Zhu L, Hua QS, Jin H. Joint-communication optimal matrix multiplication with asymmetric memories. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY, 40(3): 835-854, May 2025. DOI: 10.1007/s11390-023-3489-y, CSTR: 32374.14.s11390-023-3489-y

Joint-Communication Optimal Matrix Multiplication with Asymmetric Memories

Lin Zhu (朱　琳), *Member, CCF*, Qiang-Sheng Hua* (华强胜), *Member, CCF, IEEE* and Hai Jin (金　海), *Fellow, CCF, IEEE, Life Member, ACM*

National Engineering Research Center for Big Data Technology and System, Huazhong University of Science and Technology Wuhan 430074, China

Services Computing Technology and System Laboratory, Huazhong University of Science and Technology, Wuhan 430074 China

Cluster and Grid Computing Laboratory, Huazhong University of Science and Technology, Wuhan 430074, China

E-mail: linzhu@hust.edu.cn; qshua@hust.edu.cn; hjin@hust.edu.cn

Received June 12, 2023; accepted January 5, 2024.

Abstract Emerging hardware like non-volatile memory (NVM) and high-speed network interface cards are promising to improve the performance of matrix multiplication. However, a critical challenge in achieving high performance is the tradeoff between horizontal communication (data movement between processors) and vertical communication (data movement across memory hierarchies). In this paper, we provide an analysis in the distributed memory parallel model with additional consideration for communication between main memory and cache. We measure joint communication as the sum of the horizontal bandwidth and vertical bandwidth cost, and study the joint-communication cost of square matrix multiplication in the read-write symmetric setting (such as DRAM) and asymmetric setting (such as NVM). Specifically, we identify that in the symmetric setting, a joint-communication optimal algorithm can be directly obtained by combining the horizontally optimal and vertically optimal algorithms. We also identify that in the asymmetric setting, horizontal and vertical communications cannot be optimal at the same time, which means that there is a tradeoff between the two communications. In this case, we first present a joint-communication lower bound, and then we propose Joint-Communication Optimal Matrix Multiplication Algorithm (JOMMA), a parallel matrix multiplication algorithm whose joint-communication complexity meets the lower bound. The key idea behind JOMMA is to derive optimal matrix dimensions that each processor locally performs, which leads to determining the processor grid and an optimal schedule.

Keywords distributed algorithm, matrix multiplication, read-write asymmetric memory, vertical communication, horizontal communication

1 Introduction

Matrix multiplication is one of the most fundamental problems in numerical linear algebra, scientific computing, and high-performance computing. The increasing demands in large-scale data storage and fast processing invoke the unique question of how to design efficient parallel matrix multiplication, where the communication cost quickly becomes the bottleneck.

Fortunately, emerging networking and storage technologies, such as InfiniBand and non-volatile memory (NVM), bring a new opportunity to achieve this goal: NVM can provide data persistence while achieving comparable performance and higher density than dynamic random access memory (DRAM). Despite these useful properties, one characteristic of NVM technologies is that writing to memory is more

Regular Paper

The work was supported in part by the National Key Research and Development Program of China under Grant No. 2022ZD0115301, and the National Natural Science Foundation of China under Grant Nos. 61972447 and 61832006.

*Corresponding Author

expensive than reading from it in terms of both time and energy. Additionally, the latest released 50 GB/s InfiniBand network① delivers sub-microsecond latency and extremely high message rates. This work focuses on improving the performance of parallel matrix multiplication with the help of such new devices.

Parallel algorithm execution time is traditionally divided into three components: computation (in-cache floating-point operations), vertical communication (cache-main memory data transfer), and horizontal communication (inter-processor data exchange)[1–3]. When tackling an n-by-n conventional matrix multiplication problem across P processors, the total number of floating-point operations is n^3. Load-balanced parallel algorithms evenly distribute n^3/P floating-point operations to each processor. Since the number of floating-point operations per processor is fixed, we aim to minimize both vertical and horizontal communication to lower parallel algorithm execution time.

Many of the existing parallel matrix multiplication algorithms perform reasonably well in reducing horizontal communication. SUMMA[4] is a widely used algorithm for parallel matrix multiplication, and asymptotically minimizes horizontal communication if assuming no extra memory. The asymptotic horizontal communication lower bounds[5–7] and the memory-independent communication lower bounds[8] with tight constants have been obtained for square matrix multiplication, suggesting that known 2D and 3D[9] algorithms only optimize horizontal communication in certain memory ranges. By efficiently exploiting the available memory, the 2.5D algorithm[10] interpolates between those two results. CA3DMM[11] is a rectangular matrix multiplication parallel algorithm that is designed using a top-down approach and has near-optimal horizontal communication. Horizontal communication, however, is not the only parameter that matters; vertical communication can often be the communication bottleneck, especially in the distributed systems with NVM and InfiniBand network[12–14].

We consider a distributed memory parallel model as described in Section 3, where each processor has a two-level memory hierarchy. In addition to quantifying horizontal communication, we augment the distributed memory model with additional read bandwidth cost (data movement from main memory to cache) and write the bandwidth cost (data movement from cache to main memory) for vertical communication. Furthermore, when main memory is read-write symmetric (such as DRAM), read and write costs are considered equal[15, 16]. When main memory is read-write asymmetric (such as non-volatile memory NVM), since writing is more expensive than reading, the write cost is typically greater than the read cost. The joint-communication cost is measured as the sum of the vertical communication cost and the horizontal communication cost, and we call an algorithm "joint-communication optimal" if it can asymptotically attain the joint-communication lower bound. Similarly, "horizontally optimal" and "vertically optimal" refer to asymptotically minimizing horizontal communication and vertical communication, respectively.

To minimize the joint-communication cost, a natural idea is trying to simultaneously optimize horizontal and vertical communication. We observe that this goal can be achieved in the read-write symmetric setting, by independently employing the horizontally optimal (such as 2.5D) and vertically optimal algorithms. However, when considering more expensive writes in the read-write asymmetric setting, a vertically optimal algorithm entails the write cost reaching its lower bound, and it can be tricky to simultaneously optimize horizontal and vertical communication. Can we asymptotically reduce the joint-communication complexity by exploiting the vertical-horizontal tradeoff? Can they match the best counterparts in the asymmetric setting? These remain to be open problems in the study of joint-communication cost algorithms[15, 17].

In this paper, we provide answers to these questions. We are primarily interested in the read-write asymmetric setting and derive lower bounds for a variety of different cases (cf. Subsection 5.3). We also present JOMMA (Joint-Communication Optimal Matrix Multiplication Algorithm): an algorithm that takes a new approach to multiply two square matrices. JOMMA is joint-communication optimal for all combinations of parameters. The key idea is to derive optimal matrix dimensions of the subproblem that each processor locally processes, and thus find the processor grid and an optimal schedule. This idea comes from our observation that different subproblem matrix dimensions executed by each processor result in different horizontal and vertical communications. We use the following two methods as examples and compare the number of words read, the number of words written, the number of words transferred horizontally, and the main memory requirements in Table 1.

① https://docs.nvidia.com/networking/display/ConnectX7VPI/Introduction, May 2025.

Table 1. Comparison of Word Movement Between Method 1 and Method 2

No.	Matrix Dimension	Horizontal I/O	Vertical Read	Vertical Write	Main Memory Size
1	$n/P^{1/3}$, $n/P^{1/3}$, $n/P^{1/3}$	$O(n^2/P^{2/3})$	$O(n^3/(PS^{1/2}))$	$O(n^2/P^{2/3})$	$\Omega(n^2/P^{2/3})$
2	$n/P^{1/2}$, n, $n/P^{1/2}$	$O(n^2/P^{1/2})$	$O(n^3/(PS^{1/2}))$	$O(n^2/P)$	$\Omega(n^2/P^{1/2})$

Method 1. When using the well-known 3D algorithm to solve an n-by-n square matrix multiplication $\boldsymbol{AB}=\boldsymbol{C}$ on P processors, each processor locally computes one subproblem $\boldsymbol{A}'\boldsymbol{B}'=\boldsymbol{C}'$, where $\boldsymbol{A}'$, $\boldsymbol{B}'$, $\boldsymbol{C}'$ are $(n/P^{1/3})\times(n/P^{1/3})$ submatrices of $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, respectively. To compute the subproblem, each processor needs to access the submatrices $\boldsymbol{A}'$ and $\boldsymbol{B}'$ from other processors, which results in $O(n^2/P^{2/3})$ horizontal communication. In the local calculation of $\boldsymbol{A}'\boldsymbol{B}'=\boldsymbol{C}'$, the cache reads data from main memory to perform the calculation and writes the result back to main memory. Since the size of the output matrix $\boldsymbol{C}'$ is $n^2/P^{2/3}$, $O(n^3/(PS^{1/2}))$ reads and $O(n^2/P^{2/3})$ writes are required to execute each subproblem using Algorithm 1 (S is the cache size)[6]. Note that the size of input and output matrices cannot exceed the main memory size per processor, hence the main memory size is at least $|\boldsymbol{A}'|+|\boldsymbol{B}'|+|\boldsymbol{C}'|=3n^2/P^{2/3}$, i.e., $\Omega(n^2/P^{2/3})$.

Algorithm 1. VOMM

Require: $\boldsymbol{A}\in\mathbb{R}^{d_1\times d_2}$; $\boldsymbol{B}\in\mathbb{R}^{d_2\times d_3}$
Ensure: $\boldsymbol{C}=\boldsymbol{AB}\in\mathbb{R}^{d_1\times d_3}$

```
b = sqrt(S/3)                    ▷ divide A, B, and C into blocks of size b × b
for i = 1 to d1/b do
  for j = 1 to d3/b do
    Initialize C_ij in cache as a b × b zero matrix
    for k = 1 to d2/b do
      Load A_ik, B_kj into cache
      C_ij = C_ij + A_ik B_kj
    end for
    Store block C_ij in main memory
  end for
end for
```

Method 2. Considering the subproblem executed by each processor is $\boldsymbol{A}'\boldsymbol{B}'=\boldsymbol{C}'$, where $\boldsymbol{A}'$ is an $(n/P^{1/2})\times n$ submatrix, $\boldsymbol{B}'$ is an $n\times(n/P^{1/2})$ submatrix, and $\boldsymbol{C}'$ is an $(n/P^{1/2})\times(n/P^{1/2})$ submatrix. Then $\boldsymbol{A}'$ and $\boldsymbol{B}'$ are accessed by each processor and $\boldsymbol{C}'$ is written to main memory, hence this approach requires $O(n^2/P^{1/2})$ horizontal communication, $O(n^3/(PS^{1/2}))$ reads (by using Algorithm 1), and $O(n^2/P)$ writes. The main memory size is at least $|\boldsymbol{A}'|+|\boldsymbol{B}'|+|\boldsymbol{C}'|=2n^2/P^{1/2}+n^2/P$, i.e., $\Omega(n^2/P^{1/2})$.

From Table 1, we can see that, compared with Method 1, Method 2 has lower vertical communication at the cost of higher horizontal communication, which exhibits the horizontal-vertical tradeoff. On a parallel machine where writes are expensive, to minimize the joint-communication cost, we can reduce vertical writes in the first place. Conversely, if horizontal transfers are expensive, we reduce inter-processor communication in the first place. In addition, the limitation imposed by the main memory size on matrix dimensions increases the complexity of algorithm design. For example, Method 1 is only applicable when the main memory size is $\Omega(n^2/P^{2/3})$. In this paper, we investigate how to asymptotically minimize the sum of horizontal and vertical costs for all combinations of parameters.

Contributions. Our main contribution is a new algorithm called Joint-Communication Optimal Matrix Multiplication Algorithm, or JOMMA, which minimizes the joint-communication complexity by trading off horizontal and vertical communications. We also prove the first joint-communication lower bound for classical matrix multiplication under the asymmetric memory model in various situations, which indicates that JOMMA is asymptotically joint-communication optimal.

Paper Organization. We first introduce related work in Section 2 and present the system model in Section 3. Next, in Section 4, we demonstrate the simultaneous achievements of horizontal optimality and vertical optimality for symmetric memory. However, in Section 5, we prove that achieving horizontal and vertical optimality simultaneously is not possible and provide the corresponding joint-communication lower bounds. To optimize the joint communication, we derive optimal matrix dimensions that each processor locally performs in Section 6 and propose the JOMMA algorithm in Section 7. In Section 8, we compare the performance of JOMMA with state-of-the-art algorithms. Finally, we conclude our paper in Section 9.

2 Related Work

The most related work on this topic is that of Carson *et al.*[17] which shows that it is impossible to attain lower bounds on both interprocessor communication and writes to local memory. Based on asymmetric memories, Carson *et al.*[17] also gave a horizon-

tally optimal algorithm (2.5DMML3ooL2) and a write-optimal algorithm (SUMMAL3ooL2). However, two issues remain to be solved. The first is whether there is a general lower bound that shows how these two communications tradeoff against one another, and the second is whether there is an algorithm that can exhibit the tradeoff and asymptotically attains the lower bound. In the read-write symmetric setting, to solve the symmetric eigenvalue problem, Solomonik *et al.*[1] proposed a matrix multiplication subroutine by naturally combining horizontally[18] and vertically[19] optimal algorithms. It can be found that this subroutine is joint-communication optimal in the read-write symmetric setting, while it is not optimal in the case of asymmetric memories. Solomonik *et al.*[2] derived the tradeoffs among synchronization, bandwidth, and computational cost, while the authors did not consider data movement between levels of the memory hierarchy.

There is also some work focusing on minimizing the vertical communication of sequential matrix multiplication algorithms under asymmetric memories. Carson *et al.*[17] proposed the "write-avoiding" concept, which minimizes writes without increasing reads. Gu[16] proposed the "write-efficiency" concept for cache-oblivious matrix multiplication, which can reduce the write cost by increasing the reading cost. In our work, we do not merely consider the sequential cost. Instead, we focus on minimizing the joint-cost by studying the tradeoff between writing and horizontal communication.

