The Journal of Supercomputing (2022) 78:18051–18060
https://doi.org/10.1007/s11227-022-04581-6

A BLIS-like matrix multiplication for machine learning in the RISC-V ISA-based GAP8 processor

Cristian Ramírez[1] · **Adrián Castelló**[1] · **Enrique S. Quintana-Ortí**[1]

Accepted: 29 April 2022 / Published online: 28 May 2022

Abstract
We address the efficient realization of matrix multiplication (GEMM), with application in the convolution operator for machine learning, for the RISC-V core present in the GreenWaves GAP8 processor. Our approach leverages BLIS (Basic Linear Algebra Instantiation Software) to develop an implementation that (1) re-organizes the GEMM algorithm adapting its micro-kernel to exploit the hardware-supported dot product kernel in the GAP8; (2) explicitly orchestrates the data transfers across the hierarchy of scratchpad memories via DMA (direct memory access); and (3) operates with integer arithmetic.

Keywords Matrix multiplication · High performance · RISC-V GAP8

1 Introduction

In the last years, there is a strong interest to realize deep learning (DL) at the edge, in IoT (Internet-of-things) devices, in order to improve safety and privacy, reduce the latency experienced by the end-user, and/or decrease energy consumption [1–4]. This implies that DL technologies have to be deployed on IoT devices, with limited computational and memory capacities, restrictions in power supply and, in many cases, with strict time constraints.

Convolutional neural networks (CNNs) are the mainstream DL tool for image recognition and classification as well as signal processing [5], yet nowadays they are also being deployed for other types of DL tasks [6, 7]. From the computational point of view, the convolution is the most expensive operator in this type of DL networks

✉ Adrián Castelló
adcastel@disca.upv.es

Cristian Ramírez
cirabe@upv.es

Enrique S. Quintana-Ortí
quintana@disca.upv.es

[1] Universitat Politècnica de València, València, Spain

[7]. The IM2COL-based approach (also known as lowering) is a popular, flexible, and general-purpose approach to transform the convolution operator into a compute-intensive, cache-friendly matrix–matrix multiplication (GEMM) [8, 9]. By embedding the initial IM2COL transform into the data packing routines that are intrinsic to a high-performance realization of GEMM, the time cost of this transform can be fully amortized over the calculations and the need for an additional memory disappears [10].

While there exist many popular dense linear algebra (DLA) libraries that include high-performance realizations of the GEMM kernel (e.g., Intel MKL, AMD AOML, NVIDIA cuBLAS, GotoBLAS2, OpenBLAS, BLIS, etc.), these do not provide an effective tool to perform inference on IoT devices for a couple of reasons mainly. On the one hand, the memory footprint of these libraries is considerable (sometimes even too large for IoT appliances), as they cover a range of DLA functionality much beyond the simple GEMM kernel that is needed in DL inference. On the other hand, these DLA libraries support floating point arithmetic, while many IoT devices can only operate with integer or fixed point data.

In this work, we build the upon BLIS ideas and techniques [11] to develop an IM2COL-based convolution operator for the GAP8 platform[1] [12]. This is an ultra-low-power processing processor with a RISC-V 1+8-core cluster for computation-intensive workloads. From the point of view of optimizing the IM2COL-based convolution/GEMM, the GAP8 system presents some particular features that result in the following contributions from our work:

- We develop a BLIS-like GEMM that operates on top of the dot (scalar or inner) product, a vector kernel that is intended to receive special support in the GAP8. This is different from the approach taken in the *BLIS baseline algorithm*, which is built around the outer product. (A second major difference between the BLIS baseline algorithm and ours is the layers of the memory hierarchy where the distinct matrix operands reside when accessed from the micro-kernel.)
- We develop an instance of GEMM optimized for 8-bit integer (INT8) arithmetic that can be easily adapted for fixed point arithmetic.
- Similarly to [13], we orchestrate a careful sequence of data transfers between the different memory areas of the GAP8 via DMA transfers, integrating these movements into the BLIS packing routines.
- We perform an experimental evaluation of the GEMM realization in the FC (fabric controller) in the GAP8 platform.

