A Linear Algebra Core Design For Efficient Level-3 BLAS

Ardavan Pedram[1], Syed Zohaib Gilani[2], Nam Sung Kim[2],
Robert van de Geijn[3], Michael Schulte[4], and Andreas Gerstlauer[1]

[1]*Department of Electrical and Computer Engineering, The University of Texas at Austin*
[2]*Department of Electrical and Computer Engineering, University of Wisconsin-Madison*
[3]*Department of Computer Science, The University of Texas at Austin*
[4]*AMD Research*
[1,3]*{ardavan,gerstl}@utexas.edu, rvdg@cs.utexas.edu*
[2,4] *{gilani,nskim3l}@wisc.edu, michael.schulte@amd.com*

***Abstract*—Reducing power consumption and increasing efficiency is a key concern for many applications. It is well-accepted that specialization and heterogeneity are crucial strategies to improve both power and performance. Yet, how to design highly efficient processing elements while maintaining enough flexibility within a domain of applications is a fundamental question. In this paper, we present the design of a specialized Linear Algebra Core (LAC) for an important class of computational kernels, the level-3 Basic Linear Algebra Subprograms (BLAS). We demonstrate a detailed algorithm/architecture co-design for mapping a number of level-3 BLAS operations onto the LAC.**

Results show that our prototype LAC achieves a performance of around 64 GFLOPS (double precision) for these operations, while consuming less than 1.3 Watts in standard 45nm CMOS technology. This is on par with a full-custom design and up to $50\times$ and $10\times$ better in terms of power efficiency than CPUs and GPUs.

I. Introduction

Reducing power consumption is an increasingly important issue both in the embedded domain, where systems have to operate in highly restricted environments, and in general-purpose computing, where physical limits of technology scaling have made power walls the main roadblock to sustained performance. Furthermore, many emerging systems, increasingly demand both very high performance and power efficiency on nontraditional platforms.

Application-specific design of hardware accelerators can provide orders of magnitude improvements in power and area efficiency [1]. However, full-custom design is expensive in many aspects. The question is whether these concepts can be applied to a broader class of more general applications. If neither fine-grain programmable computing nor full-custom design are feasible, can we design specialized on-chip cores that maintain the efficiency of full custom hardware while providing enough flexibility to execute whole classes of coarse-grain operations?

The goal of our project is to design high-performance, low-power Linear Algebra Cores (LACs) that realize the level-3 BLAS kernels directly in specialized hardware. In previous work [2], we examined how this can be achieved for GEneral Matrix-Matrix multiplication (GEMM). In this paper, we generalize our design to other level-3 Basic Linear Algebra Subprograms (BLAS), demonstrating that with small micro-architectural modifications, the LAC can be extended to support the full set of BLAS operations without loss in efficiency. Furthermore, we custom design and integrate a specialized Multiply-ACcumulate (MAC) unit. We synthesize it and other key components of the LAC micro-architecture, and the resulting analysis suggests that it should be possible to achieve a performance of 50 double- and 120 single-precision GFLOPS/W in 30 GFLOPS/mm^2 in current 45nm technology. This represents two orders of magnitude improvement over current CPU architectures and an order of magnitude improvement over current GPUs.

II. Related Work

Matrix computations on general-purpose machines have been studied extensively [3]. Furthermore, in recent years, GPUs have become a popular target for acceleration. Modern general-purpose GPUs (GP-GPUs) can be effectively used for matrix computations [4]. With throughputs of more than 360 double-precision GFLOPS when running many level-3 BLAS for large matrices, GPUs utilize around 30-70% of their theoretical peak performance. However, in all cases, inherent general-purpose instruction handling overheads limit efficiencies, as indicated by low utilizations.

