General Matrix Multiply on CS-2

Qinglei Cao[1,2]

[1]Cerebras Systems, US

[2]{*qinglei.cao*}@*cerebras.net*

I. The Cerebras CS-2

The CS-2 is a system solution that consists of innovations across three dimensions: a) the second generation Cerebras Wafer Scale Engine (WSE-2) — the industry's largest and only multi-trilliontransistor processor, b) the Cerebras System and c) the Cerebras software platform. WSE-2 is the processor at the heart of the CS-2, which is the largest chip ever built, as demonstrated in Figure 1. It is the industry's only multi-trillion transistor processor, and contains more cores, more local memory, and more fabric bandwidth than any chip in history. This enables fast, flexible computation at lower latency and with less energy.

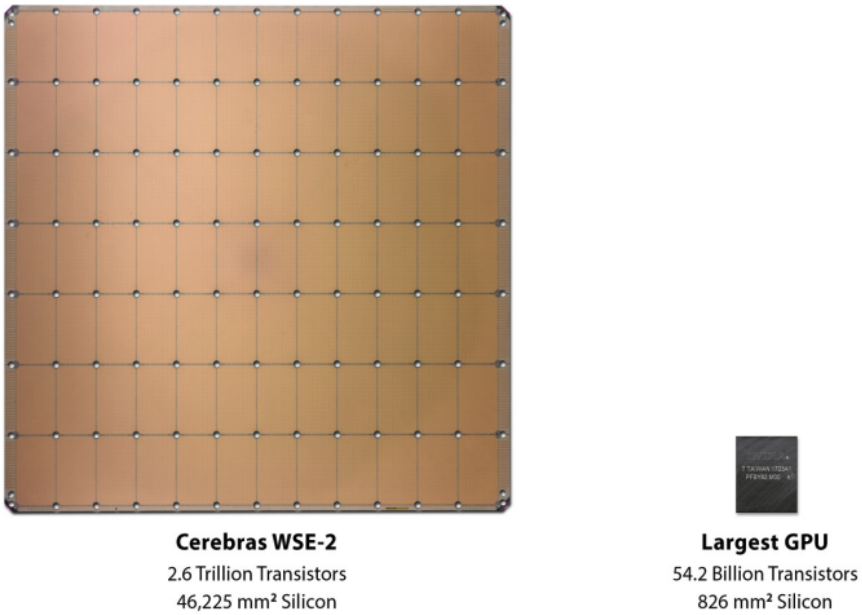


Fig. 1: The Cerebras WSE-2 and the largest Graphics Processing Unit in comparison.

	Cerebras WSE-2	A100	Cerebras Advantage
Chip size	46,225 mm²	826 mm²	**56 X**
Cores	850,000	6,912 + 432	**123 X**
On chip memory	40 Gigabytes	40 Megabytes	**1,000 X**
Memory bandwidth	20 Petabytes/sec	1,555 Gigabytes/sec	**12,862 X**
Fabric bandwidth	220 Petabits/sec	600 Gigabytes/sec	**45,833 X**

Fig. 2: Overview of the magnitude of advancement made by the Cerebras WSE-2

The WSE-2 covers 46,255 square millimeters — 56 times larger than the largest graphics processing unit. With 850,000 cores, 40 Gigabytes of on-chip SRAM, 20 petabytes/sec of memory bandwidth, and 220 petabits/sec of interconnect bandwidth, the WSE-2 contains 123 times more compute cores, 1,000 times more high-speed on-chip memory, 12,862 times more memory bandwidth and 45,833 times more fabric bandwidth than its graphics processing competitor. In effect, it provides the compute capacity of an entire cluster in a single chip, without the cost, complexity, and bandwidth bottlenecks involved with lashing together hundreds of smaller devices. A summary of the comparison is shown in Figure 2.

