OPTIMA Open-Source (OOPS) library Evaluation Results

Work package:	WP6	AFFILIATION
Author(s):	Dimitris Theodoropoulos	ICCS
	Panagiotis Miliadis	ICCS
	Panagiotis Mpakos	ICCS
	Dionisios Pnevmatikatos	ICCS

This project has received funding from the European High-Performance Computing Joint Undertaking Joint Undertaking (JU) under grant agreement No 955739. The JU receives support from the European Union’s Horizon 2020 research and innovation programme and Greece, Germany, Italy, Netherlands,Spain,Switzerland.

Executive Summary

The OOPS library delivered a set of 30 hardware kernels out of 3 libraries (BLAS, sparse algebra, CAE solvers) to allow developers to quickly map applications on FPGA-supported HPC systems. Most of the hardware kernels improve raw performance, when compared to multi-threaded software running on server-class machines. Moreover, OOPS enables implementations of HPC applications that can reduce energy consumption up to 50x when compared to server-class machines (up to 50x for L1 BLAS kernels, up to 20x for L2 kernels and up to 1.5x for L3 BLAS kernels and CAE solvers).

Table of Contents

1. Introduction

This report provides a set of results for the hardware kernels supported by the OPTIMA Open-Source (OOPS) library. OOPS delivers a set of 30 hardware kernels out of 3 libraries to allow developers to quickly map applications on FPGA-supported HPC systems. As described in the following sections, the OOPS library can increase raw performance for several BLAS kernels, sparse algebra-based algorithms, as well as CAE solvers. This is achieved at a lower energy footprint when compared to fully optimized software multi-threaded versions, such as Intel's Math Kernel Library (MKL). In a nutshell, OOPS enables the implementation of HPC applications with reduced energy consumption of up to 50x for L1 BLAS kernels, up to 20x for specific L2 BLAS kernels, up to 1.5x for specific L3 BLAS kernels, and up to 1.5x for CAE solvers, when compared to equivalent purely software versions running on server-class machines.

All kernels have been evaluated against the heavily optimized Intel's Math Kernel Library (MKL), using a different number of threads. Moreover, software runs were executed on a two-socket server with 256 GB of RAM, and two Intel Xeon Gold 5120 CPUs, each comprising 14 cores with 2 threads per core, summing up in total up to 56 threads. The FPGA card and CPU+memory power were measured using the xbutil [12] and turbostat [13] utilities respectively.

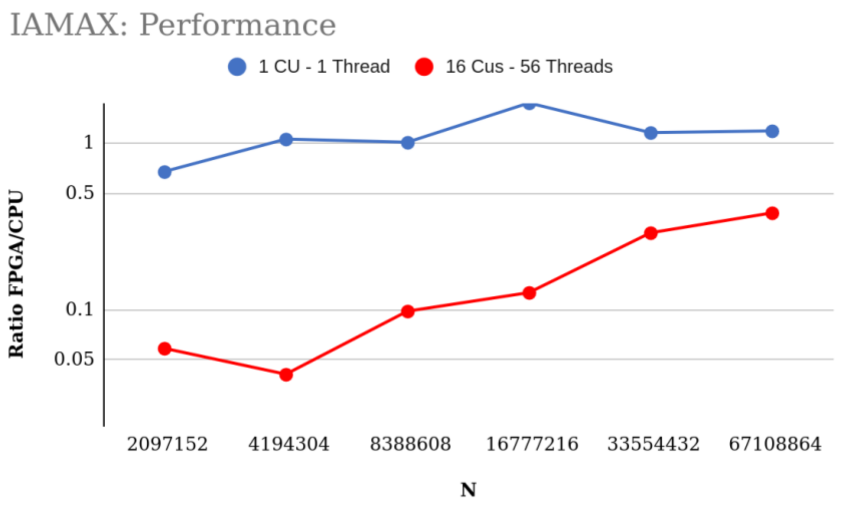
The OOPS library comprises twelve L1, eleven L2, and four L3 BLAS kernels, one sparse matrix-vector multiplication kernel, and three CAE solvers (LU decomposition, Jacobi preconditioner, and conjugate gradient calculation). It should be noted that Deliverable D5.5 describes how all OOPS kernels were implemented to target AMD's Alveo U55C FPGA cards with HBM memory.

2. Level 1 BLAS routines of the OOPS library

This section evaluates the OOPS BLAS L1 kernels in terms of scalability, performance and energy efficiency, using different vector sizes (N). For each kernel there are the following entries:

- a performance chart shows how the best hardware configuration (i.e. max number of CUs) compares vs the corresponding MKL function with a multi-thread configuration
- an energy efficiency chart that compares the performance / Watt ratio between hardware and MKL configurations.

2.1. OOPS_iamax



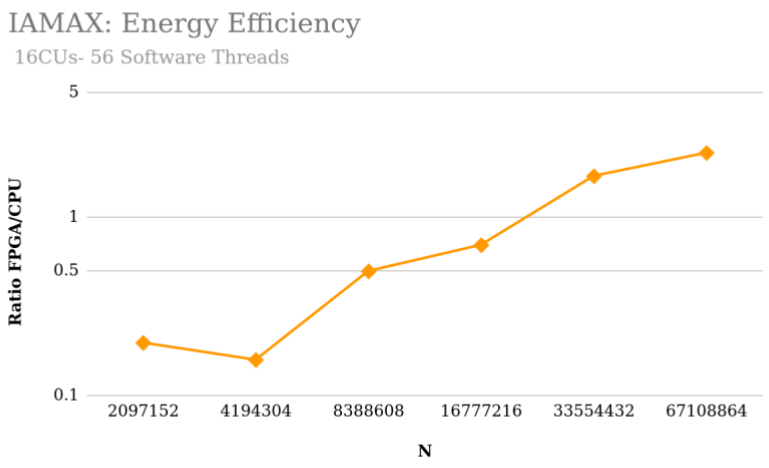
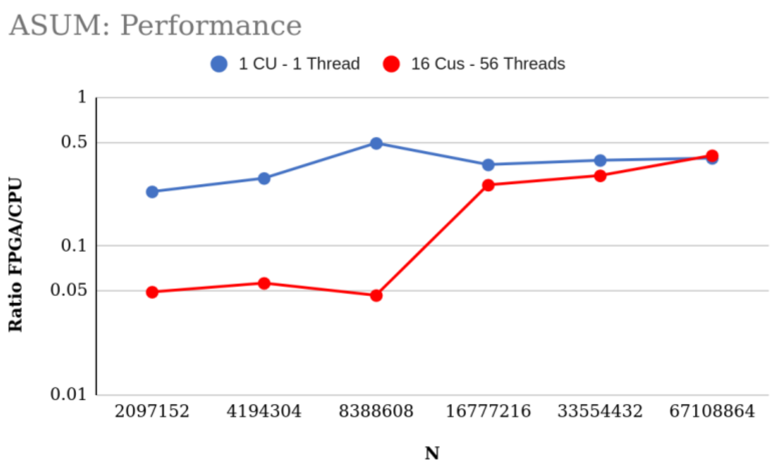


Figure 1: iamax performance and energy efficiency.

The iamax implementation supports up to 16 CUs. As shown in Figure 1, a single CU achieves 1.2x speedup, whereas the 16-CU version achieves 0.5x speedup vs single-threaded and 56-threaded software, respectively. Due to the FPGA's much less power consumption, energy efficiency is approximately up to 2x better.

2.2. OOPS_asum



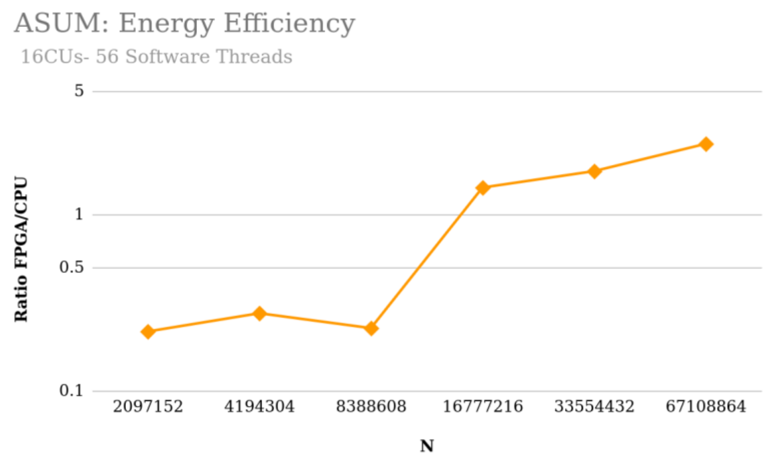
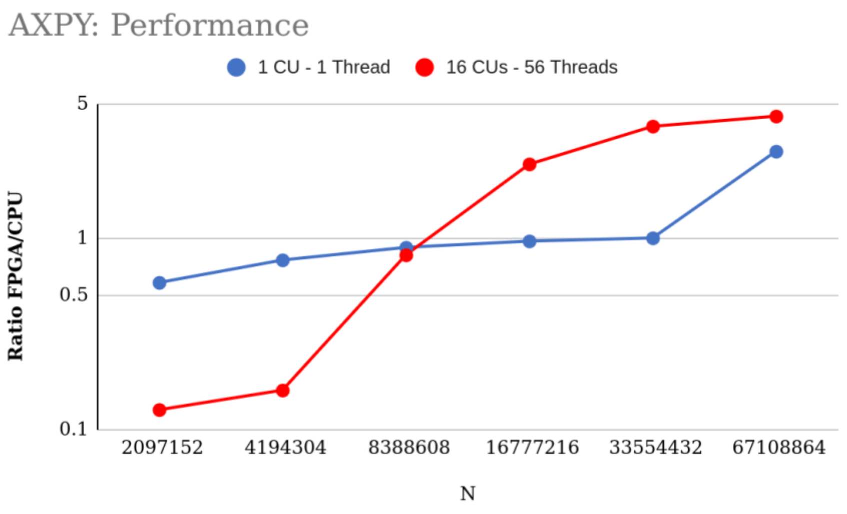


Figure 2: asum performance and energy efficiency.

The asum implementation supports up to 16 CUs. As shown in Figure 2, a single CU achieves 0.4x speedup, whereas the 16-CU version achieves 0.5x speedup vs single-threaded and 56-threaded software, respectively. Due to the FPGA's much less power consumption, energy efficiency is approximately up to 3.5x better.

2.3. OOPS_axpy



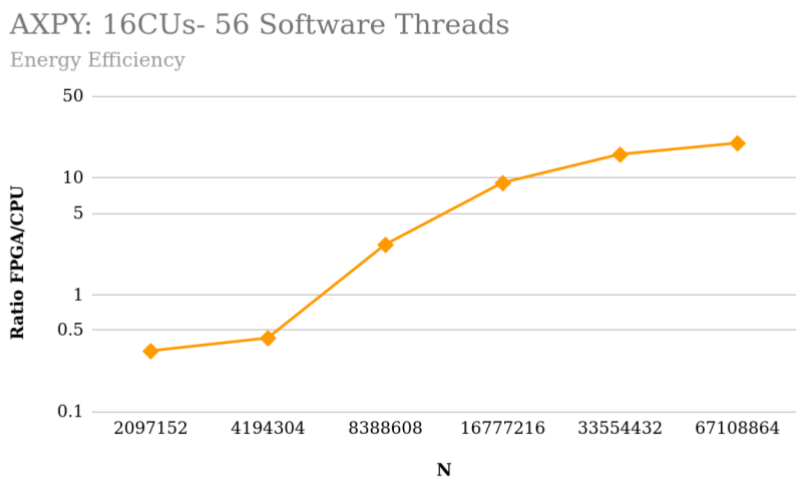
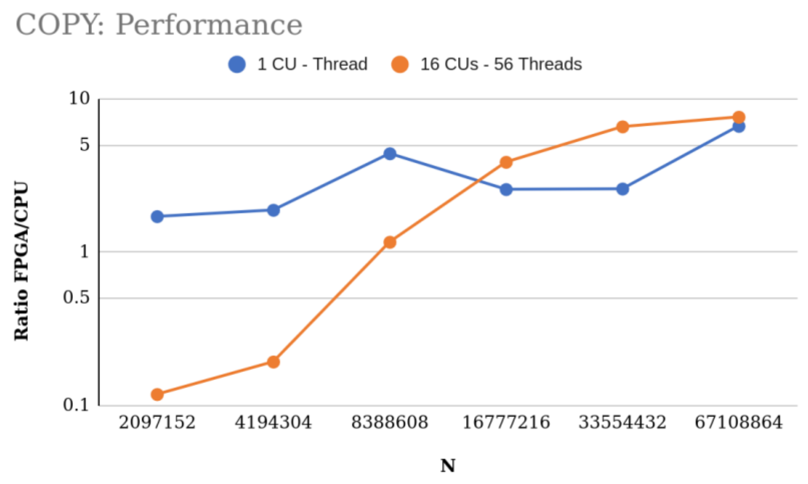


Figure 3: axpy performance and energy efficiency.

The axpy implementation supports up to 16 CUs. As shown in Figure 3, a single CU achieves 4x speedup, whereas the 16-CU version achieves 4.8x speedup vs single-threaded and 56-threaded software, respectively. Due to the FPGA's much less power consumption, energy efficiency is approximately up to 20x better.

2.4. OOPS_copy



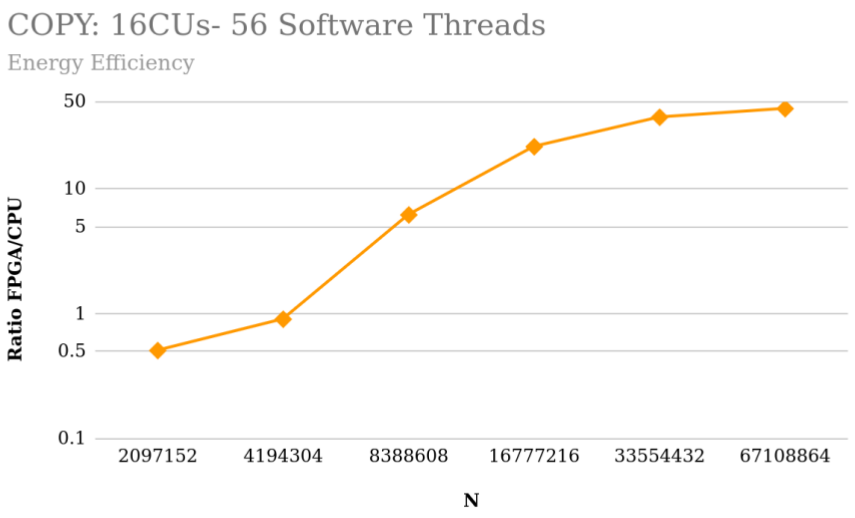
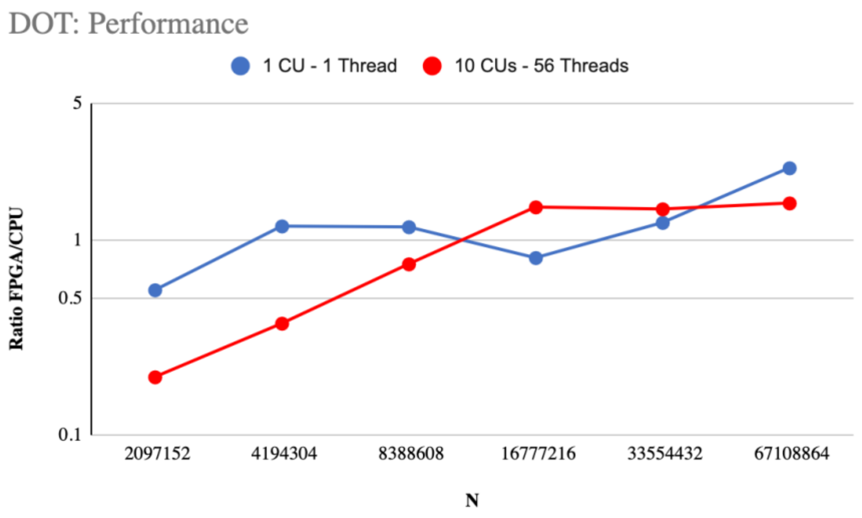


Figure 4: copy performance and energy efficiency.

The copy implementation supports up to 16 CUs. As shown in Figure 4, a single CU achieves 6x speedup, whereas the 16-CU version achieves 8x speedup vs single-threaded and 56-threaded software, respectively. Due to the FPGA's much less power consumption, energy efficiency is approximately up to 50x better.

2.5. OOPS_dot



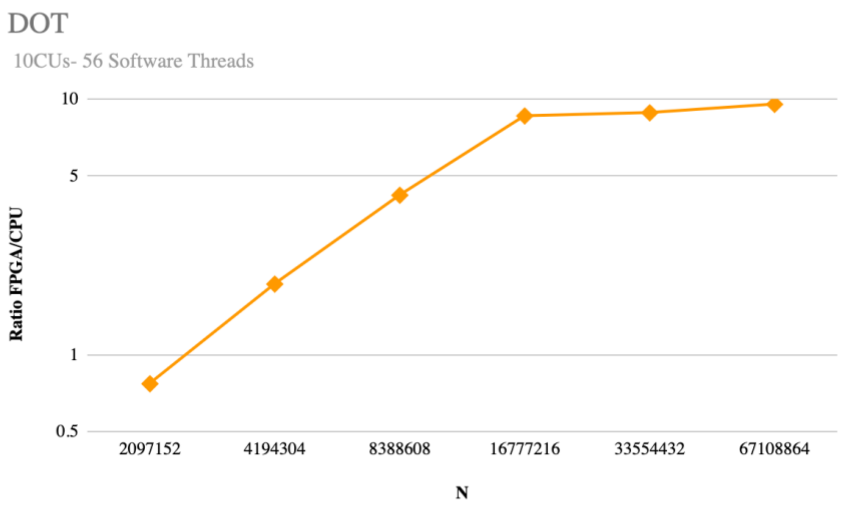
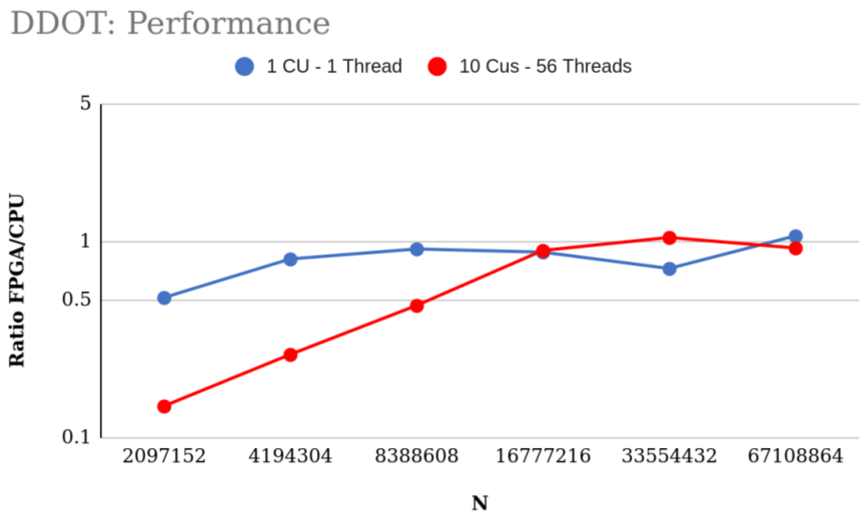


Figure 5: dot performance and energy efficiency.

The dot implementation supports up to 10 CUs. As shown in Figure 5, a single CU achieves 2.5x speedup, whereas the 10-CU version achieves 2.1x speedup vs single-threaded and 56-threaded software, respectively. Due to the FPGA's much less power consumption, energy efficiency is approximately up to 10x better.