We close this general discussion with two remarks:

- Depending on the convolution parameters, a Winograd or FFT convolution operator [14, 15] may (or may not) be more time-efficient (but also less precise) than an IM2COL-based convolution. However, a comparison of these approaches is out-of-scope for this paper.

[1] https://gwt-website-file.s3.amazonaws.com/gap8_datasheet.pdf.

```
L1   for ( j_c = 0;  j_c < n;  j_c += n_c )
L2     for ( p_c = 0;  p_c < k;  p_c += k_c ) {
         B_c ← B(p_c : p_c + k_c − 1, j_c : j_c + n_c − 1); // Pack
L3       for ( i_c = 0;  i_c < m;  i_c += m_c ) {
           A_c ← A(i_c : i_c + m_c − 1, p_c : p_c + k_c − 1); // Pack
L4         for ( j_r = 0;  j_r < n_c;  j_r += n_r )
L5           for ( i_r = 0;  i_r < m_c;  i_r += m_r )
L6             for ( p_r = 0;  p_r < k_c;  p_r += 1 ) // Micro-kernel
                 C_c(i_r : i_r + m_r − 1, j_r : j_r + n_r − 1)
                   += A_c(i_r : i_r + m_r − 1, p_r) · B_c(p_r, j_r : j_r + n_r − 1);
       }}
```

Fig. 1 High-performance implementation of the BLIS baseline algorithm for GEMM. Here C_c is a notation artifact, introduced to ease the presentation

- The major part of our paper is focused on the efficient realization of GEMM on the GAP8 processor. We emphasize that, given a high-performance realization of this kernel, transforming that into an efficient IM2COL-based convolution operator is straightforward [10].

The rest of the paper is structured as follows: In Sect. 2, we offer a brief review of the BLIS algorithm for GEMM. Next, in Sect. 3 we detail the main features of the GAP8 platform. In Sect. 4, we describe the approach to obtain a BLIS-like GEMM algorithm for the GAP8 system and evaluate the resulting routine. Finally, in Sect. 5 we close the paper with a summary of the insights gained and a discussion of future work.

2 Matrix multiplication in BLIS

Consider the general matrix–matrix multiplication (GEMM) $C \mathrel{+}= AB$, where the dimensions of the matrix operands A, B and C are $m \times k$, $k \times n$ and $m \times n$, respectively. BLIS implements this operation as five nested loops (L1–L5) around two packing routines and a micro-kernel. In addition, the latter is decomposed into an additional loop (L6) around an outer product. The realization of this *BLIS baseline algorithm* for GEMM is illustrated in Fig. 1.

The ordering of the loops in the BLIS baseline algorithm, together with a proper selection of the loop strides and dimensions of the A_c, B_c buffers [16], orchestrate the movement of the matrix operands across the processor memory hierarchy during the execution of the matrix product. Concretely, a $k_c \times n_c$ block of B is copied into a buffer B_c that is expected to reside in the L3 cache; and an $m_c \times k_c$ block of A is copied into a buffer A_c that targets the L2 cache. Furthermore, the micro-kernel operates with a $k_c \times n_r$ micro-panel of B_c (to be retrieved from the L1 cache), an $m_r \times k_c$ micro-panel of A_c (in the L2 cache), and an $m_r \times n_r$ micro-tile of C (streamed directly from the main memory); see [11, 16]. For the BLIS baseline algorithm, we will refer to the micro-panels of A_c, B_c as A_r, B_r, respectively; and the micro-tile of C as C_r.