3 Theoretical Cost Model

We model communication of the distributed memory parallel system[20, 21] as follows. We assume that the system has P processors, which are connected via a fully-connected network. As shown in Fig.1(a) and Fig.1(b), each processor has a two-level memory hierarchy, i.e., a small cache of size S, and a large main memory of size M (symmetric memory DRAM or asymmetric memory NVM). A single processor can only send/receive a message to/from one processor at a time. The data movement in this parallel model can be divided into two categories: vertical data movement across memory hierarchies, or horizontal data movement between processors. For vertical data movement, we define r to be the cost of moving a word between cache and DRAM. In addition, we define r to be the cost of reading a word from NVM to

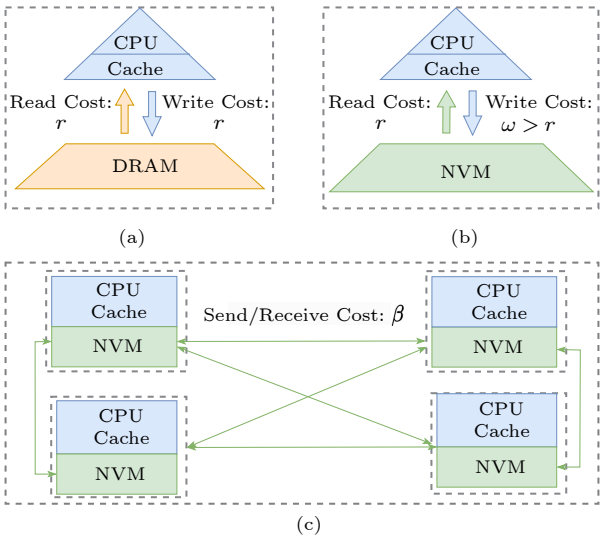

Fig.1. Memory models for sequential and parallel algorithms. (a) Symmetric memory model for sequential algorithms. (b) Asymmetric memory model for sequential algorithms. (c) Distributed asymmetric memory model for parallel algorithms.

cache and ω $(\omega > r)$ to be the cost of writing a word from cache to NVM. For horizontal data movement, we define β to be the cost of moving a word between processors (Fig.1(c)). Note that ω may be greater than β. For example, the write bandwidth of NVM for the existing architecture PilotDB is from 0.2 GB/s to 1.5 GB/s[22], while the network bandwidth of the latest InfiniBand architecture is 50 GB/s. We summarize all the notations and their definitions in Table 2.

We denote the number of words read, written, and transferred horizontally as B_r, B_w, and B_h, respectively. We measure the joint-communication cost Q in terms of the bandwidth (the number of words) along the critical path as defined in [23]. Specifically, we call the symmetric memory parallel model SPM and define $r(B_r + B_w) + \beta B_h$ to be the joint-communication cost in the SPM. We call the asymmetric memory parallel model APM and define $rB_r + \omega B_w + \beta B_h$ to be the joint-communication cost in the APM. Our model is similar to those in [15, 17], while [1] takes into account not only communication but also computation, and [24, 25] only considers horizontal communication. Throughout the paper, we require that the data layout of the input matrices and the output matrix are evenly distributed across the main memory of P processors for load balancing $(M \geqslant 3n^2/P)$.

There are five well-known collective communication operations[26, 27] used extensively in our algorithm.

Table 2. Notations and Their Definitions

Type	Symbol	Definition
Matrix	n	Matrix dimension of the initial matrix multiplication problem
	$d_1,\ d_2,\ d_3$	Matrix dimensions of subproblem solved on a processor
	$\boldsymbol{A}, \boldsymbol{B}$	Input matrices
	$\boldsymbol{C} = \boldsymbol{AB}$	Output matrix
Configuration	P	Number of processors
	M	Size of main memory ($M \geqslant 3n^2/P$)
	S	Size of cache
	r	Cost of moving a word between cache and DRAM
		Cost of reading a word from NVM to cache
	ω	Cost of writing a word from cache to NVM
	β	Cost of moving a word between processors
Cost	B_r	Number of words moved from main memory to cache
	B_w	Number of words moved from cache to main memory
	B_h	Number of words moved between processors
	Q	Joint-communication cost
Schedule	$P_x,\ P_y,\ P_z$	Dimensions of the processor grid
	P_{ijk}	Per processor index
	$P_{ij:}$	Processor group $\{P_{ij1},\ P_{ij2},\ \ldots,\ P_{ijP_z}\}$
	$\boldsymbol{\Pi}(\boldsymbol{A}),\ \boldsymbol{\Pi}(\boldsymbol{B})$	Local submatrices of $\boldsymbol{A}$ and $\boldsymbol{B}$ on a processor

- *Gather*($\boldsymbol{\Pi}(\boldsymbol{A})$, $\boldsymbol{\Pi}(\boldsymbol{X})$, i, $P_{:jk}$): all processors in $P_{:jk}$ contribute local arrays $\boldsymbol{\Pi}(\boldsymbol{A})$ to processor P_{ijk} as the local array $\boldsymbol{\Pi}(\boldsymbol{X})$.
- *Broadcast*($\boldsymbol{\Pi}(\boldsymbol{A})$, $\boldsymbol{\Pi}(\boldsymbol{X})$, k, $P_{ij:}$): root processor P_{ijk} distributes the local array $\boldsymbol{\Pi}(\boldsymbol{A})$ to every processor in $P_{ij:}$ as the local array $\boldsymbol{\Pi}(\boldsymbol{X})$.
- *Allgather*($\boldsymbol{\Pi}(\boldsymbol{A})$, $\boldsymbol{\Pi}(\boldsymbol{X})$, $P_{:jk}$): the local arrays $\boldsymbol{\Pi}(\boldsymbol{A})$ contributed by each processor in $P_{:jk}$ are gathered, and the result is broadcast to all processors P_{ijk} as $\boldsymbol{\Pi}(\boldsymbol{X})$.
- *Reduce*($\boldsymbol{\Pi}(\boldsymbol{A})$, $\boldsymbol{\Pi}(\boldsymbol{X})$, j, $P_{i:k}$): all processors in $P_{i:k}$ contribute local arrays $\boldsymbol{\Pi}(\boldsymbol{A})$ to an element-wise reduction onto root processor P_{ijk} as the local array $\boldsymbol{\Pi}(\boldsymbol{X})$.
- *Shift*($\boldsymbol{\Pi}(\boldsymbol{B})$, s, P_x, $P_{:jk}$): each processor P_{ijk} sends local array $\boldsymbol{\Pi}(\boldsymbol{B})$ to $P_{i'jk}$ via point-to-point communication, where $i' = (i + s) \bmod P_x$.

The costs of these operations can be obtained by a binomial tree or butterfly schedule[27, 28]. We summarize the costs in Table 3, where n words of data are being communicated among P processors.

Table 3. Bandwidth Costs of Communication Operations

Operation	Cost
$T_{\text{Gather}}(n, P)$	βn
$T_{\text{Broadcast}}(n, P)$	$2\beta n$
$T_{\text{Reduce}}(n, P)$	$2\beta n$
$T_{\text{Allgather}}(n, P)$	βn
$T_{\text{Shift}}(n, P)$	βn

4 Joint-Communication in SPM

In the symmetric memory parallel model, by independently employing some well-known horizontally and vertically optimal matrix multiplication algorithms, we give asymptotically tight joint-communication upper and lower bounds in this section.

Lemma 1[18]. *For a parallel square matrix multiplication $\boldsymbol{AB} = \boldsymbol{C}$ solved on P processors, where $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ are $n \times n$ matrices, at least $\Omega(n^3/(PM^{1/2}) + n^2/P^{2/3})$ words need to be moved between processors.*

Many algorithms, such as 2D[4], 2.5D[10], 3D[9], and CARMA[18], can asymptotically attain the horizontal lower bound. 2.5D and CARMA are horizontally optimal for any memory size, while 2D and 3D are optimal for $M = \Theta(n^2/P)$ and $M = \Omega(n^2/P^{2/3})$ respectively. To compute $\boldsymbol{AB} = \boldsymbol{C}$ in parallel, these algorithms usually decompose the initial problem into multiple subproblems $\boldsymbol{A'B'} = \boldsymbol{C'}$ that are executed in parallel on P processors, where $\boldsymbol{A'}$ is a $d_1 \times d_2$ matrix, $\boldsymbol{B'}$ is a $d_2 \times d_3$ matrix, and $\boldsymbol{C'}$ is a $d_1 \times d_3$ matrix. For example, 2D decomposes the n-by-n matrix multiplication problem into $P^{3/2}$ square matrix multiplication subproblems with matrix dimensions $d_1 = d_2 = d_3 = n/P^{1/2}$, and each processor solves $P^{1/2}$ subproblems.

To sequentially solve a matrix multiplication subproblem $\boldsymbol{A'B'} = \boldsymbol{C'}$ on a single processor, where $\boldsymbol{A'} \in \mathbb{R}^{d_1 \times d_2}$, $\boldsymbol{B'} \in \mathbb{R}^{d_2 \times d_3}$, $\boldsymbol{C'} \in \mathbb{R}^{d_1 \times d_3}$, at least

$\Omega(d_1d_2d_3/S^{1/2})$ words need to be moved between cache and main memory[29]. The well-known Vertically Optimal Matrix Multiplication algorithm VOMM is described in Algorithm 1[6]. The number of words read is $2\sqrt{3}d_1d_2d_3/S^{1/2}$ and the number of words written is d_1d_3.

Proposition 1. *For a parallel square matrix multiplication $\boldsymbol{AB}=\boldsymbol{C}$ performed on P processors, where $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ are $n\times n$ matrices, there exists a processor such that the word transfer between cache and memory is at least $\Omega(n^3/(PS^{1/2}))$.*

Proof. For a parallel matrix multiplication $\boldsymbol{AB}=\boldsymbol{C}$ solved on P processors, suppose a processor performs $|V|$ arithmetic operations, then at least $\Omega(|V|/S^{1/2})$ words need to be moved between cache and main memory[6], i.e., $B_r+B_w=\Omega(|V|/S^{1/2})$. Note that the total number of arithmetic operations to compute $\boldsymbol{AB}=\boldsymbol{C}$ is n^3, where $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}\in\mathbb{R}^{n\times n}$. According to the pigeonhole principle, at least one processor performs at least n^3/P arithmetic operations. Thus, for such a processor, we have $B_r+B_w=\Omega(n^3/(PS^{1/2}))$. □

Since the horizontal/vertical costs of an algorithm cannot be asymptotically lower than the horizontal/vertical cost lower bounds, a trivial joint-communication cost lower bound can be obtained by combining the horizontal and the vertical cost lower bounds. In other words, an algorithm is joint-communication optimal if it achieves both the horizontal and the vertical cost lower bounds. As the following Lemma 2 shows, this kind of algorithm can be directly obtained by combining the horizontally optimal algorithm and the sequential VOMM algorithm.

Lemma 2. *In the SPM, a joint-communication optimal algorithm can be obtained by combining the parallel 2.5D algorithm and the sequential VOMM algorithm.*

Proof. From the analysis in [10], to solve $\boldsymbol{A}^{n\times n}\boldsymbol{B}^{n\times n}=\boldsymbol{C}^{n\times n}$ in parallel on P processors, the 2.5D algorithm decomposes it into $P^{3/2}/c^{3/2}$ subproblems with problem size $n\sqrt{c}/\sqrt{P}$, where $c=\min(P^{1/3},\sqrt{PM}/n)$. Each processor handles $P^{3/2}/(Pc^{3/2})$ subproblems and applies the sequential VOMM algorithm to solve each subproblem. According to the analysis in Section 4, the number of words read and written when using VOMM to solve each subproblem is $O((n\sqrt{c}/\sqrt{P})^3/S^{1/2})$. Therefore, for each processor, the total number of words read and written is $(P^{3/2}/(Pc^{3/2}))\times O((n\sqrt{c}/\sqrt{P})^3/S^{1/2})$, i.e., $O(n^3/(P\sqrt{S}))$. From Proposition 1, this vertical communication asymptotically reaches the vertical cost lower bound. Additionally, as the 2.5D algorithm is horizontally optimal, it attains the horizontal communication's cost lower bound. As a result, the proposed algorithm achieves joint-communication optimality by satisfying both horizontal and vertical cost lower bounds. □

For the read-write symmetric memory model, since the costs of writing and reading are both r, $B_w=O(B_r)$ can achieve the asymptotic optimal vertical cost (rB_r+rB_w). However, when writing is much more expensive than reading, only less writing meets the vertical optimum. Therefore, the joint-communication optimal algorithm under the symmetric memory model may not be optimal for asymmetric memories.

5 Joint-Cost Lower Bound in APM

In this section, we discuss the joint-communication cost lower bound in the asymmetric memory model. We first give a lower bound of the vertical cost (Lemma 3). Then we show that the horizontal and the vertical optimalities cannot be achieved simultaneously (Lemma 5), which indicates that the trivial cost lower bound by naturally combining the horizontal cost lower bound and the vertical cost lower bound is not tight. Finally, we derive tighter joint-communication cost lower bounds (Theorem 1 and Theorem 2).

5.1 Vertical Cost Lower Bound

Lemma 3. *In the APM, the vertical communication cost lower bound for square matrix multiplication is $\Omega(n^3r/(PS^{1/2})+n^2\omega/P)$.*

Proof. From Proposition 1, we have $B_r+B_w=\Omega(n^3/(PS^{1/2}))$. As presented in Section 3, in the APM, the vertical cost is defined as $rB_r+\omega B_w$. As $\omega>r$, the vertical cost follows that $rB_r+\omega B_w>r(B_r+B_w)=\Omega(n^3r/(PS^{1/2}))$. Since the output matrix, of size n^2, is eventually evenly distributed among P processors, the minimum number of words written per processor is n^2/P, which implies $B_w\geqslant n^2/P$. Therefore, we have $rB_r+\omega B_w\geqslant\max(n^3r/(PS^{1/2}),n^2\omega/P)$, i.e., $\Omega(n^3r/(PS^{1/2})+n^2\omega/P)$. □

5.2 Vertically Optimal Cannot be Horizontally Optimal

This conclusion is based on the inequality pro-

posed by Loomis and Whitney[30], which describes the surface-to-volume relationship. For n-by-n matrix multiplication, there are n^3 arithmetic operations, which may be arranged into a cube $\mathcal{V}$ of size $n \times n \times n$ with the matrices $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ as its faces. The point at location (i, j, k) in the cube corresponds to the scalar multiplication $A_{ik}B_{kj}$. Let $V \subset \mathcal{V}$ denote the arithmetic operations performed by a processor, then the projections of V onto three faces correspond to the input entries of $\boldsymbol{A}$ and $\boldsymbol{B}$ that are necessary to perform V and the output entries of $\boldsymbol{C}$ which are updated. The Loomis-Whitney inequality relates the volume of V to its projections.