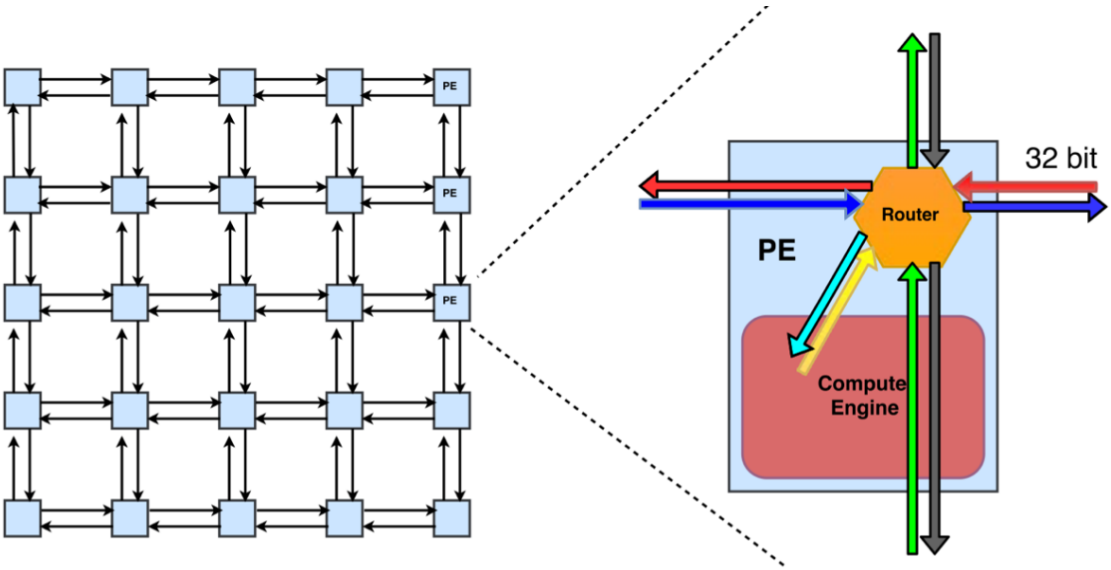


Fig. 3: WSE-2 organizations

More details, the Cerebras Swarm communication fabric creates a massive on-chip network that delivers breakthrough bandwidth and low latency, at a fraction of the power draw of traditional communication techniques that are used to aggregate servers of graphics processing units into large clusters. Swarm connects all 850,000 cores on the Cerebras WSE-2 in a 2D-mesh with 220 Petabits/sec of bandwidth (Figure 3 left). Swarm provides a hardware routing engine to each of the cores and connects them with short wires optimized for bandwidth and low-latency. The resulting fabric supports single-word active messages that can be received by the cores without any software overhead, providing flexible, allhardware communication. Each core is called processing element (PE), which is consist of a fabric router and a compute engine (CE), as shown in Figure 3 (right).

The router is a 5-port switch, with 1 port for each of the cardinal directions (North, South, East, and West), and one port for the CE. The fabric router supports 24 virtual channels, called colors. The switching unit is a wavelet, which consists of 32 data bits, one control bit, and 5 color bits. Wavelets are separated into two categories: control wavelets and data wavelets. The fabric can receive a single wavelet per cycle from each of the 5 router ports and can send one wavelet per cycle on each of the 5 router ports. The connection from fabric to compute engine is referred to as the “offramp”, and the connection from CE to fabric is called the ”onramp”. Sitting at the boundary between the CE and the fabric are a set of queues and filters:

- four filters, which can be used to selectively discard wavelets on the offramp;
- eight input queues, each of which is associated with a fabric color and serves as an extension of the fabric queue from the perspective of the CE;
- six output queues, holding wavelets generated by the CE until the fabric can accept them.

A CE is a simple compute core that runs instructions from memory. Tasks are the primary unit of programming with only one task executing at a time. There are two broad categories of tasks: wavelet-triggered tasks (WTTs) and local tasks. The scheduler can only choose to run tasks that are both unblocked and activated.

- Wavelet-triggered tasks are activated by the arrival of wavelets, whether it be data wavelets or control wavelets. Data wavelets activate the data task associated with their color, while control wavelets activate the control tasks associated with the entry-point they store in bits 0 to 5 of the index field.
- Local tasks, on the other hand, can be activated by microthreads on completion or by FIFOs

on pop or push. They can also be activated using the *actvt* instruction.

Instructions that operate on vectors, which are described using data-structure registers (DSRs), are one of the important hardware features. There are three main modes of execution: normal mode, single-step mode, and microthread mode.

- In normal mode, vector instructions run until they complete.
- Single-step mode is similar to normal, but vector operands can be looped over so operations that require multiple instructions per-element can be executed.
- Microthread mode is quite different different and is a method of processing vectors that are received or transmitted on the fabric in a manner which does not require the processor to stall waiting for inputs or outputs to become available. A microthreaded instruction is launched as part of a normal task, which then executes until it runs out of resources. If it is unable to complete execution, it then effectively forks itself off to become an independent single-instruction task. Microthreads can be run concurrently with other tasks.

Memory is a key component of every computer architecture. Memory closer to compute translates to faster calculation, lower latency, and better power efficiency for data movement. The WSE-2 has 40 Gigabytes of on-chip memory, all uniformly distributed alongside the cores, and 20 Petabytes/sec of memory bandwidth, i.e., each PE holds 48KB on-chip memory and reads/writes at the rate of 32 bits/cycle.

