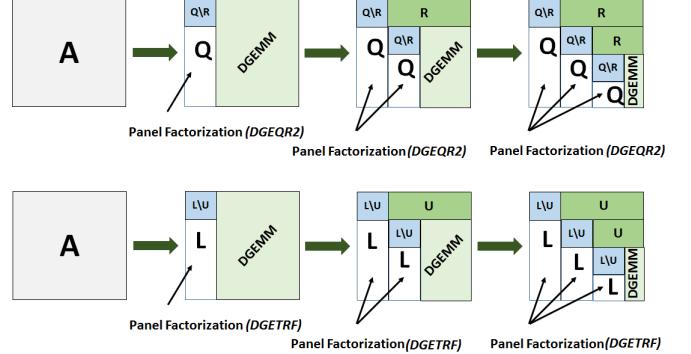


Fig. 9: QR and LU Factorizations

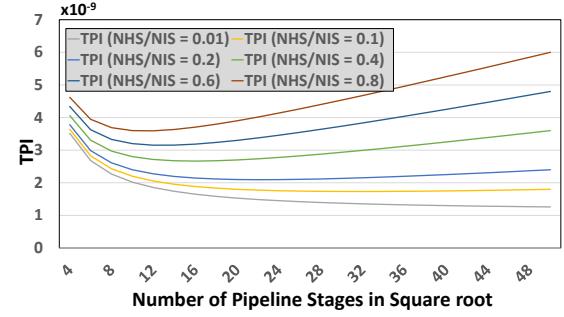


Fig. 10: TPI for Different Pipeline Stages p in Square root and $\frac{N_{HS}}{N_{IS}} = 0.01, 0.1, 0.2, 0.4, 0.6$, and 0.8

For these factorizations, it can be observed in figure 9 that the *matrix – matrix* operations (DGEMM) are dominant, and hence the optimum number of pipeline stages for multiplier and adder would remain same as derived in section 4.1. It is important to arrive at an optimum pipeline depth of divider and square root shown in the figure 11 through characterization.

In QR factorization, division and square root operations are required in panel factorization and the order of division and square root operations is $O(n^2)$ while the total operations in the factorization are $O(n^3)$. There is always dependency in the square root operation that stalls the program execution. The ratios $\frac{N_{HD}}{N_{ID}}$, and $\frac{N_{HS}}{N_{IS}}$ are observed to be high in QR factorization. With varying pipeline and varying number of hazards in the square root pipeline N_{HS} , the theoretical curve is shown in figure 10. For optimum number of stages in the divider, we expect trend that is similar to shown in the figure 10 since the number of dependency hazards in square root and divider are expected to be same in QR factorization.

In LU factorization there are multiplications, additions, and divisions. Since the occurrence of division instruction in the program is similar to the square root/divider in the QR factorization, we expect similar trend for optimum pipeline stages for divider as shown in the figure 10.

5 EXPERIMENTAL SETUP AND RESULTS

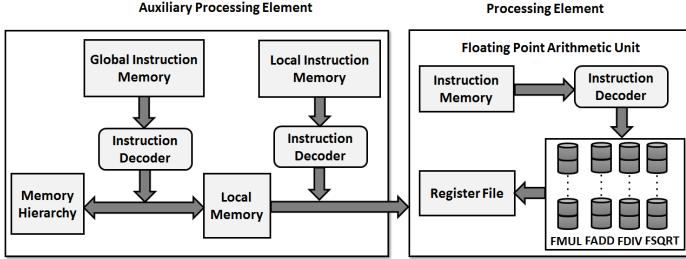


Fig. 11: Experimental Setup

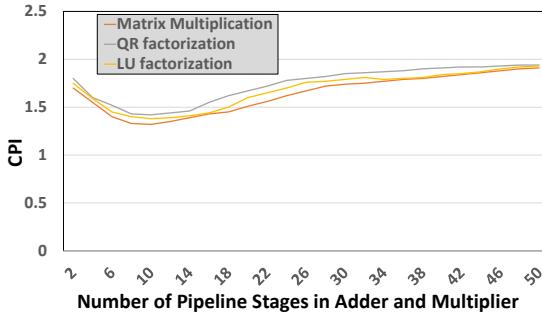


Fig. 12: CPI in Matrix Multiplication, QR factorization, and LU factorization with Varying Number of Pipeline Stages in Adder and Multiplier for a Matrix of Size 100×100