Lemma 4[30]. *Let $V \subset \mathcal{R}^3$ be a finite set of lattice points and each point (x, y, z) with integer coordinate. Let V_x be the orthogonal projection of vector space V onto the $y \times z$ plane, defined as the set of all points (y, z) for which there exists an x such that $(x, y, z) \in V$. The definitions of V_y and V_z are similar. Let $|\cdot|$ denote the cardinality of a set, then $|V| \leqslant \sqrt{|V_x| \times |V_y| \times |V_z|}$.*

Lemma 5. *In the APM, a vertically optimal algorithm cannot be horizontally optimal.*

Proof. To solve the matrix multiplication $\boldsymbol{A}^{n\times n}\boldsymbol{B}^{n\times n} = \boldsymbol{C}^{n\times n}$ on a distributed system with P processors, each processor must have a main memory size of at least $M = \Omega(n^2/P)$ to evenly store the initial input matrices. Given a vertically optimal algorithm, according to Lemma 3, we have $B_w = \Theta(n^2/P)$ for this algorithm. Moreover, by the pigeonhole principle, there exists a processor that performs at least n^3/P arithmetic operations, i.e., $|V| \geqslant n^3/P$. For this processor, since each entry of $\boldsymbol{C}$ in V_z is updated and must be written to NVM at least once, we get $|V_z| \leqslant B_w$, i.e., $O(n^2/P)$. From Lemma 4, we have $|V_x||V_y| \geqslant |V|^2/|V_z|$ and hence $\max\{|V_x|, |V_y|\} = \Omega(n^2/P^{1/2})$, which means that the processor must access at least $\Omega(n^2/P^{1/2})$ entries of $\boldsymbol{A}$ or $\boldsymbol{B}$. Since each processor initially has n^2/P entries of $\boldsymbol{A}$ and $\boldsymbol{B}$, there are at least $\Omega(n^2/P^{1/2})$ entries that need to be accessed by other processors. That is, the horizontal word transfer is at least $\Omega(n^2/P^{1/2})$, which is asymptotically higher than the horizontal word transfer cost lower bound $\Omega(n^3/(PM^{1/2})+n^2/P^{2/3})$ $(M=\Omega(n^2/P))$. Therefore, a vertically optimal algorithm cannot be horizontally optimal. □

Note that Theorem 3 in [17] gives a similar conclusion that a horizontally optimal algorithm cannot be vertically optimal.

5.3 Joint-Cost Lower Bound

By Lemma 1, the number of words moved horizontally is at least $\Omega(n^3/(PM^{1/2}) + n^2/P^{2/3})$. Accordingly, the horizontal cost lower bound can be divided into two scenarios depending on the main memory size.

- When $M = \Omega(n^2/P^{2/3})$, we have $n^3/(PM^{1/2}) = O(n^2/P^{2/3})$ and hence B_h is at least $\Omega(n^2/P^{2/3})$.
- When $M = O(n^2/P^{2/3})$, we have $n^3/(PM^{1/2}) = \Omega(n^2/P^{2/3})$ and hence B_h is at least $\Omega(n^3/(PM^{1/2}))$.

Similarly, we divide the joint-cost lower bound into two scenarios for discussion depending on the main memory size.

- When $M = \Omega(n^2/P^{1/2})$, we call it "enough memory scenario" and discuss this case in Subsection 5.3.1.
- When $M = O(n^2/P^{1/2})$, we call it "limited memory scenario" and discuss this case in Subsection 5.3.2.

Subsection 6.2 explains why this category is chosen.

5.3.1 *Joint-Cost Lower Bound with Enough Memory*

In enough memory scenario $(M = \Omega(n^2/P^{1/2}))$, by Lemma 1, the number of words transferred horizontally is at least $\Omega(n^2/P^{2/3})$. From Lemma 3, the trivial joint-cost lower bound (the sum of the horizontal and vertical lower bounds) is $\Omega((n^2\beta/P^{2/3})+(n^3r/PS^{1/2})+(n^2\omega/P))$. However, as discussed above, this lower bound is loose for asymmetric memories. A tighter lower bound is proven in Theorem 1.

Theorem 1. *Let Q be the joint-communication cost for solving $\boldsymbol{A}^{n\times n}\boldsymbol{B}^{n\times n} = \boldsymbol{C}^{n\times n}$ in the APM with enough memory. When $2\omega/\beta \leqslant 1$, Q has a lower bound of $\Omega((n^2\beta/P^{2/3}) + (n^3r/PS^{1/2}))$. For the range $1 < 2\omega/\beta < P^{1/2}$, the lower bound of Q is $\Omega((n^2\beta^{2/3}\omega^{1/3}/P^{2/3}) + (n^3r/PS^{1/2}))$. If $2\omega/\beta \geqslant P^{1/2}$, then Q has a cost lower bound of $\Omega((n^3r/PS^{1/2})+(n^2\omega/P))$.*

Proof. In the enough memory scenario $(M = \Omega(n^2/P^{1/2}))$, we categorize the cost lower bound into three cases according to the value of $2\omega/\beta$. The value of $2\omega/\beta$ is chosen based on the divide-and-conquer BFS/DFS approach[18]. Briefly, during the recursive decomposition of the initial problem, we trade off the horizontal and vertical costs by setting a parameter d and split the largest of d_1, d_2/d, d_3 (subproblem size) in half at each recursion. By calculation, a joint-cost function concerning d can be obtained, which is minimized at $d = 2\omega/\beta$. Considering that $d \leqslant 1$ is the hor-

izontal optimal scenario and $d \geqslant P^{1/2}$ is the vertical optimal scenario, we discuss the following three cases based on the size of $2\omega/\beta$ and 1, $P^{1/2}$. Interested readers can refer to [18] for more details of the above approach.

1) $2\omega/\beta \leqslant 1$. In this case, by $2\omega/\beta \leqslant 1$, we derive the $\Omega((n^2\beta/P^{2/3}) + (n^3r/PS^{1/2}))$ lower bound directly from the trivial joint-cost lower bound of $\Omega((n^2\beta/P^{2/3}) + (n^3r/PS^{1/2}) + (n^2\omega/P))$.

2) $1 < 2\omega/\beta < P^{1/2}$. By the pigeonhole principle, there exists a processor performing at least $|V| \geqslant n^3/P$ arithmetic operations, which holds for any parallel matrix multiplication algorithm. Without loss of generality, we assume that, for this processor, the number of words written is $B_w = cn^2/P$, where $c > 0$. Subsequently, based on Lemma 3 and $B_w = cn^2/P$, we ascertain that the vertical cost is bounded by $\Omega((rn^3/PS^{1/2}) + (\omega cn^2/P))$. Additionally, for this processor, as each output entry of V_z gets updated and must be written to NVM at least once, we deduce $|V_z| \leqslant B_w = cn^2/P$. Leveraging Lemma 4, we infer that $|V_x| \times |V_y| \geqslant |V|^2/|V_z|$ and thus $\max\{|V_x|, |V_y|\} \geqslant n^2/(cP)^{1/2}$, which means that this processor needs to get at least $n^2/(cP)^{1/2}$ entries of $\boldsymbol{A}$ or $\boldsymbol{B}$. Subsequently, we delineate two cases, contingent upon the value of c.

a) $0 < c < P$. Since each processor initially owns n^2/P entries of $\boldsymbol{A}$ and $\boldsymbol{B}$, for this processor, the number of entries that need to be accessed by other processors is at least $n^2/(cP)^{1/2} - n^2/P$, i.e., $\Omega(n^2/(cP)^{1/2})$. Therefore, $B_h = \Omega(n^2/(cP)^{1/2})$, and the joint-communication cost is

$$Q = \Omega\left(\frac{n^2\beta}{(cP)^{1/2}}) + \frac{n^3r}{PS^{1/2}} + \frac{cn^2\omega}{P}\right).$$

It can be found that Q is a function on c. By derivation, this function is minimized when $c^* = \arg\min Q = P^{1/3}\beta^{2/3}/(2\omega)^{2/3}$. Therefore,

$$Q = \Omega\left(\frac{n^2\beta^{2/3}\omega^{1/3}}{P^{2/3}} + \frac{n^3r}{PS^{1/2}}\right).$$

b) $c \geqslant P$. From $c \geqslant P$, $2\omega/\beta > 1$, and the trivial joint-cost lower bound

$$Q = \Omega\left(\frac{n^2\beta}{P^{2/3}} + \frac{n^3r}{PS^{1/2}} + \frac{cn^2\omega}{P}\right),$$

we have $cn^2\omega/P \geqslant n^2\omega > n^2\beta/2 \geqslant n^2\beta/2P^{2/3}$. Therefore, $Q = \Omega((n^3r/PS^{1/2}) + n^2\omega)$.

Any matrix multiplication algorithm falls into either case a or case b. Thus, we determine the joint-cost lower bound by selecting the minimum value of Q in case a and case b. Since $2\omega/\beta > 1$, we have $n^2\beta^{2/3}\omega^{1/3}/P^{2/3} = O(n^2\omega/P^{2/3}) = O(n^2\omega)$. Therefore, $Q = \Omega((n^2\beta^{2/3}\omega^{1/3}/P^{2/3}) + (n^3r/PS^{1/2}))$.

3) $2\omega/\beta \geqslant P^{1/2}$. In this case, by $2\omega/\beta \geqslant P^{1/2}$, the lower bound of $\Omega((n^3r/PS^{1/2}) + (n^2\omega/P))$ can be derived directly from the trivial joint-cost lower bound of $\Omega((n^2\beta/P^{2/3}) + (n^3r/PS^{1/2}) + (n^2\omega/P))$. □

In the enough memory scenario, the joint-communication costs of the horizontally optimal (2.5DMML3ooL2) and the vertically optimal (SUMMAL3ooL2) algorithms are given in Table 4, which shows that neither algorithm can attain the joint-cost lower bound in general.

5.3.2 *Joint-Cost Lower Bound with Limited Memory*

Note that any matrix multiplication algorithm that can be executed in the limited memory scenario can also be executed in the enough memory scenario. Therefore, the joint-cost lower bound in the enough memory scenario also holds in the limited memory scenario. However, this lower bound might not be tight in the limited memory scenario, and thus more refined analyses are needed. We give a tight joint-cost lower bound in the limited memory scenario in Theorem 2.

Theorem 2. *Let Q be the joint-communication cost for solving $\boldsymbol{A}^{n\times n}\boldsymbol{B}^{n\times n} = \boldsymbol{C}^{n\times n}$ in the APM with limited memory. When $\omega/\beta \leqslant 1$, Q is lower-bounded by*

Table 4. Joint-Communication Complexity with Enough Memory

Method	$2\omega/\beta \leqslant 1$	$1 < 2\omega/\beta < P^{1/2}$	$2\omega/\beta \geqslant P^{1/2}$
Lower bound (here)	$\Omega(\frac{n^2\beta}{P^{2/3}} + \frac{n^3r}{PS^{1/2}})$	$\Omega(\frac{n^2\beta^{2/3}\omega^{1/3}}{P^{2/3}} + \frac{n^3r}{PS^{1/2}})$	$\Omega(\frac{n^2\omega}{P} + \frac{n^3r}{PS^{1/2}})$
2.5DMML3ooL2[17]	$\Theta(\frac{n^2\beta}{P^{2/3}} + \frac{n^3r}{PS^{1/2}})$	$O(\frac{n^2\omega}{P^{2/3}} + \frac{n^3r}{PS^{1/2}})$	$O(\frac{n^2\omega}{P^{2/3}} + \frac{n^3r}{PS^{1/2}})$
SUMMAL3ooL2[17]	$O(\frac{(r+\beta)n^3}{PS^{1/2}})$	$O(\frac{(r+\beta)n^3}{PS^{1/2}})$	$O(\frac{n^2\omega}{P} + \frac{(r+\beta)n^3}{PS^{1/2}})$
JOMMA (here)	$\Theta(\frac{n^2\beta}{P^{2/3}} + \frac{n^3r}{PS^{1/2}})$	$\Theta(\frac{n^2\beta^{2/3}\omega^{1/3}}{P^{2/3}} + \frac{n^3r}{PS^{1/2}})$	$\Theta(\frac{n^2\omega}{P} + \frac{n^3r}{PS^{1/2}})$

$$\Omega\left(\beta\left(\frac{n^3}{PM^{1/2}}+\frac{n^2}{P^{2/3}})+\frac{n^3r}{PS^{1/2}}\right)\right).$$

For $1<\omega/\beta<n^2/M$, Q *has a lower bound of* $\Omega((n^3(\beta\omega)^{1/2}/PM^{1/2})+(n^3r/PS^{1/2}))$. *When* $\omega/\beta\geqslant n^2/M$, Q *exhibits a lower bound of* $\Omega((n^2\omega/P)+(n^3r/PS^{1/2}))$.

Proof. In the limited memory scenario ($M=O(n^2/P^{1/2})$), we classify the lower bound into three categories based on the value of ω/β, following a methodology similar to that presented in Theorem 1.

1) $\omega/\beta\leqslant 1$. In this case, by $\omega/\beta\leqslant 1$, the $\Omega(\beta((n^3/PM^{1/2})+(n^2/P^{2/3}))+(n^3r/PS^{1/2}))$ lower bound can be derived directly from the trivial joint-cost lower bound of

$$\Omega\left(\beta(\frac{n^3}{PM^{1/2}}+\frac{n^2}{P^{2/3}})+\frac{n^3r}{PS^{1/2}}+\frac{\omega n^2}{P}\right).$$

2) $1<\omega/\beta<n^2/M$. Assume that the subproblem performed by each processor is $\boldsymbol{A}'\boldsymbol{B}'=\boldsymbol{C}'$, where d_1,d_2 and d_3 are the matrix dimensions of the subproblem. Since the total number of arithmetic operations to solve $\boldsymbol{AB}=\boldsymbol{C}$ is n^3 and the number of operations to solve each subproblem is $d_1d_2d_3$, each processor performs $n^3/(Pd_1d_2d_3)$ subproblems. We assume $d_1d_2d_3=O(n^3/P)$ to guarantee each processor performs $\Omega(1)$ subproblems. Considering the computing of each subproblem, each processor needs to access at least one of $\boldsymbol{A}'$ or $\boldsymbol{B}'$ from other processors. Hence the total number of words moved horizontally is at least $n^3/Pd_1d_2d_3\times\min\{d_1d_2,\ d_2d_3\}$. Without loss of generality, we assume that $d_1\leqslant d_3$, then $d_1d_2\leqslant d_2d_3$, and the horizontal cost is at least $\Omega(n^3\beta/(Pd_3))$. Recall that the vertical cost of sequentially solving a subproblem with matrix dimensions d_1,d_2, and d_3 is $\Omega(d_1d_2d_3r/S^{1/2}+d_1d_3\omega)$, and the total vertical cost for each processor to sequentially solve$n^3/(Pd_1d_2d_3)$subproblemsisatleast$\Omega(n^3r/(PS^{1/2})+n^3\omega/(Pd_2))$. Therefore,

$$Q=\Omega\left(\frac{n^3\beta}{Pd_3}+\frac{n^3\omega}{Pd_2}+\frac{n^3r}{PS^{1/2}}\right).$$

From the mean inequality and $d_2d_3\leqslant M$, we get

$$\frac{n^3\beta}{Pd_3}+\frac{n^3\omega}{Pd_2}\geqslant\frac{n^3(\beta\omega)^{1/2}}{P(d_2d_3)^{1/2}}\geqslant\frac{n^3(\beta\omega)^{1/2}}{PM^{1/2}}.$$

Therefore, $Q=\Omega((n^3(\beta\omega)^{1/2}/PM^{1/2})+(n^3r/PS^{1/2}))$.