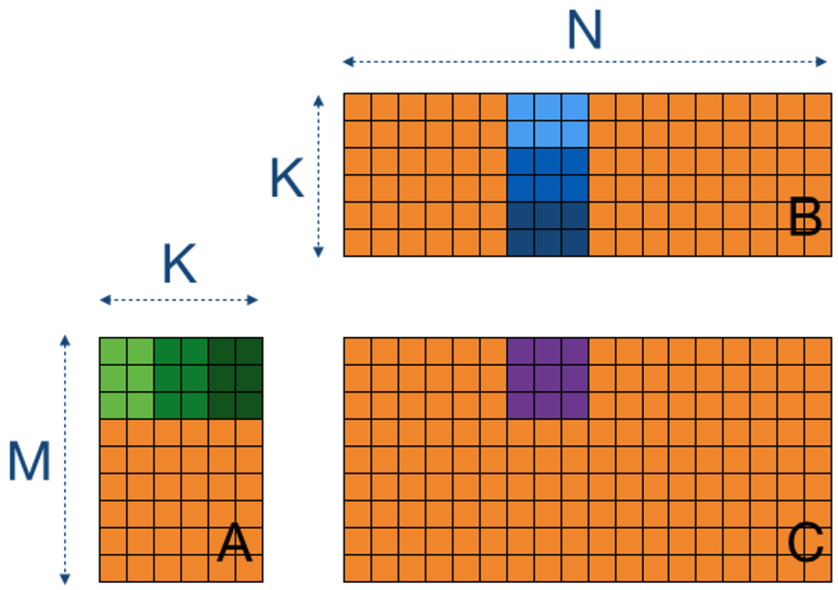
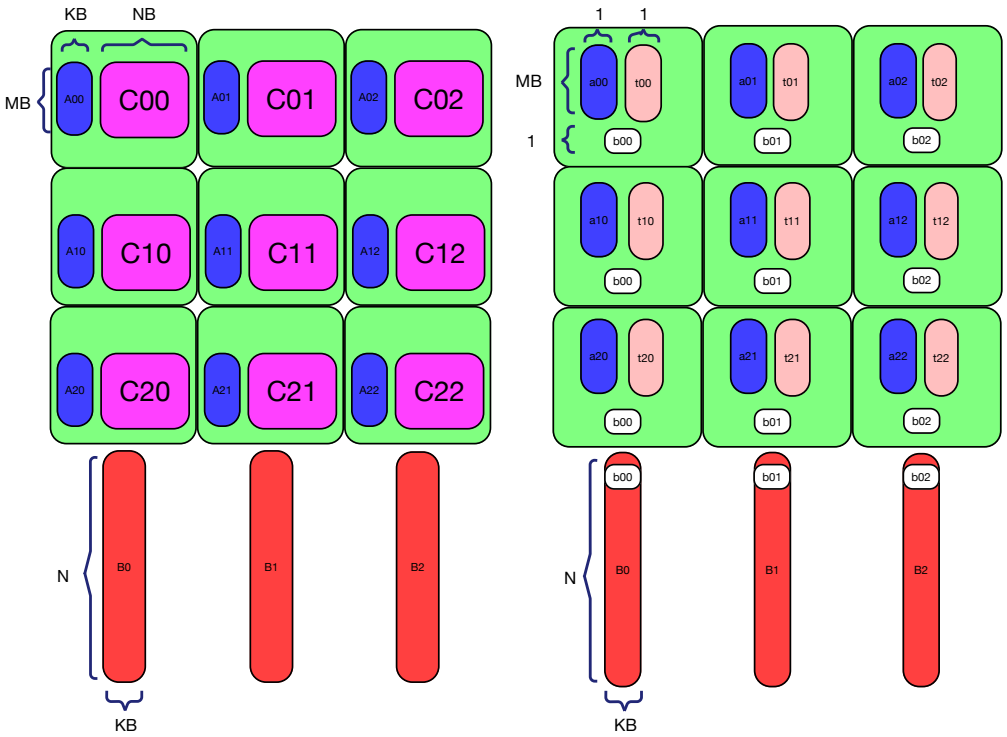


Fig. 4: GEMM example

II. General Matrix Multiply

General Matrix Multiply (GEMM) is a common but widely-used algorithm in linear algebra, machine learning, statistics, and many other domains. It is a level-3 routine in Blas Linear Algebra Subprograms (BLAS) [1], which is compute intensive with $O(N^3)$ flops and $O(N^2)$ data movement, where N is the matrix size. Figure 4 demonstrates an example of $C_{M\times N} = A_{M\times K}B_{K\times N}$. Here, a special scenario is considered, called `MatMul` kernel in weight streaming (WS). It is a variant of GEMM, where one of the inputs (A and B) is streamed in from fabric (one feature for WSE-2). Without loss of generality, let B be streamed in from fabric as weight and A and C be stored in memory as activation, to remove the transpose in the real implementation. Two algorithms are compared, i.e., the method used in the current master (called `MASTER` hereinafter) and Scalable Universal Matrix Multiplication Algorithm (`SUMMA` [2]).



(a) Data distribution.

(b) The first iteration.

Fig. 5: `MASTER`.