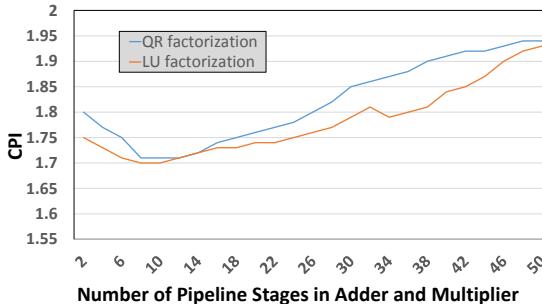


Fig. 13: CPI in QR factorization, and LU factorization with Varying Number of Pipeline Stages in Square root and Divider for a Matrix of Size 100×100

Experimental setup is shown in figure 11. As shown in the figure 11, we have a Processing Element (PE) and an Auxiliary Processing Element (APE) where PE has compute resources and load/store operations are handled by APE. PE consists of different floating point operations like multiplier, adder, square root, and divider, a small register file, an instruction memory, and an instruction decoder, and APE consists of two small instruction memories, corresponding instruction decoders, and a Local Memory (LM). The operation of PE and APE can be described as follows:

- Step 1:** Load data from upper level of memory to LM in APE
- Step 2:** Load data from LM to Register File
- Step 3:** Perform computations in the PE and store results back in the Register File

- Step 4:** Store results back from Register File to LM
- Step 5:** Store final result to the upper level of memory

TABLE 1: Comparison between LAP-PE and PE at Different Frequencies with 16KBytes of dual-ported SRAM with double precision floating point arithmetic

Architecture	Speed (GHz)	Area (mm^2)	Memory (mW)	FMAC (mW)	PE (mW)
LAP-PE	1.81	0.181	13.25	105.5	118.7
LAP-PE	0.95	0.174	6.95	31.0	38.0
LAP-PE	0.33	0.167	2.41	6.0	8.4
LAP-PE	0.20	0.169	1.46	3.4	4.8
PE	1.81	0.301	26.50	422	448.5
PE	0.95	0.28	13.90	124	137.9
PE	0.33	0.273	4.82	24	28.82
PE	0.20	0.275	2.92	13.6	16.5

As shown in the figure 11, the pipeline depths of the floating point arithmetic units are kept variable. Usually, it is not possible vary pipeline stages of the floating point unit in RTL. For simulation purpose, we use Bluespec System Verilog (BSV) that lets us incorporation of C program along with RTL registers to mimic the different number of pipeline stages for different floating point operations. The simulation environment also becomes non-synthesizable due to presence of floating point operation written in C. For simulation results, we report Cycles-per-Instructions (CPI) for varying number of pipeline stages for adder and multiplier for $matrix - matrix$ multiplication, QR factorization, and LU factorization as shown in figure 12. It can be observed in the figure 12 that our simulation results corroborate to our theoretical curve observed in section 3.

For synthesis, we use enhanced version of PE where we attach 4 multipliers and 3 adders in a reconfigurable way to enhance the performance of the PE. Table 1 presents comparison between LAP-PE and PE. It can be observed from the table 1 that PE has more area and consumes more power. This is mainly because of SRAM and DOT4 instruction. If we take GFlops/ mm^2 and GFlops/W as a performance measure as shown in table 2 then at 1.81 GHz, LAP-PE attains 19.92 GFlops/ mm^2 while PE attains 42.09 GFlops/ mm^2 . Similarly, at 0.20 GHz, LAP-PE attains 2.37 GFlops/ mm^2 while PE attains 5.09 GFlops/ mm^2 .