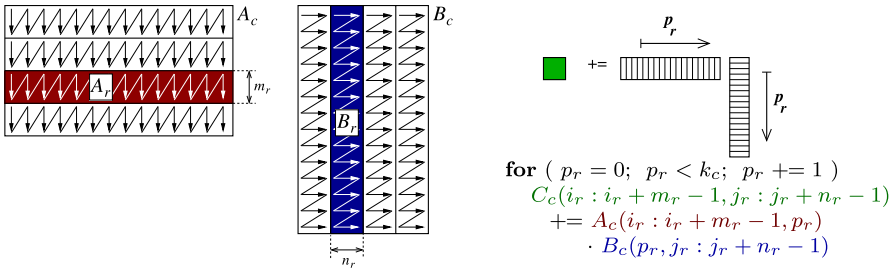

Fig. 2 Packing in the BLIS baseline algorithm for GEMM (left) and micro-kernel with *C* resident in the processor registers (right)

For high performance, the values of the cache configuration parameters m_c, n_c, k_c are either determined via extensive experimental evaluation or, alternatively, using an analytical model that takes into account the size and associativity of each cache level [16].

The copies into A_c, B_c ensure that the entries of the micro-panels A_r, B_r are accessed with unit stride from the micro-kernel (see Fig. 2), enabling the use of SIMD (single-instruction multiple-data) vector instructions [11].

To close this section, in the following we will next refer to the BLIS baseline algorithm as B3A2C0, where the letter "Z" $\in$ {A,B,C} corresponds one of the three matrix operands and the subsequent number, $i \in \{0, 1, 2, 3\}$, specifies the target cache level for that operand (with 0 referring to the processor registers).

3 An overview of the GAP8 platform

The GAP8 platform, from GreenWaves Technologies, is an IoT application processor based in the PULP (Parallel-Low-Power Processing Platform) that comprises (1) a fabric controller (FC) core for control, communications, and security functions; (2) a cluster of 8 cores designed for the execution of parallel algorithms; and 3) a specialized accelerator (HWCE). All these components share the same 512-KB L2 *memory area* (MA). Furthermore, the FC has a 16-KB L1 MA while the cluster cores and HWCE share a 64-KB multi-banked TCDM L1 (data/instruction) MA. Several DMA units allow fast transfers between MAs. The layout of the GAP8 platform is illustrated in Fig. 3.

All 1+8 cores support the same extended RISC-V instruction set architecture (ISA), including the I (integer), C (compressed instruction), M (Multiplication and division) extensions and a portion of the supervisor ISA subset. Specialized instructions exist for zero overhead hardware loops, pointer post-/pre-modified memory accesses, instructions mixing control flow with computation (e.g., min and max), multiply/subtract and accumulate, vector operations, fixed point operations, bit manipulation, and the *dot product of two vectors*. The SIMD registers in the GAP8 processor are 32-bit long (i.e., 4 INT8) and the dot product vector operations are

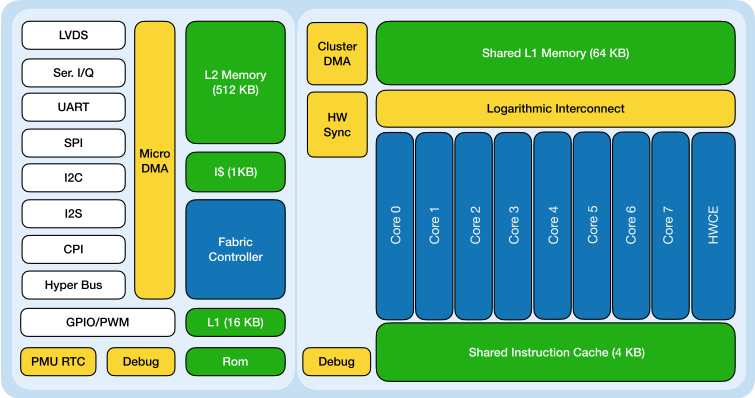

Fig. 3 GAP8 layout

simulated by software in the current SDK. Our current code targets the FC. Porting the realization of the GAP8 cluster is part of ongoing work.

According to the GAP8 manual[2], the banks of the shared L1 MA can be accessed from the cluster cores in a single cycle. In comparison, accessing data in external MAs (referred to as L3 memory,) incurs a very high cost and, therefore, should be avoided whenever possible.