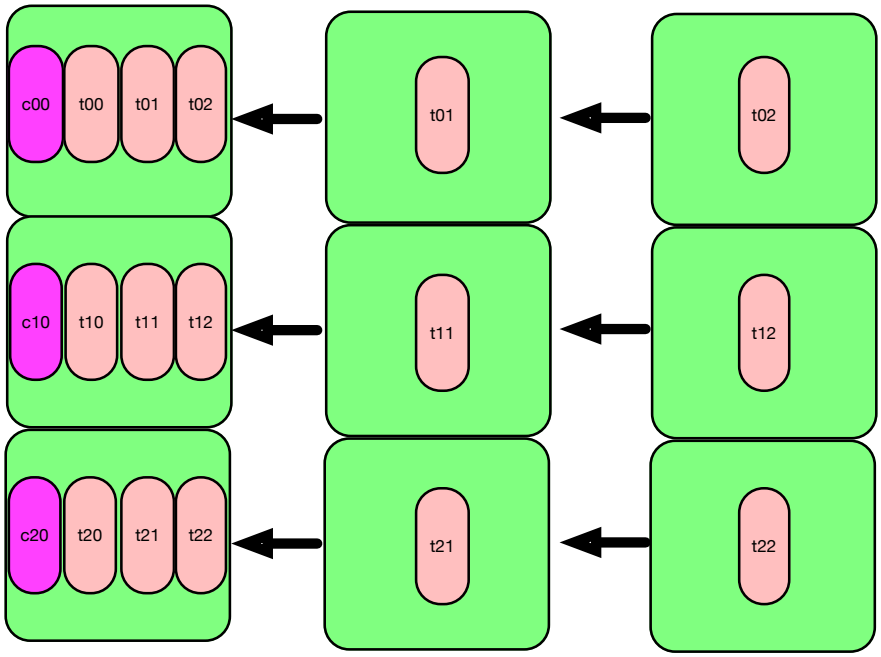


Fig. 6: Row reduction, reducing from the temporary buffer to the target column in C

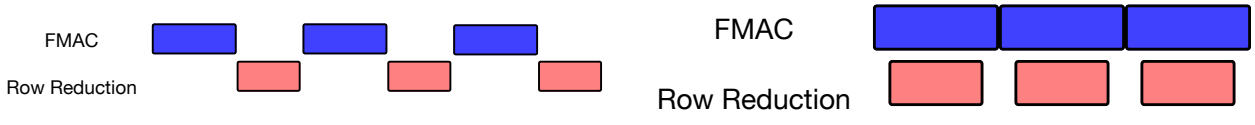
III. MASTER

A. Implementation

Figure 5a depicts how the data is distributed across PEs in MASTER, where A and C are distributed on 3×3 PEs ($PE_x = PE_y = 3$), $PE_x \times KB = K$, $PE_x \times NB = N$, and $PE_y \times MB = M$. The way what MASTER does is to calculate the final result in C one column by one column. Take the calculation of the first column in C for instance, as shown in Figure 5b. One column in B is streamed in so that each PE receives a vector of size KB. Then, $MB \times KB$ number of FMAC instructions (floating multiply-add) are operated locally, which output results to temporary buffers, holding the partial of the final results. Therefore, a row reduction is required to generate the final result of the first column in C, as shown in Figure 6 which reduces $t0*$, $t*$ and $t2*$ to the corresponding target C column sequentially.

B. Performance Analysis

The execution model of MASTER can be described in Figure 7. From mathematical perspective, $MB \times KB$ number of FMAC instructions are followed by a row reduction along horizontal split across vertical split. However, in practice, many optimizations have been proposed to approach its optimal which overlaps FMACs and row reductions, including asynchronous execution



(a) Mathematical

(b) Optimal

Fig. 7: Execution of `MASTER`

(a) Flat ring.

(b) Jump ring.

Fig. 8: Communication pattern.

(`Async_execution`) and double buffering. From the section above, it can be seen that each iteration, i.e., calculation of each column in C, is independent, as well as each row of PEs. Therefore, if only considering one iteration of the one PE which operates $FMAC$, the number of FMAC is $nb\_fmac = MB \times KB$, followed by $nb\_fsum = MB \times PE_x$ number of reduction (FSUM, floating sum). Take `Async_execution` for instance, whose purpose is to hide the overhead of row reduction by FMAC. This `Async_execution` has two aspects: (1) executing FMACs and FSUMs asynchronously (2) executing FSUMs of different PEs asynchronously. Therefore, if the communication pattern in row reduction is jump ring (see Figure 8) and the data is operated in half-precision, i.e., FP16 (SIMD = 4),

- the number of cycles of FMAC: $cycle\_fmac = nb\_fmac/4$
- the number of cycles of FSUM: $cycle\_fsum = 1 + 3 \times PE_x + MB/4$ (2 cycles to send data to its dependent and 1 cycle of FSUM, pipelined)

Hence, if discarding the potential overlap of FSUM between iterations, the efficiency it can achieve in theory is $\frac{cycle\_fma}{max(cycle\_fmac, cycle\_fsum) + MB/4}$, i.e.,

$$efficiency = \frac{MB \times KB}{max(MB \times KB, 4 + 12 \times PE_x) + MB} \tag{1}$$

(Data movement can be dealt by microthread asynchronously, while FMAC and FSUM are operated by main thread.)