2.6. OOPS_sddot



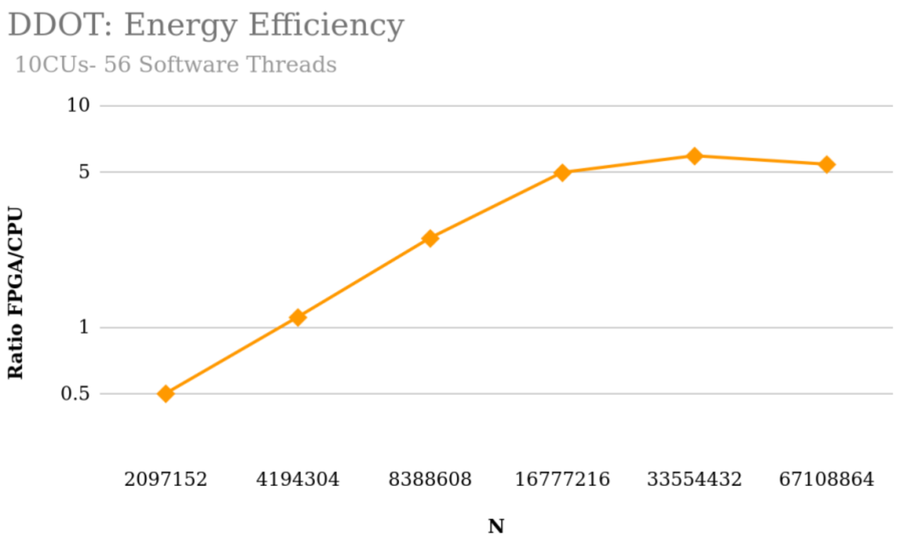
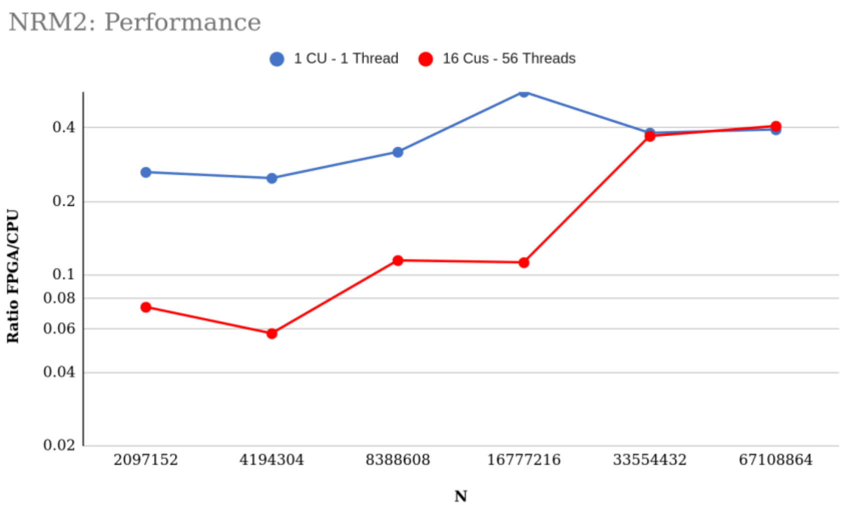


Figure 6: sddot performance and energy efficiency.

The sddot implementation supports up to 10 CUs. As shown in Figure 6, a single CU achieves 0.9x speedup, whereas the 10-CU version achieves 2.1x speedup vs single-threaded and 56-threaded software, respectively. Due to the FPGA's much less power consumption, energy efficiency is approximately up to 5x better.

2.7. OOPS_nrm2



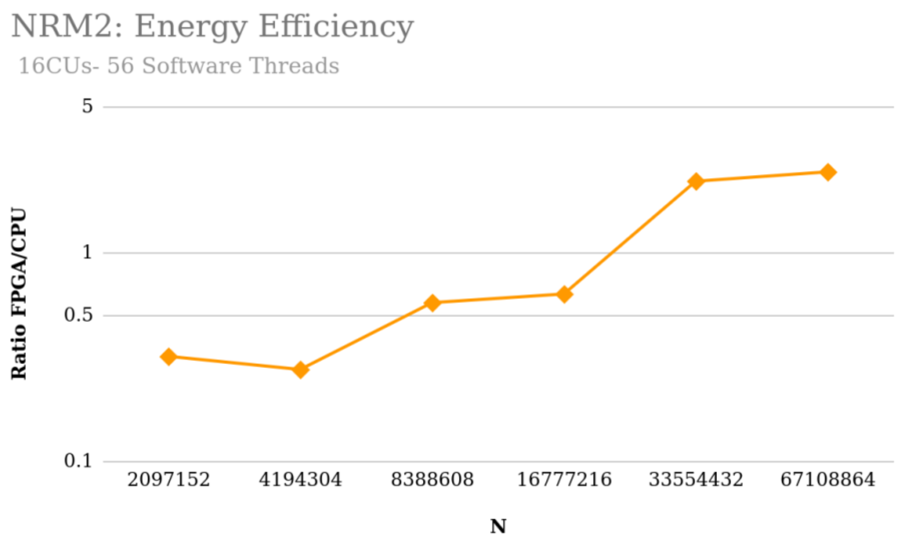
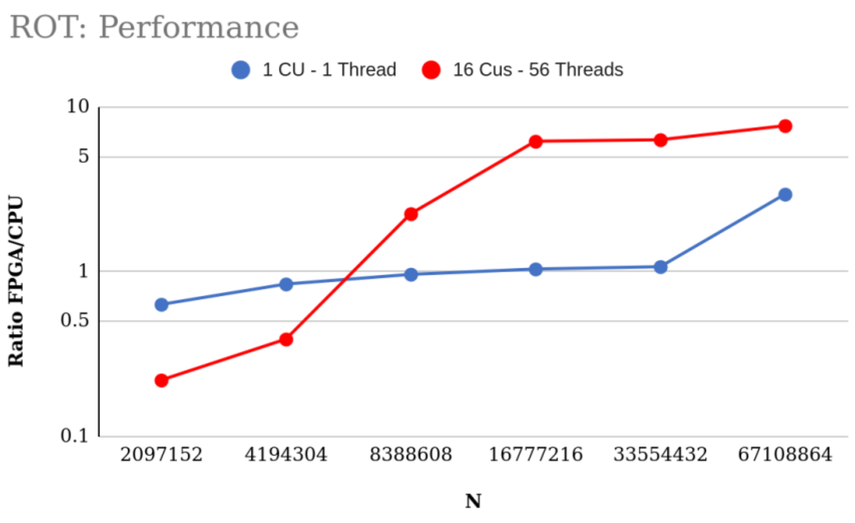


Figure 7: nrm2 performance and energy efficiency.

The nrm2 implementation supports up to 16 CUs. As shown in Figure 7, a single CU achieves 0.5x speedup, whereas the 16-CU version achieves 0.4x speedup vs single-threaded and 56-threaded software, respectively. Due to the FPGA's much less power consumption, energy efficiency is approximately up to 2.5x better.

2.8. OOPS_rot



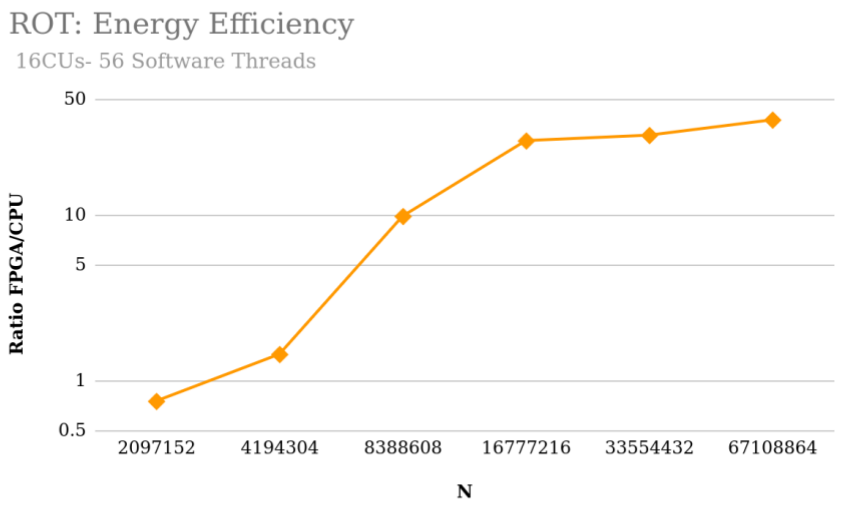
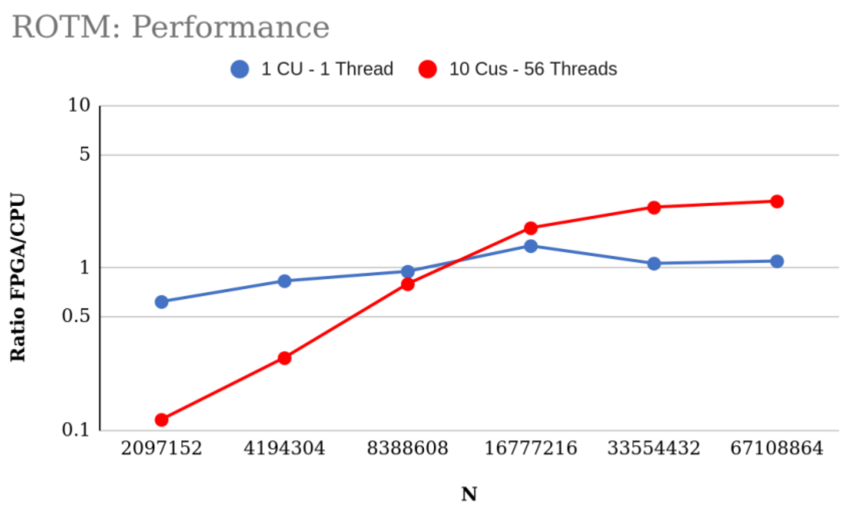


Figure 8: rot performance and energy efficiency.

The rot implementation supports up to 16 CUs. As shown in Figure 8, a single CU achieves 3.8x speedup, whereas the 16-CU version achieves 7x speedup vs single-threaded and 56-threaded software, respectively. Due to the FPGA's much less power consumption, energy efficiency is approximately up to 40x better.

2.9. OOPS_rotm



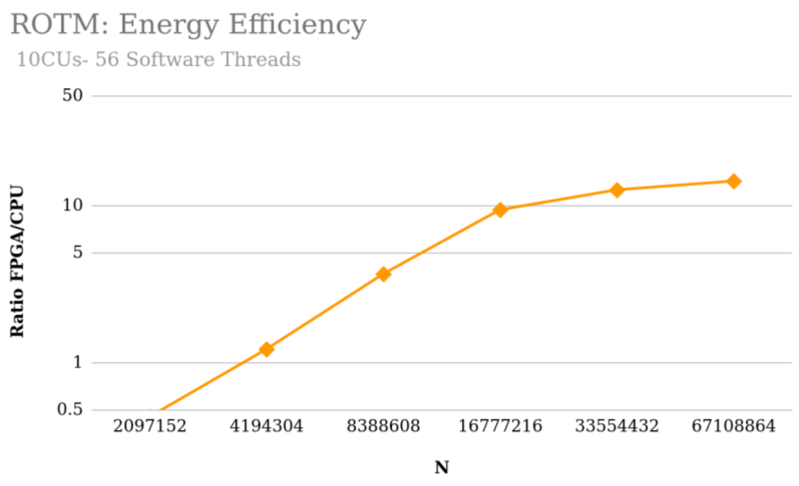
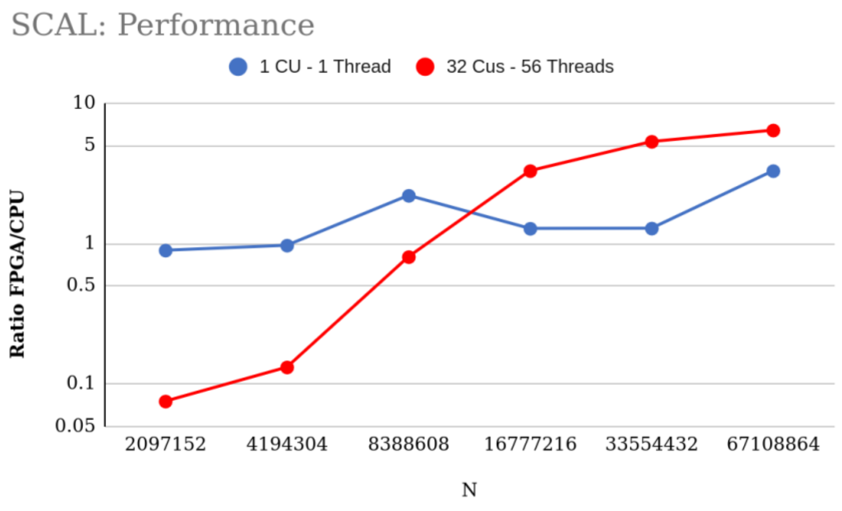


Figure 9: rotm performance and energy efficiency.

The rotm implementation supports up to 10 CUs. As shown in Figure 9, a single CU achieves 1.1x speedup , whereas the 10-CU version achieves 3.5x speedup vs single-threaded and 56-threaded software, respectively. Due to the FPGA's much less power consumption, energy efficiency is approximately up to 14x better.

2.10. OOPS_scal



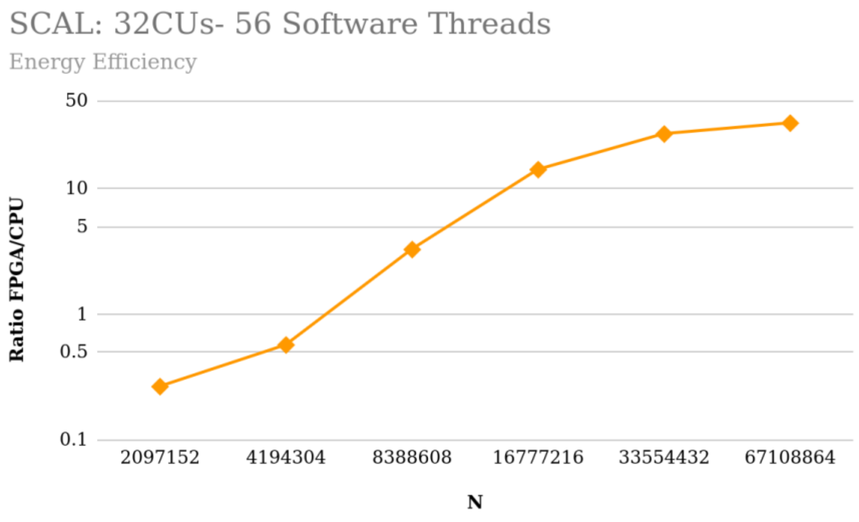
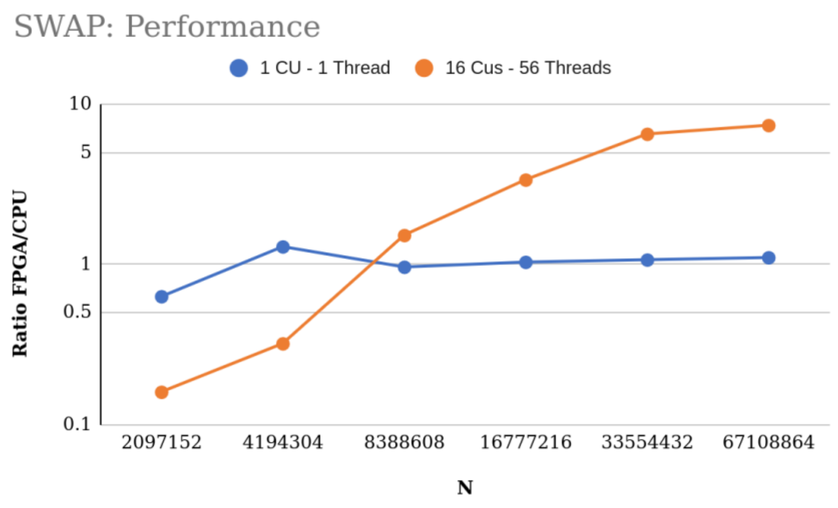


Figure 10: scal performance and energy efficiency.

The scal implementation supports up to 32 CUs. As shown in Figure 10, a single CU achieves 4x speedup, whereas the 32-CU version achieves 7x speedup vs single-threaded and 56-threaded software, respectively. Due to the FPGA's much less power consumption, energy efficiency is approximately up to 45x better.

2.11. OOPS_swap



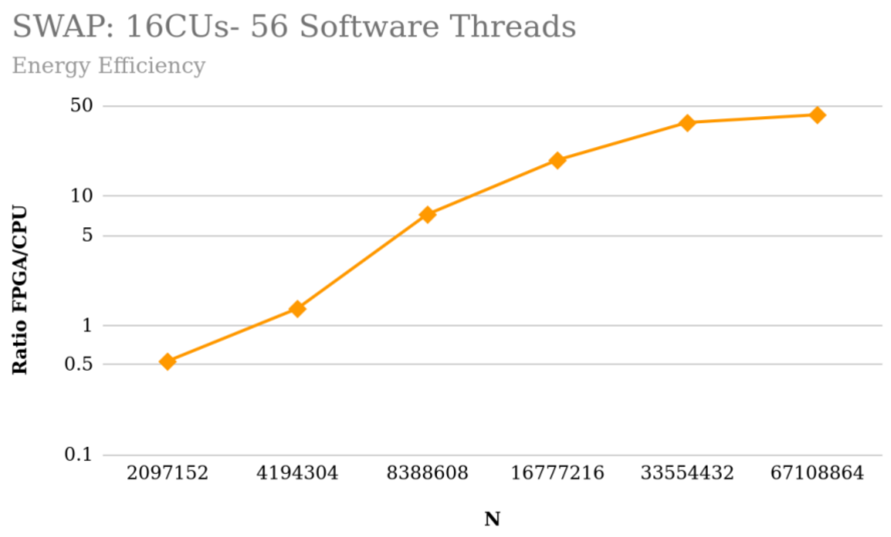
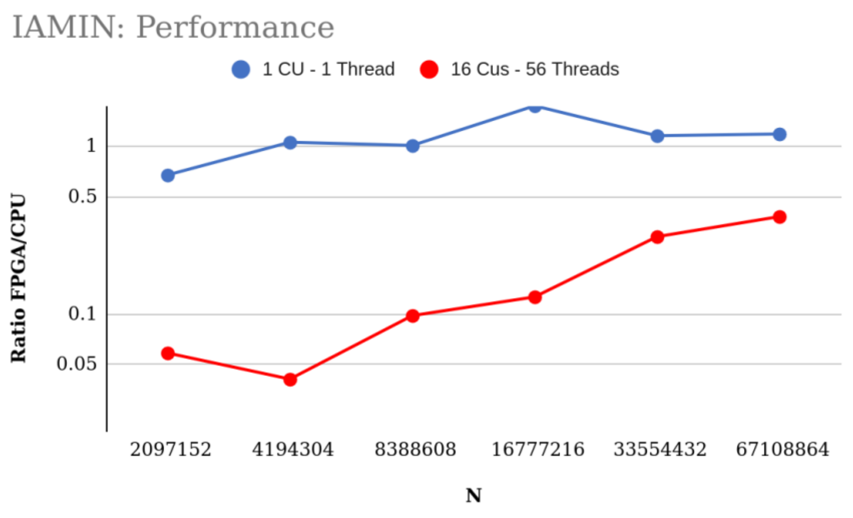


Figure 11: swap performance and energy efficiency.

The swap implementation supports up to 16 CUs. As shown in Figure 11, a single CU achieves 1.1x speedup, whereas the 16-CU version achieves 7x speedup vs single-threaded and 56-threaded software, respectively. Due to the FPGA's much less power consumption, energy efficiency is approximately up to 44x better.

2.12. OOPS_iamin



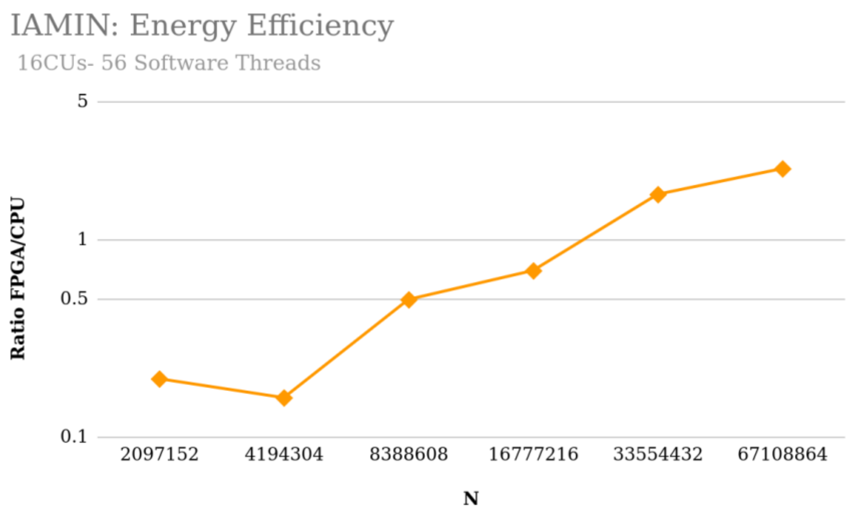


Figure 12: iamin performance and energy efficiency.