The GAP8 does not include data caches but instead relies on programmable DMA units to orchestrate data transfers to/from peripherals and in between the internal L1 and L2 MAs. Therefore, these memory spaces can be viewed as "scratchpads." The micro-DMA unit is used to transfer data to/from peripherals, including the L3 memory.

In conclusion, the GAP8 platform (1) provides special extended instructions set to optimize the performance of signal processing and machine learning algorithms, including support for the dot product vector operation; and (2) requires a careful management of the data transfers to extract optimal performance and energy efficiency. To address these, *in the next section we describe (1) how to leverage the BLIS framework in order to formulate a BLIS-like realization of* GEMM *that is based on the dot product; and (2) discuss how to accommodate the data movements in* GEMM *as part of the packing routines in a BLIS-like realization of the matrix multiplication for the GAP8 platform.*

4 Experimental results

Setup As a starting point for the experimental evaluation, we analyze the distribution of time costs for a particular GEMM of moderate dimensions $m, n, k = 1792, 1536, 1024$. Table 1 reports the execution times of the data transfers

[2] https://greenwaves-technologies.com/manuals/BUILD/HOME/html/index.html.

Table 1 Distribution of costs between the different "components" that are involved in the B3C2A0 implementation of GEMM with $m = 1792, n = 1536, k = 1024, m_c = 384, n_c = 682, k_c = 3072, m_r = 4$, and $k_r = 12$

Component	Time (s)	#Mem. accesses	Mbytes/s	Observations
Pack B_c	1.85	3.15E+06	1.62E+00	L3 (via L2) to L3
Pack C_c	4.95	2.75E+06	5.30E−01	L3 to L2
Unpack C_c	4.01	2.75E+06	6.54E−01	L2 to L3
Copy B_r	0.85	7.86E+06	8.81E+00	L3 (via L2) to L1
Stream A_r	10.80	5.51E+06	4.87E−01	L3 (via L2) to registers
Stream B_r	3.78	7.05E+08	1.78E+02	L1 to registers
Stream C_c	62.80	4.73E+08	7.18E+00	L2 to registers to L2
	Time (s)	#INT8 ops	INT8 MOPS	Observations
Arithmetic	70.09	5.64E+09	7.67E+01	Only arithmetic ops
Total	159.13	–	3.38E+01	Arithm. and mem. ops

as well as the arithmetic cost for the B3C2A0 realization of GEMM for this matrix multiplication, when setting $m_c = 384$, $n_c = 682$, $k_c = 3072$; and using a micro-kernel with $m_r \times k_r = 4 \times 12$.[3] For reference, we include the volume of memory transfers corresponding to each pack/unpack/copy routines (i.e., $B \rightarrow B_c$, $C \rightarrow C_c$, $C_c \rightarrow C$, and $B_c \rightarrow B_r$); and the data streamings from/into the processor registers performed from within the micro-kernel (in the table referred to as "Stream $A_r/B_r/C_c$"). The bottom part of the table shows the cost of the arithmetic performed in the micro-kernel when all data is already present in the processor registers. Therefore, it gives a realistic estimation of the practical peak performance that can be attained with the dot product operations, in the GAP8 FC, for a 4×12 micro-kernel.

Before we analyze the performance results, it is interesting to carry out a preliminary analysis of the GAP8 processor. Concretely, Table 1 reports that the time to compute a single INT8 operation in the GAP8 processor is $1/(76.7 \cdot 10^6) = 1.30e-08$ s, while the costs to transfer a single INT8 number from the L1 (for B_r)/L2 (for C_c) MAs to the processor registers are, respectively, given by $1/(178 \cdot 2^{10}) = 5.35e-09$ s and $1/(7.18 \cdot 2^{10}) = 1.32e-07$ s. This is relevant because, for each fused multiply-add operation, the micro-kernel for the B3C2A0 algorithm retrieves an element of B_r from the L1 MA and reads/writes a single entry of C_c from the L2 MA. The important L2 access cost (higher than the cost of the arithmetic) determines that, even though GEMM is often viewed as a compute-bound operation, this is a memory-bound kernel in the GAP8 processor.