If FSUMs between iterations are perfectly overlapped, then theoretical efficiency is:

$$efficiency = \frac{MB \times KB \times N}{(MB \times KB + MB) \times N + 12 \times PE_x} \tag{2}$$

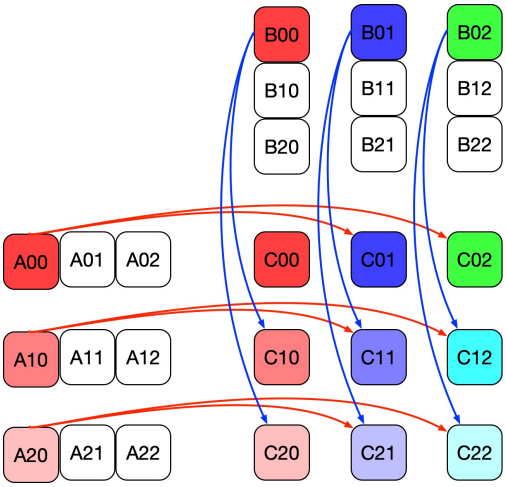
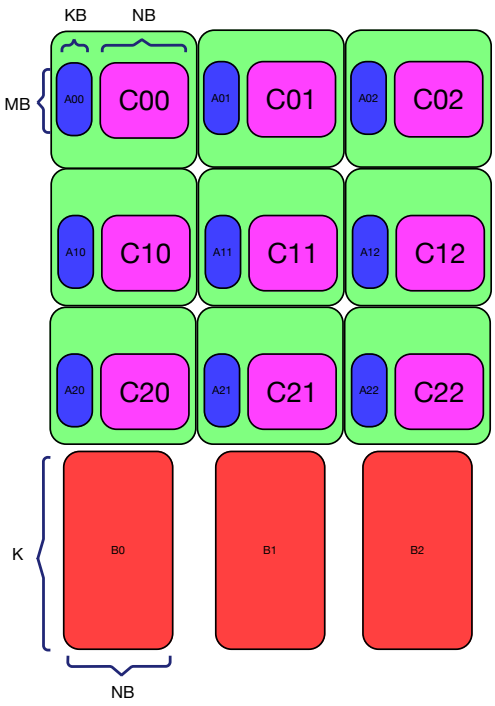


Fig. 9: Communication of SUMMA of the first iteration.

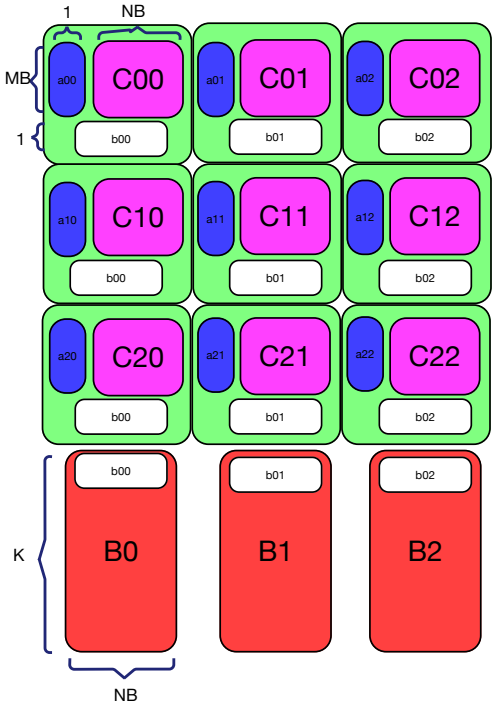
IV. SUMMA

A. Implementation

SUMMA is one of the classical algorithms to solve matrix multiply problem [2]. Figure 9

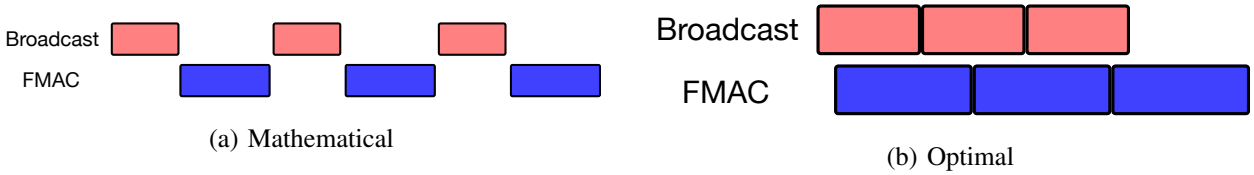


(a) Data distribution.



(b) The first iteration.

Fig. 10: SUMMA.