3) $\omega/\beta\geqslant n^2/M$. In this case, by $\omega/\beta\geqslant n^2/M$ and $M=O(n^2/P^{1/2})$, the $\Omega(n^2\omega/P+n^3r/(PS^{1/2}))$ lower bound can be directly obtained from the trivial joint-cost lower bound $\Omega((n^2\beta/P^{2/3})+(n^3r/PS^{1/2})+(n^2\omega/P))$. □

In the limited memory scenario, the joint costs of the horizontally optimal and the vertically optimal algorithms are given in Table 5, which shows that neither algorithm can attain the joint-cost lower bound in general.

6 Memory and Dimensions Analysis

In this section, we introduce why we set $M=\Omega(n^2/P^{1/2})$ as enough memory and $M=O(n^2/P^{1/2})$ as limited memory. In addition, as mentioned earlier, assuming that n-by-n matrix multiplication is decomposed into multiple subproblems $\boldsymbol{A}'\boldsymbol{B}'=\boldsymbol{C}'$ with dimensions d_1,d_2,d_3 on a parallel machine, then these dimensions demonstrate the vertical-horizontal trade-off (cf. Table 1). We use linear programming to solve the optimal solution of the three dimensions. These optimal solutions can minimize the joint communication and, therefore, guide the tunable grid and the scheduling of our algorithm.

6.1 Joint-Cost Function

To propose algorithms with low joint communication, we introduce conditions conducive to reducing joint-communication costs. Subsequently, we analyze the horizontal and the vertical communication costs of algorithms meeting the following conditions, providing their joint-communication cost function.

1) $d_1\leqslant n$, $d_2\leqslant n$, $d_3\leqslant n$.

2) $d_1d_2d_3=O(n^3/P)$ to ensure that each processor computes $\Omega(1)$ subproblems.

3) $\max\{d_1d_2,d_2d_3,d_1d_3\}=O(M)$ to ensure that the

Table 5. Joint-Communication Complexity with Limited Memory

Method	$\omega/\beta\leqslant 1$	$1<\omega/\beta<n^2/M$	$\omega/\beta\geqslant n^2/M$
Lower bound (here)	$\Omega(\frac{n^3\beta}{PM^{1/2}}+\frac{n^2\beta}{P^{2/3}}+\frac{n^3r}{PS^{1/2}})$	$\Omega(\frac{n^3(\beta\omega)^{1/2}}{PM^{1/2}}+\frac{n^3r}{PS^{1/2}})$	$\Omega(\frac{n^2\omega}{P}+\frac{n^3r}{PS^{1/2}})$
2.5DMML3ooL2[17]	$\Theta(\frac{n^3\beta}{PM^{1/2}}+\frac{n^2\beta}{P^{2/3}}+\frac{n^3r}{PS^{1/2}})$	$O(\frac{n^3\omega}{PM^{1/2}}+\frac{n^2\omega}{P^{2/3}}+\frac{n^3r}{PS^{1/2}})$	$O(\frac{n^3\omega}{PM^{1/2}}+\frac{n^2\omega}{P^{2/3}}+\frac{n^3r}{PS^{1/2}})$
SUMMAL3ooL2[17]	$O(\frac{(r+\beta)n^3}{PS^{1/2}})$	$O(\frac{n^2\omega}{P}+\frac{(r+\beta)n^3}{PS^{1/2}})$	$O(\frac{n^2\omega}{P}+\frac{(r+\beta)n^3}{PS^{1/2}})$
JOMMA (here)	$\Theta(\frac{n^3\beta}{PM^{1/2}}+\frac{n^2\beta}{P^{2/3}}+\frac{n^3r}{PS^{1/2}})$	$\Theta(\frac{n^3(\beta\omega)^{1/2}}{PM^{1/2}}+\frac{n^3r}{PS^{1/2}})$	$\Theta(\frac{n^2\omega}{P}+\frac{n^3r}{PS^{1/2}})$

sizes of input and output do not exceed the size of the main memory.

4) $d_2 \geqslant d_1 = d_3$ to reduce the number of words written since writes are more expensive than reads.

5) Computation is load-balanced.

Lemma 6. *The joint-communication cost of the algorithm satisfying the above conditions is*

$$O\left(\frac{n^3 r}{PS^{1/2}} + \frac{n^3}{P}\left(\frac{\beta}{d_1} + \frac{\beta}{d_2} + \frac{\omega}{d_2}\right)\right).$$

Proof. We first analyze the horizontal and the vertical cost of such algorithms and then give the joint-communication cost.

Horizontal Cost. Based on the total arithmetic operations (n^3) and the number of arithmetic operations per subproblem ($d_1d_2d_3$), we determine that there are $n^3/(d_1d_2d_3)$ subproblems. To maintain load balance, each processor handles $n^3/(Pd_1d_2d_3)$ subproblems ($\boldsymbol{A}'\boldsymbol{B}' = \boldsymbol{C}'$). To compute a subproblem, processors must engage in data exchange, involving at most all elements of matrices $\boldsymbol{A}'$, $\boldsymbol{B}'$, and $\boldsymbol{C}'$. Hence, the horizontal cost of solving a subproblem does not exceed $\beta(d_1d_2 + d_2d_3 + d_1d_3)$, and the total horizontal cost of each processor, addressing $n^3/(Pd_1d_2d_3)$ subproblems, remains below $n^3\beta(d_1d_2 + d_2d_3 + d_1d_3)/(Pd_1d_2d_3)$.

Vertical Cost. Processors utilize the sequential VOMM algorithm for localized subproblem resolution. Each processor handles $n^3/(Pd_1d_2d_3)$ subproblems, with each subproblem incurring a vertical cost of $O(d_1d_2d_3r/S^{1/2} + d_1d_3\omega)$ (cf. Section 4). As a result, the total vertical cost amounts to $O(n^3r/(PS^{1/2}) + n^3\omega/(Pd_2))$.

By $d_1 = d_3$ (condition 4), the joint-communication cost is

$$\begin{aligned} Q &= O\left(\frac{n^3\beta\,(d_1d_2 + d_2d_3 + d_1d_3)}{Pd_1d_2d_3} + \frac{n^3 r}{PS^{1/2}} + \frac{n^3\omega}{Pd_2}\right) \\ &= O\left(\frac{n^3 r}{PS^{1/2}} + \frac{n^3}{P}\left(\frac{\beta}{d_1} + \frac{\beta}{d_2} + \frac{\omega}{d_2}\right)\right). \end{aligned}$$

□

6.2 Memory Analysis

Generally, the larger the main memory, the larger the matrix dimensions of each subproblem, and the fewer the number of subproblems that each processor needs to execute. For example, considering the 2.5D algorithm, assuming that the main memory size per processor is $M = \Omega(cn^2/P)$, the number of subproblems performed by each processor is $O(\sqrt{P/c^3})$, where $c \leqslant P^{1/3}$.

Definition 1. *The scenario where each processor executes only $\Theta(1)$ subproblems is defined as the enough memory scenario. Conversely, the scenario where each processor executes $\Omega(1)$ subproblems is defined as the limited memory scenario.*

Lemma 7. *In the enough memory scenario, the memory size of each processor is at least $M = \Omega(n^2/P^{1/2})$.*

Proof. Let n be the size of the matrix multiplication $\boldsymbol{AB} = \boldsymbol{C}$ and d_1, d_2, d_3 the dimensions of the subproblem obtained by dividing $\boldsymbol{AB} = \boldsymbol{C}$. In the enough memory scenario, since the number of subproblems executed by each processor is $n^3/(Pd_1d_2d_3) = \Theta(1)$, we conclude that $d_1d_2d_3 = \Theta(n^3/P)$.

By $d_2 \leqslant n$ (condition 1), $d_2 \geqslant d_1 = d_3$ (condition 4), and $d_1d_2d_3 = \Theta(n^3/P)$, we have $d_1d_3 = \Omega(n^2/P)$ and $d_1d_3 = O(n^2/P^{2/3})$. Consequently, both d_1 and d_3 range from at least $\Omega(n/P^{1/2})$ to at most $O(n/P^{1/3})$.

By $d_1d_2d_3 = \Theta(n^3/P)$, $d_1 = d_3 = \Omega(n/P^{1/2})$, and $\max\{d_1d_2, d_2d_3, d_1d_3\} = O(M)$ (condition 3), we have $M = \Omega(\max\{d_1d_2,\ d_2d_3,\ d_1d_3\}) = \Omega(\max\{d_1d_2,\ d_1^2\}) = \Omega(\max\{n^3/(Pd_1),\ d_1^2\}) = \Omega(n^2/P^{1/2})$. □

6.3 Matrix Dimensions Optimization

Given the machine parameters P, S, M, r, ω, β, and problem size n, by Lemma 6, the joint-communication cost is asymptotically minimal if and only if $(\beta/d_1) + (\beta/d_2) + (\omega/d_2)$ is asymptotically minimal. Therefore, the optimization problem of minimizing the joint-communication cost is formulated as follows:

$$\begin{aligned} &\min \quad \frac{\beta}{d_1} + \frac{\beta}{d_2} + \frac{\omega}{d_2} \\ &\text{subject to:} \end{aligned} \tag{1}$$

conditions 1, 2, 3, 4, 5 (cf. Subsection 6.1).

Next, we analyze how the values of d_1, d_2, and d_3 can minimize the joint-communication cost for the enough memory scenario and the limited memory scenario.

Lemma 8. *In the enough memory scenario, if $2\omega/\beta \leqslant 1$, (1) attains its minimum when $d_1 = d_2 = d_3 = \Theta(n/P^{1/3})$. For $1 \leqslant 2\omega/\beta \leqslant P^{1/2}$, (1) is minimized when $d_1 = d_3 = \Theta((n^3\beta/2P\omega)^{1/3})$ and $d_2 = \Theta((4n^3\omega^2/P\beta^2)^{1/3})$. If $2\omega/\beta \geqslant P^{1/2}$, then (1) is minimized when $d_1 = d_3 = \Theta(n/P^{1/2})$ and $d_2 = \Theta(n)$.*

Proof. Recalling that $d_1d_2d_3 = \Theta(n^3/P)$ in the enough memory scenario and $d_1 = d_3$ (condition 4), we have $d_2 = \Theta(n^3/(Pd_1{}^2))$. Therefore,

$$O\left(\frac{\beta}{d_1}+\frac{\beta}{d_2}+\frac{\omega}{d_2}\right)=O\left(\frac{\beta}{d_1}+\frac{P\beta {d_1}^2}{n^3}+\frac{P\omega {d_1}^2}{n^3}\right). \tag{2}$$

Parallel to the lower bound proof in Theorem 1, we classify the upper bound into three cases based on the value of $2\omega/\beta$.

1) $2\omega/\beta \leqslant 1$. In this case, (2) is $O(\beta/d_1 + P\beta {d_1}^2/n^3)$, which is a function of d_1. By derivation, this function is minimized when $d_1 = \Theta(n/P^{1/3})$. Therefore, from $d_1d_2d_3 = \Theta(n^3/P)$ (in the enough memory scenario) and $d_1 = d_3$ (condition 4), (1) is minimized when $d_1 = d_2 = d_3 = \Theta(n/P^{1/3})$.

2) $2\omega/\beta > 1$. In this case, (2) is $O(\beta/d_1 + P\omega {d_1}^2/n^3)$, which is a function of d_1. By derivation, this function is minimized when $d_1 = \Theta((n^3\beta/2P\omega)^{1/3})$. However, given $d_1 = d_3$, $d_2 \leqslant n$, and $d_1d_2d_3 = \Theta(n^3/P)$, d_1 is constrained to the range of $\Omega(n/P^{1/2})$. If $(n^3\beta/(2P\omega))^{1/3} > n/P^{1/2}$, (1) minimizes at $d_1 = \Theta((n^3\beta/(2P\omega))^{1/3})$; otherwise, it minimizes at $d_1 = \Theta(n/P^{1/2})$.

a) $1 < 2\omega/\beta < P^{1/2}$. In this case, we have $(n^3\beta/(2P\omega))^{1/3} > n/P^{1/2}$. Thus (1) is minimized when $d_1 = d_3 = \Theta((n^3\beta/(2P\omega))^{1/3})$ and $d_2 = \Theta((4n^3\omega^2/(P\beta^2))^{1/3})$.

b) $2\omega/\beta \geqslant P^{1/2}$. In this case, we have $(n^3\beta/(2P\omega))^{1/3} \leqslant n/P^{1/2}$. Thus (1) is minimized when $d_1 = d_3 = \Theta(n/P^{1/2})$ and $d_2 = \Theta(n)$. □

Lemma 9. *In the limited memory scenario, if $\omega/\beta \leqslant 1$, (1) attains its minimum when $d_1 = d_2 = d_3 = \Theta(\min(n/P^{1/3}, M^{1/2}))$. For $1 \leqslant \omega/\beta \leqslant P^{1/2}$, (1) is minimized when $d_1 = d_3 = \Theta(\sqrt{M\beta/\omega})$ and $d_2 = \Theta(\sqrt{M\omega/\beta})$. If $\omega/\beta \geqslant P^{1/2}$, then (1) achieves its minimum value when $d_1 = d_3 = \Theta(M/n)$ and $d_2 = \Theta(n)$.*

Proof. In alignment with the lower bound proof presented in Theorem 2, we similarly categorize the upper bound into three cases based on the value of ω/β.

