Performance evaluation and modelling of single-precision matrix multiplication on Cerebras CS-2

Ryunosuke Matsuzaki
Dept. of Computer Science, Meiji University
Kawasaki, Kanagawa, Japan
ce245011@meiji.ac.jp

Daichi Mukunoki
Independent researcher
daichi.mukunoki@gmail.com

Takaaki Miyajima
Dept. of Computer Science, Meiji University
Kawasaki, Kanagawa, Japan
takaaki.miyajima@cs.meiji.ac.jp

***Abstract*—Although recent supercomputers have been improving their computational performance, achieving performance scaling with respect to the number of nodes is not easy due to long inter-node communication latency. Many attempts have been made to hide communication latency and maintain strong scalability even for dense matrix multiplication. Matrix multiplication is an ideal candidate for benchmarking the performance of supercomputers. The Cerebras CS-2 system is an accelerator for deep learning with the world's largest chip, the wafer-scale engine 2 (WSE-2). The WSE-2 can be considered a distributed memory system that comes with 745500 processing elements connected in a low-latency 2D mesh topology. This paper presents the effective maximum performance, weak and strong scaling performance, and proposes a performance model for single-precision matrix multiplication on CS-2. We observed the maximum performance of 349.0 TFlops/s (matrix size: 33000×33000) and a weak scaling efficiency of 1.00. The mean absolute percentage error between our performance model and the actual measurement was 4.7% at matrix size is around 2000×2000.**

***Index Terms*—Cerebras CS-2, matrix multiplications**

I. Introduction

Dense matrix multiplication is one of the key computational kernels and is often used for benchmarking the computational performance of supercomputers as a computation-intensive task. As the number of nodes in supercomputers increases, it is no longer easy to maintain strong scalability even for matrix multiplication due to long inter-node communication latency [1]. It is known that the communication time becomes larger than the computation time when the problem size is not large enough. The Cerebras CS-2 system achieves higher scalability in parallel computing through one clock cycle communication latency between processing elements (PEs). Although performance evaluations in several specific applications has been conducted [2] [3] [4], performance analysis of fundamental kernels such as matrix multiplication has not been progressed. This paper presents the maximum performance, weak and strong scaling performance, and proposes a performance model for single-precision matrix multiplication on CS-2.

II. Background

A. Cerebras CS-2 System

The CS-2 system is equipped with one wafer-scale engine 2 (WSE-2), the world's largest chip at the time of this writing. The WSE-2 boasts a homogeneous array of PEs and operates on a distributed memory architecture featuring a simple 2-D mesh topology. With a staggering total of 745500 PEs running at 850 MHz, arranged in a layout of 750 along the x-axis and 994 along the y-axis. Each PE comprises a router and a Compute Element (CE). The CE comprises a 48 KB local scratchpad memory and a six-stage in-order pipeline computing core. They are connected by two 64-bit read ports and one write port. Coherency among PEs is not guaranteed and there is no conventional shared memory like in CPUs and GPUs. The local scratchpad memory can be accessed in one clock cycle if no bank conflicts occur. Each router can efficiently transfer 32-bit data (either as one 32-bit or two 16-bit transmissions) to neighbouring PEs within a single cycle. Notably, communication and computation are seamlessly overlapped, with communication operations not consuming processor cycles.

Cerebras SDK and Cerebras System Language (CSL) are provided to program scientific applications on CS-2 [5]. Built-in functions perform operations such as fused multiply-accumulate (FMAC) on up to four-dimensional tensors. A set of functions for collective communication in the x-/y-axis direction is also provided as a library. For example, *@fmacs* and *broadcast* functions perform single-precision floating-point FMAC operation and broadcast a message among PEs, respectively. Point-to-point communication is not provided for CS-2. Because the routing must be determined at compile time, it is impractical to implement communications where the destination changes dynamically.