(a) Mathematical

(b) Optimal

Fig. 11: Execution of SUMMA

shows the communication of the first iteration of SUMMA on 3×3 PEs/processes/blocks, which broadcasts A horizontally, broadcasts B vertically, and operates outer-produce between A and B. It can be seen that SUMMA consists of K number of outer-product, and the vector sizes of each outer-product are M and N. Towards the special scenario where B is streamed in, Figure 10a describes how the data is distributed, and Figure 10b shows the process of the first iteration, where A and C are distributed on 3×3 PEs ($PE_x = PE_y = 3$), $PE_x \times KB = K$, $PE_x \times NB = N$, and $PE_y \times MB = M$.

B. Performance Analysis

The execution model of SUMMA can be described in Figure 11 regarding its mathematical definition and the optimal goal. As shown above, it can be seen: (1) each outer-product is independent; (2) the tile at the top right corner is the last to execute. Therefore, if considering the first iteration of that PE in FP16, the number of FMAC is $nb\_fmac = MB \times NB$, using $nb\_fmac/4$ cycles. For the communication overhead, that PE needs $PE_x + MB/2$ cycles (pipelined) to receive one piece of A from West and $PE_y + NB/2$ cycles (pipelined) for the weight (B) from South.

First, let's see the efficiency that the mathematical definition can achieve. If the two broadcasts of A and B can not be overlapped, then, the theoretically efficiency is

$$efficiency = MB \times NB/(MB \times NB + 4 \times PE_x + 2 \times MB + 4 \times PE_y + 2 \times NB) \quad (3)$$

If that two broadcasts can be overlapped, then

$$efficiency = MB \times NB/(MB \times NB + max(4 \times PE_x + 2 \times MB, 4 \times PE_y + 2 \times NB)) \quad (4)$$

Next, let's consider the optimal one, where FMAC and the two broadcasts can be overlapped, then

$$efficiency = MB \times NB/max(MB \times NB, 4 \times PE_x + 2 \times MB, 4 \times PE_y + 2 \times NB) \quad (5)$$

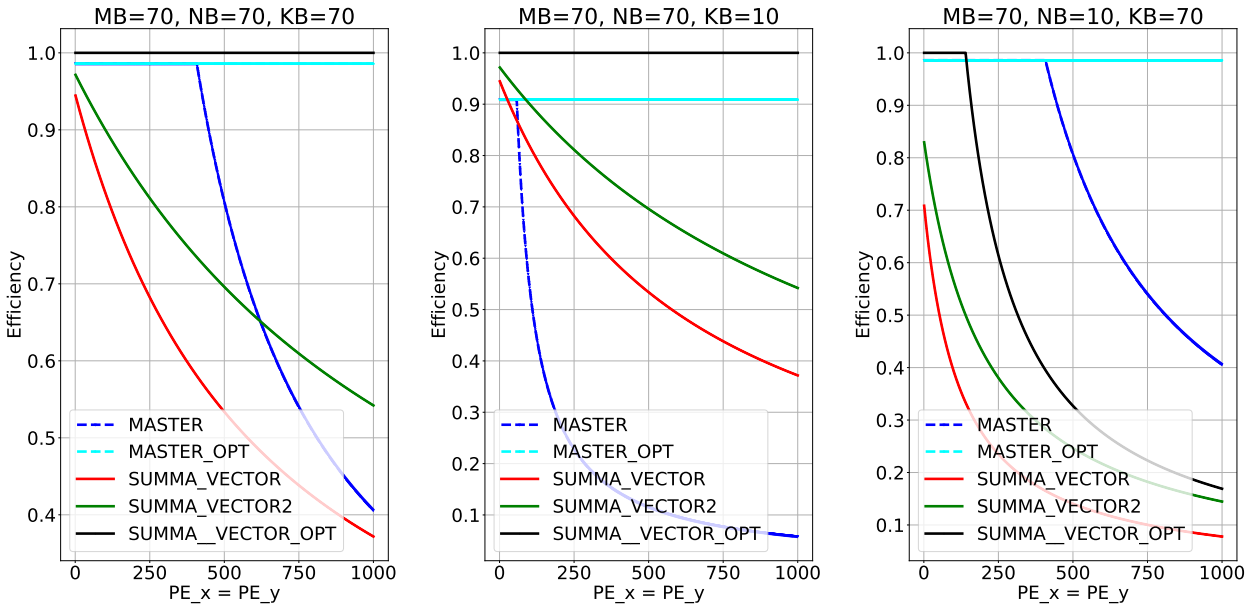


Fig. 12: Performance simulation. "MASTER": Equation 1; "MASTER_OPT": Equation 2; "SUMMA_VECTOR": Equation 3; "SUMMA_VERTOR2": Equation 4; "SUMMA_VECTOR_OPT": Equation 5.

V. Experiments

First, we simulate the theoretical performance `MASTER` and `SUMMA` based on the equations above, as shown in Figure 12 on three scenarios which varies NB and KB (10 and 70) but keeps $MB = 70$. We can see that, in theory, `MASTER` and `SUMMA` suits different scenarios while with the observation that on full scale wafer, (1)"MASTER_OPT" is better in the most cases; (2) "SUMMA_VECTOR_OPT" needs intensive computations to overlap the communication.