1) $\omega/\beta \leqslant 1$. In this case, $O(\beta/d_1 + \beta/d_2 + \omega/d_2) = O(\beta/d_1 + \beta/d_2)$. As $d_1d_2 = O(M)$ (condition 3), applying the Arithmetic-Geometric Mean inequality, we ascertain that $O(\beta/d_1 + \beta/d_2)$ achieves its minimum when $d_1d_2 = \Theta(M)$. Substituting $d_1 = \Theta(M/d_2)$, we find that $O(\beta/d_1 + \beta/d_2) = O(\beta d_2/M + \beta/d_2)$, a function of d_2. Upon differentiation, this function reaches its minimum when $d_2 = \Theta(M^{1/2})$. However, for the case of $M = \Omega(n^2/P^{2/3})$, we have $d_1 = d_2 = d_3 = \Theta(M^{1/2}) = \Omega(n/P^{1/3})$. This results in $d_1d_2d_3 = \Omega(n^3/P)$, indicating that condition 2 is not satisfied. In this case, given $d_1d_2d_3 = O(n^3/P)$ and $d_1 = d_3$, the expression $O(\beta/d_1 + \beta/d_2)$ attains its minimum when $d_1^2d_2 = \Theta(n^3/P)$. By substituting $d_2 = \Theta(n^3/Pd_1^2)$, we have $O(\beta/d_1 + \beta/d_2) = O(\beta/d_1 + \beta Pd_1^2/n^3)$, which reaches its minimum when $d_1 = \Theta(n/P^{1/3})$. Hence, (1) achieves its minimum when $d_1 = d_2 = d_3 = \Theta(\min\{n/P^{1/3}, M^{1/2}\})$.

2) $\omega/\beta > 1$. In this case, $O(\beta/d_1 + \beta/d_2 + \omega/d_2) = O(\beta/d_1 + \omega/d_2)$. Given $d_1d_2 = O(M)$ according to condition 3, employing the Arithmetic-Geometric Mean inequality reveals that $O(\beta/d_1 + \omega/d_2)$ attains its minimum when $d_1d_2 = \Theta(M)$. By substituting $d_1 = \Theta(M/d_2)$, we find that $O(\beta/d_1 + \omega/d_2) = O(\beta d_2/M + \omega/d_2)$, a function of d_2. Upon differentiation, this function reaches its minimum when $d_2 = \Theta(\sqrt{M\omega/\beta})$. However, d_2 is bounded by $d_2 \leqslant n$. Hence, if $\sqrt{M\omega/\beta} < n$, (1) minimizes at $d_2 = \Theta(\sqrt{M\omega/\beta})$; otherwise, (1) minimizes at $d_2 = \Theta(n)$.

a) $1 < \omega/\beta < n^2/M$. In this case, we have $\sqrt{M\omega/\beta} < n$. Thus (1) is minimized when $d_2 = \Theta(\sqrt{M\omega/\beta})$ and $d_1 = d_3 = \Theta(\sqrt{M\beta/\omega})$ $(d_1 = d_3 = \Theta(M/d_2))$.

b) $\omega/\beta \geqslant n^2/M$. In this case, we have $\sqrt{M\omega/\beta} \geqslant n$. Thus (1) is minimized when $d_2 = \Theta(n)$ and $d_1 = d_3 = \Theta(M/n)$ $(d_1 = d_3 = \Theta(M/d_2))$. □

7 Joint-Communication Optimal Algorithm

Algorithm 2 is a brief description of the Joint-Communication Optimal Matrix Multiplication Algorithm (JOMMA). See Algorithm 3 and Algorithm 4 for more details.

Algorithm 2. JOMMA

Require: $\boldsymbol{A}$, $\boldsymbol{B}$, P, M, S, r, ω, β
Ensure: $\boldsymbol{C} = \boldsymbol{AB}$
1: **if** enough memory **then**
2: Call *EM-JOMMA*
3: **end if**
4: **if** limited memory **then**
5: Call *LM-JOMMA*
6: **end if**

7.1 Enough Memory

In the enough memory scenario, we employ EM-JOMMA (cf. Algorithm 3) to solve matrix multiplication.

7.1.1 Data Layout

P processors are arranged in a $P_x \times P_y \times P_z$ cubic grid, where $P_xP_yP_z = P$ and P_{ijk} is the processor

at coordinate (i, j, k) $(i = \{1, 2, \ldots, P_x\}, j = \{1, 2, \ldots, P_y\}, k = \{1, 2, \ldots, P_z\})$. Initially, the $n \times n$ input matrix $\boldsymbol{A}$ is evenly distributed to the 2D slice $P_{:1:}$ $(i \in \{1, 2, \ldots, P_x\}, j = 1,$ and $k \in \{1, 2, \ldots, P_z\})$, and each processor P_{i1k} owns an $(n/P_x) \times (n/P_z)$ block $\boldsymbol{A}(i, k)$ as its local matrix $\boldsymbol{\Pi(A)}$ (cf. Fig.2(a)). Similarly, the $n \times n$ input matrix $\boldsymbol{B}$ is evenly distributed to the 2D slice $P_{1::}$, and each processor P_{1jk} owns an $(n/P_z) \times (n/P_y)$ block $\boldsymbol{B}(k, j)$ as its local matrix $\boldsymbol{\Pi(B)}$ (cf. Fig.2(b)). The output matrix $\boldsymbol{C}$ can be divided into $P_x \times P_y$ blocks, and the algorithm terminates when each processor P_{ij1} on the 2D slice $P_{::1}$ has finished computing $\boldsymbol{C}(i, j) = \sum_{k=1}^{P_z} \boldsymbol{A}(i, k)\boldsymbol{B}(k, j)$ (cf. Fig.2(c)).

Note that although the above input matrices are initially load-imbalanced, we can easily rearrange them for load-balancing. For example, let the block $\boldsymbol{A}(i, k)$ owned by P_{i1k} be scattered over the P_y processors in $P_{i:k}$. Similarly, let the block $\boldsymbol{B}(k, j)$ owned by P_{1jk} be scattered over the P_x processors in $P_{:jk}$. These two scatter operations do not affect the asymptotic joint-communication cost.

7.1.2 Scheduling

From the above data layout, the dimensions of the subproblem solved by each processor are $d_1 = n/P_x$, $d_2 = n/P_z$, and $d_3 = n/P_y$. Therefore, we set the values of P_x, P_y, and P_z according to Lemma 8, thus being able to asymptotically minimize the joint-communication cost (lines 1–9 in Algorithm 3). For example, when $2\omega/\beta \leqslant 1$, we set $P_x = P_y = P_z = P^{1/3}$ such that $d_1 = d_2 = d_3 = n/P^{1/3}$.

After determining the parameters of the tunable processor grid, the EM-JOMMA algorithm has three steps.

Algorithm 3. EM-JOMMA

Require: $\boldsymbol{A}$ is divided into $P_x \times P_z$ matrix blocks, and P_{i1k} has $\boldsymbol{A}(i, k)$ as $\boldsymbol{\Pi(A)}$; $\boldsymbol{B}$ is divided into $P_z \times P_y$ matrix blocks, and P_{1jk} owns $\boldsymbol{B(k, j)}$ as $\boldsymbol{\Pi(B)}$

Ensure: $\boldsymbol{C} = \boldsymbol{AB} \in \mathbb{R}^{n \times n}$

1: **if** $((2\omega)/\beta) \leqslant 1$ **then**
2: $\quad P_x = P_y = P_z = P^{\frac{1}{3}}$
3: **end if**
4: **if** $1 < ((2\omega)/\beta) < P^{1/2}$ **then**
5: $\quad P_x = P_y = ((2\omega P)/\beta)^{\frac{1}{3}}, P_z = ((P\beta^2)/(4\omega^2))^{\frac{1}{3}}$
6: **end if**
7: **if** $((2\omega)/\beta) \geqslant P^{1/2}$ **then**
8: $\quad P_x = P_y = P^{1/2}, P_z = 1$
9: **end if**
10: $Broadcast(\boldsymbol{\Pi(A)}, \boldsymbol{\Pi(X)}, 1, P_{i:k})$
11: $Broadcast(\boldsymbol{\Pi(B)}, \boldsymbol{\Pi(Y)}, 1, P_{:jk})$
12: $\boldsymbol{\Pi(Z)} \leftarrow VOMM(\boldsymbol{\Pi(X)}, \boldsymbol{\Pi(Y)})$
13: $Reduce(\boldsymbol{\Pi(Z)}, \boldsymbol{\Pi(C)}, 1, P_{ij:})$

Step 1. The EM-JOMMA algorithm performs two broadcast operations, $Broadcast(\boldsymbol{\Pi(A)}, \boldsymbol{\Pi(X)}, 1, P_{i:k})$ (cf. line 10 in Algorithm 3 and Fig.2(a)) and $Broadcast(\boldsymbol{\Pi(B)}, \boldsymbol{\Pi(Y)}, 1, P_{:jk})$ (cf. line 11 in Algorithm 3 and Fig.2(b)), so that each processor P_{ijk} accesses $\boldsymbol{A}(i, k)$ as $\boldsymbol{\Pi(X)}$ and $\boldsymbol{B}(k, j)$ as $\boldsymbol{\Pi(Y)}$.

Step 2. Each processor P_{ijk} calls the VOMM algorithm to compute $\boldsymbol{\Pi(Z)} = \boldsymbol{C}_{ijk} = \boldsymbol{A}(i, k)\boldsymbol{B}(k, j)$ (cf. line 12 in Algorithm 3).

Step 3. The EM-JOMMA algorithm performs the reduction operation, $Reduce(\boldsymbol{\Pi(Z)}, \boldsymbol{\Pi(C)}, 1, P_{ij:})$ (cf. line 13 in Algorithm 3 and Fig.2(c)), so that the processor P_{ij1} can compute $\boldsymbol{\Pi(C)} = \boldsymbol{C}(i, j) = \sum_{k=1}^{P_z} \boldsymbol{C}_{ijk}$.

7.1.3 Joint-Communication Cost of EM-JOMMA

By $T_{\text{VOMM}}(d_1, d_2, d_3)$ we denote the vertical communication cost of computing $\boldsymbol{AB} = \boldsymbol{C}$ using VOMM, where $\boldsymbol{A}$ is a $d_1 \times d_2$ matrix, $\boldsymbol{B}$ is a $d_2 \times d_3$

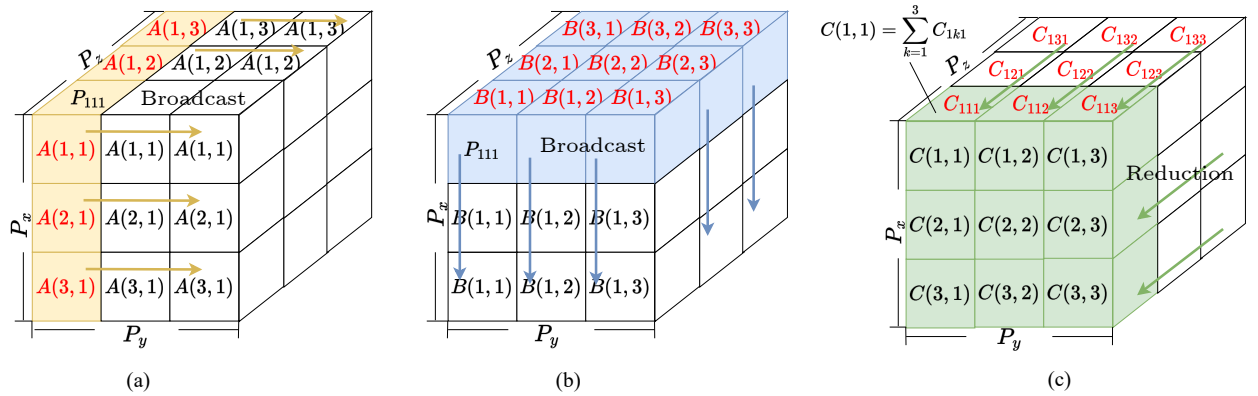

Fig.2. Two broadcast operations and a reduction operation. (a) Processor P_{i1k} broadcasts $\boldsymbol{\Pi(A)}$ along $P_{i:k}$. (b) Processor P_{1jk} broadcasts $\boldsymbol{\Pi(B)}$ along $P_{:jk}$. (c) A reduction along $P_{ij:}$ onto root P_{ij1} sums each $\boldsymbol{C_{ijk}}$.

matrix and $\boldsymbol{C}$ is a $d_1 \times d_3$ matrix, then $T_{\text{VOMM}}(d_1,\ d_2,\ d_3) = 2\sqrt{3}d_1d_2d_3r/S^{1/2} + \omega d_1d_3$ (See Algorithm 1).

Below we show that in the APM with enough memory, for any value of ω and β, EM-JOMMA is joint-communication optimal.

Considering Algorithm 3, we find $|\boldsymbol{\Pi}(\boldsymbol{A})| = |\boldsymbol{\Pi}(\boldsymbol{X})| = n^2/(P_xP_Z)$, $|\boldsymbol{\Pi}(\boldsymbol{B})| = |\boldsymbol{\Pi}(\boldsymbol{Y})| = n^2/(P_yP_z)$, and $|\boldsymbol{\Pi}(\boldsymbol{Z})| = |\boldsymbol{\Pi}(\boldsymbol{C})| = n^2/(P_xP_z)$. Referring to Table 3, the joint-communication cost function of the EM-JOMMA algorithm is

$$Q = T_{\text{Broadcast}}\left(\frac{n^2}{P_xP_z}, P_y\right) + T_{\text{Broadcast}}\left(\frac{n^2}{P_yP_z}, P_x\right) + T_{\text{VOMM}}\left(\frac{n}{P_x}, \frac{n}{P_z}, \frac{n}{P_y}\right) + T_{\text{Reduce}}\left(\frac{n^2}{P_xP_y}, P_z\right). \quad (3)$$

Substituting the provided values of P_x, P_y, and P_z from EM-JOMMA into (3), we derive the joint-communication cost as follows, which asymptotically matches the lower bound established in Theorem 1.