B. Parallel distributed matrix multiplication

Scalable Universal Matrix Multiplication Algorithm (SUMMA) is a widely used parallel distributed matrix multiplication algorithm [6]. It consists of a local matrix product using FMAC operations and communication consisting of broadcast operations in the x-/y- directions. Compared to the Cannon algorithm, it requires lower communication bandwidth but causes higher latency due to

979-8-3503-5554-3/24/$31.00 ©2024 IEEE
DOI 10.1109/SCW63240.2024.00101

the need for broadcasts with a distance equal to the matrix size [7]. As the number of nodes increases while the size of matrices remains the same, the number of transferred data becomes small, but the communication latency cost becomes non-negligible. Fine-grained communication often becomes an obstacle to scaling on recent supercomputers due to its long internode communication latency. On the other hand, broadcast in the x-/y- directions on a 2D mesh is a suitable communication pattern for systems with 2D mesh topology such as CS-2. The low latency inter-PE communication of CS-2 can mitigate the impact of distant broadcast communication.

The Cerebras SDK provides a SUMMA implementation for CS-2 named "GEMM with Collective Operations". [8]. In this program, the input matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are decomposed into sub-matrices that fit into the PE's memory size of 48 KB. Then the sub-matrices are sent from the host machine to the PEs on the CS-2, and the CS-2 performs SUMMA. Finally, the CS-2 returns the result matrix $\boldsymbol{C}$ to the host machine. In this implementation, the *broadcast* function is used for x-/y-direction broadcast communication and *@fmacs* function is used for local matrix multiplication.

III. Performance Evaluation

We evaluate the maximum performance, weak and strong scaling performance of SUMMA on CS-2 using the "GEMM with Collective Operations" program. The size of matrices to be computed is $M \times M$, and the size of submatrices to be computed on each PE is $Mt \times Mt$. The purpose of this paper is to understand the performance characteristics of CS-2 and not to evaluate the performance of the algorithm, so only square matrices are used for evaluation. Evaluating the performance of other shapes is a topic for future research. Since only square matrices are evaluated, not all the PEs on WSE-2 are used for maximum performance. Specifically, 750×750 out of 750×994 PEs are used. Because the program requires two temporally buffers equivalent to the size of the matrix, each PE consumes a total of 5 buffers $\times Mt \times Mt \times$ 4 bytes of memory. In addition, the alignment of matrices is not optimised, and bank conflicts are not avoided.

We use the CS-2 system with Cerebras SDK version: 1.0.0 and WSE-2 software version: 1.6.1 from TED AI Lab, Tokyo Electron Device Ltd. The cycle counter provided by the Cerebras SDK was used for the time measurement. The elapsed time was calculated as the elapsed cycles divided by the operating frequency of 850 MHz. The computational performance (Flops/s) was calculated as the number of operations, $2 \times M^3$, divided by the elapsed time. As matrix multiplication is commonly used as a part of certain applications, the performance evaluation only covers the computation inside the CS-2. It excludes the communication time of input and output data between the host CPU and the CS-2.

A. Maximum performance

The CS-2 achieved a maximum performance of 349.0 TFlops/s with $750 \times 750 = 562500$ PEs (75.5% of the total 750×994 PEs) for the matrix size of $M = 33000$ with $Mt = 44$ ($44 \times 750 = 33000$). In this case, the memory footprint on each PE is 44×44×4 bytes×5 buffers=38.7 KB and 21.78 GB on the overall computation. This is 80.6% of the total 27.0 GB of memory for the 562500 PEs used and 60.8% of the total 35.78 GB of memory for the entire CS-2.