Next, Table I details some experimental and theoretical performance results. Most of these experiments come from `test_perf` in test_matmul. "Exp" shorts for experimental, and "Theo" shorts for theoretical. Numbers in bold are scenarios that `SUMMA` is better than `MASTER`. From this table, we can see that

- the theoretical bound of `MASTER` possibly is between Equation 1 and Equation 2;
- the theoretical bound of `SUMMA` without optimization possibly is between Equation 3 and Equation 4;
- these results roughly follow the trends of their theoretical bounds.

TABLE I: Performance results.

K (hin)	N (hout)	M (batch)	PE_x (PE_y)	Exp MASTER (cycles)	Exp SUMMA (cycles)	Exp Ratio MASTER/SUMMA	Theo Eq1 MASTER (cycles)	Theo Eq2 MASTER (cycles)	Theo Eq3 SUMMA (cycles)	Theo Eq4 SUMMA (cycles)
128	128	128	16	5859	6946	0.84	6656	2352	7168	4608
256	128	128	16	8874	13762	0.64	6656	4400	14336	9216
128	256	128	16	11430	8996	**1.27**	13312	4656	9216	7168
256	256	128	16	17535	17863	0.98	13312	8752	18432	14336
128	128	256	16	8002	10365	0.77	6912	4656	10240	7168
256	128	256	16	12116	20626	0.58	8704	8752	20480	14336
128	256	256	16	15803	14470	**1.09**	13824	9264	14336	11264
256	256	256	16	23982	28816	0.83	17408	17456	28672	22528
128	128	384	16	10830	13875	0.78	7168	6960	13312	9728
256	128	384	16	16969	27636	0.61	13056	13104	26624	19456
128	256	384	16	21469	20022	**1.07**	15872	13872	19456	15872
256	256	384	16	33729	39924	0.84	26112	26160	38912	31744
40	40	40	10	1289	1215	**1.06**	1320	230	1120	640
80	80	80	10	3636	3848	0.94	2720	1470	3520	2400
720	20	720	10	27517	114820	0.23	26280	26310	92160	59040
20	720	720	10	79132	28500	**2.77**	38880	38910	27760	26840

VI. MatMul in Machine Learning

VII. Possible Optimizations towards SUMMA

A. Block instead of Vector

If using block instead of vector, then the number of iteration is PE_x instead of K. Therefore, if still considering the top right corner PE and FP16, in each iteration, the number cycles of FMAC is: $MB*NB*KB/4$; while it needs $PE_x+MB\times KB/4$ cycles to receive A (activation) and $PE_y + NB \times KB/2$ cycles to receive B (weight).

In this case, Equation 3 becomes:

$$MB\times NB\times KB/(MB\times NB\times KB+4\times PE_x+2\times MB\times KB+4\times PE_y+2\times NB\times KB) \quad (6)$$

Equation 4 becomes:

$$MB\times NB\times KB/(MB\times NB\times KB+max(4\times PE_x+2\times MB\times KB, 4\times PE_y+2\times NB\times KB)) \quad (7)$$