- $2\omega/\beta \leqslant 1$,

$$Q = O\left(\frac{n^2\beta}{P^{2/3}} + \frac{n^3r}{PS^{1/2}}\right).$$

- $1 < 2\omega/\beta < P^{1/2}$,

$$Q = O\left(\frac{n^3r}{PS^{1/2}} + \frac{n^2\omega^{1/3}\beta^{2/3}}{P^{2/3}}\right).$$

- $2\omega/\beta \geqslant P^{1/2}$,

$$Q = O\left(\frac{n^3r}{PS^{1/2}} + \frac{n^2\omega}{P}\right).$$

7.2 Limited Memory

In the limited memory scenario, we present LM-JOMMA (cf. Algorithm 4) to solve matrix multiplication.

7.2.1 Data Layout

P processors are arranged in a cubic grid $P_x \times P_y \times P_z$, $P_xP_yP_z = P$. Initially, the $n \times n$ input matrix $\boldsymbol{A}$ is divided into $P_x \times P_y$ blocks, and the $n \times n$ matrix $\boldsymbol{B}$ is divided into $P_y \times P_x$ blocks. Let these blocks be evenly distributed on the 2D slice $P_{::1}$. Specifically, the processor P_{ij1} owns an $(n/P_x) \times (n/P_y)$ block $\boldsymbol{A}(i,\ j)$ and an $(n/P_y) \times (n/P_x)$ block $\boldsymbol{B}(j,\ i)$ (cf. Fig.3(a)). In addition, the output matrix $\boldsymbol{C}$ can be divided into $P_x \times P_x$ blocks, where the size of each block is $(n/P_x) \times (n/P_x)$.

7.2.2 Scheduling

From the above data layout, the dimensions of the subproblem solved by each processor are $d_1 = n/P_x, d_2 = n/P_y$, and $d_3 = n/P_x$. Therefore, we set the values of P_x, P_y, and P_z according to Lemma 9, thus being able to asymptotically minimize the joint-communication cost (lines 1–9 in Algorithm 4). For example, when $\omega/\beta \geqslant n^2/M$, we set $P_x = n^2/M$, $P_y = 1$, and $P_z = P/(P_xP_y) = PM/n^2$ such that $d_1 = d_3 = M/n$ and $d_2 = n$.

Algorithm 4. LM-JOMMA

Require: $\boldsymbol{A}$ and $\boldsymbol{B}$ are evenly distributed on the 2D slice $P_{::1}$; processor P_{ij1} has $\boldsymbol{A}(i,\ j)$ as $\boldsymbol{\Pi}(\boldsymbol{A})$ and $\boldsymbol{B}(j,\ i)$ as $\boldsymbol{\Pi}(\boldsymbol{B})$; $\boldsymbol{\Pi}(\boldsymbol{C}) = \varnothing$

Ensure: $\boldsymbol{C} = \boldsymbol{AB} \in \mathbb{R}^{n\times n}$

1: **if** $\omega/\beta \leqslant 1$ **then**
2: $c = \min(P^{\frac{1}{3}}, \sqrt{PM}/n),\ P_x = P_y = \sqrt{P/c},\ P_z = c$
3: **end if**
4: **if** $1 < \omega/\beta < n^2/M$ **then**
5: $P_x = n\omega^{\frac{1}{2}}/M^{\frac{1}{2}}\beta^{\frac{1}{2}}, P_y = n\beta^{\frac{1}{2}}/M^{\frac{1}{2}}\omega^{\frac{1}{2}}, P_z = PM/n^2$
6: **end if**
7: **if** $\omega/\beta \geqslant n^2/M$ **then**
8: $P_x = n^2/M, P_y = 1, P_z = PM/n^2$
9: **end if**
10: $Broadcast(\boldsymbol{\Pi}(\boldsymbol{A}), \boldsymbol{\Pi}(\boldsymbol{X}), 1, P_{ij:})$
11: $Broadcast(\boldsymbol{\Pi}(\boldsymbol{B}), \boldsymbol{\Pi}(\boldsymbol{Y}), 1, P_{ij:})$
12: $Shift(\boldsymbol{\Pi}(\boldsymbol{Y}), (k-1)\lceil P_x/P_z\rceil, P_y, P_{:jk})$
13: **for** $m = 1$ to $\lceil P_x/P_z\rceil$ **do**
14: $Shift(\boldsymbol{\Pi}(\boldsymbol{Y}), 1, P_x, P_{:jk})$
15: $\boldsymbol{\Pi}(\boldsymbol{Z}) \leftarrow VOMM(\boldsymbol{\Pi}(\boldsymbol{X}), \boldsymbol{\Pi}(\boldsymbol{Y}))$
16: $j' = m \bmod P_y$
17: $Reduce(\boldsymbol{\Pi}(\boldsymbol{Z}), \boldsymbol{\Pi}(\boldsymbol{U}), j', P_{i:k})$
18: $\boldsymbol{\Pi}(\boldsymbol{C}) = \{\boldsymbol{\Pi}(\boldsymbol{C}), \boldsymbol{\Pi}(\boldsymbol{U})\}$
19: **end for**

Before introducing scheduling, we give the following Lemmas 10 and 11, which are used to design Algorithm 4. Lemma 10 details the computation of $\boldsymbol{AB} = \boldsymbol{C}$ on the 2D slice $P_{::1}$, while Lemma 11 outlines the computation of $\boldsymbol{AB} = \boldsymbol{C}$ on the cubic processor grid assuming each 2D slice contains a copy of input matrices $\boldsymbol{A}$ and $\boldsymbol{B}$.

Lemma 10. *When matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ are evenly distributed on the 2D slice $P_{::1}$, computing $\boldsymbol{C} = \boldsymbol{AB}$ on $P_{::1}$ can be accomplished using a sequence of iterative P_x shift operations, denoted as $Shift(\boldsymbol{\Pi}(\boldsymbol{B}), 1, P_x, P_{:jk})$.*

Proof. Initially, each processor row $P_{i:1}$ on the 2D slice $P_{::1}$ owns $\boldsymbol{A}(i,:)$ and $\boldsymbol{B}(:,i)$ (cf. Fig.3(a)), en-

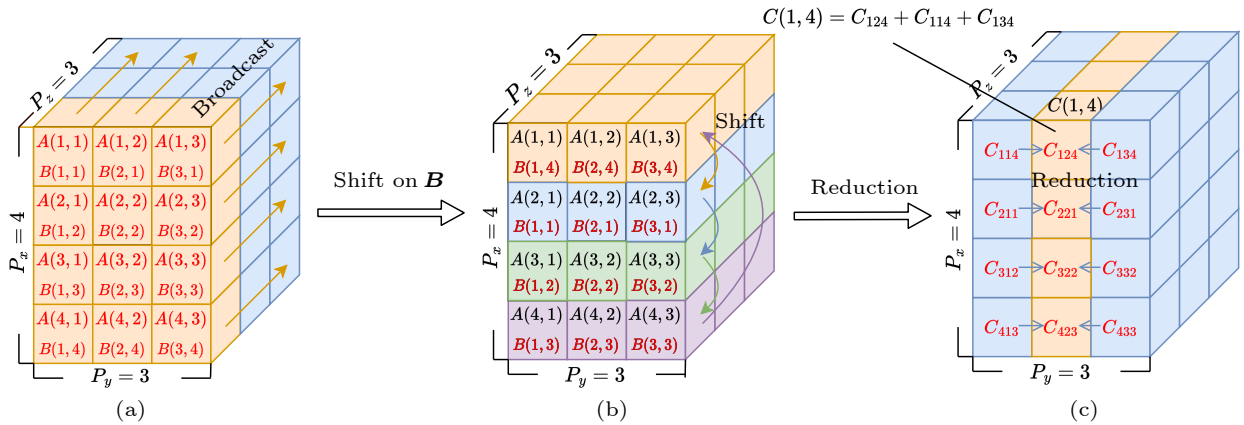

Fig.3. Broadcast, shift, and reduction operations. (a) $\boldsymbol{A}$ and $\boldsymbol{B}$ are initially evenly distributed on $P_{::1}$ and the broadcast operations of lines 10 and 11 in Algorithm 4. (b) The data layout after performing one shift operation $Shift(\boldsymbol{\Pi}(\boldsymbol{B}), 1, P_x, P_{:jk})$ on (a). (c) The reduction operation of line 17 in Algorithm 4 when $m = 2$.

abling the computation of the block $\boldsymbol{C}(i,i) = \sum_{j=1}^{P_y} \boldsymbol{A}(i,j)\boldsymbol{B}(j,i)$. Through a circular shift operation $Shift(\boldsymbol{\Pi}(\boldsymbol{B}), 1, P_x, P_{:jk})$ (cf. Fig.3(b)), the processor row $P_{i:1}$ accesses $\boldsymbol{B}(:,i')$ and can compute the block $\boldsymbol{C}(i,i') = \sum_{j=1}^{P_y} \boldsymbol{A}(i,j)\boldsymbol{B}(j,i')$, where $i' = (i-1)$. As there are P_x processor rows within the 2D slice $P_{::1}$ and $\boldsymbol{C}$ is partitioned into $(P_x)^2$ blocks, the computation of $\boldsymbol{C}$ can be achieved through iterative execution of P_x shift operations. □

Lemma 11. *When $\boldsymbol{A}$ and $\boldsymbol{B}$ are replicated on every 2D slice $P_{::k}$ for all $k \in \{1, 2, \ldots, P_z\}$, then the computation of $\boldsymbol{C} = \boldsymbol{AB}$ on the cubic processor grid can be achieved by initially performing a shift operation $Shift(\boldsymbol{\Pi}(\boldsymbol{B}), (k-1)\lceil P_x/P_z \rceil, P_x, P_{:jk})$, followed by $\lceil P_x/P_z \rceil$ additional shift operations $Shift(\boldsymbol{\Pi}(\boldsymbol{B}), 1, P_x, P_{:jk})$.*

Proof. According to Lemma 10, we can calculate $\boldsymbol{C}$ on the 2D slice $P_{::1}$ through the execution of P_x shift operations, $Shift(\boldsymbol{\Pi}(\boldsymbol{B}), 1, P_x, P_{:jk})$. After each shift operation, a computational task follows, where each processor within the 2D slice $P_{::k}$ computes a matrix multiplication subproblem (as depicted in Figs.3(a) and 3(b)). If each 2D slice on the cubic grid has a copy of $\boldsymbol{A}$ and $\boldsymbol{B}$, P_x computational tasks can be distributed to these P_z 2D slices for parallel execution. Hence, the computation of $\boldsymbol{C}$ on the cubic processor grid can be accomplished by performing $\lceil P_x/P_z \rceil$ shift operations. More precisely, the required shift operations on the 2D slice $P_{::k}$ correspond to those performed on the 2D slice $P_{::1}$ from the $((k-1)\lceil P_x/P_z \rceil + 1)$-th to the $(k\lceil P_x/P_z \rceil + 1)$-th. Hence, an initial shift operation, denoted as $Shift(\boldsymbol{\Pi}(\boldsymbol{B}), (k-1)\lceil P_x/P_z \rceil, P_x, P_{:jk})$, is required on every 2D slice $P_{::k}$, for all $k \in \{1, 2, \ldots, P_z\}$, before executing the $\lceil P_x/P_z \rceil$ subsequent shift operations. □

The above Lemma 11 leads to the LM-JOMMA algorithm, which has the following four steps.

Step 1. The LM-JOMMA algorithm performs two operations, $Broadcast(\boldsymbol{\Pi}(\boldsymbol{A}), \boldsymbol{\Pi}(\boldsymbol{X}), 1, P_{ij:})$ and $Broadcast(\boldsymbol{\Pi}(\boldsymbol{B}), \boldsymbol{\Pi}(\boldsymbol{Y}), 1, P_{ij:})$ (cf. lines 10 and 11 in Algorithm 4 and Fig.3(a)), so that $\boldsymbol{A}$ and $\boldsymbol{B}$ are replicated on each 2D slice $P_{::k}$, $\forall k \in \{1, 2, \ldots, P_z\}$.

Step 2. From Lemma 11, all processors perform an initial shift operation $Shift(\boldsymbol{\Pi}(\boldsymbol{B}), (k-1)\lceil P_x/P_z \rceil, P_x, P_{:jk})$ (cf. line 12 in Algorithm 4) such that each 2D slice $P_{::k}$ can perform the appropriate computations.

Step 3. All processors iteratively perform $\lceil P_x/P_z \rceil$ shift operation on $\boldsymbol{B}$ (cf. lines 13–15 in Algorithm 4 and Fig.3(b)) such that $\boldsymbol{C}$ can be computed completely.

Step 4. After each circular shift (line 16 in Algorithm 4) and local computations (line 17 in Algorithm 4) are performed, the computation result needs to be summed for each processor row $P_{i:k}$. For the m-th shift operation ($m = \{1, 2, \ldots, \lceil P_x/P_z \rceil\}$), we specify $P_{ij'k}$ as the root processor of processor row $P_{i:k}$, where $j' = m \mod P_y$ (cf. line 16 in Algorithm 4). Then, the LM-JOMMA algorithm performs a reduction operation (cf. line 17 in Algorithm 4 and Fig.3(c)) such that the root processor can sum the local results of this processor row.

7.2.3 *Joint-Communication Cost of LM-JOMMA*

Below we prove that in the APM with limited memory, for any value of ω and β, LM-JOMMA is joint-communication optimal.

Considering Algorithm 4, we find $|\boldsymbol{\Pi}(\boldsymbol{A})| = |\boldsymbol{\Pi}(\boldsymbol{X})| = n^2/(P_xP_y)$, $|\boldsymbol{\Pi}(\boldsymbol{B})| = |\boldsymbol{\Pi}(\boldsymbol{Y})| = n^2/(P_xP_y)$, and $|\boldsymbol{\Pi}(\boldsymbol{Z})| = |\boldsymbol{\Pi}(\boldsymbol{U})| = |\boldsymbol{\Pi}(\boldsymbol{C})| = n^2/P_x^2$. Referring

to Table 3, the joint-communication cost function of the LM-JOMMA algorithm is

$$Q = 2T_{\text{Broadcast}}\left(\frac{n^2}{P_xP_y}, P_z\right) + T_{\text{Shift}}\left(\frac{n^2}{P_xP_y}\right) + \lceil\frac{P_x}{P_z}\rceil T_{\text{VOMM}}\left(\frac{n}{P_x}, \frac{n}{P_y}, \frac{n}{P_x}\right) + \lceil\frac{P_x}{P_z}\rceil T_{\text{Shift}}\left(\frac{n^2}{P_xP_y}\right) + \lceil\frac{P_x}{P_z}\rceil T_{\text{Reduce}}\left(\frac{n^2}{P_x^2}, P_y\right). \quad (4)$$

By substituting the provided values of P_x, P_y, and P_z from LM-JOMMA into (4), we obtain the joint-communication cost as follows, which asymptotically matches the lower bound established in Theorem 2.