The iamin implementation supports up to 16 CUs. As shown in Figure 12, a single CU achieves 1.1x speedup, whereas the 16-CU version achieves 0.4x speedup vs single-threaded and 56-threaded software, respectively. Due to the FPGA's much less power consumption, energy efficiency is approximately up to 3.5x better.

3. Level 2 BLAS routines of the OOPS library

The following section evaluates the OOPS BLAS L2 kernels in terms of scalability, performance and energy efficiency. Each routine is evaluated in terms of performance and energy efficiency, utilizing different vector and matrix sizes on the x-axis in each chart. Performance indicates the peak achieved giga-floating point operations per second (GFLOPs), calculated through the following formula for L2 kernels: 2*N*N/execution_time(s). Meanwhile, energy efficiency denotes the achieved performance of the kernel per watt. Each chart compares the kernels of the OOPS library with the corresponding MKL routines, utilizing a multi-thread configuration. Finally, our evaluation incorporates a variable number of compute units, illustrating the scalability of the designs through the partitioning of the initial dataset.

3.1. OOPS_gemv

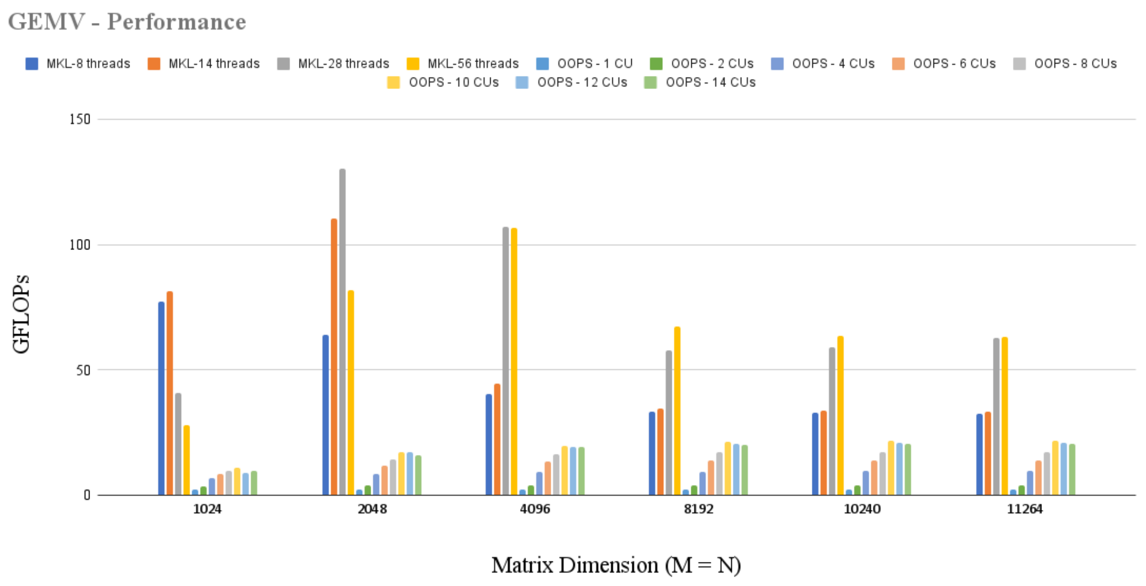


Figure 13: gemv performance.

The final version of the GEMV routine features 14 compute units. As shown in Figure 13, the optimal performance is achieved with 10 compute units, exhibiting raw performance approximately three lower than the MKL library operating on a 56-thread configuration. The addition of more compute units does not yield a substantial improvement in performance, as the frequency must be reduced to adhere to the routing constraints during compilation and generation of the final bitstream.

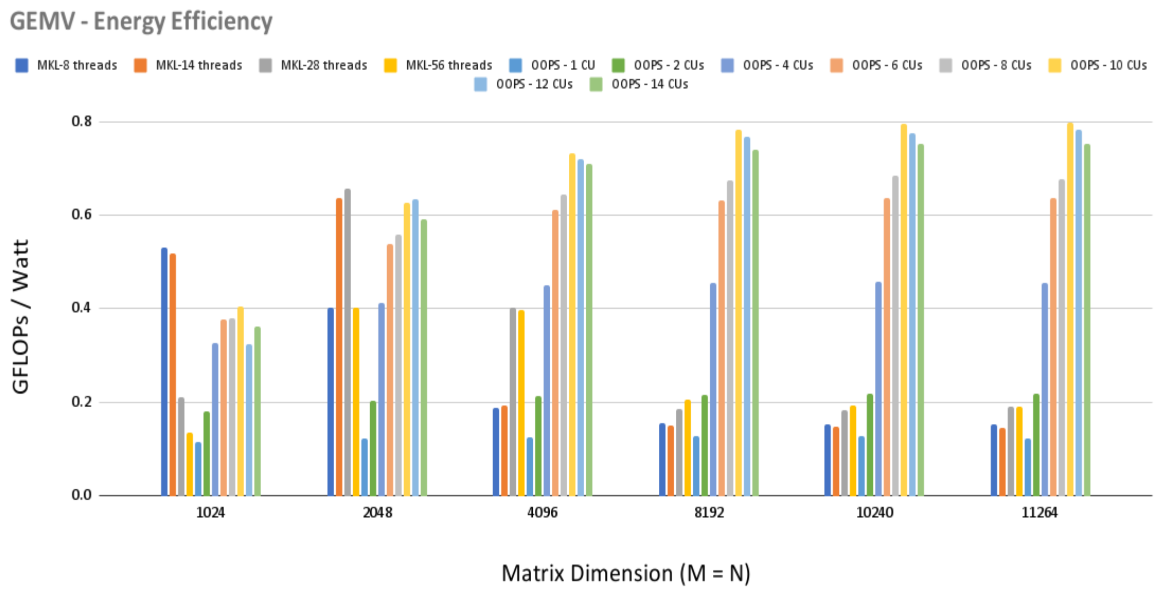


Figure 14: gemv energy efficiency.

Nonetheless, our design exhibits notable energy efficiency in comparison to the corresponding MKL subroutine. As shown in Figure 14, the configuration featuring 10 compute units demonstrates a noteworthy performance per watt ratio, approximately four times greater than the 56-thread configuration. This underscores the genuine potential of FPGA devices, delivering competitive performance alongside server-class and high-end CPUs but with significantly lower power consumption. Notably, our server-class CPU maintains competitiveness with the FPGA solely when handling small matrices, where all values can be accommodated within the caches.

3.2. OOPS_trmv

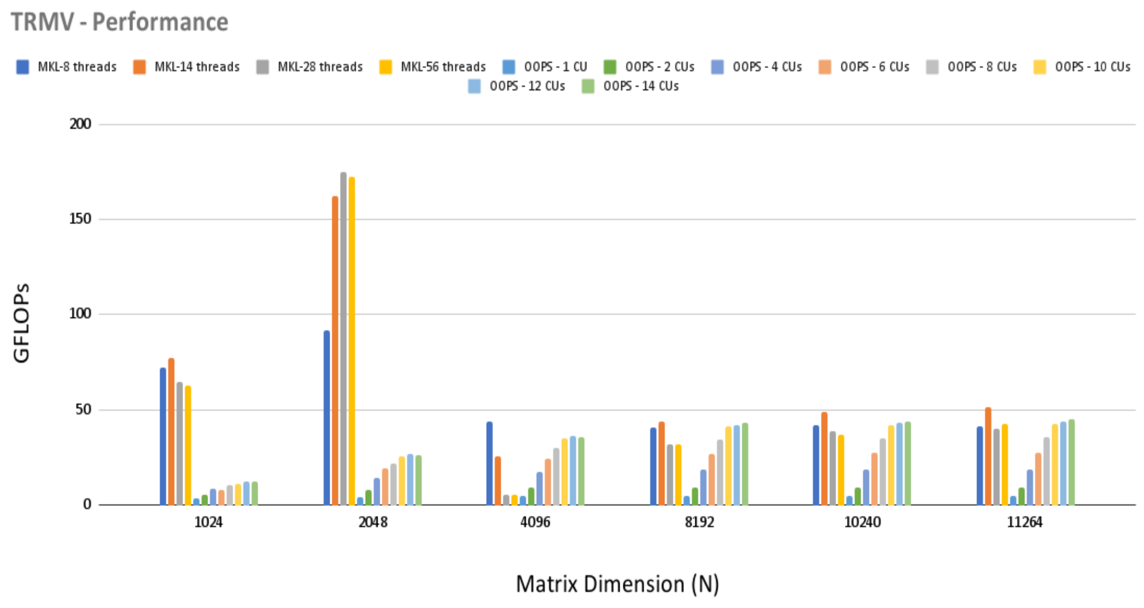


Figure 15: trmv performance.

The final version of the TRMV routine features 14 compute units. As shown in Figure 15, TRMV, being a less intricate kernel compared to the GEMV kernel, allows our design to attain peak performance by deploying 14 compute units, fully leveraging the available memory ports provided by the Alveo U55c platform. As depicted in the figure above, OOPS_TRMV demonstrates competitive raw performance when compared to the equivalent MKL routine, regardless of the thread configuration. The MKL library exhibits superior performance only in configurations with small dimension sizes, where all vectors and matrices can be accommodated within the caches.

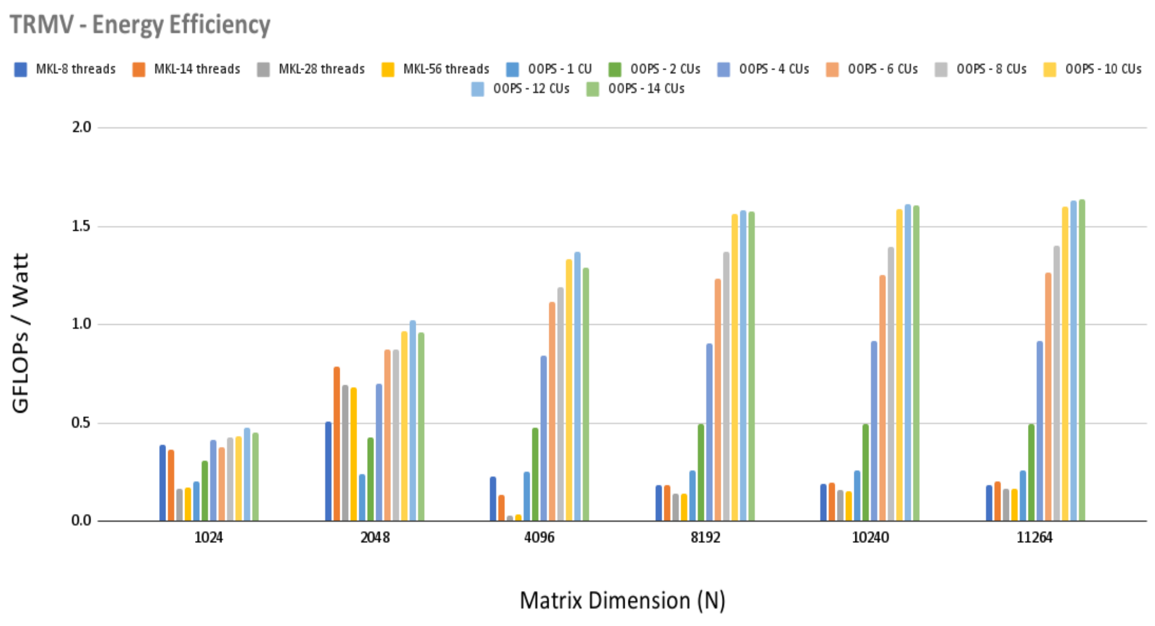


Figure 16: trmv energy efficiency.

Given that our TRMV kernel delivers competitive performance comparable to a server-class CPU, our design showcases outstanding energy efficiency relative to the corresponding MKL subroutine. Specifically, as shown in Figure 16, our configuration with 14 compute units demonstrates a performance per watt ratio 6.5x times greater, achieving equivalent GFLOPs performance to the server-class CPU but with significantly reduced power consumption. This underscores the potential of FPGA devices as a viable alternative for executing linear algebra subroutines.

3.3. OOPS_symv

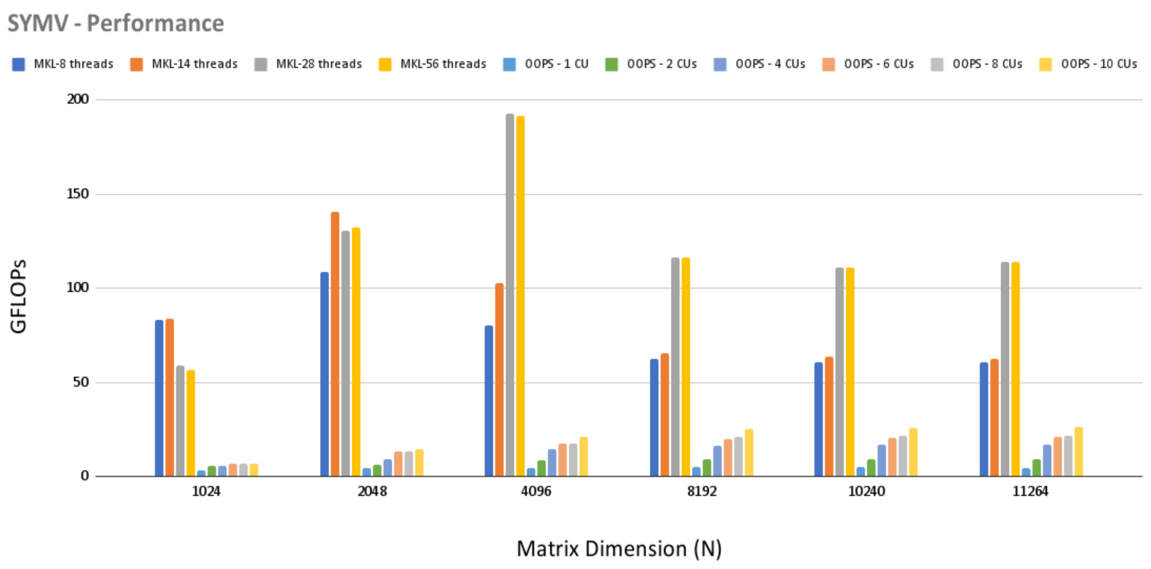


Figure 17: symv performance.

The final version of the SYMV routines features 10 compute units, representing the configuration that achieves optimal performance. As shown in Figure 17, the raw performance is roughly 4.5x lower than that of the MKL library (N=11264), which operates on a 56-thread configuration. This disparity arises from the necessity to reduce the frequency to comply with routing constraints during the generation of the bitstream.

The SYMV kernel poses a higher level of complexity, requiring the storage of all partial results from computing the symmetric part of the initial matrix. Consequently, this demands the allocation of more resources within the FPGA fabric, prompting a reduction in the operating frequency of the kernel. This adjustment is made to accommodate an increase in compute units necessary to align with the available memory ports. Furthermore, the area overhead resulting from the storage of partial results contributes to an increase in the latency of each compute unit, given that the design spans across multiple SLRs to align with routing constraints.

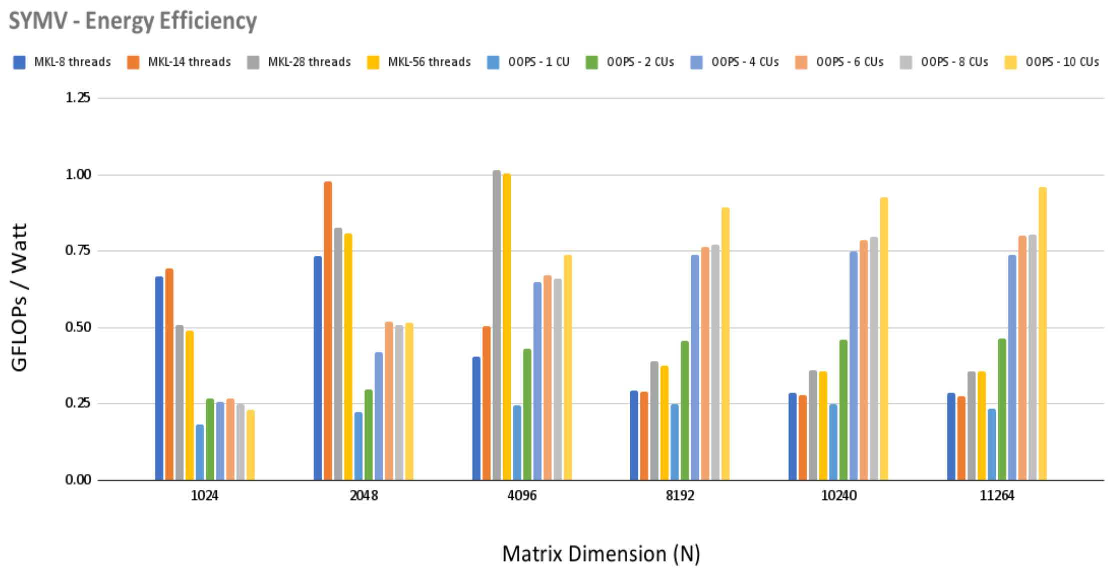


Figure 18: symv energy efficiency.

While our SYMV routine falls behind its MKL counterpart on a server-class CPU, our configuration with 10 compute units showcases a notable performance per watt ratio, approximately 3x better than the 56-thread configuration, as shown in Figure 18. This emphasizes the capability of our design to deliver a superior performance per watt ratio, even though it may exhibit less efficient raw performance. In conjunction with preceding kernels, the server-class CPU remains competitive in terms of energy efficiency primarily when dealing with small matrices, where all values are stored within the caches.

3.4. OOPS_trsv

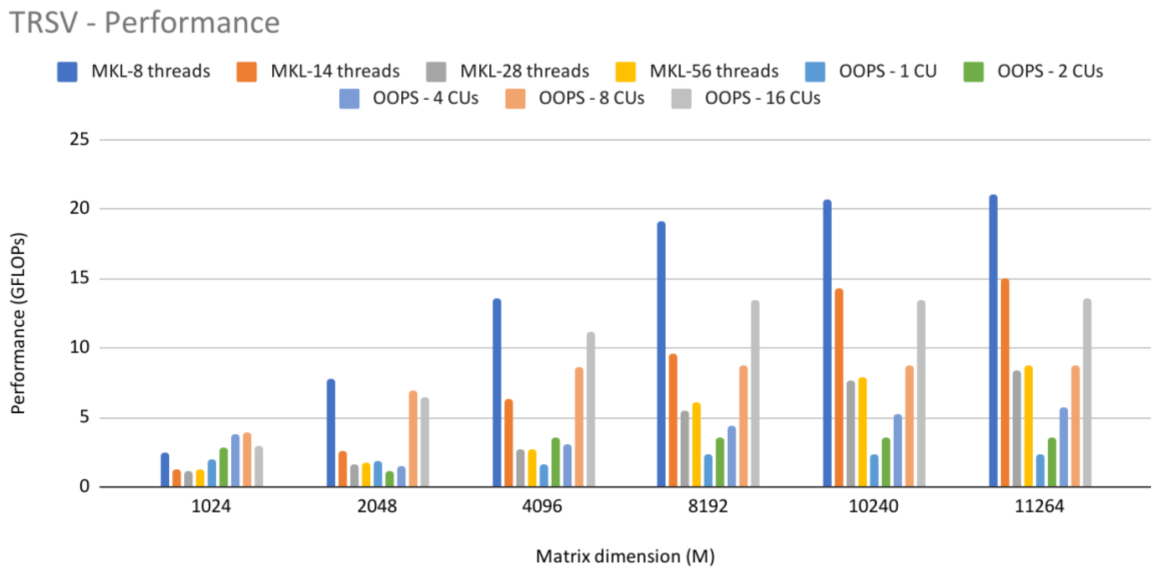


Figure 19: trsv performance.