Over the years, many more specialized, parallel architectures for high-performance computing have been proposed and in most cases benchmarked using GEMM and other level-3 BLAS as a prototypical applications. Systolic arrays were popular in the 80s [5]. With increasing memory walls, recent approaches have brought the computation units closer to memory, including hierarchical clustering of shared memory tiles [6] or network-on-chip architectures [7]. Despite such optimizations, utilizations still range from 60% down to less than 40% with increasing numbers of tiles.

Specialized realizations of BLAS routines on FPGAs have been explored [8]. Such approaches show promising results [9], [10], but are limited by inefficiencies in area, performance, and power due to programmable routing and logic overheads. In contrast, our design is optimized to support all level-3 BLAS kernels at high performance and high efficiency using dedicated, fixed cores.

III. Design of a Linear Algebra Core

We start by briefly reviewing the LAC proposed in [2] and illustrated in Figure 1. It consists of a 2D array of $n_r \times n_r$ Processing Elements (PEs), each of which has a MAC unit with a local accumulator, local storage, simple distributed control, and bus interfaces to communicate data

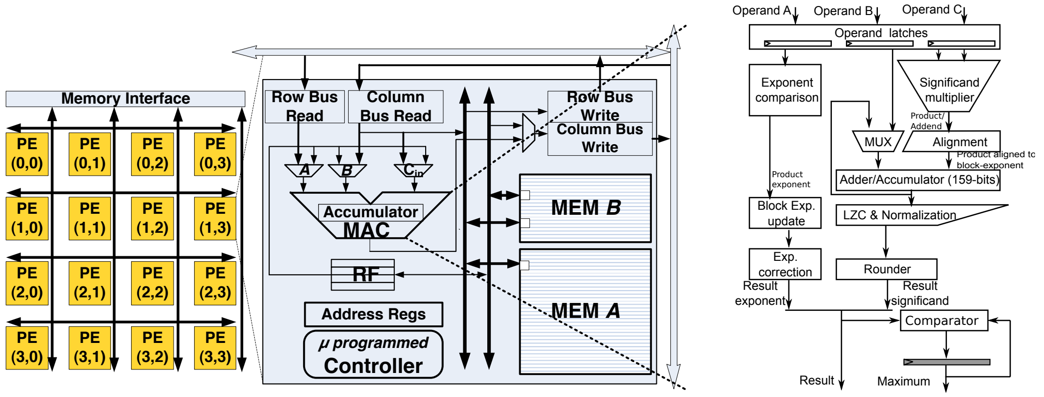

Fig. 1. Core architecture. Each PE contains control logic, SRAM, and MAC unit and is connected to row and column broadcast buses.

within rows and columns. In the following, we describe the core micro-architecture and the mapping of representative BLAS operations onto it. For illustrative purposes, we focus on the case of a mesh with $n_r \times n_r = 4 \times 4$ PEs.

Details of the PE-internal architecture are shown in Figure 1 (right). Each PE has a MAC unit to perform the inner dot-product computations that are key to almost all BLAS operations. Apart from preloading accumulators with initial values, all accesses to elements of a $n_r \times n_r$ matrix being updated are performed directly inside the MAC units, avoiding the need for any register file or memory accesses.

We utilize pipelined MAC units that can typically achieve a throughput of one MAC operation per cycle by utilizing a fast feedback path to postpone normalization of results until after the last accumulation. Leveraging a Fused MAC (FMA) unit with delayed normalization also significantly decreases power consumption while increasing accuracy. The FMA unit incorporated within each PE is designed using the technique presented in [11]. We extend that design to support double-precision floating-point operands and a higher precision for the intermediate result. Figure 1 illustrates the double-precision FMA unit employed within each PE. Key aspects of the design include the fast feedback path between the Adder/Accumulator output and its input, along with the use of a block exponent to help avoid rounding and normalization in intermediate results.

In our design, we connect PEs by horizontal and vertical broadcast busses. Interconnect is realized as simple, data-only busses that do not require overhead for address decoding or complex control. Power and delay analysis shows that we can have one cycle bus latency for LAC sizes up to 16×16 PEs.

