Time-Power-Energy Balance of BLAS kernels in modern FPGAs

Federico Favaro[1], Ernesto Dufrechou[2], Juan P. Oliver[1], and Pablo Ezzatti[2]

[1] Instituto de Ingeniería Eléctrica, Facultad de Ingeniería
Universidad de la República, Montevideo, Uruguay
{ffavaro,jpo}@fing.edu.uy
[2] Instituto de Computación, Facultad de Ingeniería
Universidad de la República, Montevideo, Uruguay
{edufrechou,pezzatti}@fing.edu.uy

Abstract. Numerical Linear Algebra (NLA) is a research field that in the last decades has been characterized by the use of kernel libraries that are de facto standards. One of the most remarkable examples, in particular in the HPC field, is the Basic Linear Algebra Subroutines (BLAS). Most BLAS operations are fundamental in multiple scientific algorithms because they generally constitute the most computationally expensive stage. For this reason, numerous efforts have been made to optimize such operations on various hardware platforms. There is a growing concern in the high-performance computing world about power consumption, making energy efficiency an extremely important quality when evaluating hardware platforms. Due to their greater energy efficiency, Field-Programmable Gate Arrays (FPGAs) are available today as an interesting alternative to other hardware platforms for the acceleration of this type of operation. Our study focuses on the evaluation of FPGAs to address dense NLA operations. Specifically, in this work we explore and evaluate the available options for two of the most representative kernels of BLAS, i.e. GEMV and GEMM. The experimental evaluation is carried out in an Alveo U50 accelerator card from Xilinx and an Intel Xeon Silver multicore CPU. Our findings show that even in kernels where the CPU reaches better runtimes, the FPGA counterpart is more energy efficient.

Keywords: dense numerical linear algebra · energy-efficiency · HPC · matrix-matrix multiplication.

1 Introduction

The Numerical Linear Algebra (NLA) is one of the most important fields in scientific computing. A support for this claim is the existence of several computational kernels involved in the most widespread benchmarks. One of the most notorious is the Linpack benchmark [13] that is employed to define the Top500 list [2]. This benchmark is based on the LU-factorization operation to compute

the peak performance reached by a specific combination of a hardware platform and software implementations.

The LU-factorization is part of the LAPACK specification [3] and typically these kinds of methods are built over BLAS kernels. This philosophy of developing several layers of kernels specifications, has guided the dense NLA landscape since the 70s. Firstly, with the BLAS-1 specification [18], later with BLAS-2 [12] and BLAS-3 [11], and subsequently LAPACK and ScaLAPACK [7], this field offers a the facto standard for the definition and interoperativity of its basic kernels.

In recent years, another constraint that emerged in the HPC field, and in partcular in NLA, is the energy consumption required to compute the different kernels [14, 5, 15]. This situation motivated, among other things, the development of the Green 500 list [1]. This effort reorders the Top500 hardware platform considering, instead of the attainable peak performance, the ratio between performance and energy consumption (e.g. GFLOPs per watts). Thus, in the last decade the energy consumption of both, algorithm and hardware platforms, has been a matter of utmost importance.

Field-Programable Gate Arrays (FPGAs) technology is gaining attention in the HPC community. As a reconfigurable device, FPGAs are very efficient for implementing parallel algorithms. Even though they offer lower memory bandwidth and clock frequencies, FPGAs are becoming more powerful, narrowing the gap with other heterogeneous platforms like GPUs. They are still behind in raw computing power with GPUs, but given their considerably lower power consumption, there is an active topic of research for energy efficiency on FPGAs. In addition to the hardware upgrades, High-Level Synthesis (HLS) tools are becoming more and more refined which enables higher productivity and (may provide) more access to non-hardware experts.

In the previously described context we advance in the study of the potential of FPGAs to address NLA operations. More in detail, in this work we explore and evaluate the available options for two of the most representative kernels of BLAS, i.e. GEMV and GEMM. The experimental evaluation was carried out in an Alveo U50 acelerator card from Xilinx and an Intel Xeon Silver multi-core CPU. It shows that even in kernels where the CPU reaches better runtimes, the FPGA counterpart requires less energy consumption.