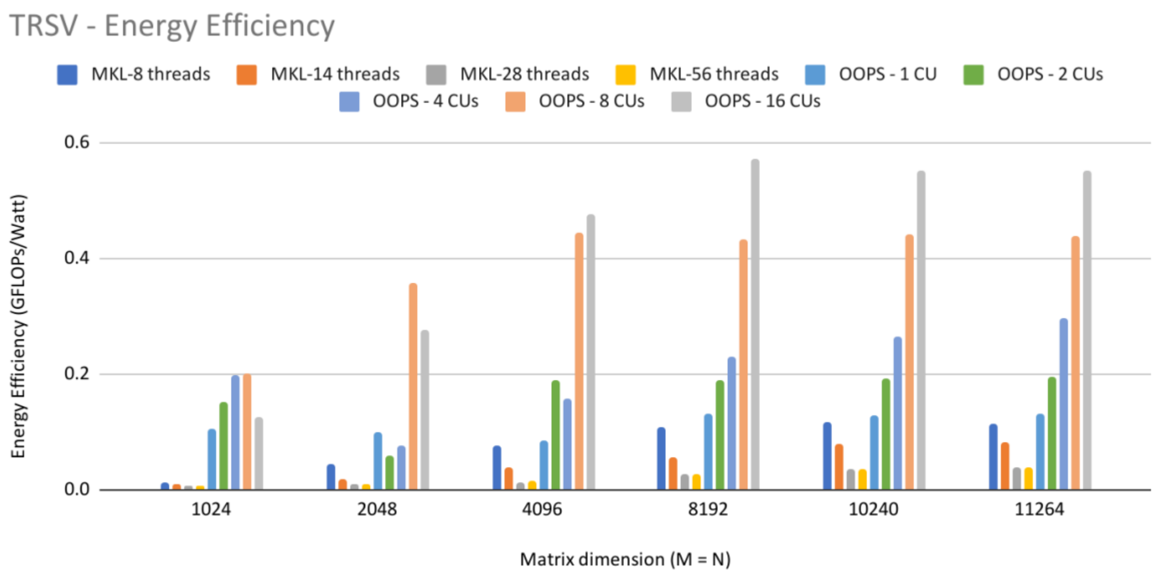


Figure 20: trsv energy efficiency.

The final version of the TRSV kernel features 16 Compute Units. As shown in Figure 19, due to the significant data dependency in the x vector, where each computed value is utilized in subsequent iterations, communication is essential among Compute Units. This communication is facilitated through Broadcast streams, enabling point-to-point forwarding of required x values. Each Compute Unit is tasked with computing a specific segment of the x vector, following load balancing to determine the area of A each Compute Unit will operate on. In comparison to a 56-thread server-class CPU and the Intel MKL library, our kernel performs up to 2x for small matrix sizes (e.g. N=1024) with respect to the MKL function. However, as the matrix size increases (e.g N=11264), the trsv kernel is 0.66x and 2.6x in comparison to 8-thread and 56-thread execution respectively. Moreover, notable gains in energy efficiency are evident, with up to 10x improvements, compared to the CPU, as shown in Figure 20.

3.5. OOPS_gbmv

The final version of GBMV routine features 10 compute units. Banded matrices refer to sparse matrices wherein the non-zero values reside along the diagonals of the original matrix. In our experimental setup, we explore two test cases where the non-zero values constitute 5% and 0.5% of the matrix dimension, respectively. These test cases serve to demonstrate the performance of our kernel in scenarios ranging from a highly sparse matrix to a relatively dense one.

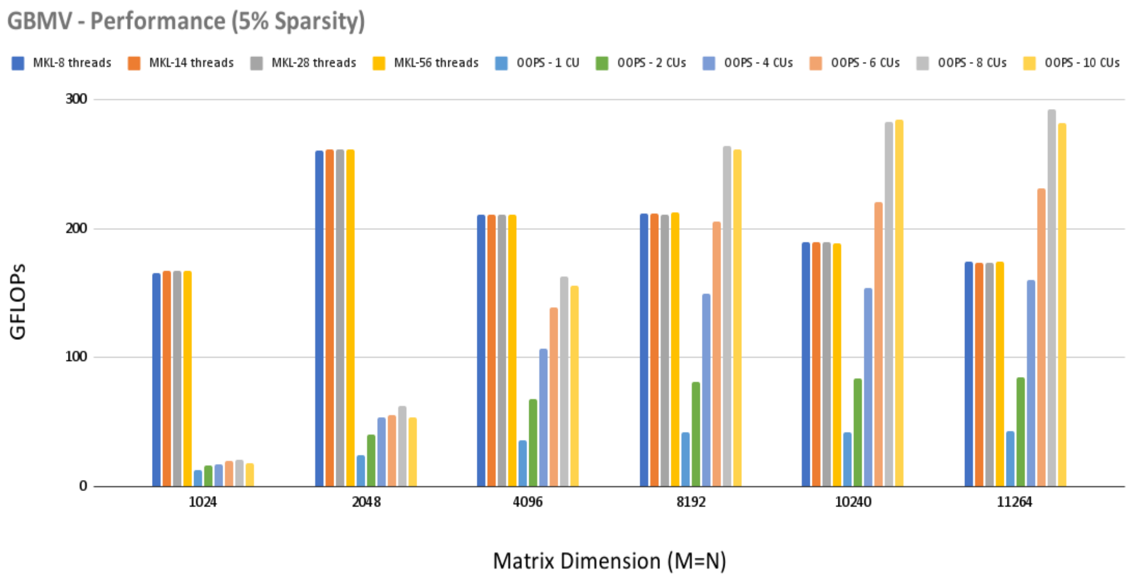


Figure 21: gbmv performance for 5% sparsity.

As shown in Figure 21, when non-zero values constitute 5% of the matrix, GBMV attains peak performance with eight compute units, showcasing a raw performance that is approximately 1.7x more efficient than the MKL library operating with a 56-thread configuration. The raw performance of our kernel experiences a slight decline when scaling to ten compute units for two reasons: a) the frequency is reduced to comply with routing constraints in generating the final bitstream, and b) the sparsity of the matrix, combined with the partitioning of the dataset across all compute units, does not fully leverage the parallel nature of FPGAs. Nevertheless, both configurations demonstrate higher performance compared to a server-class CPU.

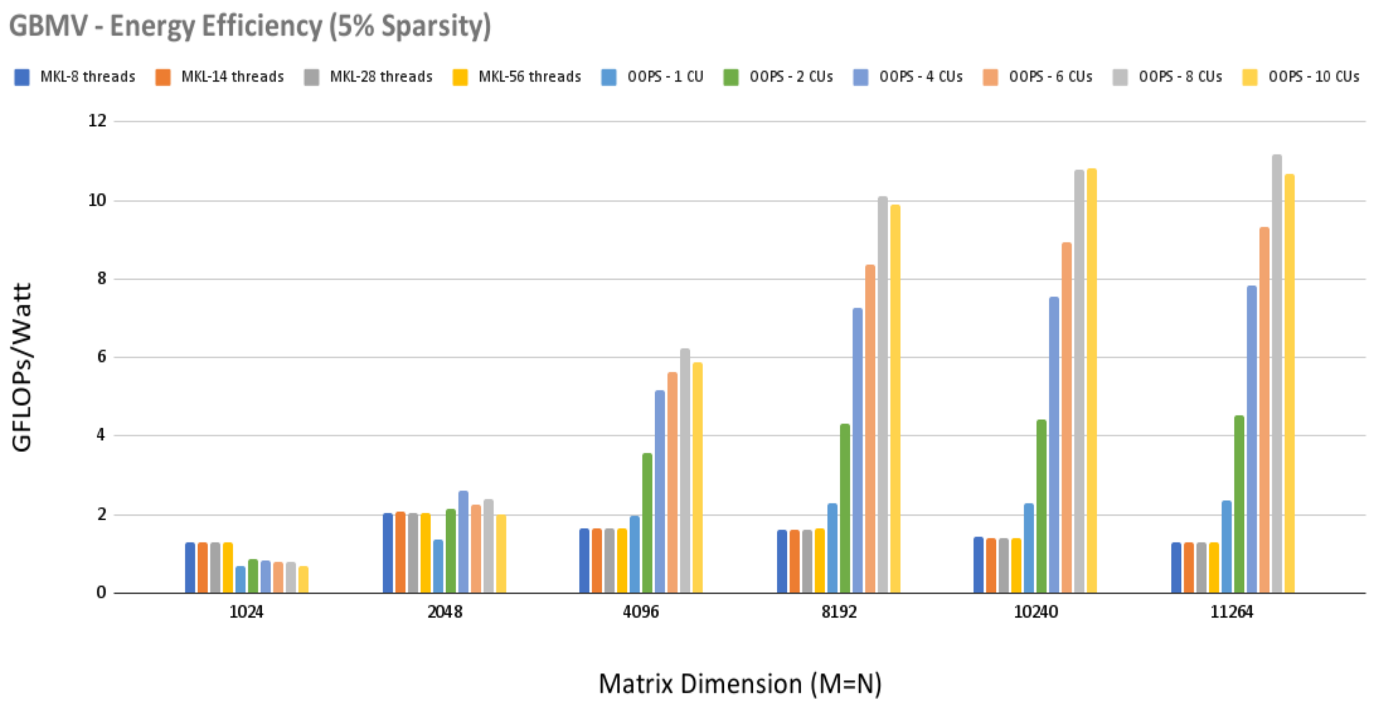


Figure 22: gbmv energy efficiency for 5% sparsity.

As shown in Figure 22, considering that GBMV achieves higher performance comparable to a server-class CPU, our kernel showcases outstanding energy efficiency when compared to the corresponding MKL subroutine. More precisely, our configuration with eight compute units exhibits a performance per watt ratio that is 10x greater. This underscores the potential of FPGA devices as a compelling alternative for executing banded matrices.

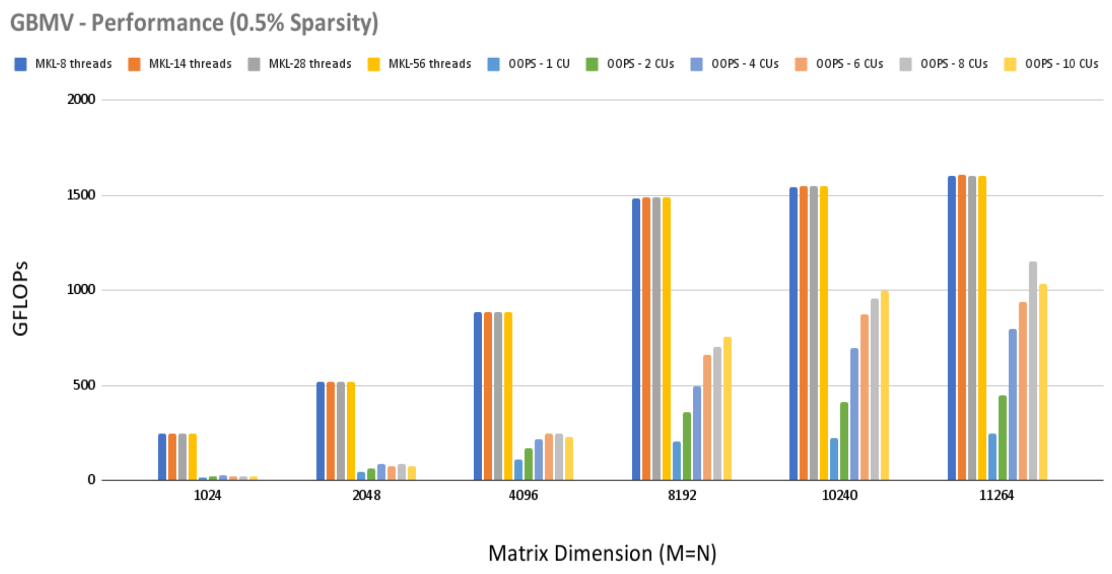


Figure 23: gbmv performance for 0.5% sparsity.

Conversely, when non-zero values account for 0.5% of the initial matrix, GBMV exhibits lower performance compared to its MKL counterpart, as shown in Figure 23. The decline in performance can be attributed primarily to two factors: a) all non-zero values can be accommodated within the

CPU caches, enabling quicker retrieval than DDR memory, and b) the highly sparse nature of the matrix does not effectively capitalize on the parallel and pipelined architecture of our kernel, given the relatively small number of iterations.

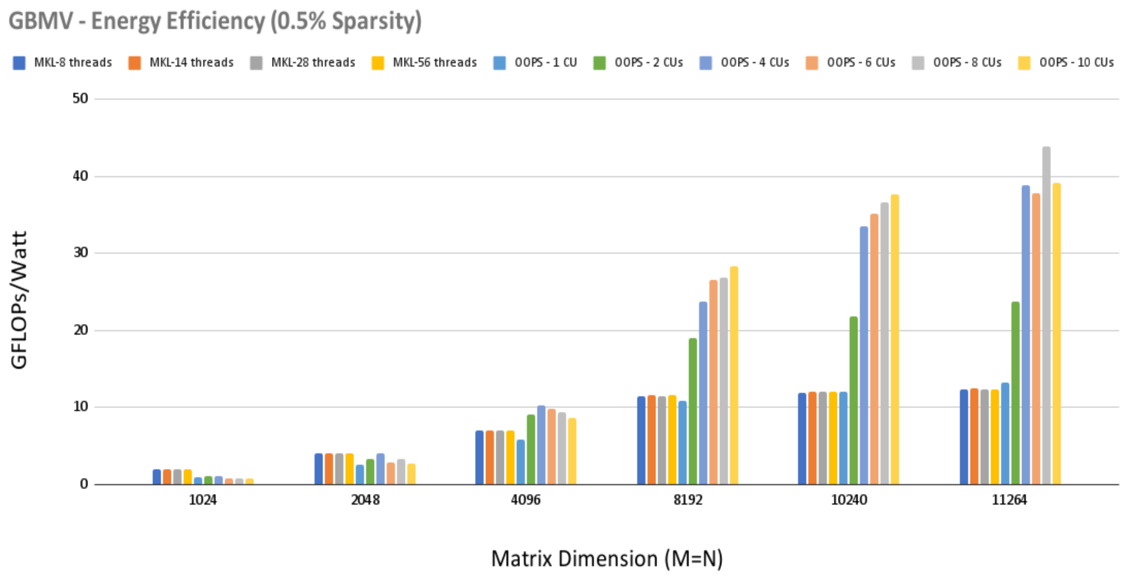


Figure 24: gbmv energy efficiency for 0.5% sparsity.

Nevertheless, our GBMV routine maintains its exceptional energy efficiency relative to the MKL library when executed on a server-class CPU. As shown in Figure 24, our kernel achieves 3.5x better performance per watt ratio, highlighting the ability of our design to deliver competitive performance even in highly sparse matrices, while consuming less power.

3.6. OOPS_tbmv

The final version of the TBMV routine features 14 compute units. As previously detailed, our experimental setup involves two test cases wherein non-zero values account for 5% and 0.5% of the matrix dimension. In this context, these configurations pertain to the number of elements in either the lower or upper part of the triangular matrix, as opposed to the GBMV subroutine.

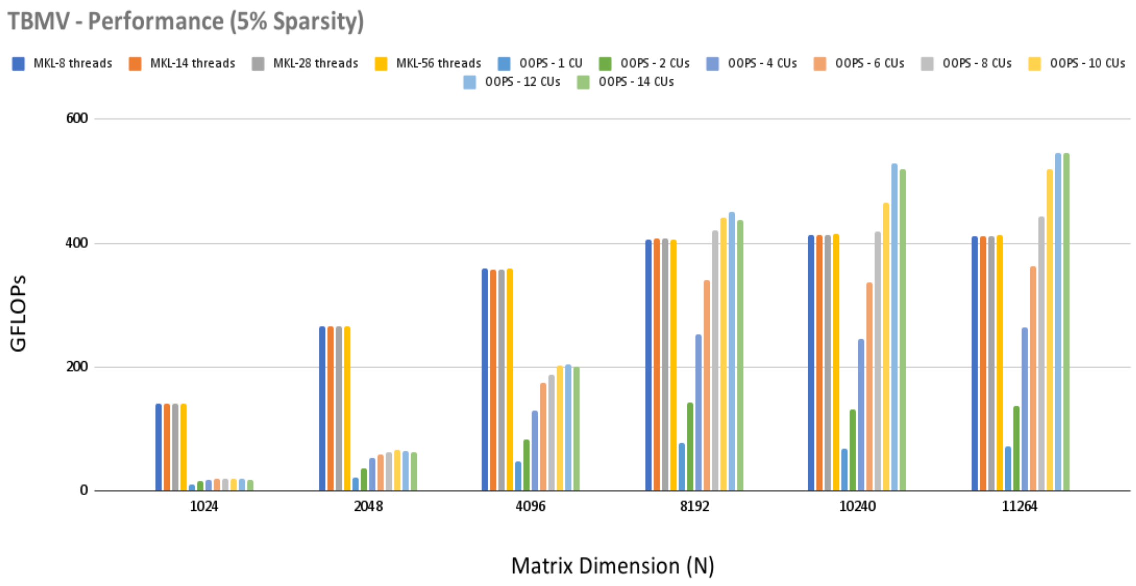


Figure 25: tbmv performance for 5% sparsity.

As shown in Figure 25, when non-zero values account for 5% of the matrix, TBMV achieves peak performance with twelve and fourteen compute units, demonstrating a raw performance that is 1.32x greater than the MKL library operating with a 56-thread configuration. However, our kernel encounters challenges in scaling beyond twelve compute units due to the reduced frequency imposed by routing constraints. Despite this limitation, both configurations demonstrate higher performance compared to the server-class CPU.

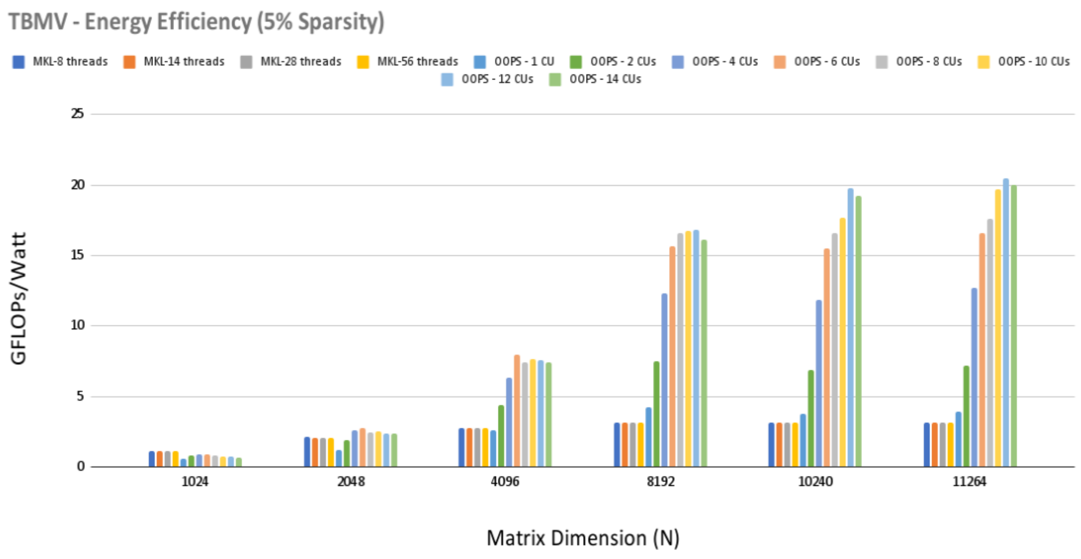


Figure 26 tbmv energy efficiency for 5% sparsity.

Given that TBMV outperforms its MKL counterpart, our design exhibits remarkable energy efficiency, as shown in Figure 26. Specifically, our configuration with eight compute units demonstrates a 6.5x greater performance-per-watt ratio, highlighting the potential of FPGAs to efficiently handle both triangular and general banded matrices with significantly reduced power consumption.