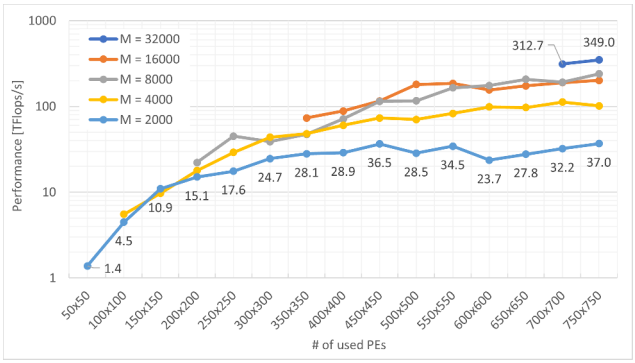

Fig. 1: Strong scaling: Trends of computational performance at $M \approx 2000$ to 32000.

The ratio of computation and communication to the total processing time is 98% and 2%, respectively, It indicates that computation is a more limiting factor for performance than communication. 349.0 TFlops/s is 73.8% of the attainable peak performance of 477.5 TFlops/s with 750×750 PEs. Note that the attainable peak performance is the peak performance of the FMAC operations on a single PE multiplied by the number of PEs used, not a theoretical performance. The major reason for the performance decrement is that Mt is 44, which is too small to achieve the peak performance of FMAC operations. The peak performance of the FMAC operations is obtained when processing more than 128 elements continuously, and performance drops by around 10% from the peak performance when processing 44 elements. It has also been known that the performance of the FMAC operations can only be improved by about 5% at 32 or 64 elements, even if bank conflict is avoided [9]. Therefore, we believe that increasing the number of elements to be processed contiguously is necessary to improve computational performance.

B. Strong scaling

We measure the strong scaling performance when the number of PEs used is increased while keeping the problem size the same. The number of PEs used is varied from 50×50 to 750×750 in increments of 50. The matrix size M is set to around 2000, 4000, 8000, 16000, and 32000, but the actual size varies slightly depending on the number of PEs used since M may not be divisible by the number of PEs. For example, when 600 PEs are used for a computation of $M \approx 4000$, the actual matrix size is $M = 4200$ with $Mt = 7$.

Figures 1 and 2 show the trend of the computational performance of each matrix size, and the breakdown of the processing time at $M \approx 2000$, respectively. Figure 1 shows

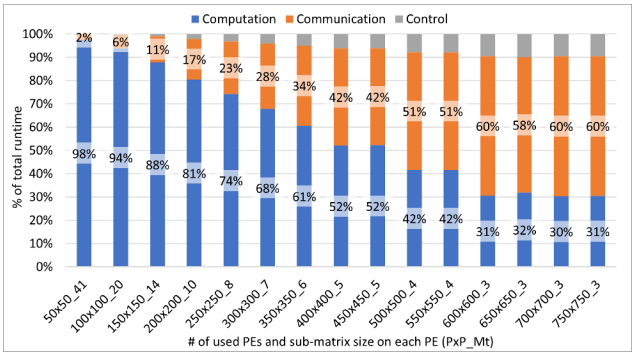

Fig. 2: Strong scaling: Performance breakdown of the processing time at $M \approx 2000$.

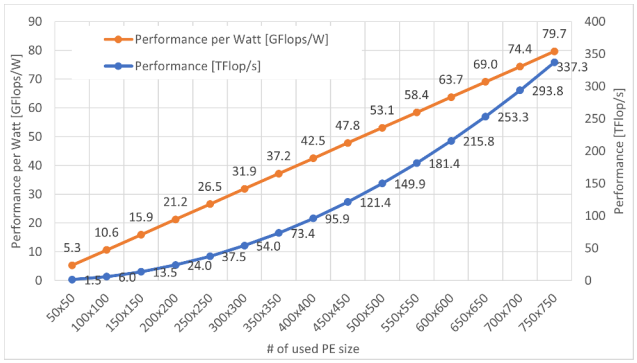

Fig. 3: Weak scaling: computational and performance per watt at $Mt = 47$.