The rest of the paper is structured as follows. In Section 2 we summarize the main concepts of BLAS and the use of FPGAs platforms for HPC. Later, in Section 3, we present the different versions of both studied kernels. This is followed by the experimental evaluation in Section 4. Finally, in Section 5 we close with the main concluding remarks and some lines of future work.

2 FPGAs and NLA

In this section we briefly introduce the BLAS specifications and the main concepts related with the use of FPGAs for HPC.

2.1 BLAS

Numerical Linear Algebra (NLA) is a research field that in the last decades has been characterized by the use of kernel-libraries that are de facto standards. One of the most remarkable examples, in particular in the HPC field, is the Basic Linear Algebra Subroutines (BLAS) [8]. This library has become essential for HPC due to its efficiency, portability and availability. BLAS is composed of routines for computing common linear algebra operations. It is organized into levels according to the degree of complexity. The level 1 involves scalar, vector and vector-vector operations, the level 2 includes matrix-vector operations, and the level 3 performs matrix-matrix operations. BLAS libraries have become one of the main building blocks in linear algebra applications, such as solving linear system of equations, linear least square problems or eigenvalue problems. Two of the most important operations are GEMM and GEMV from levels 3 and 2 respectively. We describe these operations next.

GEMM This operation belongs to Level 3 of the BLAS specification [10] and is defined as follows:

$$C = \alpha A * B + \beta C \tag{1}$$

where A, B and C are matrices and α and β are scalars. This kernel is considered the main building block in dense linear algebra because many other operations can be expressed in terms of several GEMM invocations [6].

GEMV This operation is defined as follows:

$$y = \alpha A * x + \beta y \tag{2}$$

where A is a matrix, x and y are vectors and α and β are scalars. GEMV belongs to Level 2 of the BLAS specification.

2.2 FPGAs

FPGAs are composed of a matrix of configurable logic blocks (or logic elements) and hardcoded blocks such as memories, hardware adder/multipliers (DSPs) and clock managers. On top of that, a programmable routing structure enables the interconnection of the different blocks. They also feature several programmable input/output pins that allow interfacing with the outside world.

To *program* an FPGA means that an actual electrical circuit is synthesized inside the device through the programmable logic's interconnection-elements and hard-coded blocks. This allows very low latency (as there is little to none control overhead), great flexibility (as they can be reprogrammed in the field), and fine-grained parallelism. From a technological perspective, FPGAs stand somewhere in between Application-Specific Integrated Circuits (ASICs) and general-purpose processors. One of the main advantages of FPGAs with respect to ASICs is that the first can be reprogrammed after the manufacturing process.

The clock's operating frequency of a given design depends on the synthesized circuit, but it is usually lower than other heterogeneous devices. FPGAs also offer lower peak floating-point performance than GPUs and even multi-core CPUs, and less memory bandwidth, but this may change at some point, as FPGA manufacturers are making big efforts to compete with GPU performance in these contexts.

Traditionally, FPGAs have been a good alternative in fixed-point, dataflow streaming applications, where they reach high speeds at excellent energy efficiency. Also, as opposed to CPUs and GPUs they natively support arbitrary precision bitwidths. However, their poor performance in floating-point arithmetic, in addition to the complex design flow kept them apart from the mainstream HPC world. This started to change recently, as modern high-end FPGA devices offer up to millions of logic elements, thousands of DSP blocks (that allow TFLOP performance) and high-bandwidth memory (HBM). These characteristics, in combination with the HLS tools available, are making these devices increasingly attractive in the HPC domain.

A brief review of the state of the art about the use of FPGAs to compute dense numerical algebra kernels can be found in F. Favaro et al. [17, 16].

3 Evaluated kernels

3.1 Vitis libraries

Xilinx offers an extensive set of performance-optimized, open source libraries for use with Vitis software. Their repository includes common topics such as math, linear algebra, statistics, data management, and also domain specific libraries for image processing, computer vision, data compression, etc. For linear algebra, Xilinx developed Vitis BLAS Library, which is an FPGA implementation of the Basic Linear Algebra Subroutines (BLAS).