- $\omega/\beta \leqslant 1$,

$$Q = O\left(\beta\left(\frac{n^3}{PM^{1/2}} + \frac{n^2}{P^{2/3}}\right) + \frac{n^3 r}{PS^{1/2}}\right).$$

- $1 < \omega/\beta < n^2/M$,

$$Q = O\left(\frac{n^3 r}{PS^{1/2}} + \frac{n^3\omega^{\frac{1}{2}}\beta^{\frac{1}{2}}}{PM^{\frac{1}{2}}}\right).$$

- $\omega/\beta \geqslant n^2/M$,

$$Q = O\left(\frac{n^3 r}{PS^{1/2}} + \frac{n^2\omega}{P}\right).$$

8 Evaluation

In this section, we conduct a detailed evaluation of the communication costs associated with JOMMA, 2.5DL3ooL2, SUMMAL3ooL2, and 2.5D-COWE[16] (Cache-Oblivious Write Efficient Algorithm), utilizing numerical analysis simulation results rather than actual implementations on distributed systems. Below we provide a brief overview of 2.5DL3ooL2, SUMMAL3ooL2, and 2.5D-COWE.

2.5DL3ooL2 combines the 2.5D and VOMM approaches, offering an asymptotically optimal horizontal cost for n-by-n matrix multiplication. In the enough memory scenario, it follows the scheduling pattern of Algorithm 3 when $\omega/\beta \leqslant 1$, and in the limited memory scenario, it adheres to the scheduling pattern of Algorithm 4 when $2\omega/\beta \leqslant 1$. This uniformity arises from JOMMA's ability to asymptotically minimize horizontal costs when horizontal word transfers incur significant expenses, as demonstrated in Tables 4 and 5.

SUMMAL3ooL2 is a SUMMA algorithm variant, preserving the initial data layout of SUMMA. That is, P processors are arranged in a $P^{1/2} \times P^{1/2}$ grid and each processor owns local matrices $\mathbf{\Pi}(\boldsymbol{A})$ and $\mathbf{\Pi}(\boldsymbol{B})$ with size $(n/P^{1/2}) \times (n/P^{1/2})$. For SUMMA, there are $P^{1/2}$ rounds. Each processor participates in two broadcast operations on $\mathbf{\Pi}(\boldsymbol{A})$ and $\mathbf{\Pi}(\boldsymbol{B})$ in each round and then performs local matrix multiplication $\mathbf{\Pi}(\boldsymbol{A}) \cdot \mathbf{\Pi}(\boldsymbol{B})$ (cf. Fig.4). For SUMMAL3ooL2, to minimize writes, each processor participates in two broadcast operations on submatrices of $\mathbf{\Pi}(\boldsymbol{A})$ and $\mathbf{\Pi}(\boldsymbol{B})$ in each round, and the size of each submatrix is $(S/3)^{1/2} \times (S/3)^{1/2}$. In this way, the problem size matches the cache size S, and the outputs not fully computed during the round can be stored in the cache until fully computed, and then written to main memory. Since each processor computes a matrix multiplication of size $d_1 = d_2 = d_3 = (S/3)^{1/2}$ per round, the number of arithmetic operations performed per round is $P(S/3)^{3/2}$. Hence the number of rounds is $n^3/(P(S/3)^{3/2})$.

2.5D-COWE combines 2.5D, known for horizontal optimization, with COWE[16], a cache-oblivious (cache size is unknown) divide-and-conquer algorithm that is vertically optimal in the sequential and asymmetric setting. To sequentially solve $\boldsymbol{A}'\boldsymbol{B}' = \boldsymbol{C}'$ on a single processor, with matrices $\boldsymbol{A}' \in \mathbb{R}^{d_1 \times d_2}$, $\boldsymbol{B}' \in \mathbb{R}^{d_2 \times d_3}$, and $\boldsymbol{C}' \in \mathbb{R}^{d_1 \times d_3}$, COWE recursively divides the largest one of the three dimensions: d_1, $d_2 r^{2/3}/\omega^{2/3}$, d_3, into halves, resulting in two subproblems. The recursion terminates when the subproblem

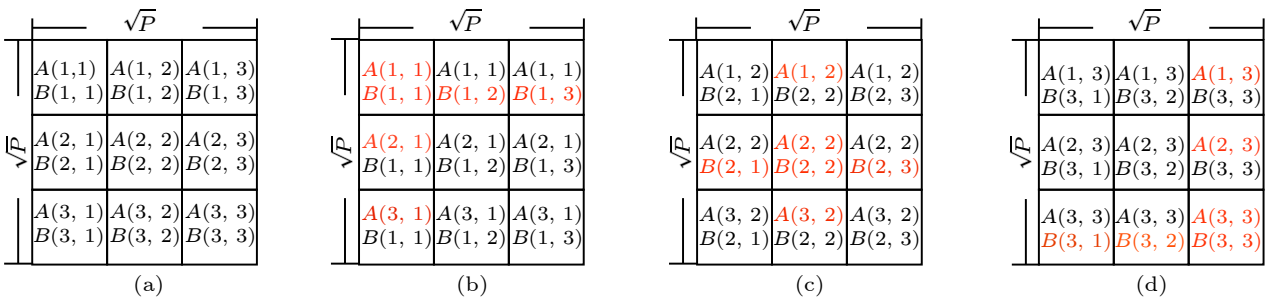

Fig.4. Initial data layout on nine processors for SUMMA and local data of each processor after executing broadcast operations in each round. (a) Initial layout. (b) Round 1. (c) Round 2. (d) Round 3.

size matches the cache capacity. While COWE offers applicability to scenarios with unknown cache sizes, it incurs a higher vertical cost compared with VOMM.

Although there are other existing algorithms such as ScaLAPACK[31], CARMA[18], and COSMA[32], we do not use them as the baselines since these algorithms are used to handle rectangular matrix multiplication and do not perform better than 2.5D for square matrix multiplication[32].

From the previous algorithm description, we analyze the exact costs of these four algorithms and summarize the costs of operations used in the four algorithms in Tables 6 and 7. It should be noted that due to the balanced initial data layout, an additional gather operation is required before the broadcast operation for JOMMA, i.e., an allgather operation.

Given the parameters P, M, S, r, ω, β, and the problem size n, we compare the performance of these four algorithms by evaluating 18 groups of configurations. These configurations cover all the scenarios discussed in this paper. In realistic situations, the write bandwidth of NVM is from 0.20 GB/s to 2.20 GB/s, and the read bandwidth is from 0.63 GB/s to 6.80 GB/s[22, 33]. The data throughput of the InfiniBand architecture NVIDIA Quantum-2 is from 25 GB/s to 50 GB/s. Therefore, in our simulation, we take the value of ω/β from the set {0.5, 1, 2, 8, 16}. In addition, the values of other parameters meet the following restrictions: 1) $n^2/P^{1/2} < M < n^2$ for enough memory; 2) $n^2/P < M < n^2/P^{1/2}$ for limited memory; and 3) $S \leqslant n^2/P$.

Fig.5 illustrates our simulation results. It can be found that in all the tested scenarios, the horizontal costs of 2.5DL3ooL2 and 2.5D-COWE are equal and both are the lowest, while SUMMAL3ooL2 has the lowest vertical cost and JOMMA has the lowest joint-cost. Additionally, 2.5DL3ooL2 outperforms 2.5D-COWE in vertical cost reduction, attributed to VOMM's reduced write demands despite its cache-aware nature (requiring knowledge of cache size). In nearly all examined scenarios, SUMMAL3ooL2 demonstrates notably inferior performance due to its utilization of an irrational horizontal scheduling strategy to achieve vertical optimality. The analysis in Tables 6 and 7 reveals that the horizontal cost of SUMMAL3ooL2 contains a factor n^3, whereas that of the other algorithms contains a factor n^2.

In Fig.5, for small ω/β values (Figs.5(a) and 5(d)), JOMMA and 2.5DL3ooL2 achieve the lowest joint costs, resulting in a 3× speedup compared with SUMMAL3ooL2. This underperformance of SUMMAL3ooL2 is attributed to its excessive horizontal communication. Moreover, the horizontal cost typically surpasses the vertical cost, especially when enough memory is available. When ω/β is large (Figs.5(c) and 5(f)), JOMMA and SUMMAL3ooL2 exhibit equal and the lowest vertical communication costs. Since writing is overly expensive, SUMMAL3ooL2 may achieve a lower joint cost than 2.5DL3ooL2 and 2.5D-COWE. In this scenario, JOMMA demonstrates approximately 3x, 2.1x, and 1.3x performance improvements compared with SUMMAL3ooL2, 2.5D-COWE, and 2.5DL3ooL2, respectively. The horizontal costs of JOMMA, 2.5D-COWE, and 2.5DL3ooL2 significantly increase, nearly by a factor of k^2, as the problem size

Table 6. Exact Costs with Enough Memory

Method	Allgather	Broadcast	Reduce	VOMM/COWE
2.5DL3ooL2	$\frac{n^2\beta}{P_xP_z} + \frac{n^2\beta}{P_yP_z}$	0	$\frac{2n^2\beta}{P_xP_y}$	$\frac{n^2\omega}{P_xP_y} + \frac{2\sqrt{3}n^3r}{PS^{1/2}}$
SUMMAL3ooL2	0	$\frac{4\sqrt{3}n^3\beta}{PS^{1/2}}$	0	$\frac{n^2\omega}{P} + \frac{2\sqrt{3}n^3r}{PS^{1/2}}$
2.5D-COWE	$\frac{n^2\beta}{P_xP_z} + \frac{n^2\beta}{P_yP_z}$	0	$\frac{2n^2\beta}{P_xP_y}$	$\frac{3\sqrt{3}n^3r^{2/3}\omega^{1/3}}{PS^{1/2}}$
JOMMA	$\frac{n^2\beta}{P_xP_z} + \frac{n^2\beta}{P_yP_z}$	0	$\frac{2n^2\beta}{P_xP_y}$	$\frac{n^2\omega}{P_xP_y} + \frac{2\sqrt{3}n^3r}{PS^{1/2}}$

Table 7. Exact Costs with Limited Memory

Method	Allgather	Broadcast	Reduce	Shift	VOMM/COWE
2.5DL3ooL2	$\frac{2n^2\beta}{P_xP_y}$	0	$\frac{2n^2\beta}{P_xP_z}$	$\frac{n^2\beta}{P_xP_y} + \frac{n^2\beta}{P_yP_z}$	$\frac{n^2\omega}{P_xP_z} + \frac{2\sqrt{3}n^3r}{PS^{1/2}}$
SUMMAL3ooL2	0	$\frac{4\sqrt{3}n^3\beta}{PS^{1/2}}$	0	0	$\frac{n^2\omega}{P} + \frac{2\sqrt{3}n^3r}{PS^{1/2}}$
2.5D-COWE	$\frac{2n^2\beta}{P_xP_y}$	0	$\frac{2n^2\beta}{P_xP_z}$	$\frac{n^2\beta}{P_xP_y} + \frac{n^2\beta}{P_yP_z}$	$\frac{3\sqrt{3}n^3r^{2/3}\omega^{1/3}}{PS^{1/2}}$
JOMMA	$\frac{2n^2\beta}{P_xP_y}$	0	$\frac{2n^2\beta}{P_xP_z}$	$\frac{n^2\beta}{P_xP_y} + \frac{n^2\beta}{P_yP_z}$	$\frac{n^2\omega}{P_xP_z} + \frac{2\sqrt{3}n^3r}{PS^{1/2}}$