Local storage in each PE consists of a larger single-ported memory, a smaller dual-ported memory, and a small register file with one write and two read ports. Typically in dense linear algebra problems, access patterns are predictable and in most cases sequential. Hence, only simple, auto-incrementing address generators are required.

LAC control is distributed and each PE has a state machine that drives a predetermined, hardcoded sequence of communication, storage and computation steps for each supported BLAS operation. The basic state machine in each PE requires two address registers, one loop counter and less than 10 states per BLAS operation.

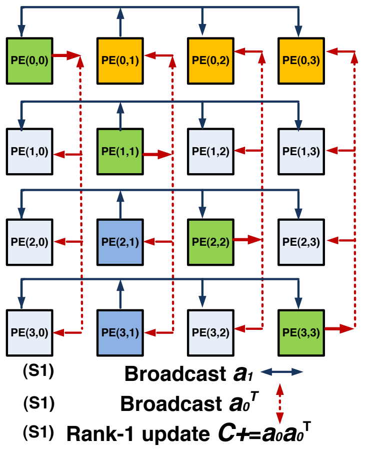

Fig. 2. Second iteration of a 4×4 SYRK on LAC.

IV. Algorithm Mapping

In previous work, we showed how GEMM is mapped on to the LAC [2]. In the following, we will describe the mapping of TRiangular Solve with Multiple Right-hand sides (TRSM), SYmmetric Rank-K update (SYRK), and SYmmetric Rank-2K update (SYR2K) onto our LAC architecture. These are representative of the full class of level-3 BLAS operations. It is commonly accepted that if these operations perform well, all level-3 BLAS operations can be mapped efficiently [3]. The important observation is that all operations require similar computations and data movements.

A. SYRK and SYR2K

The SYRK operation computes $C := C + AA^T$ with a rectangular matrix $A \in \mathbb{R}^{n\times m}$, updating only the lower triangular part of the symmetric matrix $C \in \mathbb{R}^{n\times n}$.

At the lowest level, an unblocked algorithm is utilized. To compute the SYRK of a $n_r \times n_r$ sub-matrix of C stored in the accumulators of the LAC from a $n_r \times k_c$ sub-matrix of A, three different operations take place in the same cycle in each iteration. Figure 2 illustrates the second ($i = 1$) iteration of a SYRK operation. The ith column of PEs broadcasts the values of the ith column of A, a_i, across the row busses, where the PEs in each row keep a copy of these values in their register file for use in the next iteration. At the same time, the values a_{i-1} from the previous iteration are transposed along the diagonal PEs by broadcasting them over the column

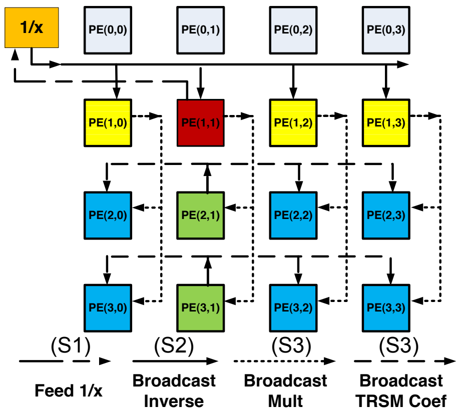

Fig. 3. Second iterations of a 4×4 TRSM on LAC.

busses. Hence, all PEs now have copies of elements of a_{i-1} and a_{i-1}^T, and a rank-1 update is performed to compute $C := C + a_{i-1} \times a_{i-1}^T$. The a_{i-1}^T is also kept in $(i-1)$th row of PEs to store A^T. This is repeated for $i = 0, \ldots, k_c$ cycles.

A bigger SYRK for C of size $m_c \times m_c$ and A of size $m_c \times k_c$ can be blocked into smaller subproblems using a lower order SYRK (mentioned above) to update the diagonal $n_r \times n_r$ lower triangular blocks of C and produce the transpose of the corresponding $n_r \times k_c$ panels of A in a single iteration. Most of the computations are thereby cast into typical GEMM operations using the produced panel of A^T and the remaining panels of A.