Performance analysis We can highlight a few of aspects by inspecting the table. First, as could be expected from a blocked algorithm for GEMM, the larger volumes of memory transfers involve those data that are closer to the processor

[3] We also experimented with 4×4 and 4×8 micro-kernels, but these options were discarded due to the considerably higher costs of transferring C_c between the L2 MA and the registers.

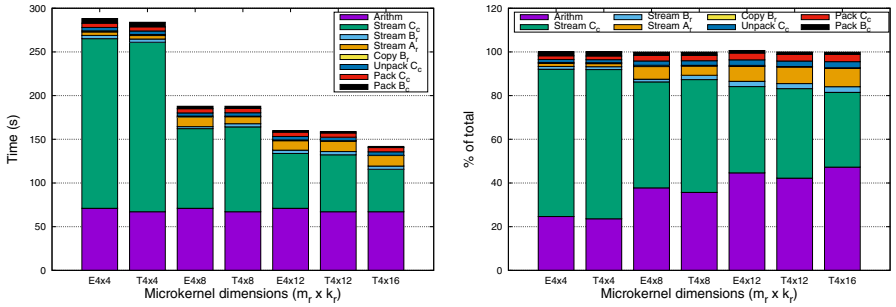

Fig. 4 Theoretical and experimental time and distribution of costs among the different components of the algorithm using micro-kernels of dimension 4×4, 4×8, and 4×12. (We also include the theoretical analysis for a 4×16 micro-kernel.) The labels starting with "E" and "T" below each bar distinguish between results using experimentation of the analytical model, respectively. The left-hand plot shows the total execution time while the right-hand plot reports the percentage of time

registers. The transfer rates are also coherent, reporting higher bandwidth for the data movements between the L1 MA and the processor registers, followed by the transfers between the L2 MA and the registers, and so on. In this sense, the distinct transfer rates observed for the packings involving B_c and C_c can be explained because the former operates with "sub-vectors" of dimension $k_r = 12$ while the latter operates with sub-vectors of sizes $m_r = 4$, improving the efficiency of the DMR mechanism by a factor of $k_r/m_r = 3$. Also, note the different transfer rates for the copy of B_r as this does not incur the overhead of re-organizing data required by the packing/unpacking routines.

Second, adding up all costs, the total execution time is 159.13 seconds for this particular problem, selection of cache configuration parameters, and micro-kernel. This offers a rate of 33.8 MOPS (millions of INT8 operations per second), which is lower than the practical peak of 76.7 MOPS if we only consider the arithmetic, attaining about 44% of that peak. For this particular problem, the results in the table expose a clear bottleneck due to the transfers of C_c between the L2 MA and the processor registers.

To further investigate this, we developed an analytical model that estimates the partial costs due to each component, and validated its accuracy using the data from the table as well as that of several other executions using different cache configuration parameters and/or distinct (4×4, 4×8) micro-kernels. (The relative errors for all tested cases remained below 2%.) We then used this model to estimate that, for example, using a 4×16 micro-kernel and properly scaling the dimensions of m_c, n_c, k_c to fit the buffers into the memory system, the performance could be improved to 39.8 MOPS for the same problem dimension. Figure 4 reports some of the results attained from experiments with other micro-kernels and the analytical model. At this point, we would like to note that developing manually optimized micro-kernels is a complex task that we are working to transform into an automatic procedure. Given that the analytical model provides accurate estimations of the performance of the algorithm when combined with a particular micro-kernel, we did not see much purpose in creating a large variety of micro-kernels.