Similarly, at 1.81 GHz, LAP-PE attains 29.7 GFlops/W while PE attains 28.281 GFlops/W. At 0.95 GHz, LAP-PE attains 46.4 GFlops/W while in PE it is 48.54 GFlops/W. At 0.2 GHz, LAP-PE achieves 51.1 GFlops/W while PE achieves 84.84 GFlops/W.

It can be concluded from above observations that PE performs better than LAP-PE at lower frequencies. This is mainly because lower power consumed by double precision floating point operations at low frequencies.

6 CONCLUSION

We presented theoretical framework to arrive at an optimum number of pipeline stages for adder, multiplier, square root, and divider for BLAS and LAPACK. We presented characterization of BLAS and LAPACK to estimate parameters. The estimated parameters were used to arrive at theoretical curves. We also presented a PE that has extensible pipelines in the simulation environment. Through simulations, we show that our theoretical results corroborates to our simulation results. We synthesize PE with RTL of floating point unit and show better performance

TABLE 2: Comparison of LAP-PE and PE

Speed	LAP-PE (GFlops/mm ²)	LAP-PE (GFlops/W)	PE (GFlops/mm ²)	PE (GFlops/W)
1.81	19.92	29.7	42.09	28.24
0.95	10.92	46.4	23.75	48.54
0.33	3.95	57.8	8.46	82.5
0.2	2.37	51.1	5.09	84.84

than the most recent custom realization of BLAS and LAPACK. Through our theoretical framework and experimental studies, it was shown that for domain specific platforms, it is possible and advisable to first derive an optimum pipeline depth theoretically for better performance of the platform. The theoretical framework presented can be extended with more precise determination of parameters like γ and N_H . Near accurate determination of these parameters would result in better estimation of the optimum number of pipeline stages in domain specific platforms.