Finally, the LAC uses very similar principles as for SYRK to perform the SYR2K operation and its blocked versions. The SYR2K produces $C := C + AB^T + BA^T$ by cross-multiplying rectangular matrices $A, B \in \mathbb{R}^{n \times m}$ by their transpose to update the lower triangular part of the symmetric matrix $C \in \mathbb{R}^{n \times n}$. The amount of both communication and computation is doubled in this case.

B. TRSM

The TRSM operation solves a system of equations $LX = B$, with lower triangular matrix $L \in \mathbb{R}^{n \times n}$ and rectangular matrix $B \in \mathbb{R}^{n \times m}$ for $X \in \mathbb{R}^{n \times m}$, such that upon completion $X = L^{-1}B$.

Figure 3 shows the mapping of an unblocked down-looking TRSM algorithm for a $n_r \times n_r$ sub-matrix of B and lower triangular $n_r \times n_r$ diagonal sub-matrix of L, both stored in the registers of the LAC (with $n_r \times n_r$ PEs). The LAC is augmented with an inverse unit that computes $f(x) = 1/x$. In each iteration $i = 0 \ldots n_r - 1$, the algorithm performs three steps $S1$ through $S3$, where the figure shows the second such iteration ($i = 1$). In $S1$ and $S2$, the element $\lambda_{i,i}$ of L in PE(i,i) is updated with its inverse. The result is broadcast within the ith PE row and used to multiply into the elements of the corresponding row of matrix B (effectively dividing row elements of B by $\lambda_{i,i}$). In S3, the results of those computations are broadcast within their respective columns to be multiplied by the corresponding column of L (which is broadcast within the respective rows) in order to perform a rank-1 update that subtracts the result of this multiplication from the remaining lower part of matrix B. This completes the current iteration, which is repeated for $i = 0, \ldots, n_r - 1$. Given a MAC unit with p pipeline stages, this $n_r \times n_r$ TRSM takes $2pn_r$ cycles. Due to the data dependencies between different PEs within and between iterations, each element has to go through p stages of MAC units while other stages are idle.

A bigger TRSM problem for L of size $m_c \times m_c$ and B of size $m_c \times k_c$ can be blocked into smaller subproblems using a standard blocked up-looking algorithm. In each iteration, $n_r \times k_c$ panels of B are replaced with values of X computed from the corresponding diagonal $n_r \times n_r$ blocks of L using the basic TRSM algorithm on the LAC as described above. Most of the computations are thereby cast into typical GEMM operations as was described previously.

V. Implementation Results and Estimations

We have developed both simulation and analytical performance models of the LAC. In addition, we synthesized key components to estimate overall area and power consumption. We validated the performance model and LAC operation in general by developing a cycle-accurate simulator. The simulator is configurable in terms of PE pipeline stages, bus latencies, and memory and storage sizes. Furthermore, using power consumption numbers for the components, our simulator is able to produce an accurate power profile of the overall execution. We accurately modeled the cycle-by-cycle control and data movement for GEMM and TRSM, and we verified functional correctness of the produced results.

A. Performance Results

Details of analytical performance models and LAC operation for GEMM can be found in [2]. GEMM operation typically achieves the best utilization and hence performance among all other level-3 BLAS. Figure 4 shows a comparison of selected level-3 BLAS for $n_r \in \{4, 8\}$, $m_c = k_c$ and $n = 512$. We can observe that for a PE memory size of 20KBytes and off-core memory bandwidth of 4 B/cycles, GEMM, TRSM, SYRK, and SYR2K achieve 100%, 95%, 90%, and 85% utilization, respectively.