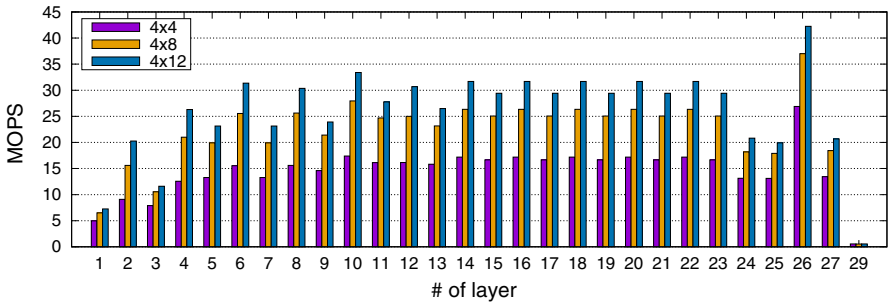

Fig. 5 MOPS rates attained for the convolutional layers in MobileNet-v1

From these results, we recognize that there is a clear path to improve the performance of our GEMM realization by integrating a prefetching scheme within the micro-kernel that overlaps the access to C_c from within it with the arithmetic operations. We plan to investigate this as our next immediate step.

Finally, Fig. 5 reports the MOPS rates attained for the convolution layers appearing in the MobileNet-v1 [17] CNN, when processed using the IM2COL approach followed by our GAP8-specific implementation of GEMM, for three different micro-kernel sizes. This figure confirms the superiority of the 4×12 micro-kernel for the practical cases appearing in DL, but also the practical use of the GEMM realization as, in many of the cases, the throughput rate remains between 25 and 55 MOPS.

Comparison with other GEMM algorithms. The family of BLIS-like algorithms for GEMM comprises 6 variants which are obtained by re-ordering the loops of the algorithm in Fig. 1 (though BLIS only implements one: the baseline algorithm) [18–20]. We next briefly discuss the main features of these alternative realizations:

- Variant B3C2A0 is obtained by swapping the roles of operands B and C with respect to the C3B2A0 realization presented in this work. As a result, B is now re-packed into the L3 memory (as B_c), and from there copied into the L1 MA (as B_r); C is packed into the L2 MA (as C_c); and a micro-tile of A remains in the processor registers during the execution of the micro-kernel, with the latter cast in terms of a sequence of dot products. Given the bottleneck that we identified in the access to the L2 MA, and the fact that this variant needs to both read and write the entries of C from that level of the memory, we can expect that, in general, this variant incurs in higher memory access costs than our C3B2A0 algorithm, which only needed to read B from the L2 MA.
- Variant B3A2C0 (BLIS baseline algorithm) was already discussed in Sect. 2. From the point of view of the memory access to the L2 MA, this variant should present a similar behavior as that of our realization C3B2A0 (depending on the GEMM operands' dimensions and provided there is a proper selection of the cache configuration parameters). However, as argued earlier, B3A2C0 implements its micro-kernel as a sequence of outer products, which have no software/hardware support in the GAP8 processor. Therefore, we can expect higher arithmetic costs for the BLIS baseline algorithm in the GAP8 processor.

- The remaining 3 variants (C3A2B0, A3B2C0, A3C2B0) are obtained by simply swapping the roles of A and B in the three algorithms previously discussed, and they should no significant advantage over them. For example, A3C2B0 is the "twin" of the B3C2A0 algorithm proposed in this paper. As both A and B are input operands with "symmetric" roles in GEMM, the fact that either A or B is the operand that resides in a certain level of the memory hierarchy should be irrelevant (again, provided it is not affected by the GEMM operands' dimensions and the cache configuration parameters are selected properly).

5 Concluding remarks

We have described a member of the BLIS family of algorithms for GEMM especially designed to exploit the hardware support for the dot product kernel provided by the RISC-V core that is integrated into the GreenWaves GAP8 processor. Our approach operates with 8-bit integer arithmetic in order to offer an energy-efficient inference tool for CNNs. The experimental analysis demonstrates that a careful orchestration of the data transfers across the GAP8 memory hierarchy is crucial to obtain fair performance. The path ahead of us is broad. As part of future work, we plan to develop more sophisticated micro-kernels that overlap data transfers with arithmetic, as well as exploit the 8-core cluster in the GAP8 platform.