The library provides three levels of implementations (not to be confused with the BLAS levels organization): primitives (L1), kernels (L2), and software APIs (L3). L1 provides parametrized C++ implementations (to be compiled with HLS) of the basic operations found in BLAS. These primitives include modules for computations and for data movement. The first ones have streaming interfaces and carry out the operations, while the second ones move data between on-chip memory and the computation modules. This allows the programmer to construct high-performance logic by interconnecting computation and data mover modules. L2 offers kernel implementation examples aimed at host code developers. L3 provides C/C++ and Python APIs to allow software developers to accelerate BLAS operations using pre-built FPGA images.

For this work we evaluated the BLAS function kernels from L2. These kernels share the same top function, which has only two ports to communicate with external memory (DRAM, HBM or PLRAM). The kernels consist of an instruction processing block, a computation unit (e.g. GEMM), and a timer unit.

GEMM **basic:** The architecture of this kernel is composed of the following blocks:

- Systolic array: Implemented using L1 primitives. Its size depends on the datatype and the memory interface. For single precision floating point and 512 bits interface it corresponds to 16×16.
- Data movers: These blocks get data from global memory and send it to the computation blocks, and vice versa.
- Transpose modules: One of the matrices must be transposed before entering the systolic array. This block also acts as a buffer to reuse data.

GEMM **multiple compute units (MCU):** This kernel is implemented as two parallel instances (compute units) of the previous kernel. Each compute unit has its own dedicated HBM channel. The provided version of this kernel uses four compute units and its intended for the Alveo U250 board. In order to fit the design in the ALveo U50 board only 2 instances could be used. Also the DDR memory had to be changed for HBM.

GEMV **basic:** This kernel follows the same structure as GEMM basic, but with a custom processing block to perform GEMV operation.

GEMV **streaming:** This kernel does not follow the aforementioned BLAS kernels unique function architecture. Instead, it makes efficient use of the high-bandwidth memory. To maximize throughput, it instantiates 16 parallel GEMV compute blocks and connects each one to an individual HBM channel.

3.2 Matrix-matrix multiplication (MMM)

In order to obtain a point of comparison for the results of Vitis BLAS, we included in the evaluation a state of the art implementation for GEMM developed by J. de Fine Licht et al. [9]. They propose a matrix-matrix multiplication (MMM) implementation on FPGA aimed at minimizing off-chip data movement (by reusing data stored in fast on-chip memory) and maximizing performance (computations per I/O operation). They start with a general model for computation, I/O and resource utilization to create a hardware architecture that is highly optimized for the resource available on a target device.

Their I/O model assumes a parallel machine consisting on p processors, each one with S words of fast private memory. To perform an arithmetic operation, each processor must have all operands in its fast memory. They model the MMM algorithm as a computation directed acyclic graph (CDAG), where each vertex corresponds to a unique value during the execution and the edges represent data dependencies between them.

They constrain their model based on FPGA available resources and characteristics (number of ports and limited fan-out) to maximize the computation

throughput and favour routability. They reach a logic hierarchy which encapsulates various FPGA resources and guides the implementation to minimize I/O and maximize performance. The implementation follows a systolic array architecture, were N_p processing elements (PE) consume pre-fetched elements of the matrices A and B in a stream-like fashion. Each PE holds N_c compute units (CU) and each one of them is capable of producing one output product (partial result of matrix C) every clock cycle. The PEs are encapsulated in compute tiles, which are in turn grouped in block tiles. On top, a memory tile encapsulates the block tiles, using all available memory blocks of the FPGA. The parallelism is determined by the number of compute units.

Their implementation is done in HLS C++, is flexible (parametrized), portable, scalable (adaptable to different FPGAs with different number of resources and with different characteristics) and open source (rare for highly tuned FPGA implementations).

They tested the implementation for different configurations of tiles size and number of CUs and for various data types, measuring performance and energy efficiency. Their design achieved 409 GOp/s 32-bit floating point performance, and 1.5 TOp/s 8-bit integer performance, utilizing more than 80% of hardware resources in a Xilinx VCU1525 accelerator board.

4 Experimental evaluation

4.1 Setup

We used the following hardware for the experiments:

- An Alveo U50 FPGA accelerator card from Xilinx. The FPGA is based on the UltraScale+ architecture and includes 872K look-up tables, 1743K registers, 28 MB of internal RAM, and 5952 DSP blocks. The chip also has 8 GB of HBM RAM. The designs for this platform were compiled using Xilinx Vitis 2020.2.
- A system with an Intel Xeon Silver 4208 CPU with 8-cores running at 2.1 GHz, and 80 GB of RAM. The CPU implementations make use of Intel MKL library, using all 8 cores (8 threads) with SMT disabled and AVX2 instructions. This device is capable of AVX512, but we experimentally determined that using this feature in multicore execution severely limits the operating frequency of the cores, which degrades the performance.