The LAC shows utilizations for TRSM, SYRK and SYR2K that are close to what GEMM can achieve. The reason why none of the other operations reach 100% utilization is that their basic operations do not fully utilize all the PEs. This is due to the triangular shape of the diagonal blocks in each of these cases. However, since lower-order terms only form a fraction of all computations, the overall performance ends up close to peak as the size of problem grows.

TRSM achieves better performance for smaller problem sizes, even though the computation of the triangular part of the lower order term of TRSM is less efficient than SYRK. The difference between SYRK and TRSM is in the bandwidth demand. SYRK needs more bandwidth than TRSM for the same problem size. In small problems, the amount of bandwidth directly affects the performance and results in a higher utilization for TRSM. By contrast, SYRK has higher utilization of the lower order term and better performance in bigger problem sizes. For example, with 25 Kbytes of local

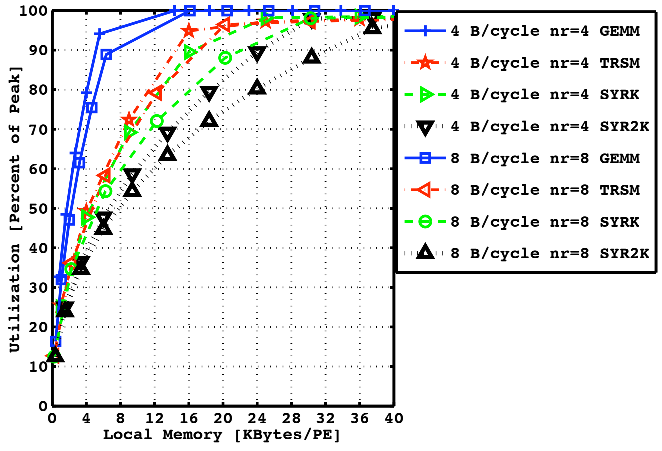

Fig. 4. Utilizations for representative level-3 BLAS operations.

TABLE I
45NM PERFORMANCE, POWER AND AREA FOR A LAP PE WITH 16+4=20 KB SRAM.

	Speed [GHz]	FMAC Depth	Area PE [mm^2]	Memory [mW]	FMAC [mW]	PE [mW]
SP	1.67	5	0.083	20.28	13.46	34.74
	1.43	4	0.092	10.27	14.05	25.32
	1.25	4	0.092	8.97	11.80	21.77
	1.1	4	0.082	7.84	9.52	18.36
DP	2	6	0.110	23.02	56.34	80.36
	1.43	5	0.101	17.71	38.73	57.44
	1.25	4	0.101	15.84	35.96	52.79
	1.11	4	0.103	8.58	31.24	40.81

memory per PE, SYRK with 98% utilization overtakes TRSM with 96% utilization.

SYR2K performs worse than SYRK as is expected for this operation. For the same PE memory size, only a smaller SYR2K operation can be mapped on the LAC. A typical level-3 BLAS has $O(n^2)$ communication and $O(n^3)$ computation complexity. The SYR2K operation doubles the amount of communication and computation, which is not bandwidth efficient compared to solving a bigger SYRK problem.

B. Area and Power Efficiency

We studied the feasibility of a LAP implementation in 45nm bulk CMOS technology. MAC and inverse units were synthesized using Synopsys Design Compiler and a 45nm TSMC standard cell library. The pipeline depths is kept constant until there is a timing failure, at which point the number of pipeline stages are increased. An increase in pipeline depth can also mean a reduction in power, since more timing slack is available for each stage resulting in less power hungry logic. We measured power consumption using PowerCompiler assuming random inputs. Area and power numbers for memories and buses were obtained from CACTI-6.0 [12]. Since GEMM results in the highest utilization and load, we used access patterns of the GEMM algorithm obtained from our simulator to estimate SRAM power consumption.

Table I shows the estimated area and power efficiency of a PE at different design points at peak performance for GEMM. Running at a clock frequency of 2 GHz, a 4×4 LAC is estimated to achieve 50 double-precision (DP) GFLOPS/W. The corresponding area efficiency and energy-delay values are 36 mm^2/GFLOPS and 5 mW/$GFLOPS^2$.