REFERENCES

- [1] T. N. Mudge, “The specialization trend in computer hardware: technical perspective,” *Commun. ACM*, vol. 58, no. 4, p. 84, 2015. [Online]. Available: <http://doi.acm.org/10.1145/2735839>
- [2] A. Pedram, S. Z. Gilani, N. S. Kim, R. A. van de Geijn, M. J. Schulte, and A. Gerstlauer, “A linear algebra core design for efficient level-3 blas,” in *ASAP*, 2012, pp. 149–152.
- [3] 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*, Dec 2011, pp. 45–50.
- [4] S. Das, K. T. Madhu, M. Krishna, N. Sivanandan, F. Merchant, S. Natarajan, I. Biswas, A. Pulli, S. K. Nandy, and R. Narayan, “A framework for post-silicon realization of arbitrary instruction extensions on reconfigurable data-paths,” *Journal of Systems Architecture - Embedded Systems Design*, vol. 60, no. 7, pp. 592–614, 2014. [Online]. Available: <http://dx.doi.org/10.1016/j.sysarc.2014.06.002>
- [5] A. Pedram, A. Gerstlauer, and R. A. van de Geijn, “Algorithm, architecture, and floating-point unit codesign of a matrix factorization accelerator,” *IEEE Trans. Computers*, vol. 63, no. 8, pp. 1854–1867, 2014. [Online]. Available: <http://dx.doi.org/10.1109/TC.2014.2315627>
- [6] M. H. Ionica and D. Gregg, “The movidius myriad architecture’s potential for scientific computing,” *IEEE Micro*, vol. 35, no. 1, pp. 6–14, 2015. [Online]. Available: <http://dx.doi.org/10.1109/MM.2015.4>
- [7] Z. E. Rákossy, F. Merchant, A. A. Aponte, S. K. Nandy, and A. Chattopadhyay, “Efficient and scalable cgra-based implementation of column-wise givens rotation,” in *ASAP*, 2014, pp. 188–189.
- [8] Z. E. Rákossy, F. Merchant, A. A. Aponte, S. K. Nandy, and A. 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*, 2014, pp. 1–6. [Online]. Available: <http://dx.doi.org/10.1109/VLSI-SoC.2014.7004166>
- [9] E. Sprangle and D. Carmean, “Increasing processor performance by implementing deeper pipelines,” in *Proceedings of the 29th Annual International Symposium on Computer Architecture*, ser. ISCA ’02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 25–34. [Online]. Available: <http://dl.acm.org/citation.cfm?id=545215.545219>
- [10] V. Srinivasan, D. Brooks, M. Gschwind, P. Bose, V. Zyuban, P. N. Strenski, and P. G. Emma, “Optimizing pipelines for power and performance,” in *Proceedings of the 35th Annual ACM/IEEE International Symposium on Microarchitecture*, ser. MICRO 35. Los Alamitos, CA, USA: IEEE Computer Society Press, 2002, pp. 333–344. [Online]. Available: <http://dl.acm.org/citation.cfm?id=774861.774897>
- [11] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional architecture for fast feature embedding,” *arXiv preprint arXiv:1408.5093*, 2014.
- [12] B. 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>.
- [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 cgbras,” in *XIVth International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS 2014, Agios Konstantinos, Samos, Greece, July 14-17, 2014*, 2014, pp. 225–232. [Online]. Available: <http://dx.doi.org/10.1109/SAMOS.2014.6893215>
- [14] 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, Mumbai, India, January 5-9, 2014*, 2014, pp. 258–263. [Online]. Available: <http://dx.doi.org/10.1109/VLSID.2014.51>
- [15] F. Merchant, A. Maity, M. Mahadurkar, K. Vatwani, I. Munje, M. Krishna, S. Nalesh, N. Gopalan, S. Raha, S. Nandy, and R. Narayan, “Micro-architectural enhancements in distributed memory cgbras for lu and qr factorizations,” in *VLSI Design (VLSID), 2015 28th International Conference on*, Jan 2015, pp. 153–158.
- [16] 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 *29th International Conference on VLSI Design, VLSID 2016, Kolkata, India, January 4-8, 2016 (in press)*.
- [17] F. Merchant, N. Choudhary, S. K. Nandy, and R. Narayan, “Efficient realization of table look-up based double precision floating point arithmetic,” in *29th International Conference on VLSI Design, VLSID 2016, Kolkata, India, January 4-8, 2016*.
- [18] M. J. Flynn, P. Hung, and K. W. Rudd, “Deep-submicron microprocessor design issues,” *IEEE Micro*, vol. 19, no. 4, pp. 11–22, Jul. 1999. [Online]. Available: <http://dx.doi.org/10.1109/40.782563>
- [19] A. Hartstein and T. R. Puzak, “The optimum pipeline depth for a microprocessor,” in *29th International Symposium on Computer Architecture (ISCA 2002), 25-29 May 2002, Anchorage, AK, USA*, 2002, pp. 7–13. [Online]. Available: <http://dx.doi.org/10.1109/ISCA.2002.1003557>
- [20] ———, “Optimum power/performance pipeline depth,” in *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 36. Washington, DC, USA: IEEE Computer Society, 2003, pp. 117–. [Online]. Available: <http://dl.acm.org/citation.cfm?id=956417.956566>
- [21] A. Pedram, A. Gerstlauer, and R. A. van de Geijn, “Floating point architecture extensions for optimized matrix factorization,” in *21st IEEE Symposium on Computer Arithmetic, ARITH 2013, Austin, TX, USA, April 7-10, 2013*, 2013, pp. 49–58. [Online]. Available: <http://dx.doi.org/10.1109/ARITH.2013.21>
- [22] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users’ Guide*, 3rd ed. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999.
- [23] N. J. Higham, “Exploiting fast matrix multiplication within the level 3 BLAS,” *ACM Trans. Math. Softw.*, vol. 16, no. 4, pp. 352–368, 1990. [Online]. Available: <http://doi.acm.org/10.1145/98267.98290>