We performed the characterization of performance and energy consumption as follows:

- In the Alveo U50 FPGA, the board has internal sensors that provide current, voltage, and temperature readings while the kernel is running. The driver Xilinx Runtime (XRT) sends these values to the host.
- In the Intel Xeon processor, we measured CPU and memory power consumption using RAPL (which provides an estimate of the dissipated power based on performance counters and a device power model).

- All power measurements were automated using PMlib [4]. Results are an average of readings collected during 2 minutes of execution, with an equal warm-up time before measuring.
- The runtime measurements are the average of multiple iterations of the kernels.

4.2 Experimental results and discussion

For the experimental evaluation, and as a baseline, we employ the BLAS implementations offered by MKL library on CPU.

All the results summarized in this section are the average of 10 independent executions. Also, in all cases we used single precision floating point.

The resource utilization of the implemented FPGA kernels is shown in Table 1.

Table 1. Resource utilization in percentage of available resources for the implemented FPGA kernels.

		Utilization (%)				
Type	**Available**	**GEMV** basic	**GEMV** Streaming	**GEMM** basic	**GEMM** MCU	**MMM**
LUTs	870016	14.02	22.83	37.39	61.16	42.45
Registers	1740032	8.98	17.10	28.30	47.59	32.33
Block RAM	1344	16.22	16.07	18.34	22.84	53.27
DSPs	5940	0.29	9.87	20.94	41.82	46.13

In the first experiment we evaluate the computational performance reached by the different versions for the GEMV operation over square matrices of: 128, 256, 512, 1024, 2048, 4096 and 8192 columns. Specifically, Figure 1 presents the GFLOPs achieved by all the evaluated variants.

Considering the obtained experimental results for the GEMV kernel, and with focus on the FPGAs variants, firstly we can say that the basic version is a non-competitive option. This implementation has very low levels of parallelism, because it performs the dot product on vectors of 16 elements. Also, due to a carried-dependency issue in the computation loop, it ends operating 4 times slower than intended. Next, for the small test cases the MMM reaches better runtimes than the Streaming variant, which has a poor performance for small matrices. However, for dimension bigger than 1024 the result is reversed. More in detail, the performance of the MMM variant is stagnant since matrices of 1024 while the Streaming counterpart is growing even for the largest matrices. The Streaming variant provides 16 times more parallelism than the basic version and takes advantage of the HBM on the Alveo board. The CPU version offers the best peak of performance for matrices of 1024 columns, but in the largest test

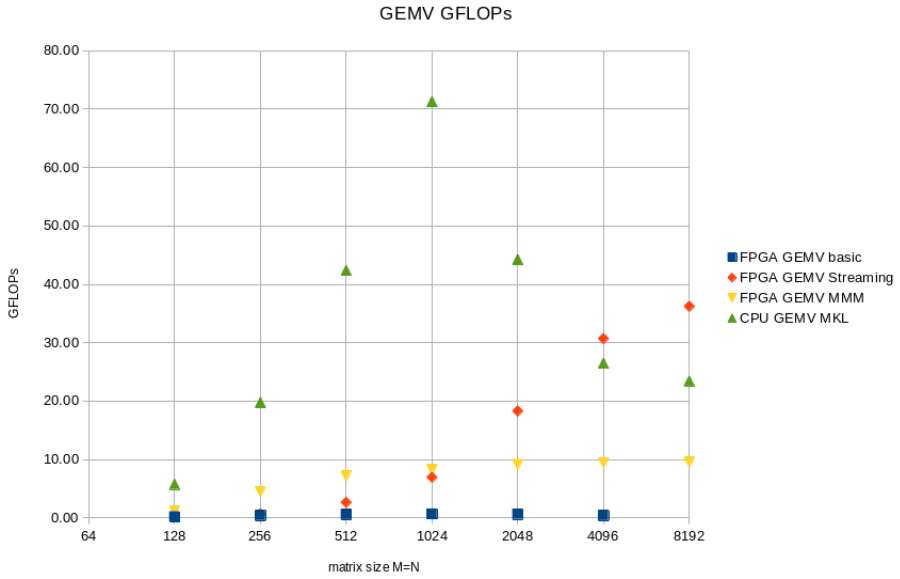

Fig. 1. Achieved performance (GFLOPs) of the GEMV kernels for different matrix sizes.

cases the performance is degraded. This result is reasonable taking into account the effects of the use of the cache memories.