Acknowledgements This work was supported by the research project PID2020-113656RB-C22 of MCIN/AEI/10.13039/501100011033. C. Ramírez is a "Santiago Grisolía" fellow supported by *Generalitat Valenciana*. Adrián Castelló is a FJC2019-039222-I fellow supported by MCIN/AEI/10.13039/501100011033. This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under Grant Agreement No. 955558. The JU receives support from the European Union's Horizon 2020 research and innovation program, and Spain, Germany, France, Italy, Poland, Switzerland, Norway.

References

1. Hazelwood K et al (2018) Applied machine learning at Facebook: a datacenter infrastructure perspective. In: IEEE International Symposium on High Performance Computer Architecture, pp 620–629
2. Park J et al (2018) Deep learning inference in Facebook data centers: characterization, performance optimizations and hardware implications. arXiv:1811.09886
3. Wu C et al (2019) Machine learning at Facebook: understanding inference at the edge. In: International Symposium on High Performance Computer Architecture, pp 331–344

4. Yi S, Li C, Li Q (2015) A survey of fog computing: concepts, applications and issues. In: Proceedings of the 2015 Workshop on Mobile Big Data, ser. Mobidata'15, pp 37–42
5. Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS), vol 1, pp 1097–1105
6. Pouyanfar S et al (2018) A survey on deep learning: algorithms, techniques, and applications. ACM Comput Surv 51(5):92:1-92:36
7. Sze V et al (2017) Efficient processing of deep neural networks: a tutorial and survey. Proc IEEE 105(12):2295–2329
8. Chellapilla K, Puri S, Simard P (2006) High performance convolutional neural networks for document processing. In: Tenth International Workshop on Frontiers in Handwriting Recognition
9. Georganas E et al (2018) Anatomy of high-performance deep learning convolutions on SIMD architectures. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, ser. SC '18. IEEE Press
10. San Juan P et al (2020) High performance and portable convolution operators for multicore processors. In: Proceedings of the 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), pp 91–98
11. Van Zee FG, van de Geijn RA (2015) BLIS: a framework for rapidly instantiating BLAS functionality. ACM Trans Math Softw 41(3):14:1-14:33
12. Flamand E et al (2018) GAP-8: A RISC-V SoC for AI at the edge of the IoT. In: IEEE 29th Interantional Conference on Application-Specific Systems, Architectures and Processors, pp 1–4
13. Ali M et al (2012) Level-3 blas on the ti c6678 multi-core dsp. In: 2012 IEEE 24th International Sympsoium on Computer Architecture and High Performance Computing
14. Lavin A, Gray S (2016) Fast algorithms for convolutional neural networks. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp 4013–4021
15. Zlateski A, Jia Z, Li K, Durand F (2019) The anatomy of efficient FFT and Winograd convolutions on modern CPUs. In: Proceedings of the ACM International Conference on Supercomputing, ser. ICS '19, pp 414–424
16. Low TM et al (2016) Analytical modeling is enough for high-performance BLIS. ACM Trans Math Softw 43(2)
17. Howard AG, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, Andreetto M, Adam H (2017) Mobilenets: efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861
18. Gunnels JA, Gustavson FG, Henry GM, van de Geijn RA (2004) A family of high-performance matrix multiplication algorithms. In: Proceedings of the 7th International Conference on Applied Parallel Computing: State of the Art in Scientific Computing, ser. PARA'04. Berlin, Heidelberg, pp 256–265. https://doi.org/10.1007/11558958_30
19. Smith TM, van de Geijn RA (2019) The MOMMS family of matrix multiplication algorithms. CoRR, vol. abs/1904.05717. arXiv:1904.05717
20. Castelló A, Igual FD, Quintana-Ortí ES (2022) Anatomy of the BLIS family of algorithms for matrix multiplication. In: 30th Euromicro Workshop on Parallel, Distributed and Networked Processing PDP 2022, to appear