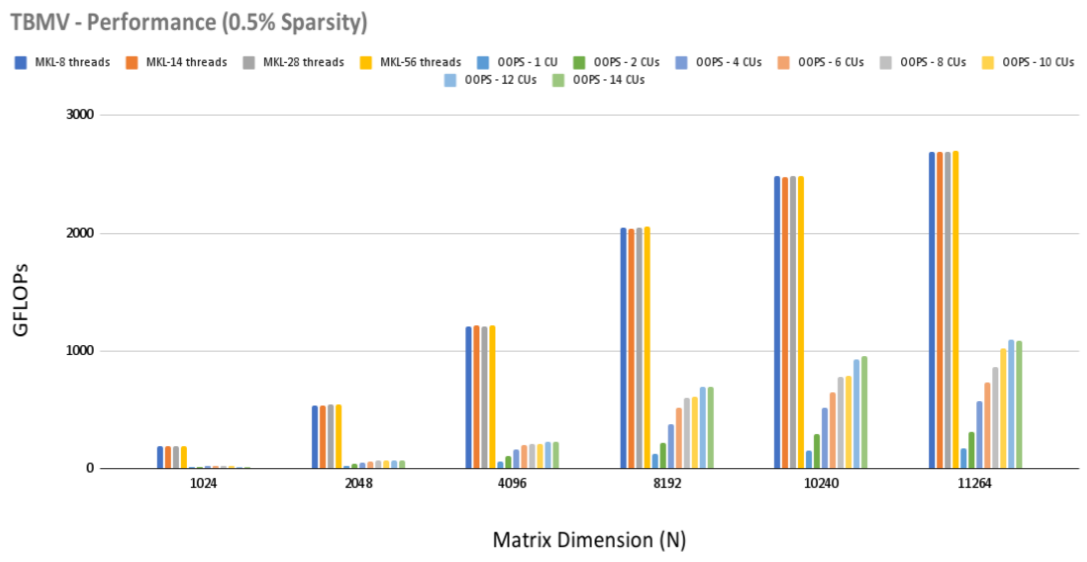


Figure 27¨ tbmv performance for 0.5% sparsity.

On the other hand, when non-zero values represent 0.5% of the triangular matrix, TBMV displays similar behavior to GBMV, as shown in Figure 27. Our design demonstrates inferior performance compared to its MKL counterpart, primarily due to two factors: a) the highly sparse nature of the matrix allows the non-zero values to be accommodated within the caches, and b) the dataset cannot effectively exploit the parallel and pipelined architecture of our kernel.

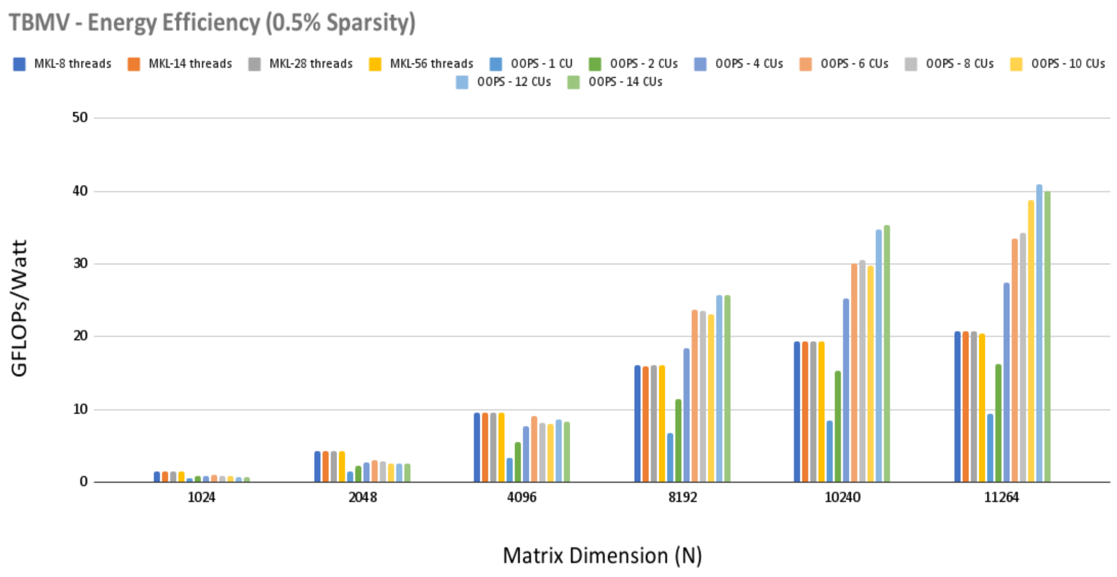


Figure 28: tbmv energy efficiency for 0.5% sparsity.

Even though highly sparse matrices do not fully exploit the capabilities of our kernel, our design still yields a higher performance-per-watt ratio. Specifically, as shown in Figure 28, our configuration with eight compute units demonstrates a 2x increase in energy efficiency, showcasing the ability of our kernel to deliver great performance with substantially less power consumption.

3.7. OOPS_sbmv

The final version of the SBMV routine features 10 compute units. As previously detailed, our experimental setup involves two test cases wherein non-zero values account for 5% and 0.5% of the matrix dimension. In this context, these configurations pertain to the number of elements in either the lower or upper part of the triangular matrix, as opposed to the GBMV subroutine.

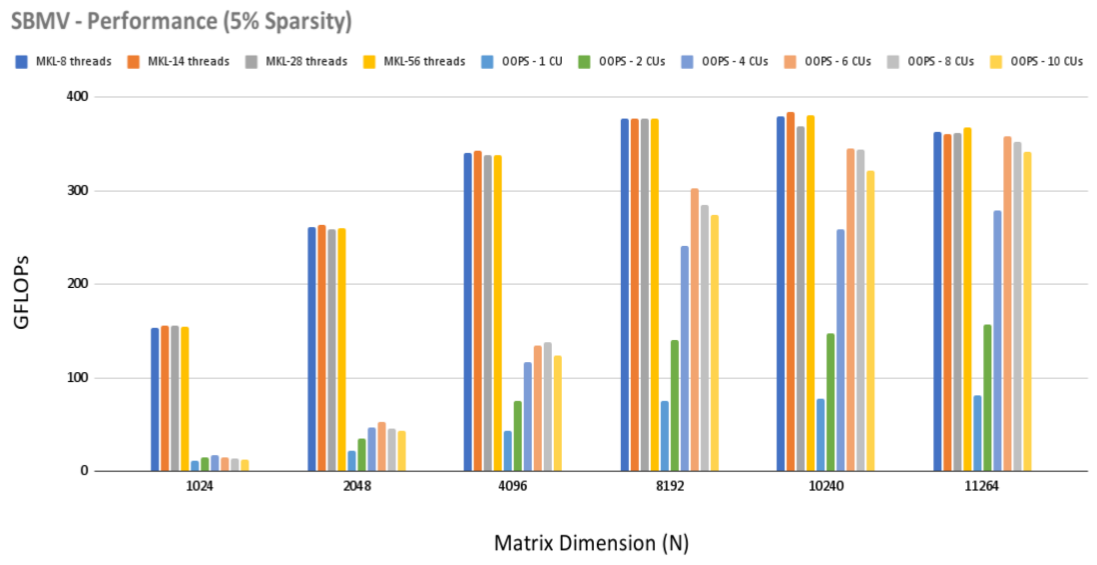


Figure 29: sbmv performance for 5% sparsity.

When non-zero values account for 5% of the matrix, SBMV achieves its optimal performance with 6 compute units, as shown in Figure 29. Our design demonstrates competitive performance when compared to the corresponding MKL subroutine, which operates in a 56-thread configuration. However, our kernel encounters limitations in scalability when additional compute units are instantiated. The decline in performance can be attributed to the necessity of storing all partial results generated during the computation of the symmetric part of the matrix. Akin to SYMV, the storage of partial results contributed to increased latency for each compute unit, as the design spans to multiple SLRs, to align with the resource requirements.

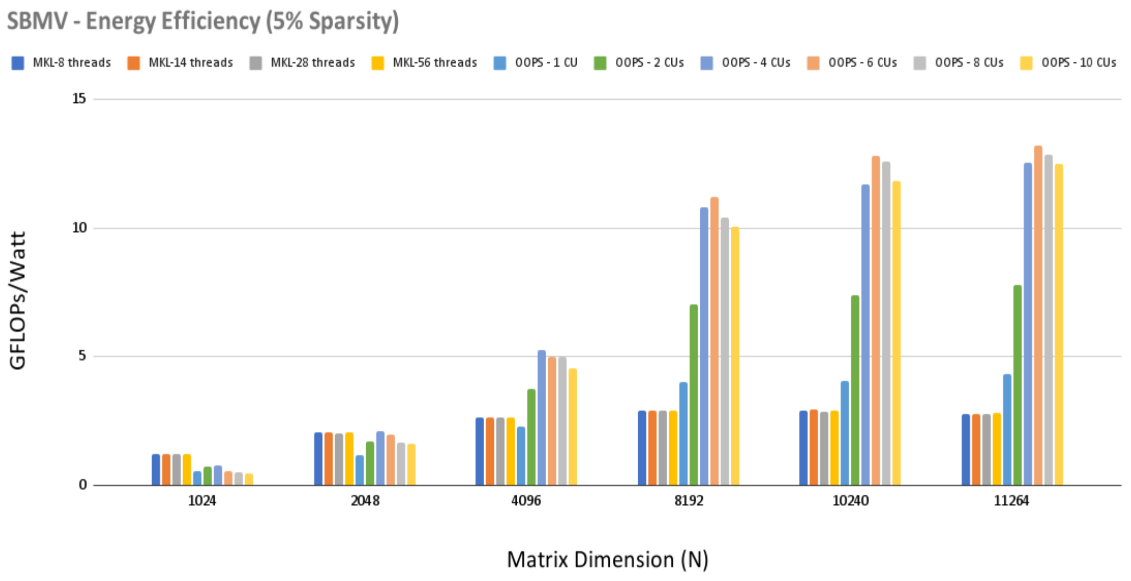


Figure 30: sbmv energy efficiency for 5% sparsity.

Given that SBMV provides competitive performance with its MKL counterpart, our design exhibits remarkable energy efficiency. Specifically, as shown in Figure 30, our configuration with six compute units demonstrates a 4.7x greater performance-per-watt ratio, highlighting the potential of FPGAs to efficiently handle both symmetric banded matrices with significantly reduced power consumption.

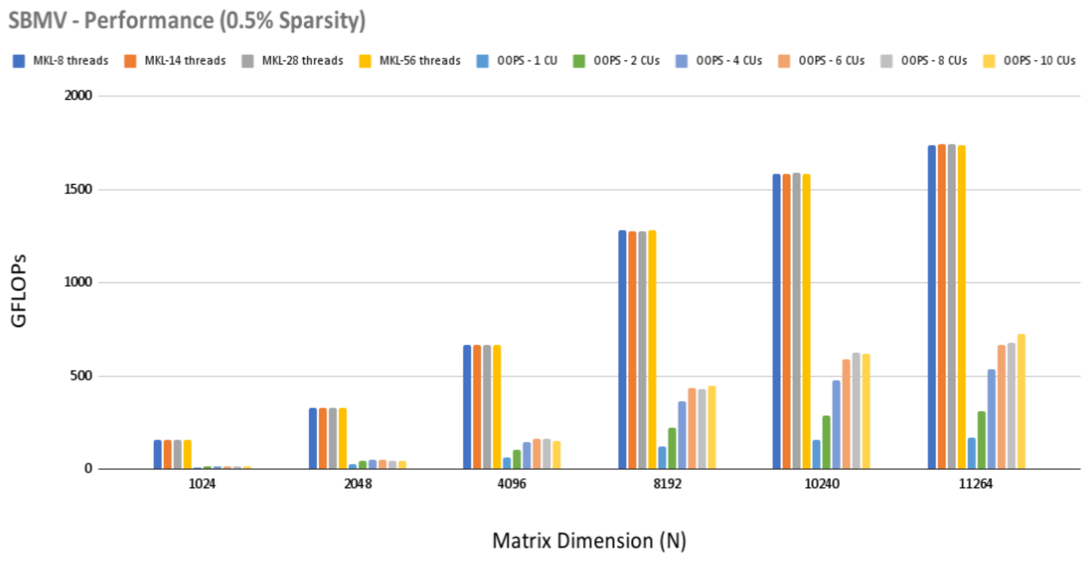


Figure 31: sbmv performance for 0.5% sparsity.

However, our design encounters challenges in achieving competitive performance when non-zero values comprise 5% of either the lower or upper triangular matrix, as shown in Figure 31. This observation is consistent with both TBMV and GBMV, underscoring the inefficiency of FPGAs to keep up with server-class CPUs in highly sparse matrices.

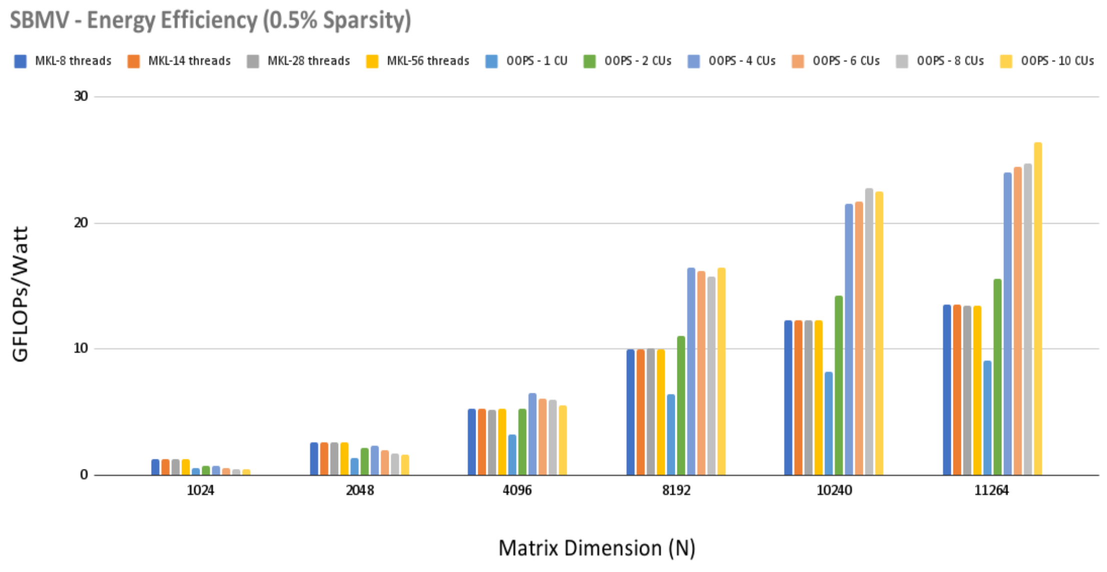


Figure 32: sbmv energy efficiency for 0.5% sparsity.

Nevertheless, our SBMV kernel yields a higher performance-per-watt ratio even in highly sparse matrices. More precisely, our configuration with ten compute units demonstrates 2x increased energy efficiency, showcasing the ability of our kernel to process symmetric banded matrices with substantially less power consumption, as shown in Figure 32.

3.8. OOPS_tbsv

The final version of our TBSV routine features 16 compute units. As previously detailed, our experimental setup involves two test cases where non-zero values account for 5% and 0.5% of the matrix dimension. In this context, these configurations pertain to the number of elements in either the lower or upper part of the triangular matrix.

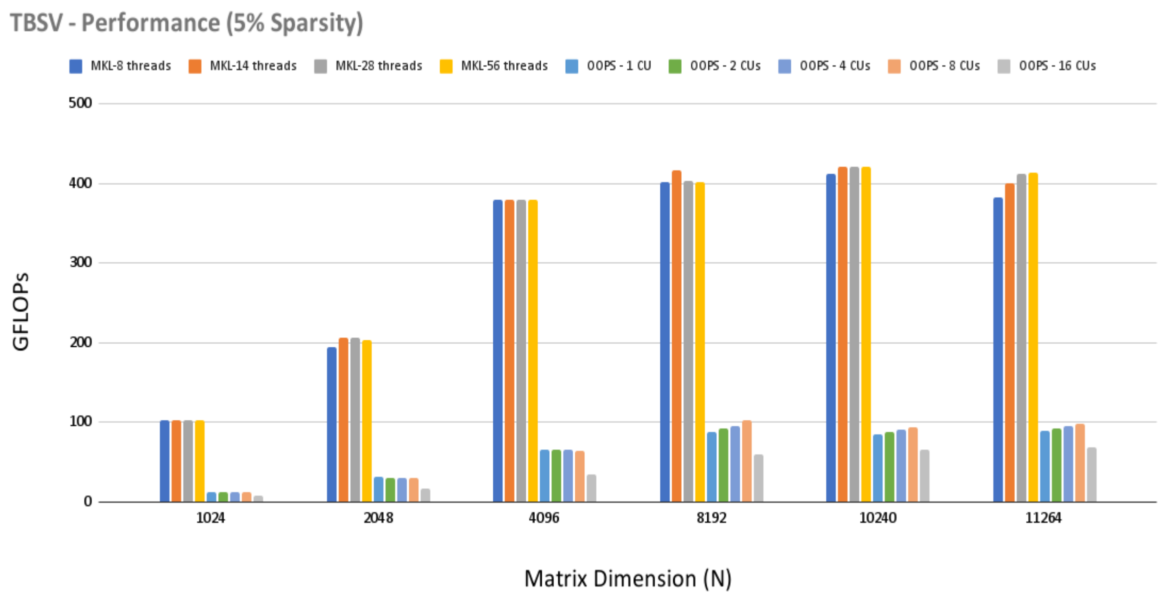


Figure 33: tbsv performance for 5% sparsity.

In comparison to previous implementations, TBSV exhibits poor performance, even when non-zero values constitute 5% of the triangular matrix. Moreover, the kernel falls short of achieving sufficient scalability, as shown in Figure 33. Notably, a configuration featuring sixteen compute units demonstrates performance decline compared to configuration with eight of four compute units. This decline can be attributed to three key factors: a) TBSV exhibits strong data dependency between subsequent iterations, hindering the ability to achieve high parallel in compute the values of the X vector; b) The performance is further hindered by data transfers between compute units, necessitated by the transmission of results from initial compute units to subsequent ones for compute successive values of the X vector; c) The variable K significantly influences the coverage between compute units, indicating the level of coarse-grain parallelism achievable by our kernel. In our experiments K represents the 5% of the triangular matrix, limiting the ability of compute units to operate in parallel. Consequently, only a fraction (K%) of iteration can be executed concurrently at any given time, resulting in poor scalability for our kernel.

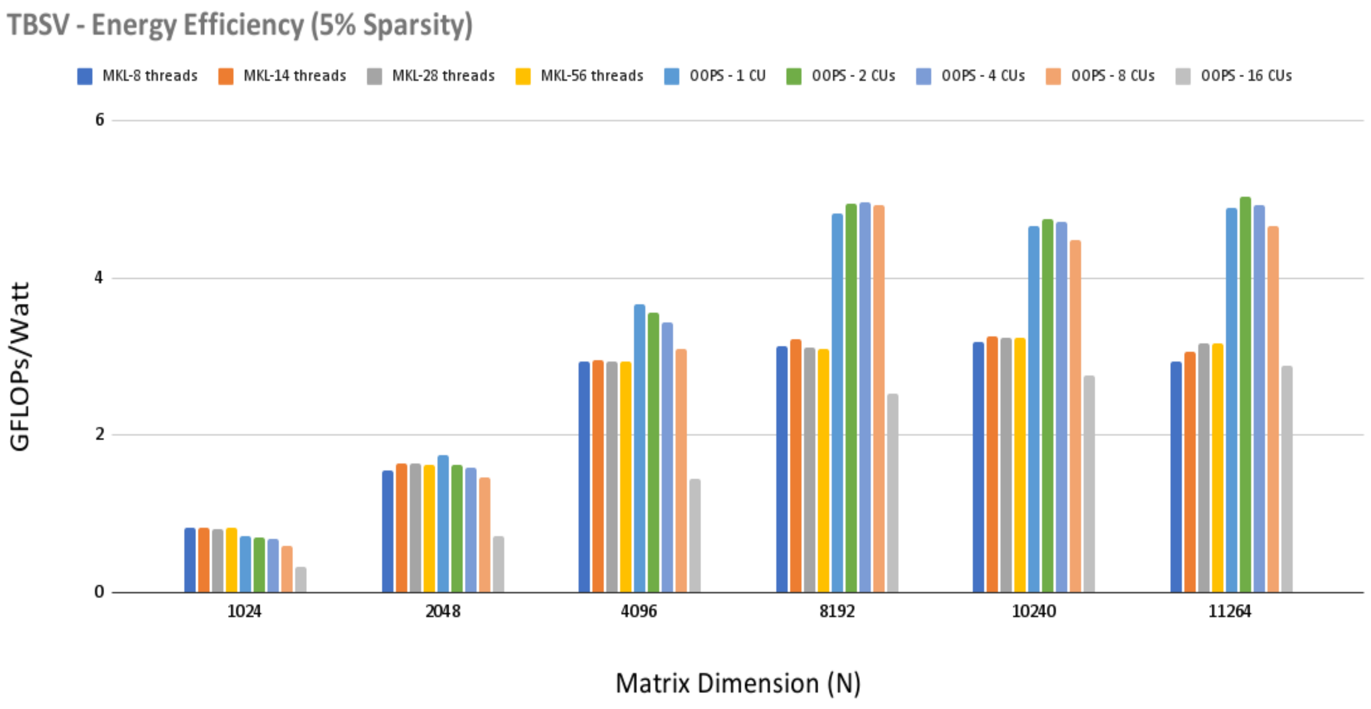


Figure 34: tbsv energy efficiency for 5% sparsity.