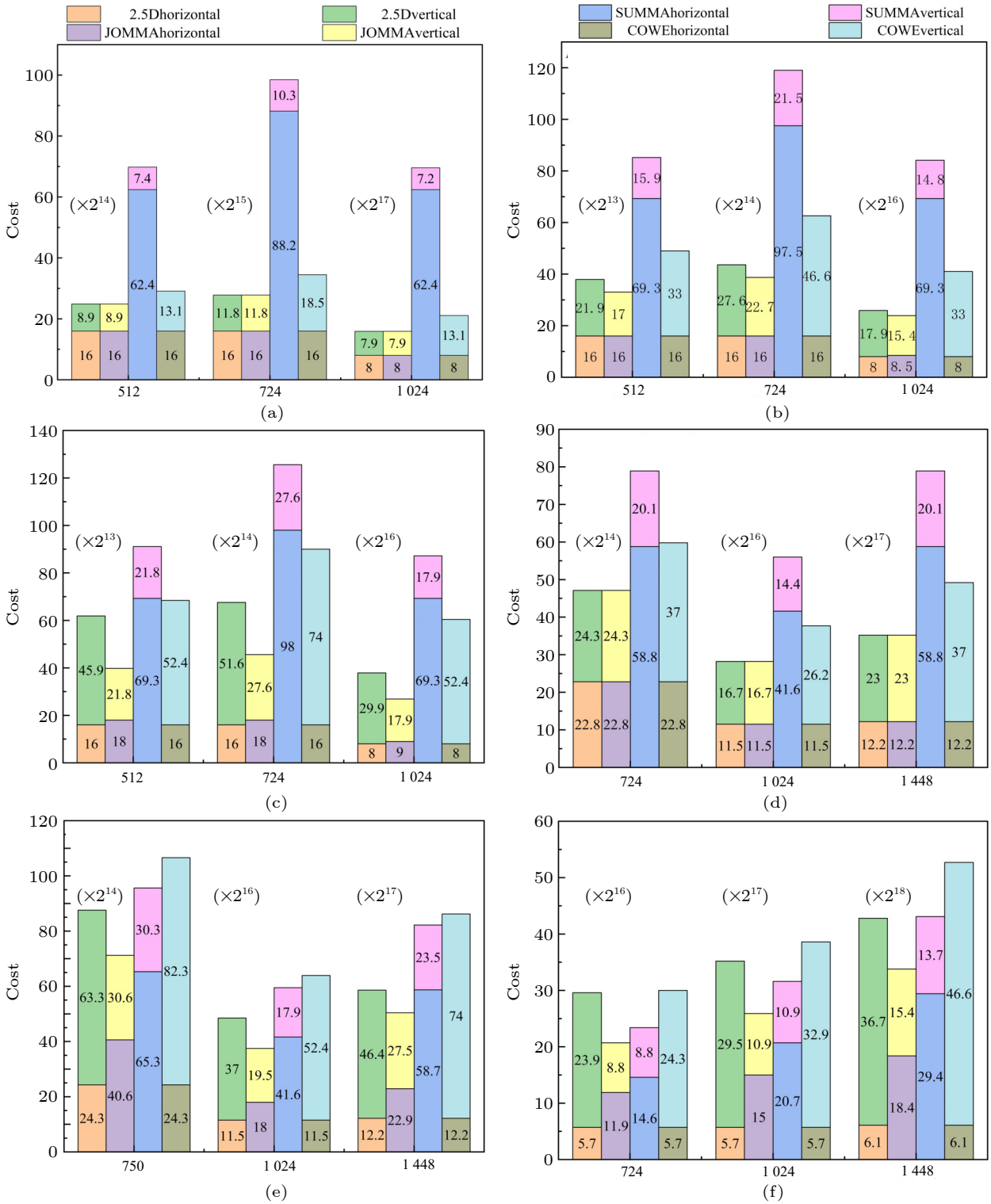

Fig.5. Exact horizontal and vertical costs of 2.5DL3ooL2, JOMMA, SUMMAL3ooL2, and 2.5D-COWE with different machine parameters. For a given problem size n in each figure, the four bars from left to right represent the costs of 2.5DL3ooL2, JOMMA, SUMMAL3ooL2, and 2.5D-COWE, respectively. Each bar's bottom and top parts represent the horizontal and vertical costs, respectively. $P = 2^6$, $S = 2^{12}$, $r = 1$ (a) Enough memory and $2\omega/\beta \leqslant 1$. $M = 2^{18}$, $\omega = 2$, $\beta = 4$. (b) Enough memory and $1 < 2\omega/\beta \leqslant P^{1/2}$ $M = 2^{18}$, $\omega = 4$, $\beta = 2$. (c) Enough memory and $2\omega/\beta \geqslant P^{1/2}$. $M = 2^{18}$, $\omega = 16$, $\beta = 2$. (d) Limited memory and $\omega/\beta \leqslant 1$. $M = 2^{16}$, $\omega = 2$, $\beta = 2$. (e) Limited memory and $1 < \omega/\beta < n^2/M$. $M = 2^{16}$, $\omega = 16$, $\beta = 2$. (f) Limited memory and $\omega/\beta \geqslant n^2/M$. $M = 2^{16}$, $\omega = 32$, $\beta = 2$.

scales up by a factor of k. This increase is primarily attributed to the presence of a factor of n^2 in the horizontal cost function (Tables 6 and 7). When ω/β assumes a moderate value (Figs.5(b) and 5(e)), SUMMAL3ooL2 and 2.5D-COWE exhibit suboptimal performance attributed primarily to excessive horizontal or vertical communication. In this scenario, JOMMA exhibits a performance advantage over the second-best approach (2.5DL3ooL2), achieving a 1.3x speedup. This speedup is expected to increase with larger datasets and processor sizes. For instance, in the enough memory scenario with $2\omega/\beta \geqslant \sqrt{P}$, JOMMA's write count is reduced by a factor of $P^{1/3}$ compared with 2.5DL3ooL2.

9 Conclusions

Motivated by the observation that the horizontal and vertical lower bounds cannot be simultaneously attained for asymmetric memories, in this paper, we investigated how to optimize joint-communication by balancing horizontal communication and writing. We proved the first joint-communication lower bound for classical matrix multiplication, and proposed a joint-communication optimal algorithm that matches the lower bound.

For n-by-n matrix multiplication, there is no tradeoff between communication and computation due to load balancing and a fixed total number of arithmetic operations. However, such a tradeoff may exist for other problems such as sparse iterative solvers[34]. In addition, for asymmetric memories, we leave the discussion of rectangular matrix multiplication as our future work. It will be interesting to extend our work to graph computing and other linear algebra problems. Finally, implementing JOMMA in real distributed systems equipped with asymmetric memories to see the gains of JOMMA over the horizontally optimal ones will also be interesting future work.

Conflict of Interest The authors declare that they have no conflict of interest.

References

[1] Solomonik E, Ballard G, Demmel J, Hoefler T. A communication-avoiding parallel algorithm for the symmetric eigenvalue problem. In *Proc. the 29th ACM Symposium on Parallelism in Algorithms and Architectures*, Jul. 2017, pp.111–121. DOI: 10.1145/3087556.3087561.

[2] Solomonik E, Carson E, Knight N, Demmel J. Tradeoffs between synchronization, communication, and computation in parallel linear algebra computations. In *Proc. the 26th ACM Symposium on Parallelism in Algorithms and Architectures*, Jun. 2014, pp.307–318. DOI: 10.1145/2612669.2612671.

[3] Ma L, Solomonik E. Efficient parallel CP decomposition with pairwise perturbation and multi-sweep dimension tree. In *Proc. the 35th IEEE International Parallel and Distributed Processing Symposium*, May 2021, pp.412–421. DOI: 10.1109/IPDPS49936.2021.00049.

[4] Van De Geijn R A, Watts J. SUMMA: Scalable universal matrix multiplication algorithm. *Concurrency: Practice and Experience*, 1997, 9(4): 255–274. DOI: 10.1002/(SICI)1096-9128(199704)9:4<255::AID-CPE250>3.0.CO;2-2.

[5] Ballard G, Demmel J, Holtz O, Lipshitz B, Schwartz O. Brief announcement: Strong scaling of matrix multiplication algorithms and memory-independent communication lower bounds. In *Proc. the 24th Annual ACM Symposium on Parallelism in Algorithms and Architectures*, Jun. 2012, pp.77–79. DOI: 10.1145/2312005.2312021.

[6] Ballard G, Demmel J, Holtz O, Schwartz O. Minimizing communication in numerical linear algebra. *SIAM Journal on Matrix Analysis and Applications*, 2011, 32(3): 866–901. DOI: 10.1137/090769156.

[7] Irony D, Toledo S, Tiskin A. Communication lower bounds for distributed-memory matrix multiplication. *Journal of Parallel and Distributed Computing*, 2004, 64(9): 1017–1026. DOI: 10.1016/j.jpdc.2004.03.021.

[8] Daas H A, Ballard G, Grigori L, Kumar S, Rouse K. Brief announcement: Tight memory-independent parallel matrix multiplication communication lower bounds. In *Proc. the 34th ACM Symposium on Parallelism in Algorithms and Architectures*, Jul. 2022, pp.445–448. DOI: 10.1145/3490148.3538552.

[9] Agarwal R C, Balle S M, Gustavson F G, Joshi M, Palkar P. A three-dimensional approach to parallel matrix multiplication. *IBM Journal of Research and Development*, 1995, 39(5): 575–582. DOI: 10.1147/rd.395.0575.

[10] Solomonik E, Demmel J. Communication-optimal parallel 2.5D matrix multiplication and LU factorization algorithms. In *Proc. the 17th International Euro-ParConference on Euro-Par 2011 Parallel Processing*, Aug. 29–Sept. 2, 2011, pp.90–109. DOI: 10.1007/978-3-642-23397-5_10.

[11] Huang H, Chow E. CA3DMM: A new algorithm based on a unified view of parallel matrix multiplication. In *Proc. the 2022 International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2022. DOI: 10.1109/SC41404.2022.00033.

[12] Chen Y, Lu Y, Yang F, Wang Q, Wang Y, Shu J. FlatStore: An efficient log-structured key-value storage engine for persistent memory. In *Proc. the 25th International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2020, pp.1077–1091. DOI: 10.1145/3373376.3378515.

[13] Wei X, Xie X, Chen R, Chen H, Zang B. Characterizing

and optimizing remote persistent memory with RDMA and NVM. In *Proc. the 2021 USENIX Annual Technical Conference*, Jul. 2021, pp.523–536.

[14] Taranov K, Rothenberger B, De Sensi D, Perrig A, Hoefler T. NeVerMore: Exploiting RDMA mistakes in NVMeoF storage applications. In *Proc. the 2022 ACM SIGSAC Conference on Computer and Communications Security*, Nov. 2022, pp.2765–2778. DOI: 10.1145/3548606.3560568.

[15] Carson E, Demmel J, Grigori L, Knight N, Koanantakool P, Schwartz O, Simhadri H V. Write-avoiding algorithms. Technical Report UCB/EECS-2015-163, EECS Department, University of California, 2015. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-163.html, May 2025.

[16] Gu Y. Write-efficient algorithms [Ph. D. Thesis]. Carnegie Mellon University, Pittsburgh, 2018.

[17] Carson E C, Demmel J, Grigori L, Knight N, Koanantakool P, Schwartz O, Simhadri H V. Write-avoiding algorithms. In *Proc. the 2016 IEEE International Parallel and Distributed Processing Symposium*, May 2016, pp.648–658. DOI: 10.1109/IPDPS.2016.114.

[18] Demmel J, Eliahu D, Fox A, Kamil S, Lipshitz B, Schwartz O, Spillinger O. Communication-optimal parallel recursive rectangular matrix multiplication. In *Proc. the 27th IEEE International Symposium on Parallel and Distributed Processing*, May 2013, pp.261–272. DOI: 10.1109/IPDPS.2013.80.

[19] Frigo M, Leiserson C E, Prokop H, Ramachandran S. Cacheoblivious algorithms. In *Proc. the 40th Annual Symposium on Foundations of Computer Science*, Oct. 1999, pp.285–298. DOI: 10.1109/SFFCS.1999.814600.

[20] Ballard G, Demmel J, Holtz O, Lipshitz B, Schwartz O. Communication-optimal parallel algorithm for Strassen's matrix multiplication. In *Proc. the 24th Annual ACM Symposium on Parallelism in Algorithms and Architectures*, Jun. 2012, pp.193–204. DOI: 10.1145/2312005.2312044.

[21] Ballard G, Buluç A, Demmel J, Grigori L, Lipshitz B, Schwartz O, Toledo S. Communication optimal parallel multiplication of sparse random matrices. In *Proc. the 25th Annual ACM Symposium on Parallelism in Algorithms and Architectures*, Jul. 2013, pp.222–231. DOI: 10.1145/2486159.2486196.

[22] Ruan C, Zhang Y, Bi C, Ma X, Chen H, Li F, Yang X, Li C, Aboulnaga A, Xu Y. Persistent memory disaggregation for cloud-native relational databases. In *Proc. the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2023, pp.498–512. DOI: 10.1145/3582016.3582055.

[23] Yang C Q, Miller B P. Critical path analysis for the execution of parallel and distributed programs. In *Proc. the 8th International Conference on Distributed*, Jun. 1988, pp.366–373. DOI: 10.1109/DCS.1988.12538.

[24] Hua Q S, Qian L, Yu D, Shi X, Jin H. A nearly optimal distributed algorithm for computing the weighted girth. *Science China Information Sciences*, 2021, 64(11): 212101. DOI: 10.1007/s11432-020-2931-x.

[25] Jia L, Hua Q S, Fan H, Wang Q, Jin H. Efficient distributed algorithms for holistic aggregation functions on random regular graphs. *Science China Information Sciences*, 2022, 65(5): 152101. DOI: 10.1007/s11432-020-2996-2.

[26] Chan E, Heimlich M, Purkayastha A, van De Geijn R. Collective communication: Theory, practice, and experience. *Concurrency and Computation: Practice and Experience*, 2007, 19(13): 1749–1783. DOI: 10.1002/cpe.1206.

[27] Thakur R, Rabenseifner R, Gropp W. Optimization of collective communication operations in MPICH. *The International Journal of High Performance Computing Applications*, 2005, 19(1): 49–66. DOI: 10.1177/1094342005051521.

[28] Hutter E, Solomonik E. Communication-avoiding CholeskyQR2 for rectangular matrices. In *Proc. the 2019 IEEE International Parallel and Distributed Processing Symposium*, May 2019, pp.89–100. DOI: 10.1109/IPDPS.2019.00020.

[29] Hong J W, Kung H T. I/O complexity: The red-blue pebble game. In *Proc. the 13th Annual ACM Symposium on Theory of Computing*, May 1981, pp.326–333. DOI: 10.1145/800076.802486.

[30] Loomis L H, Whitney H. An inequality related to the isoperimetric inequality. *Bulletin of the American Mathematical Society*, 1949, 55(10): 961–962. DOI: 10.1090/S0002-9904-1949-09320-5.

[31] Choi J, Dongarra J J, Pozo R, Walker D W. ScaLAPACK: A scalable linear algebra library for distributed memory concurrent computers. In *Proc. the 4th Symposium on the Frontiers of Massively Parallel Computation*, Oct. 1992, pp.120–127. DOI: 10.1109/FMPC.1992.234898.

[32] Kwasniewski G, Kabić M, Besta M, VandeVondele J, Solcà R, Hoefler T. Red-blue pebbling revisited: Near optimal parallel matrix-matrix multiplication. In *Proc. the 2019 International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2019, Article No. 24. DOI: 10.1145/3295500.3356181.

[33] Song Y, Kim W H, Monga S K, Min C, Eom Y I. Prism: Optimizing key-value store for modern heterogeneous storage devices. In *Proc. the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar. 2023, pp.588–602. DOI: 10.1145/3575693.3575722.

[34] Demmel J, Hoemmen M, Mohiyuddin M, Yelick K. Avoiding communication in sparse matrix computations. In *Proc. the 2008 IEEE International Symposium on Parallel and Distributed Processing*, Apr. 2008. DOI: 10.1109/IPDPS.2008.4536305.

Lin Zhu is currently pursuing his Ph.D. degree in computer science from Huazhong University of Science and Technology, Wuhan. His research interests include parallel algorithms and distributed computing.

Qiang-Sheng Hua received his B.S. and M.S. degrees in computer science from Central South University, Changsha, in 2001 and 2004, respectively, and his Ph.D. degree in computer science from The University of Hong Kong, Hong Kong, in 2009. He is currently a professor with Huazhong University of Science and Technology, Wuhan. He is interested in the algorithmic aspects of parallel and distributed computing.

Hai Jin is a Chair Professor of Computer Science and Engineering at Huazhong University of Science and Technology, Wuhan. Jin received his Ph.D. degree in computer engineering from Huazhong University of Science and Technology, Wuhan, in 1994. His research interests include computer architecture, virtualization technology, distributed computing, big data processing, network storage, and network security.