Later, we evaluate the energy consumption implied by the different GEMV implementations. In this line, Figure 2 presents the GFLOPs per watt achieved by the evaluated variants over the same range of matrices. Similar to the performance evaluation, the energy study allows us to conclude that the fastest GEMV version is, in general, the most energy-efficient option. However, in all cases the FPGAimplementations require less power than the CPU counterpart. Additionally, the MMM version uses, on average, less power than the Streaming variant. Finally, and as a remarkable aspect, the FPGA version outperforms (in the energy-consumption perspective) the CPU counterpart for the three largest dimensions.

The experimental results for the performance reached by the GEMM kernels is shown in Figure 3. For this operation, the CPU variant is faster than the FPGA counterparts for all the evaluated matrix dimensions. With focus on the FPGA, the basic variant is far below the other versions in performance. For the basic and MCU GEMM kernels, performance climbs to a maximum around dimensions 1024 and then starts to degrade. The cause for this performance loss for the bigger sizes was not fully determined and needs to be further investigated. The MMM variant also peaks around size 1024 but the for bigger sizes the performance remains constant.

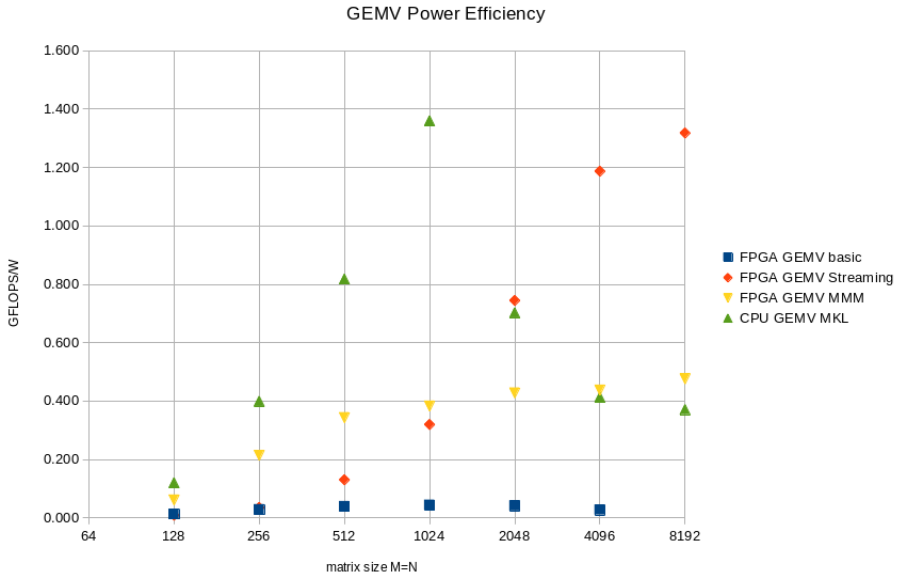

Fig. 2. Energy efficiency (GFLOPs/W) of the GEMV kernels for different matrix sizes.

The energy consumption results for the GEMM kernels are summarized in Figure 4. Contrary to the performance results achieved in the GEMV experiment, in this case the FPGA outperforms the CPU counterpart for six of seven matrix dimensions. This situation remarks the energy efficiency offered by the FPGA platforms, specially in this context where the CPU is faster than other versions. Neither of the Vitis BLAS versions manages to outperform the CPU in this case (except for the smallest matrix size). This is expected since the evaluated kernels for GEMM were designed for bigger FPGAs boards and are not optimized for the Alveo U50 platform (contrary to the Streaming GEMVwhich was designed for this board).