Despite the reduced performance of our TBSV kernel compared to its MKL counterpart, it achieves higher performance-per-watt ratio in all tested configurations, as shown in Figure 34. Our designed kernel delivers 1.58x increased energy efficiency, showcasing its ability to operate solver algorithms using banded matrices with significantly reduced power consumption. Additionally, our results confirm the limited scalability of the TBSV design, as energy efficiency diminishes by instantiating additional compute units. This decline is attributed to the increased power consumption from the increased FPGA area consumption, without a corresponding increase in performance.

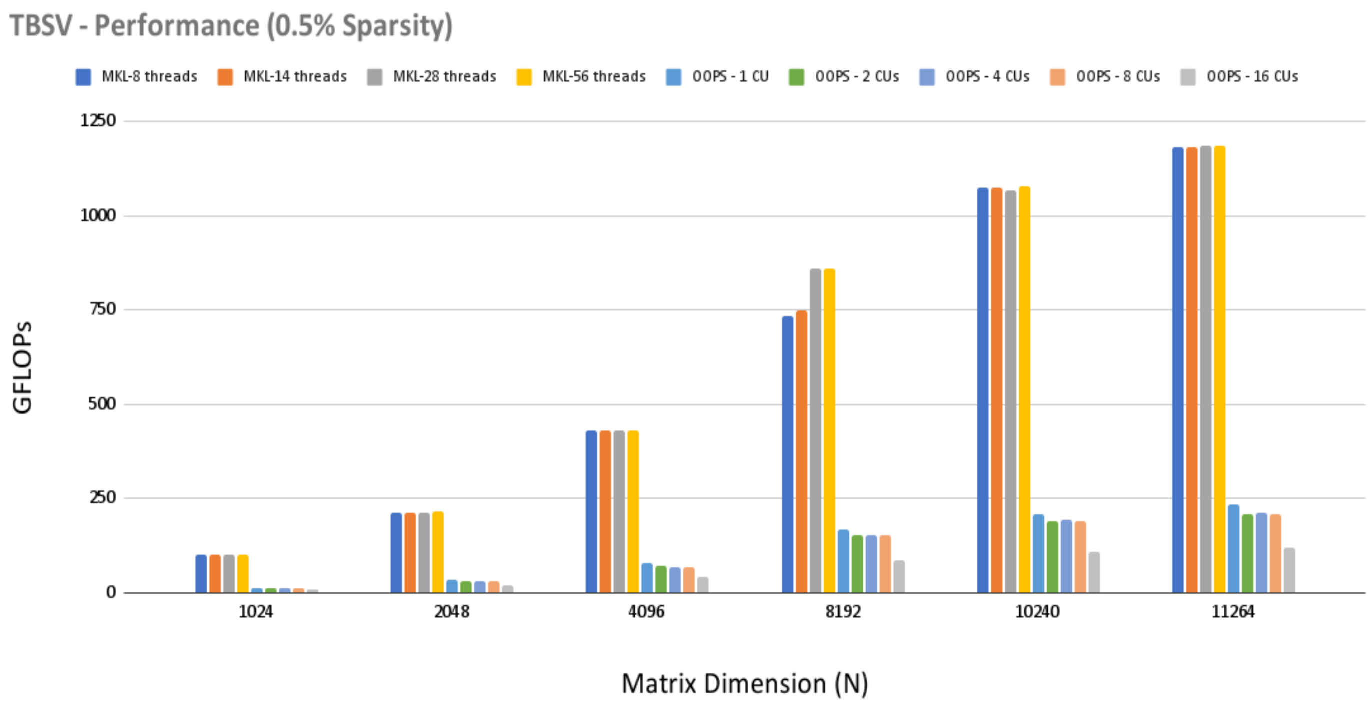


Figure 35: tbsv performance for 0.5% sparsity.

When the non-zero elements account for 0.5% of the triangular matrix, TBSV demonstrates reduced performance compared to the previous configuration, as shown in Figure 35. In this experiment, K represents only the 0.5% of the triangular matrix, constraining even more the ability of compute units to operate in parallel. Our results confirm the suboptimal performance of FPGAs when confronted with data dependencies among successive iteration and low levels of parallelism.

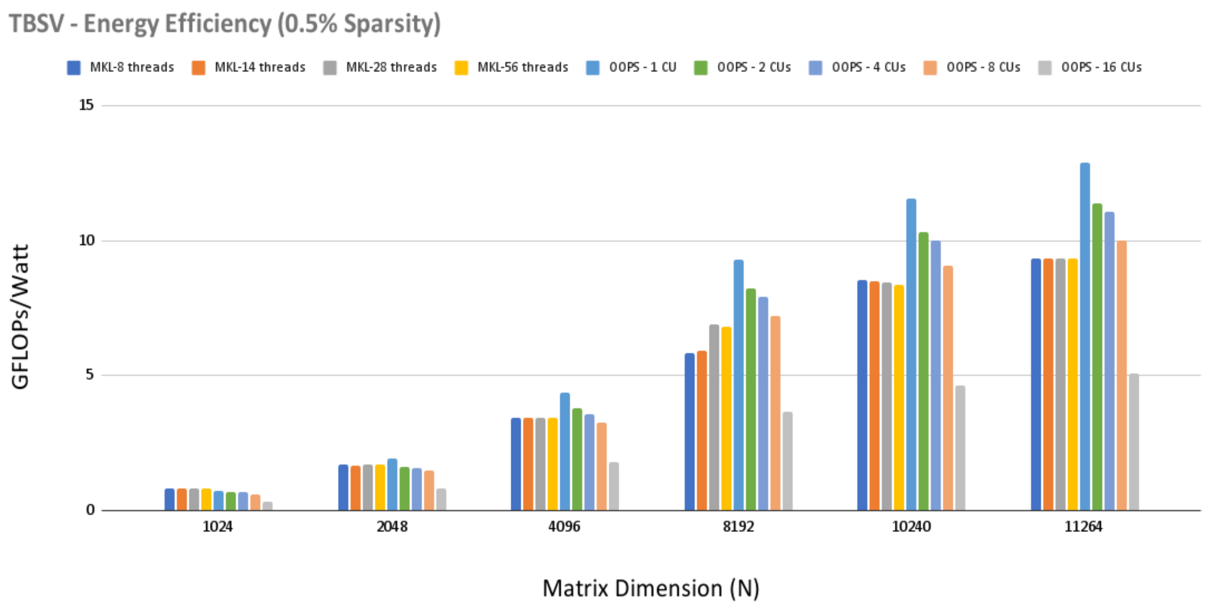


Figure 36: tbsv energy efficiency for 0.5% sparsity.

Similar trends are noted in the experiments when K is set at 0.5% of the original triangular matrix, as shown in Figure 36. The kernel exhibits a modest improvement in energy efficiency across all examined configurations, compared to the MKL library in a server-class CPU. Furthermore, the performance-per-watt ratio experiences a decline with the addition of extra compute units, attributed to the limited scalability of the kernel.

3.9. OOPS_tpmv

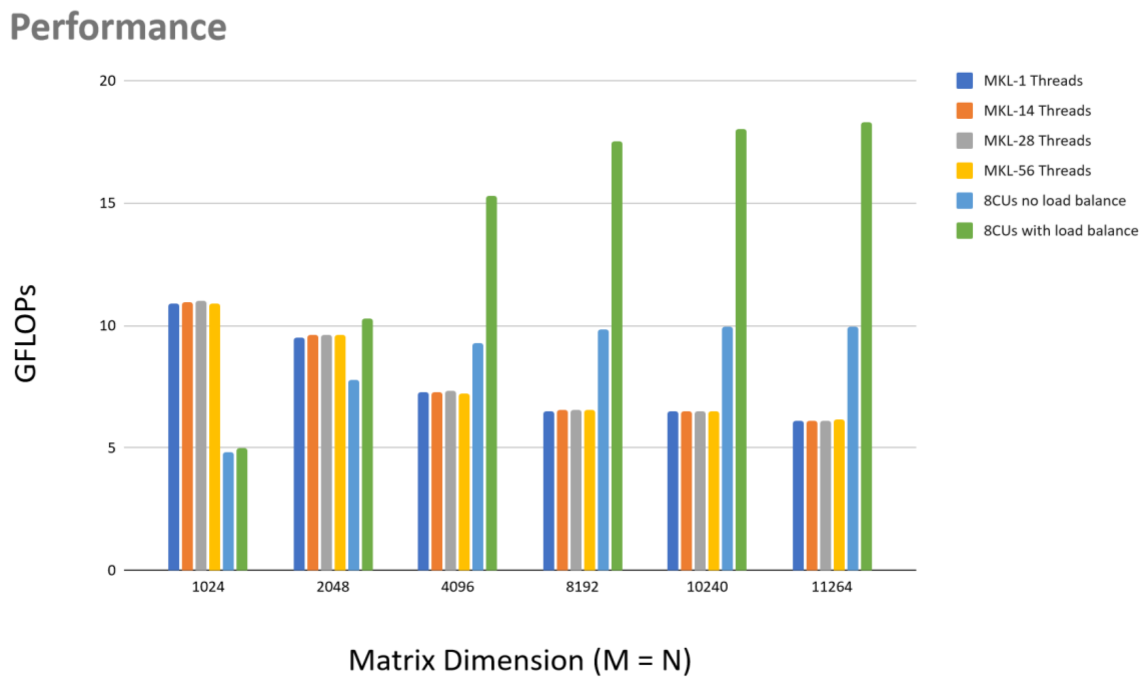


Figure 37: tpmv performance.

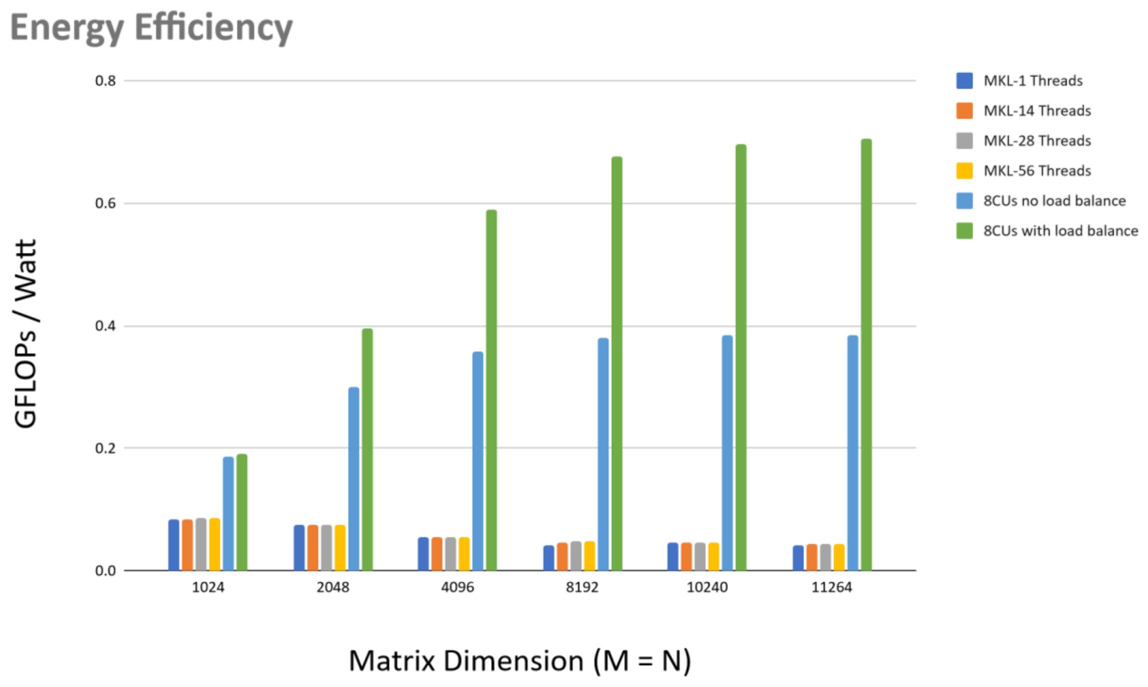


Figure 38: tpmv energy efficiency.

Figures 37 and 38 above provide the performance and energy efficiency of tpmv (with 8 internal CUs) when compared to the MKL software function from 1 to 56 threads. Tests included with and without load balancing, i.e. whether elements from the input matrix are equally distributed to the available CUs or not respectively. In the former case, the load balancer distributes equally the input matrix elements to the available CUs, whereas in the latter case, each CU processes (almost) the same number of rows. However, due the fact that the input matrix is triangular, the first CU processes less data compared to the last one.

As shown, the OOPS kernel can be up to 1.4x and almost 3x faster for large matrix sizes (N=11264) compared to the MKL 56 thread software implementation, without and with load balancer respectively. Moreover, due the FPGA's low power consumption, the kernel energy efficiency without and with load balancing can be up to 6.7x and 12x improved, compared to the optimized software version.

3.10. OOPS_spmv Symmetric Packed Matrix-Vector) [1]

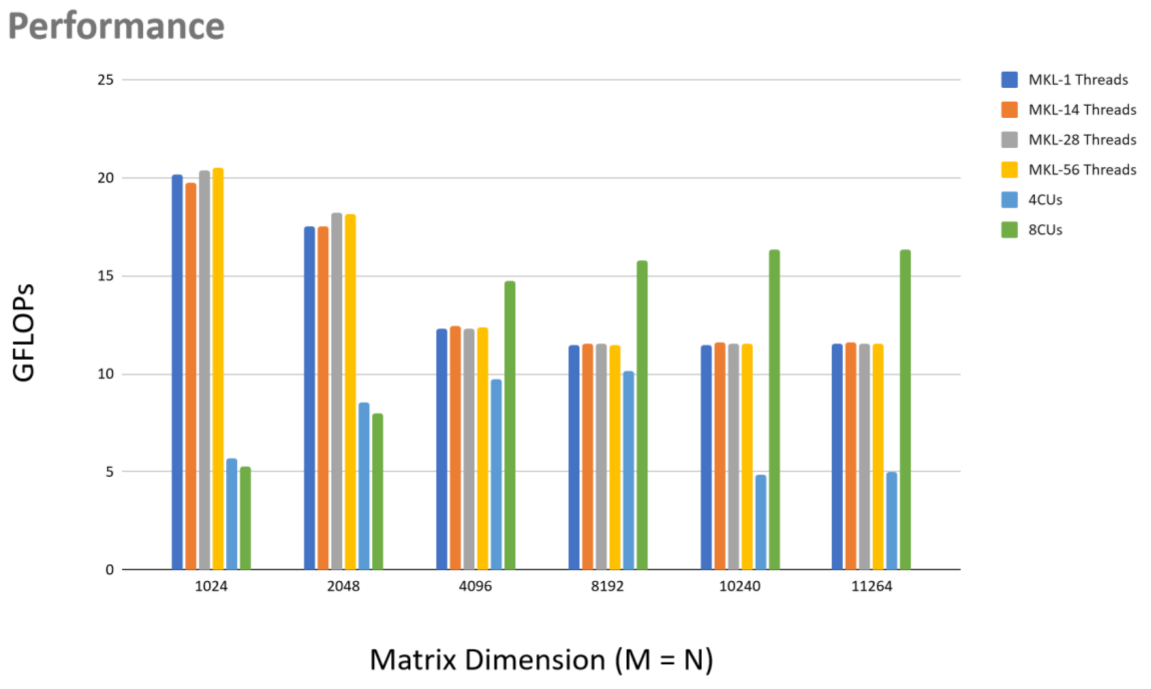


Figure 39: spmv performance.

[1] Symmetric Packed Matrix-Vector as per BLAS terminology. Overloaded term with SpMV for Sparse Matrix Vector multiplication.

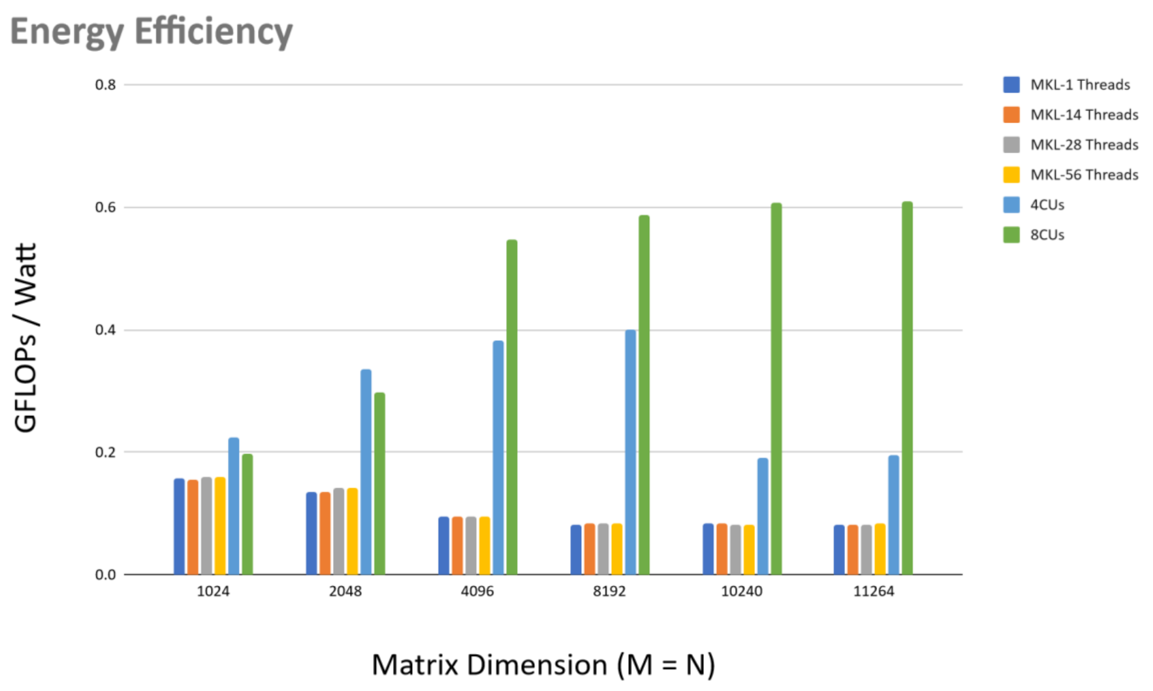


Figure 40: spmv energy efficiency.

Figures 39 and 40 provide the performance and energy efficiency of spmv (with 4 and 8 internal CUs) when compared to the MKL software function from 1 to 56 threads. As shown, the OOPS kernel can be up to 1.3x faster compared to the MKL 56 thread software implementation, for large matrix sizes (N=11264). Moreover, due the FPGA's low power consumption, the kernel energy efficiency is up to 6x better compared to the optimized software version.