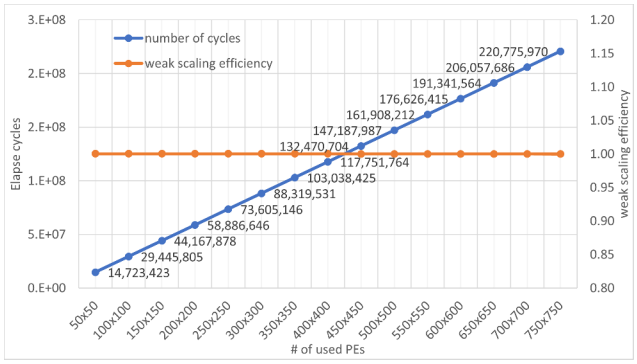

Fig. 4: Weak scaling: Elapsed cycles and weak scaling efficiency per watt at $Mt = 47$.

that the computational performance progressively increases when the M becomes larger. A noteworthy observation in Figure 2 is that the communication still accounts for 60% even when $Mt = 3$ which is extremely small. When the number of PEs is increased while $Mt = 3$, the ratio of calculation to communication remains almost the same. This is because communication between PEs is one cycle. In the general case for other supercomputers, communication latency dominates total processing time, and computational performance asymptotes as Mt becomes smaller. The effective performance decreases as M becomes smaller since the execution efficiency of FMAC operations decreases as Mt becomes smaller as explained in Section III-A, and the ratio of communication to the total time increases. Lastly, the computation and communication time becomes smaller when Mt is smaller, and thus the control cost becomes relatively visible.

C. Weak scaling

For weak scaling measurement, the computational performance, power efficiency (GFlops/W), and parallelisation efficiency were measured while keeping Mt equal to 47 and increasing the number of PEs used. As with strong scaling, the number of PEs used was varied in increments of 50 from 50×50 to 750×750. The maximum matrix size that can be computed on CS-2 is 47×750 = 35250.

Figure 3 shows the computational performance and performance per watt. Figure 4 shows the elapsed cycles and weak scaling efficiency. Computational performance increased linearly with the number of PEs while achieving a weak scaling efficiency of 1.00. Weak scaling efficiency is calculated as follows. When the number of PEs used is increased from 50×50 to 100×100, the amount of computation increases by a factor of 8 while the number of used PEs increases by a factor of 4. Consequently, the elapsed cycles doubled when the parallel efficiency is 1.00. From Figure 4, when the problem size is 2^3 times bigger, and the number of PEs used is 2^2 times larger, the elapsed cycle exactly doubles. From this result, the weak scaling efficiency is calculated as 1.00.

The performance per watt is improved linearly with the number of PEs used, up to 79.66 GFlops/W. It can be considered a very high effective number because the power-to-performance ratio calculated from the specification is approximately 90.00 GFlops/W. Note that the ratio of computation and communication to total processing time was 98% and 2%, respectively, in all cases. The computational performance of each PE can be calculated as 0.6 GFlops/s when dividing the computational performance by the number of PEs used.

IV. Performance Model

Our fundamental idea for creating a performance model is to sum the time required for each processing step (x-/y-axis broadcast and FMAC) because each step runs sequentially.

From preliminary evaluations [9], a model for elapsed cycles of FMAC operations (T_{fmac}) and broadcast communication (T_{bcast}) can be expressed as $T_{fmac} = I \times 2 + 10$ and $T_{bcast} = I \times 1.60 + 250$, respectively, where I is the number of elements processed consecutively. The absolute error between the T_{fmac} and the measured values was at most 2 cycles in the range from 1 to 2048 elements. The mean absolute percentage error (MAPE) between the performance model and the actual measurement was 3.6%. A possible reason for the very high accuracy of the model is that the CU is an in-order pipeline, and all memory access latency is 1 cycle. MAPE between the