5 Conclusions

In this article we have revisited the use of non-traditional HPC hardware to compute BLAS kernels. Specifically, we review the available kernels to compute the GEMV and GEMM kernels in FPGAs and also we extend and tune some other variants of these kernels. The experimental evaluation carried out over an Alveo U50 FPGA board shows that, in general, the CPU version outperforms in GFLOPs the FPGAs counterparts but the use of FPGAs offer more efficient variants from the energy consumption perspective. These results are very relevant. First, due to the importance of the energy consumption as a restriction in

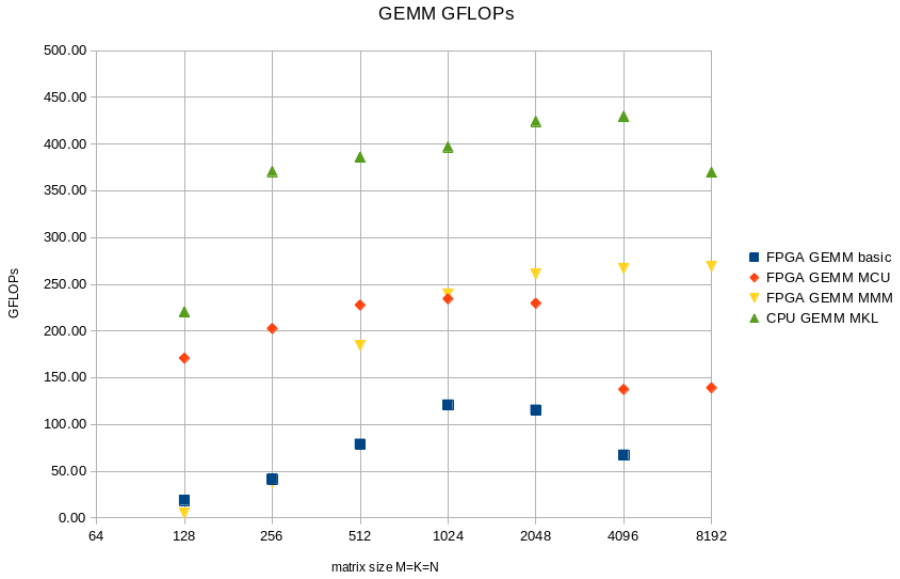

Fig. 3. Achieved performance (GFLOPs) of the GEMM kernels for different matrix sizes.

the HPC field and second, considering the years of development of CPU implementations compared to the recent focus on this kind of methods for FPGAs.

As a future line of work we identify several perspectives, following we describe the most important ones.

- Firstly, is mandatory to extend our study to include other FPGAs with different characteristics, among other Intel FPGAs.
- Secondly, the comparison should be done with other heterogeneous hardware platforms. In particular, comparing the FPGA kernels with implementations on cutting-edge GPUs and particularly low-consuming devices (such as ARM processors).
- Also, it would be interesting to complement the GFLOPs and GFLOPs per watt metrics with other perspectives, maybe the most important being the learning curve in FPGA design and the design and compilation times of the FPGAs implementations.

Acknowledgements The researchers were supported by Universidad de la República and the PEDECIBA. We acknowledge the ANII – MPG Independent Research Groups: "Efficient Hetergenous Computing" with the CSC group.

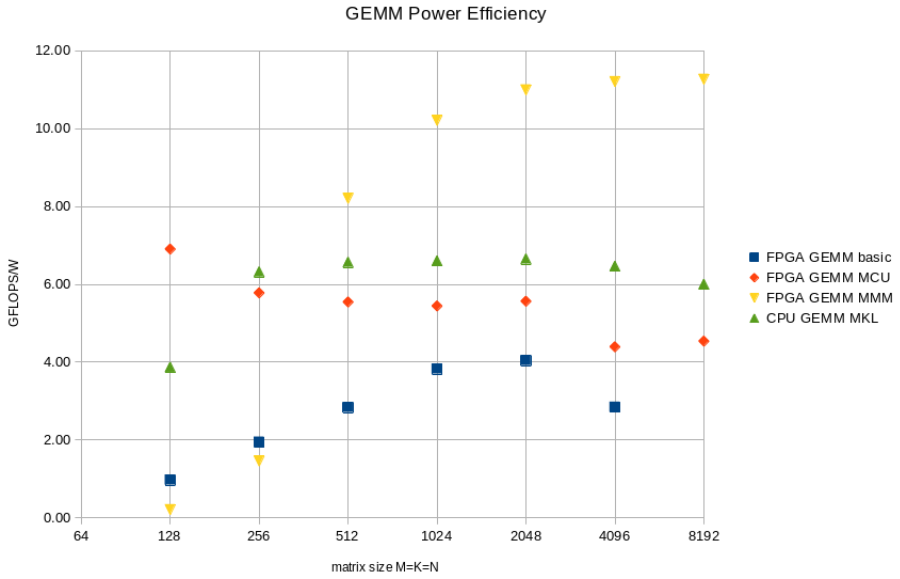

Fig. 4. Energy efficiency (GFLOPs/W) of the GEMM kernels for different matrix sizes.