3.11. OOPS_tpsv

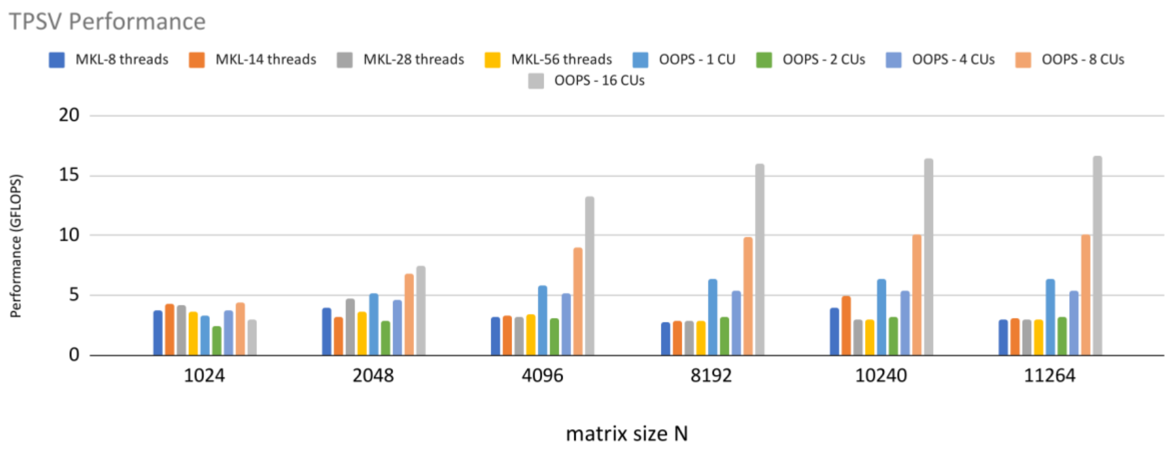


Figure 41: tpsv performance.

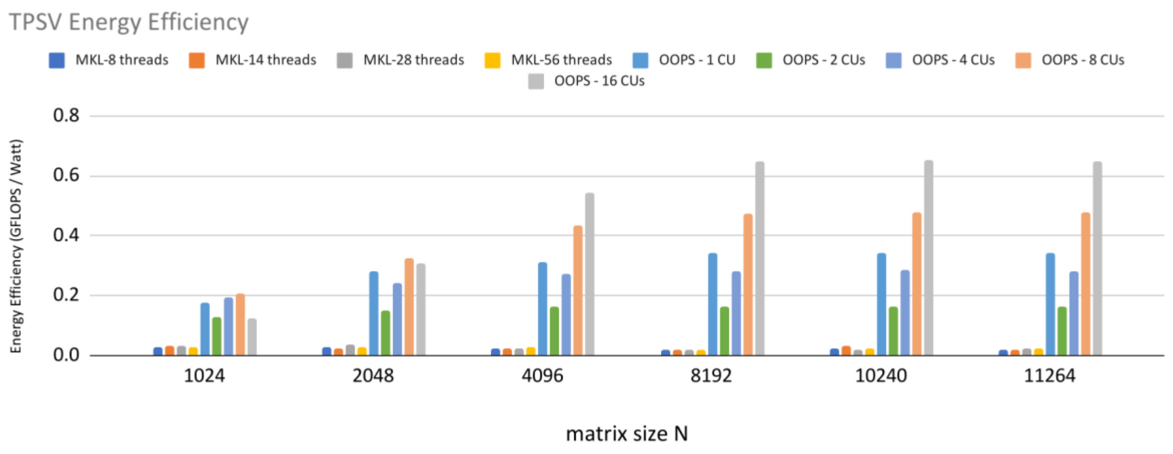


Figure 42: tpsv energy efficiency.

Figures 41 and 42 provide the performance and energy efficiency of tpsv, ranging from 1 to 16 CUs, when compared to the MKL software function from 1 to 56 threads. As shown, lower number of MKL threads provides better performance, compared to more threads, such as 56. As such, the MKL 8 thread software implementation is up to 30% faster compared to the 16 CUs, for small matrix sizes (N=1024). However, for larger sizes, the OOPS kernel with 16 CUs can be up to 3x faster compared to the software version with 56 threads. Moreover, due the FPGA's low power consumption, the kernel energy efficiency is up to 2.8x better compared to the optimized software version.

4. Level 3 BLAS routines of the OOPS library

4.1. OOPS_gemm

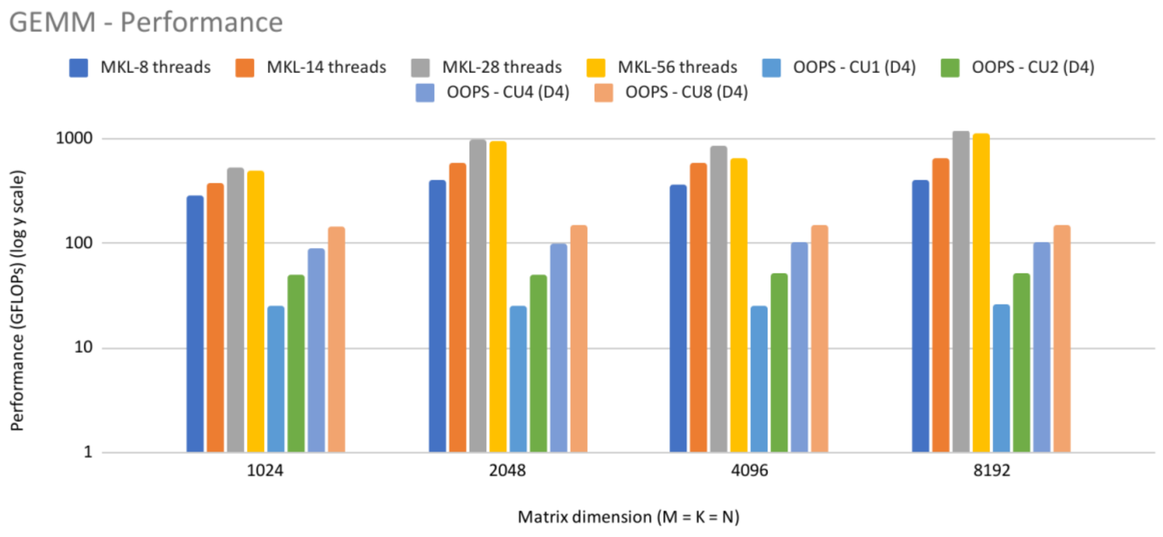


Figure 43: gemm performance

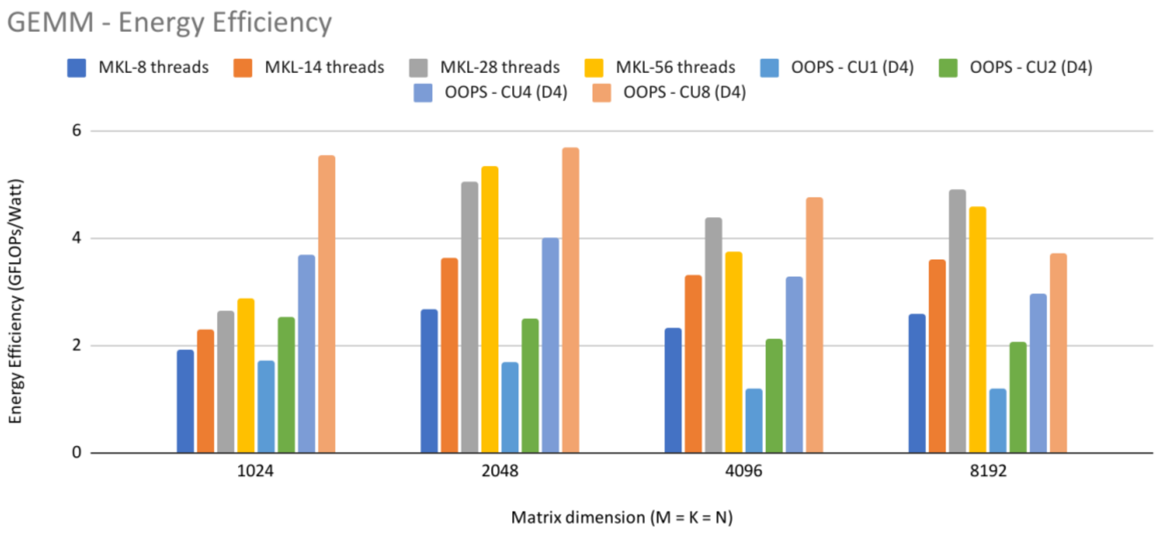


Figure 44: gemm energy efficiency

Figures 43 and 44 show the gemm performance and energy efficiency. The final version of the gemm kernel is configured with eight independent Compute Units, each utilizing a systolic design characterized by a depth factor denoted as D. Each Compute Unit is assigned with a segment of matrices A and C, and an individual copy of matrix B, enabling their independent operation. The systolic design allows for reuse of elements of matrix A, with each level of the systolic design employing a different computation sub-kernel. Optimal performance is achieved when D attains a value of 4, showcasing 5.5x lower performance relative to the Intel MKL library running on a 56-thread server-class system. On the other hand, the FPGA's low operational power leads to higher energy efficiency across nearly all matrix sizes.

4.2. OOPS_trmm

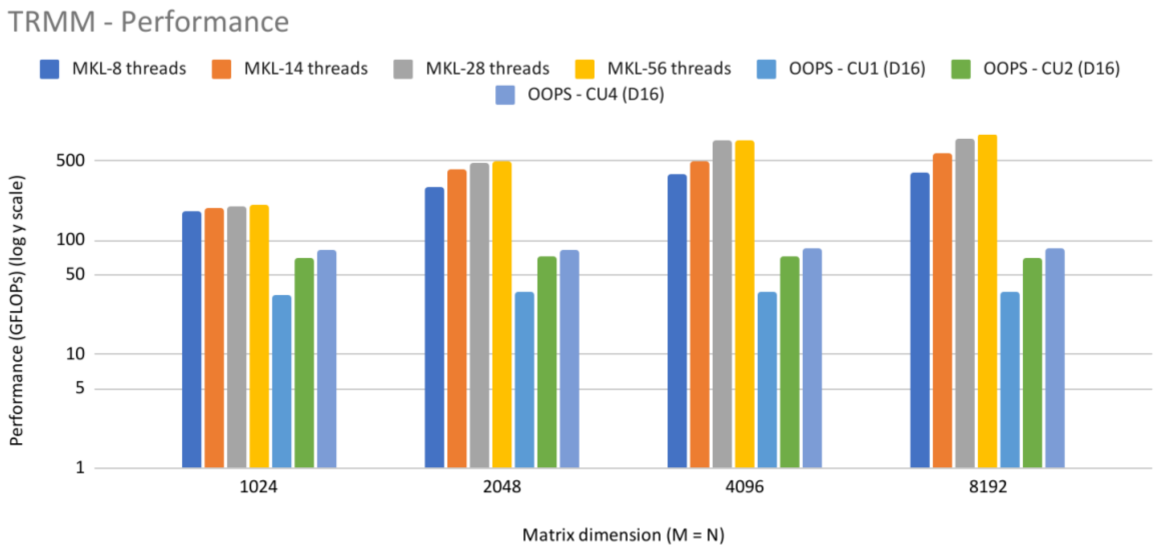


Figure 45: trmm performance.

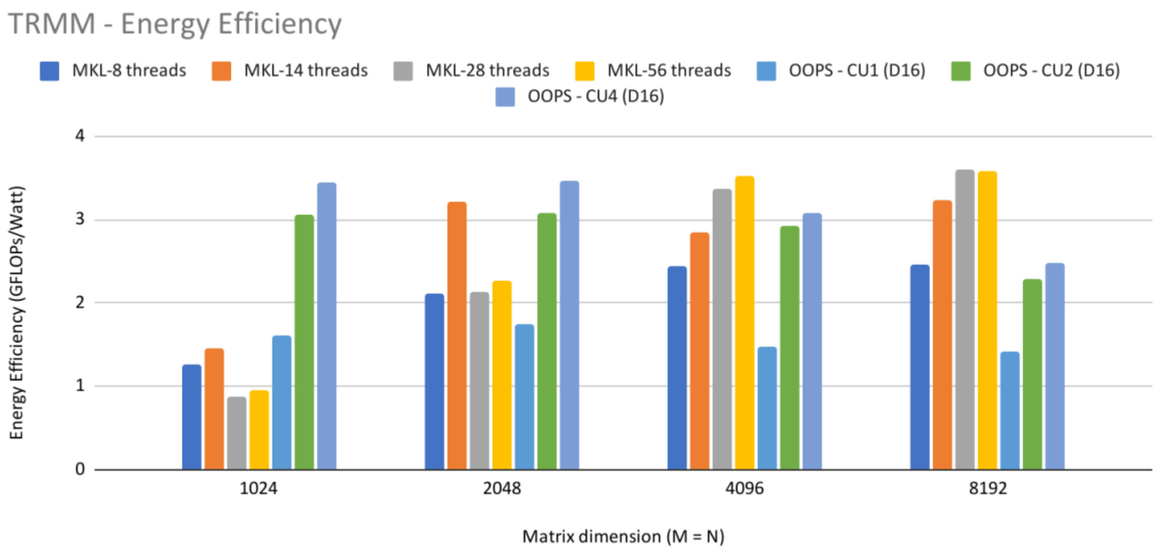


Figure 46: trmm energy efficiency.

Figures 45 and 46 show the trmm performance and energy efficiency. The final version of the trmm kernel is configured with four Compute Units, each employing a D-level systolic design, the same as in gemm kernel. Each Compute Unit is assigned an individual copy of matrix A and a segment of matrix B (input and output), enabling their independent operation. Optimal performance is achieved when D attains a value of 16, showcasing a 6.6x lower performance relative to the Intel MKL library running on a 56-thread system. The observed performance decline arises from the necessity to pad the triangular matrix A with zeros during FPGA reading, ensuring the seamless execution of the systolic design. As previously noted, the FPGA's low operational power leads to comparable energy efficiency with the CPU implementation.

4.3. OOPS_symm

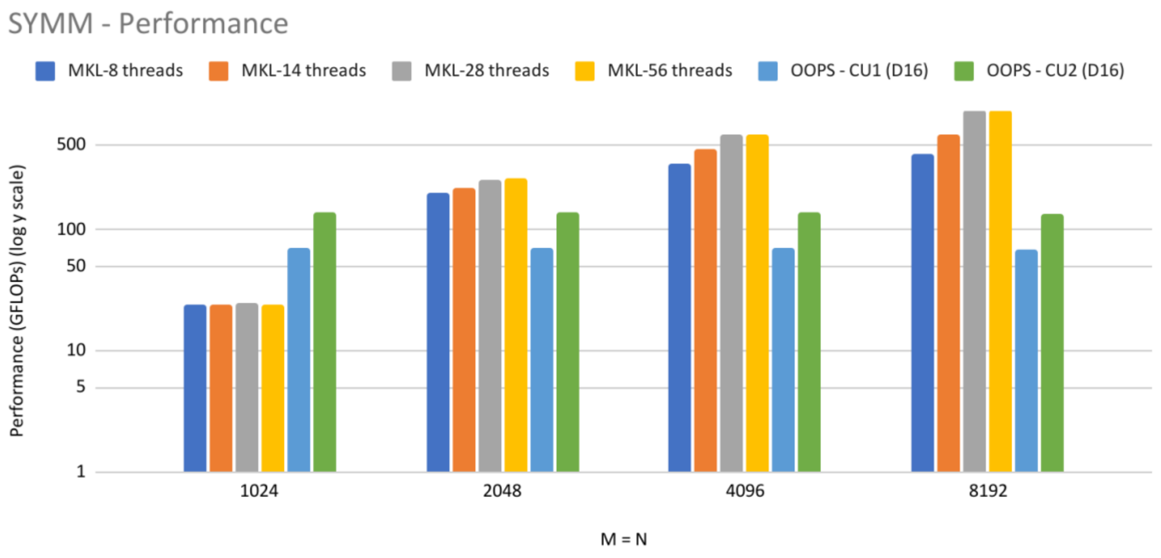


Figure 47: symm performance.

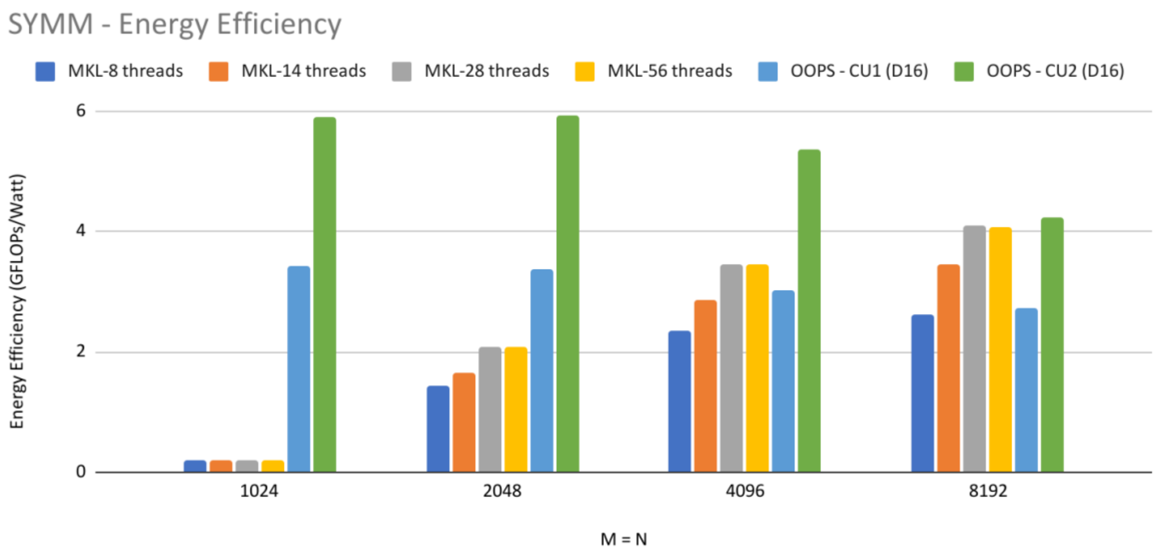


Figure 48: symm energy efficiency.

Figures 47 and 48 show the symm performance and energy efficiency. The final version of the SYMM kernel is configured with two Compute Units, each employing a systolic design characterized by a depth factor, denoted as D, similar to the gemm kernel. Each Compute Unit is assigned an individual copy of matrix A and a segment of matrices B and C (input and output), enabling their independent operation. Optimal performance is achieved when D attains a value of 16. Due to the intricate nature of the 16-level systolic design, additional Compute Units could not be synthesized while meeting timing constraints. In terms of performance, it is observed to be 1/3 compared to the Intel MKL library executing on a 56-thread system. Conversely, the energy efficiency metric for this kernel demonstrates an 8.9x improvement relative to the specific CPU, despite operating slower than the CPU.

4.4. OOPS_trsm

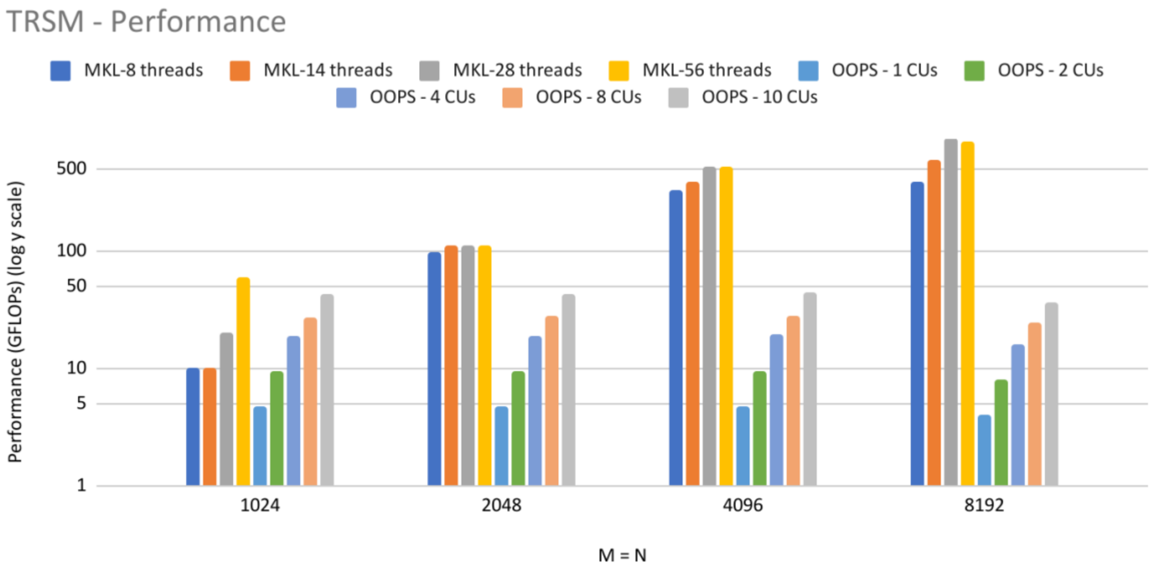


Figure 49: trsm performance.