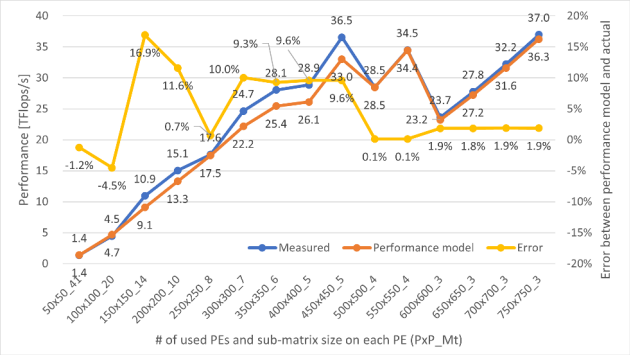

Fig. 5: Estimated performance by performance model, actual measured performance, and error in estimated performance relative to actual performance at $M \approx 2000$

T_{bcast} and the actual measurement was 2.0% in the range of 1 to 4096 transferred elements. The constant number of 250 cycles is considered an overhead for router configuration, etc.

The total elapsed cycles (T_{total}) becomes as follows, where T_{comp}, T_{comm}, and T_{cntl}, P are cycles for computation, communication, control overhead, and the number of used PE in the x-axis, respectively. We found that the T_{comp} takes longer than the FMAC operations with $Mt \times Mt$ elements. In particular, the function *@increment_dsd_offset* calculating a start address for DSD operation took too long to ignore.

$$\begin{aligned} T_{total} &= \{T_{comp} + T_{comm} + T_{cntl}\} \times P \\ &= \{(T_{fmac} + 0.5 \times N_{loop} + 10) + T_{bcast} + 100\} \times P \end{aligned}$$

Finally, the computational performance model of matrix multiplication F_{perf} [Flops/s] becomes follows.

$$F_{perf} = \frac{2 \times M^3 \times (850 \times 10^6)}{\{(T_{fmac} + 0.5 \times N_{loop} + 10) + T_{bcast} + 100\} \times P}$$

Figure 5 shows the estimated performance by the performance model, the actual measured performance, and the error in the estimated performance relative to the actual performance at $M \approx 2000$. Our performance model estimates actual performance very accurately, with a MAPE of 4.7% at $M \approx 2000$. When the sub-matrix size is larger than five, the error becomes large. The main reason for this is that the actual time of two broadcasts in the x-/y- direction differed from the estimated T_{bcast} as the number of transferred elements increased. We assume that as the broadcast in the two directions cannot be fully overlapped, the difference from the model became larger as the data size increased.

V. Discussion

Several parallel distributed matrix multiplication algorithms have been proposed, such as SUMMA. It has been reported that SUMMA can no longer improve strong scalability on recent supercomputers [1] [10]. The 2.5D algorithm proposed by Solomonik et al. [11] avoids communication and optimises the amount of communication compared with SUMMA by replicating input matrices. However, it requires extra memory for replication. There are more advanced algorithms, such as COSMA [10], and CARMA [12], which optimise load balancing and the amount of communication even for tall-skinny matrices. These algorithms can be combined with communication overlapping techniques through blocking. However, communication overlapping does not work well when the number of nodes is increased for strong scaling. This is because the computation time becomes shorter, and the communication time cannot be hidden [13] [10].

The result in Section III suggests that the CS-2 can maintain scalability without using advanced algorithms like other supercomputers. This is because the broadcast in the x-/y-direction of SUMMA is best suited to the 2D mesh topology of CS-2, and the low-latency inter-PE communication of CS-2 minimises the impact of distant communication. SUMMA gives optimal communication for certain matrix shapes, such as square matrices, when there is no extra memory.It also perfectly balances the load for any matrix dimension. Even if the amount of communication is reduced using the 2.5D algorithm or communication is overlapped by blocking, no significant performance improvement can be expected from the results of Figure 2. Both techniques require extra memory; thus, the negative impact is more significant for CS-2, where the local memory is small. Please recall that each PE only has 48KB of local memory, and our evaluation uses more than 80% of them at most.