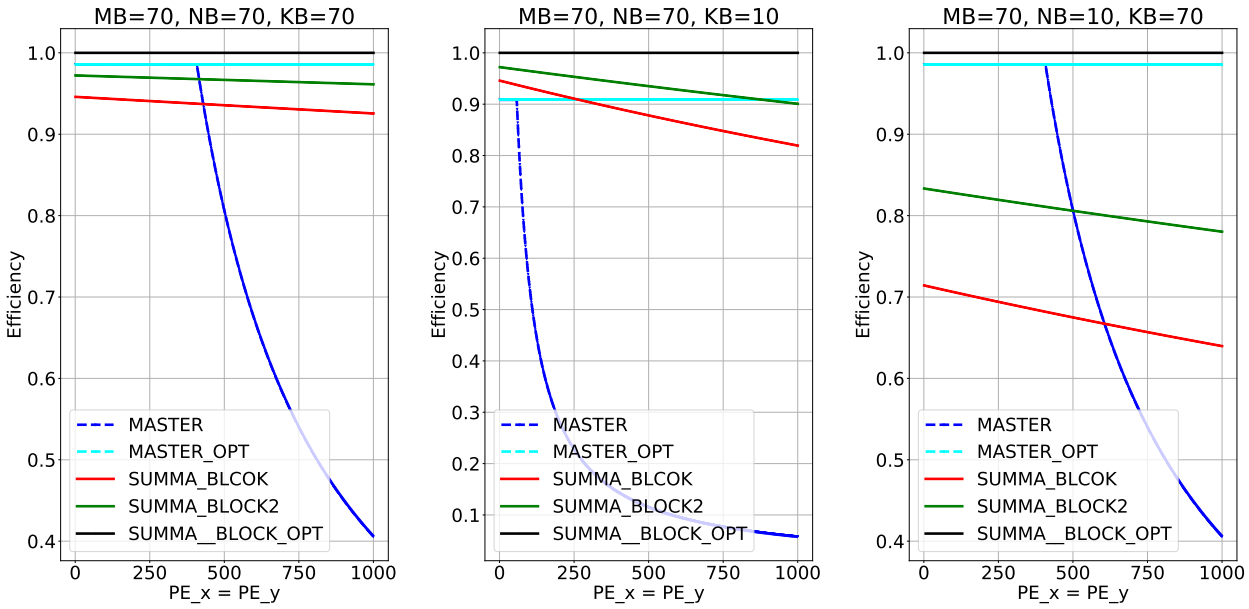


Fig. 13: Performance simulation. "MASTER": Equation 1; "MASTER_OPT": Equation 2; "SUMMA_BLOCK": Equation 6; "SUMMA_BLOCK2": Equation 7; "SUMMA_BLOCK_OPT": Equation 8.

Equation 5 becomes:

$$MB \times NB \times KB / max(MB \times NB \times KB, 4 \times PE_x + 2 \times MB \times KB, 4 \times PE_y + 2 \times NB \times KB) \quad (8)$$

This way relaxes the urgency of the communication and creates more rooms to overlap communication and computation, and Fig. 13 updates Fig. 12 with this optimization, which is called "BLOCK" instead of "VECTOR".

B. Ideal

The ideal `SUMMA` should be (1) communication perfectly by computation (2) pipelined. Therefore, the total number of cycles for a MatMul problem on FP16 is $max(MB \times NB \times K/4, PE_x + MB \times K/2, PE_y + NB \times K/2)$: Equation 5 becomes:

$$MB \times NB \times K / max(MB \times NB \times K, 4 \times PE_x + 2 \times MB \times K, 4 \times PE_y + 2 \times NB \times K) \quad (9)$$

Fig. 14 mimics three scenarios of square matrices with relative small sizes. **However, this is the ideal, and I'm not sure whether it's achievable or not.**

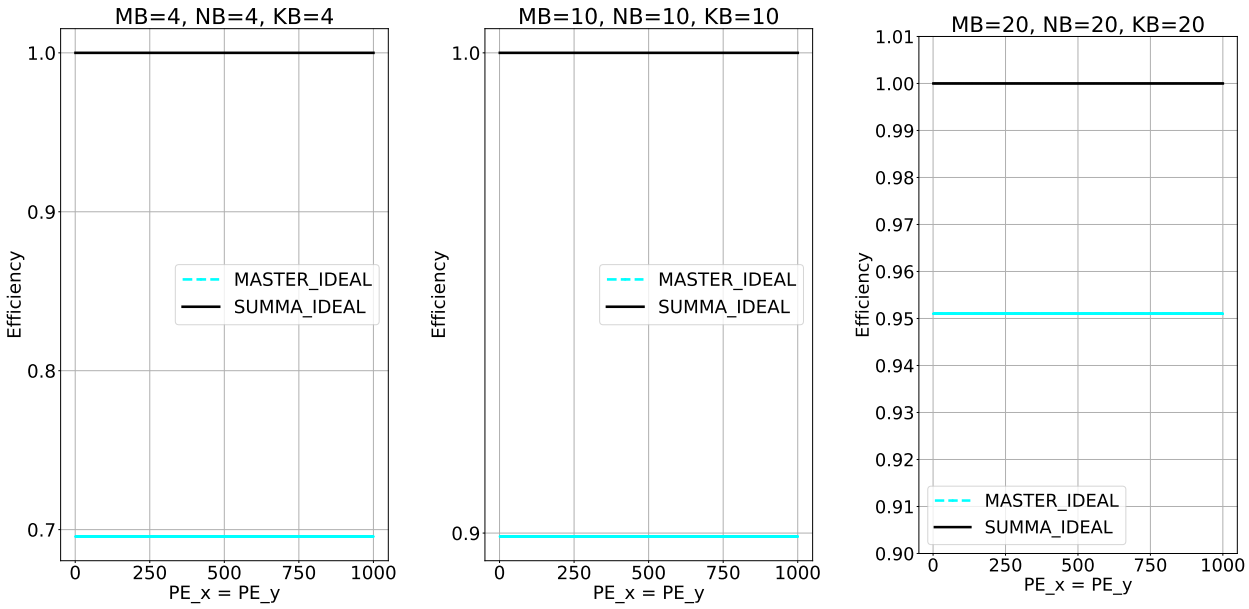


Fig. 14: Performance simulation. “MASTER_IDEAL”: Equation 2; “SUMMA_IDEAL”: Equation 9.

VIII. Summary

References

[1] BLAS. [Online]. Available: https://netlib.org/blas/

[2] R. A. Van De Geijn and J. Watts, “Summa: Scalable universal matrix multiplication algorithm,” *Concurrency: Practice and Experience*, vol. 9, no. 4, pp. 255–274, 1997.