References

1. The Green500 list, 2022. Available at http://www.green500.org.
2. The top500 list, 2022. Available at http://www.top500.org.
3. E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, Jack J. Dongarra, J. Du Croz, S. Hammarling, A. Greenbaum, A. McKenney, and D. Sorensen. *LAPACK Users' guide*. SIAM, 3rd edition, 1999.
4. Sergio Barrachina, Maria Barreda, Sandra Catalán, Manuel F Dolz, Germán Fabregat, Rafael Mayo, and ES Quintana-Ortí. An integrated framework for power-performance analysis of parallel scientific workloads. *Energy*, pages 114–119, 2013.
5. Peter Benner, Pablo Ezzatti, Enrique S. Quintana-Ortí, and Alfredo Remón. On the impact of optimization on the time-power-energy balance of dense linear algebra factorizations. In Rocco Aversa, Joanna Kolodziej, Jun Zhang, Flora Amato, and Giancarlo Fortino, editors, *Algorithms and Architectures for Parallel Processing - 13th International Conference, ICA3PP 2013, Vietri sul Mare, Italy, December 18-20, 2013, Proceedings, Part II*, volume 8286 of *Lecture Notes in Computer Science*, pages 3–10. Springer, 2013.
6. Paolo Bientinesi, John A. Gunnels, Margaret E. Myers, Enrique S. Quintana-Ortí, Tyler Rhodes, Robert A. van de Geijn, and Field G. Van Zee. Deriving dense linear algebra libraries. *Formal Aspects of Computing*, 25(6):933–945, Nov 2013.
7. L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. SIAM, 1997.

8. L Susan Blackford, Antoine Petitet, Roldan Pozo, Karin Remington, R Clint Whaley, James Demmel, Jack Dongarra, Iain Duff, Sven Hammarling, Greg Henry, et al. An updated set of basic linear algebra subprograms (BLAS). *ACM Transactions on Mathematical Software*, 28(2):135–151, 2002.
9. Johannes de Fine Licht, Grzegorz Kwasniewski, and Torsten Hoefler. Flexible communication avoiding matrix multiplication on FPGA with high-level synthesis. *In Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'20), February 23-25, 2020, Seaside, CA, USA*, December 2020.
10. J. J. Dongarra, Jeremy Du Croz, Sven Hammarling, and I. S. Duff. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Softw.*, 16(1):1–17, March 1990.
11. Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Iain Duff. A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, 16(1):1–17, March 1990.
12. Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Richard J. Hanson. An extended set of FORTRAN basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, 14(1):1–17, March 1988.
13. Jack J. Dongarra, Piotr Luszczek, and Antoine Petitet. The linpack benchmark: Past, present, and future, 2002.
14. J. Dongarra *et al.* The international ExaScale software project roadmap. *Int. J. of High Performance Computing & Applications*, 25(1):3–60, 2011.
15. Pablo Ezzatti, Enrique S. Quintana-Ortí, Alfredo Remón, and Jens Saak. Power-aware computing. *Concurrency and Computation: Practice and Experience*, 31(6):e5034, 2019. e5034 cpe.5034.
16. Federico Favaro, Ernesto Dufrechou, Pablo Ezzatti, and Juan Pablo Oliver. Energy-efficient algebra kernels in FPGA for High Performance Computing. 21, Oct 2021.
17. Federico Favaro, Juan P. Oliver, Ernesto Dufrechou, and Pablo Ezzatti. Understanding the Performance of Elementary NLA Kernels in FPGAs. In *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 479–482, 2020.
18. C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for Fortran usage. *ACM Transactions on Mathematical Software*, 5(3):308–323, September 1979.