VI. Conclusion

This paper aimed to evaluate the performance of the single-precision matrix multiplication and model its performance to understand the applicability of CS-2 to large-scale scientific computations. Our evaluation result showed good strong scalability, including the case where the sub-matrix size on each PE was extremely small (3×3). Communication took up 60% of the total processing time in this case. Highly accurate performance models for FMAC operations, broadcast communications and matrix multiplication were also presented.

References

[1] D. Mukunoki and T. Imamura, "Performance analysis of 2d-compatible 2.5d-pdgemm on knights landing cluster," in *Computational Science – ICCS 2018*, Y. Shi, H. Fu, Y. Tian, V. V. Krzhizhanovskaya, M. H. Lees, J. Dongarra, and P. M. A. Sloot, Eds. Cham: Springer International Publishing, 2018, pp. 853–858.

[2] M. Orenes-Vera, I. Sharapov, R. Schreiber, M. Jacquelin, P. Vandermersch, and S. Chetlur, "Wafer-scale fast fourier transforms," in *Proceedings of the 37th ACM International Conference on Supercomputing*, ser. ICS '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 180–191. [Online]. Available: https://doi.org/10.1145/3577193.3593708

[3] M. Jacquelin, M. Araya-Polo, and J. Meng, "Scalable distributed high-order stencil computations," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '22. IEEE Press, 2022.

[4] K. Rocki, D. Van Essendelft, I. Sharapov, R. Schreiber, M. Morrison, V. Kibardin, A. Portnoy, J. F. Dietiker, M. Syamlal, and M. James, "Fast stencil-code computation on a wafer-scale processor," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '20. IEEE Press, 2020.

[5] Cerebras Systems, "Documentation for Developing with CSL - SDK Documentation (1.0.0)," https://sdk.cerebras.net, accessed: 2024-02-05.
[6] R. A. van de Geijn and J. Watts, "Summa: Scalable universal matrix multiplication algorithm," USA, Tech. Rep., 1995.
[7] E. Solomonik and J. Demmel, "Matrix multiplication on multidimensional torus networks," in *High Performance Computing for Computational Science - VECPAR 2012*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 201–215.
[8] Cerebras Systems, "GEMM with Collective Operations - SDK Documentation (1.0.0)," https://sdk.cerebras.net/csl/code-examples/benchmark-gemm-collectives, accessed: 2024-02-05.
[9] T. Miyajima, R. Matsuzaki, and L. Fukuoka, "Stream benchmark on cerebras wse-2 (poster)," in *ISC High Performance 2024 Research Paper Proceedings (39th International Conference)*, no. 10, mar 2024.
[10] G. Kwasniewski, M. Kabić, M. Besta, J. VandeVondele, R. Solcà, and T. Hoefler, "Red-blue pebbling revisited: near optimal parallel matrix-matrix multiplication," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3295500.3356181
[11] E. Solomonik and J. Demmel, "Communication-optimal parallel 2.5d matrix multiplication and lu factorization algorithms," in *Euro-Par 2011 Parallel Processing*, E. Jeannot, R. Namyst, and J. Roman, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 90–109.
[12] J. Demmel, D. Eliahu, A. Fox, S. Kamil, B. Lipshitz, O. Schwartz, and O. Spillinger, "Communication-optimal parallel recursive rectangular matrix multiplication," in *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, 2013, pp. 261–272.
[13] E. Georganas, J. Gonzalez-Dominguez, E. Solomonik, Y. Zheng, J. Tourino, and K. Yelick, "Communication avoiding and overlapping for numerical linear algebra," in *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2012, pp. 1–11.

ACKNOWLEDGEMENT

This work was supported by Japan Science and Technology Agency (JST) as part of Adopting Sustainable Partnerships for Innovative Research Ecosystem (ASPIRE), Grant Number JPMJAP2341. This work was supported by JSPS KAKENHI Grant Number 24K14972.