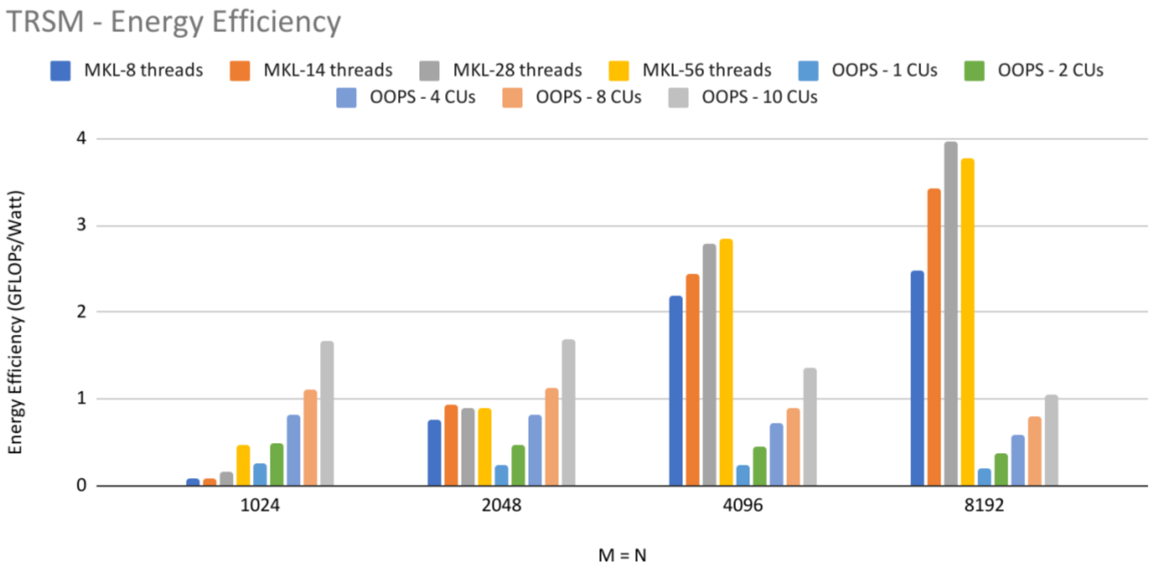


Figure 50: trsm energy efficiency.

Figures 49 and 50 show the trsm performance and energy efficiency. The final version of the trsm kernel is configured with ten Compute Units, with each Compute Unit being assigned an individual copy of matrix A and a segment of matrices B and X, enabling their independent operation. Additionally, each Compute Unit employs two internal computation kernels to achieve workload balance, as presented for the trsv kernel. While using more internal computation kernels results in a more complex design operating at a slower frequency without any significant performance benefit, two kernels are found to be sufficient for the trsm kernel. This kernel demonstrates competitive performance only for small matrix sizes compared to the Intel MKL library running on a 56-thread server-class system. However, the FPGA's low operational power provides a 1.5x improvement in energy efficiency on average compared to the CPU.

5. Sparse Matrix - Vector (SpMV) kernel

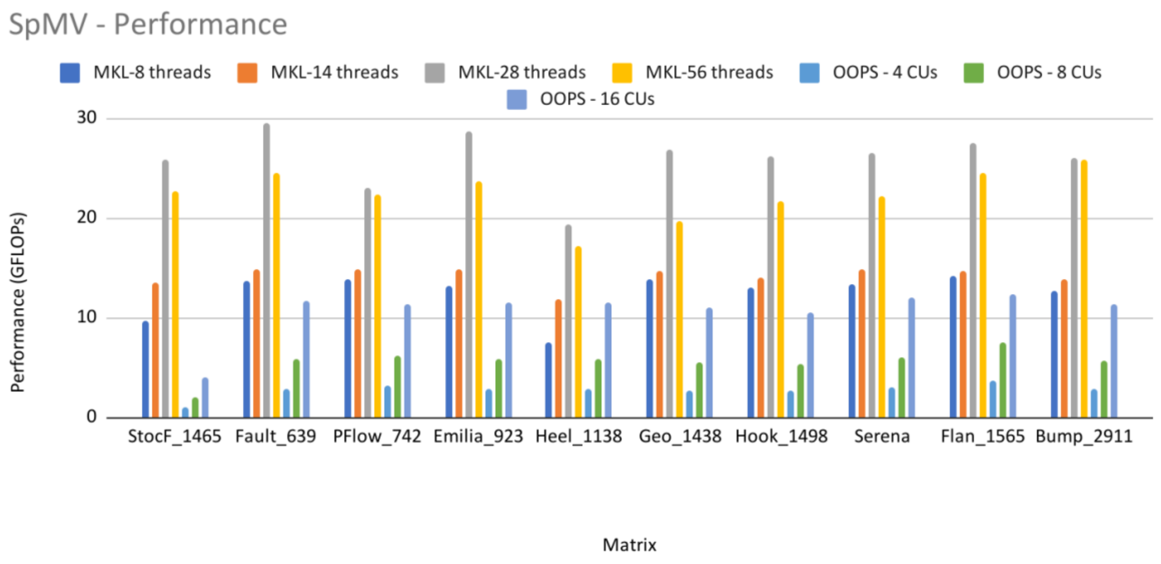


Figure 51: SpMV performance.

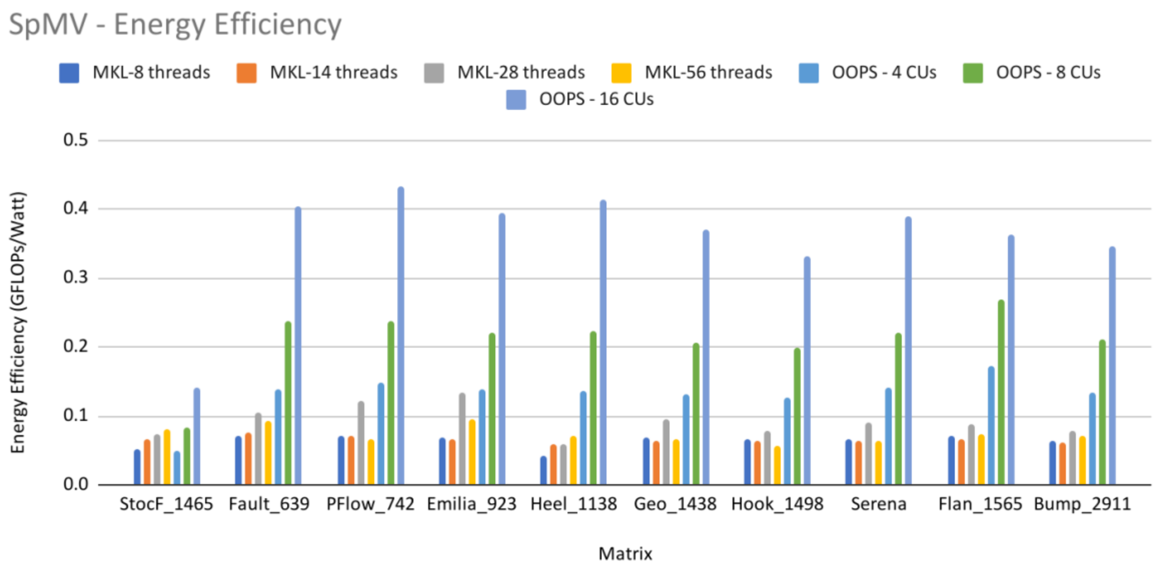


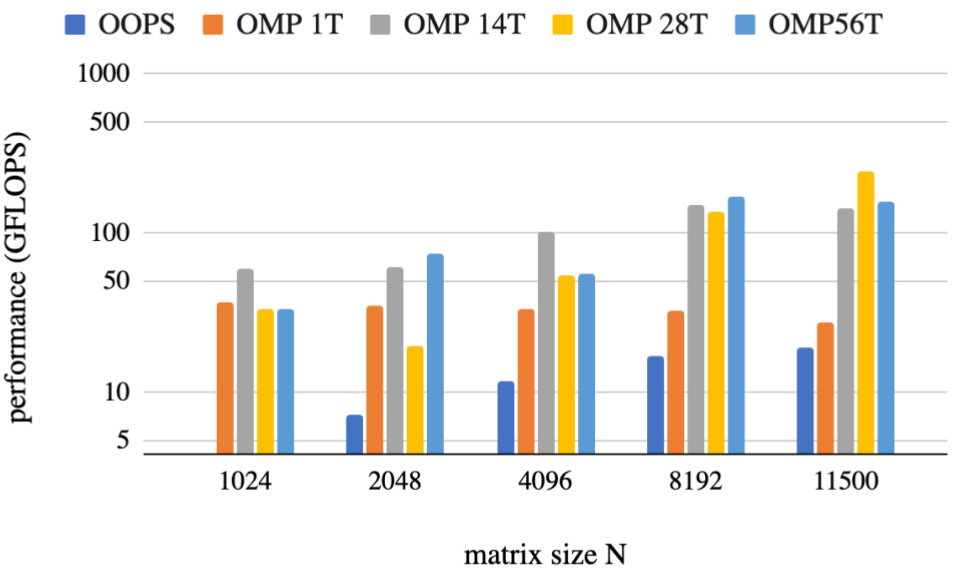
Figure 52: SpMV energy efficiency.

Figures 51 and 52 show the SpMV performance and energy efficiency. The final version of the SpMV kernel is configured with sixteen Compute Units, each assigned a segment of the output vector along with the respective elements of the matrix, enabling their independent operation. Given the memory-bound nature of this kernel, efficient utilization of available HBM bandwidth is crucial. To achieve this, input vector elements are streamed to the FPGA instead of being randomly accessed. In terms of performance, it achieves half the performance compared to the Intel MKL library running on a 56-thread server-class system. However, the FPGA's low operational power results in a 5x improvement in energy efficiency on average compared to the CPU.

6. General Computer-Aided Engineering (CAE) solvers

This section provides the evaluation results for matrix LU decomposition and Jacobi's preconditioner. The Conjugate Gradient kernel evaluation is in Section 4.

6.1. LU decomposition



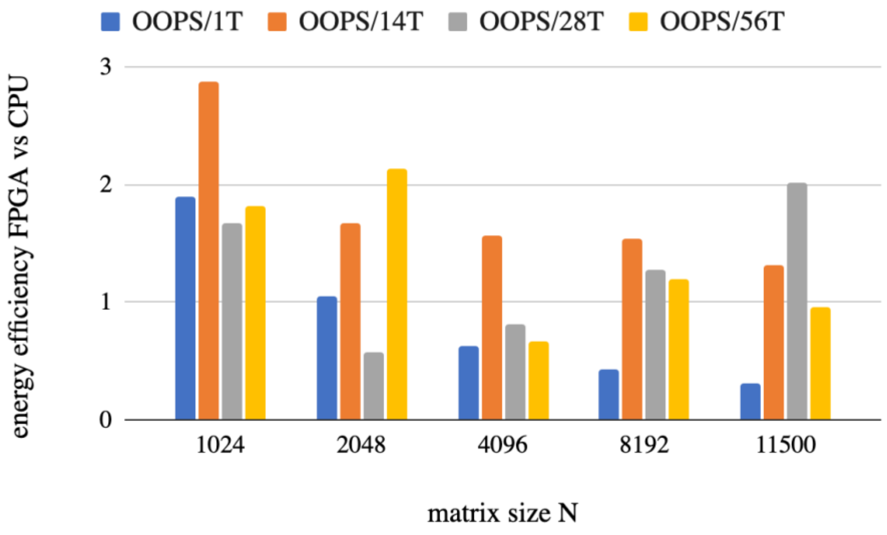


Figure 53: lu performance and energy efficiency.

The LU decomposition implementation supports up to 16 CUs. As shown in Figure 53, a single CU achieves 1.1x better performance, whereas the 16-CU version achieves 0.4x performance vs single-threaded and 56-threaded software, respectively. It should be noted that the execution time includes data transfers between the host and device, as well as intermediate data stores in the HBM.

As described in D5.5, the LU decomposition algorithm is based on iteratively processing smaller "snapshots" of the original matrix, thus intermediate results cannot be stored in the FPGA's BRAM

/ URAM. Instead, the OOPS LU kernel overlaps snapshot storage with iteration processing to increase performance, and partially hide the intermediate HBM access overheads. In terms of energy, due to the FPGA's much less power consumption (up to 27 Watts), energy efficiency varies; there are cases where the CPU is more energy efficient up to 2.7x (e.g. 1 thread for N=11500), and others where the OOPS LU kernel consumes up to 2.9x less energy (e.g. 14 threads for N=1024).

6.2. Jacobi preconditioner (applyJ)

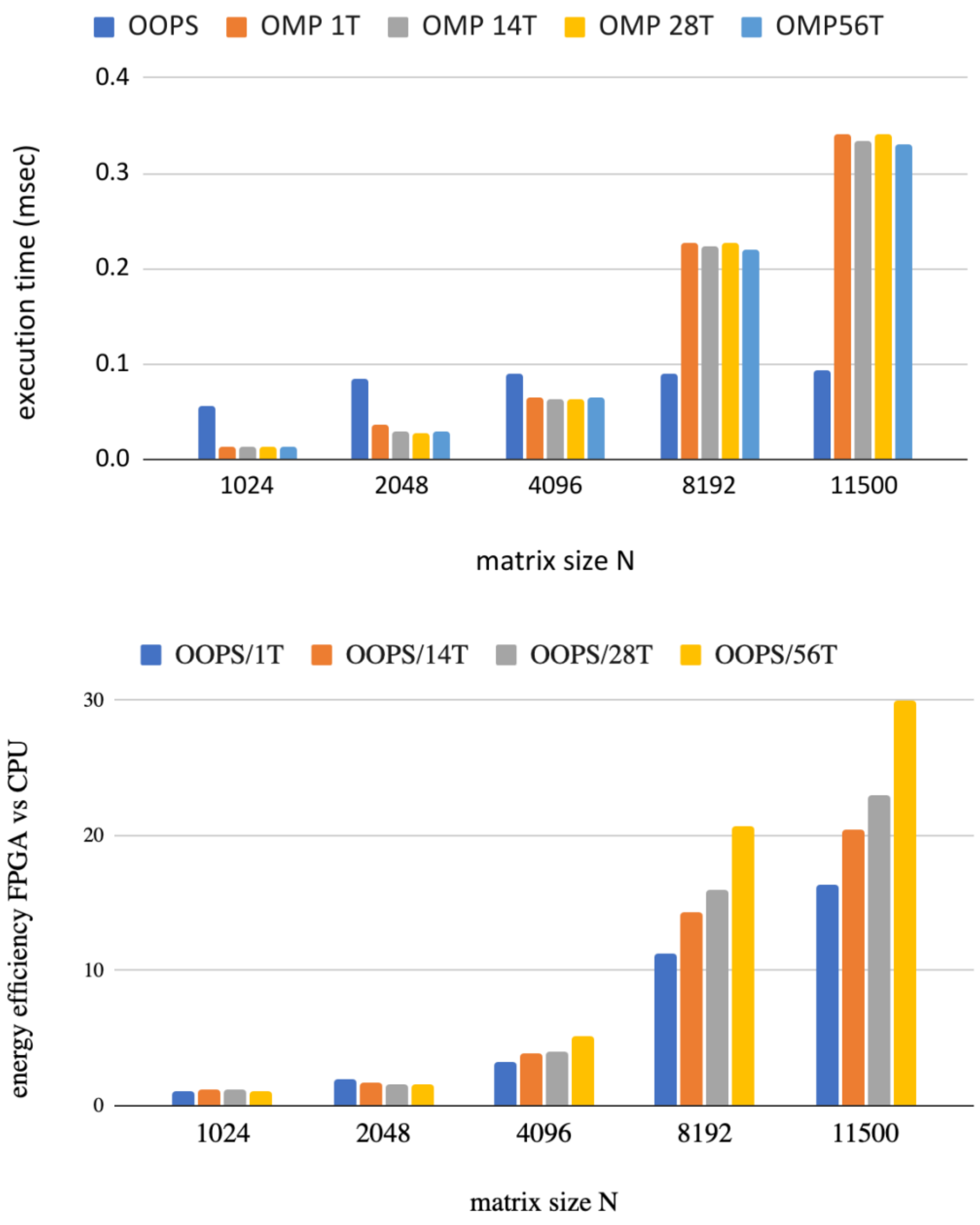


Figure 54: applyJ performance and energy efficiency.

Figure 54 shows the Jacobi preconditioning execution time and energy efficiency for different matrix sizes. For small sizes, the optimized software version can be faster up to 4.2x compared to the OOPS jacobi kernel. However, for larger sizes, such as N=11500 the FPGA version can be faster up to 3.5x. Moreover, in terms of energy, the low power consumption of the Alveo card (up to 28 Watts) leads to up to 30x better energy efficiency compared to server-class CPUs.

7. Resource utilization

This section provides the resource utilization for all OOPS kernels, considering the maximum number of CUs that can fit in an Alveo u55c FPGA. This is a more important aspect in the case of the libraries, than in the case of the overall applications, since we expect the end users to utilize those libraries together with the rest of their code; as a result, it is highly desirable to be aware of the percentage of the reconfigurable resources that will be available after the libraries are implemented in the FPGA.

Library	Kernel	FFs (%)	LUTs (%)	BRAMs (%)	DSPs (%)
BLAS L1	iamax	8	11	14	0
	asum	5	9	15	0
	axpy	11	14	34	15
	copy	3	5	21	0
	dot	4	7	17	2
	sddot	4	7	17	2
	nrm2	4	7	17	2
	rot	21	29	54	46
	rotm	13	18	35	29
	scal	11	14	53	18
	swap	6	10	41	0
	iamin	8	11	14	0
BLAS L2	gemv	15	26	21	21
	trmv	16	35	21	12

	symv	21	38	39	24
	trsv	12	9	50	15
	gbmv	14	27	14	15
	tbmv	16	34	21	13
	sbmv	21	42	39	24
	tbsv	16	35	31	15
	tpmv	7	9	8	7
	spmv	12	20	28	23
	tpsv	9	14	52	15
BLAS L3	gemm	26	22	37	40
	trmm	37	32	52	60
	symm	20	17	28	32
	trsm	15	12	77	20
Sparse algebra	SpMV	14	11	36	10
CAE solvers	Japply	7	8	20	23
	LU	12	16	25	18

The OOPS library enables energy-efficient HPC for BLAS and CAE solvers. When compared to Intel's Math Kernel Library (MKL), for all BLAS L1 kernels, OOPS provides a superior set of hardware implementations with respect to energy efficiency. Moreover, BLAS L2 kernels, and the Jacobi preconditioner achieve better performance and energy efficiency compared to the corresponding MKL's software functions. With respect to BLAS L3, LU decomposition, and SpMV kernels, in certain cases the OOPS implementations achieve lower raw performance compared to MKL; however, they still support more energy-efficient implementations. Overall, the improved energy efficiency provided by OOPS is mainly attributed to (i) the fact that an Alveo card on average consumes less than 30W (static+logic power), whereas the CPU and memory modules power was measured to consume up to more than 200W, and (ii) the dataflow pipelined implementations that increase overall throughput even with clock speeds that are 1 order of magnitude lower (e.g. 250 MHz FPGA clock vs 2.5 GHz CPU clock). To sum up, the OOPS library supports in total 30 kernels out of 3 libraries. Moreover, it enables performance and energy-efficient HPC implementations that can reduce energy consumption (when compared to server-class machines) up to 50x for L1 BLAS kernels, up to 20x for specific L2 BLAS kernels, up to 1.5x for specific L3 BLAS kernels, and up to 1.5x for CAE solvers.