TABLE II
LAC EFFICIENCY FOR LEVEL-3 BLAS ALGORITHMS AT 1.1 GHZ.

Algorithm	$\frac{W}{mm^2}$	$\frac{GFLOPS}{mm^2}$	$\frac{GFLOPS}{W}$	Utilization
GEMM $n_r = 4$	0.397	21.61	54.4	100%
TRSM $n_r = 4$	0.377	20.53	51.7	95%
SYRK $n_r = 4$	0.357	19.45	49.0	90%
SYR2K $n_r = 4$	0.314	17.07	43.0	79%
GEMM $n_r = 8$	0.397	21.61	54.4	100%
TRSM $n_r = 8$	0.377	20.53	51.7	95%
SYRK $n_r = 8$	0.346	18.80	47.3	87%
SYR2K $n_r = 8$	0.290	15.77	39.7	73%

With GEMM being an operation that exhibits amble parallelism and locality and that has been studied extensively through careful and tedious hand-tuning on conventional architectures, many systems, including our LAC, are able to achieve close to peak performance. However, in contrast to other architectures, we show that we are able to sustain such utilization rates, performance and efficiencies for almost all other, more complex linear algebra operations. Table II summarizes detailed performance and area efficiencies of the LAC for all presented level-3 BLAS operations at 1.1 GHz.

VI. CONCLUSION

This paper presents the algorithm/architecture co-design of a linear algebra core for level-3 BLAS. Our core is general and flexible in supporting all representative level-3 BLAS operations while synthesis and power estimation results show that it can provide order of magnitude improved efficiencies compared to other architectures. Our analysis clearly shows the fundamental architectural tradeoffs for efficient execution of linear algebra computations.

We have begun to generalize our design towards other, more complicated linear algebra operations like LU and Cholesky factorization. Our simulator is already able to run the latter. The conclusion is that with minor extensions to PEs, such as inverse square root units, the LAC can be generalized to accommodate these operations.

REFERENCES

[1] R. Hameed et al., "Understanding sources of inefficiency in general-purpose chips," *ISCA '10*, 2010.
[2] A. Pedram *et al.*, "A high-performance, low-power linear algebra core," *ASAP '11*, pp. 35–41, 2011.
[3] K. Goto et al, "High-performance implementation of the level-3 BLAS," *ACM Trans. Math. Softw.*, vol. 35, no. 1, pp. 1–14, 2008.
[4] V. Volkov and J. Demmel, "Benchmarking GPUs to tune dense linear algebra," *SC 2008*, 2008.
[5] K. Johnson *et al.*, "General-purpose systolic arrays," *Computer*, vol. 26, no. 11, pp. 20 – 31, 1993.
[6] J. Kelm et al., "Rigel: an architecture and scalable programming interface for a 1000-core accelerator," *ISCA '09*, 2009.
[7] S. Vangal et al., "An 80-tile sub-100-w teraflops processor in 65-nm cmos," *IEEE J. of Solid-State Circuits*, vol. 43, no. 1, 2008.
[8] L. Zhuo and V. Prasanna, "High-performance designs for linear algebra operations on reconfigurable hardware," *IEEE Trans. on Computers*, vol. 57, no. 8, 2008.
[9] M. Parker, "High-performance floating-point implementation using FPGAs," in *MILCOM*, 2009.
[10] J.-W. Jang *et al.*, "Energy- and time-efficient matrix multiplication on FPGAs," *IEEE Trans on VLSI Systems,*, vol. 13, no. 11, 2005.
[11] S. Gilani *et al.*, "Energy-efficient floating-point arithmetic for software-defined radio architectures," in *ASAP2011*.
[12] N. Muralimanohar *et al.*, "Architecting efficient interconnects for large caches with cacti 6.0," *IEEE Micro*, vol. 28, pp. 69–79, January 2008.