ANALYSIS OF PARALLEL MATRIX MULTIPLICATION ALGORITHMS

WENJING CUN, LIHUA PEI, AND YUEFAN DENG

Abstract. A phase diagram is created to map and to analyze all the published parallelizations of the naïve matrix multiplication algorithms with computational complexity of $O(N^3)$, and to design new parallelizations, by using decomposition of a 3-dimensional supercube. The properties of these parallel matrix multiplication algorithms are analyzed by theoretical reasoning and several groups of numerical experiments on computer clusters of different numbers of nodes and sizes of buffers for matrix dimensions.

1. Introduction

High-performance computing (HPC) algorithms for linear algebra are critical for many applications in science, engineering and commerce. Matrix multiplication (MM) is at the heart of many linear algebra problems. Much efforts have been devoted to discussing the general method for MM and its parallelization. Several classical parallelization methods have attracted wide acceptance.

Though existing practical PMM algorithms such as SUMMA [1] and PUMMA [2], etc have been designed to deal with various general conditions, they are still not general enough or optimal enough. Recent advances in PMM, or parallel linear algebra in general, exploited heavily on recursions to reduce the total complexity including the Strassen's MM [3] and to divide the PMM into multiple recursive layers with simpler dimensions to optimize the communications among nodes [4]. Based on task mapping [10], we divide the PMM operations with the assumptions that they can be fully processed non-recursively for various general cases with different conditions.

It is desirable to design a map for organizing the families of these parallel matrix multiplication (PMM) algorithms. Using the metric of parallel efficiency of the lattice, we evaluate the upper bounds of the algorithms as a function of the given dimensions of the matrices involved and the computer system parameters.A new PMM algorithm we design is to introduce the adaptivities to the traditional PMM algorithms for handling different practical cases including the varying dimensions of matrices, number of computer nodes, buffer sizes, etc.

Inspired by research on sparse tensor space [8], we introduce a new PMM algorithms based on decomposition of a 3D-supercube to generalize the PMM algorithms with computational complexity $O(N^3)$. By constructing an analytical space, or phase diagram, we map and analyze those algorithms, for comparison and other analyses as well as new designs. Our 3D-supercube PMM algorithm may help construct a denser space for faster computation by avoiding the unnecessary and

Date: July 30, 2018.

Key words and phrases. Parallel computation, Matrix theory.

redundant calculations. This scheme allows optimizing the parallelization to be adaptive to the practical conditions, and to be helpful in predicting and speeding up the tensor production [7]. Furthermore, our approach may help design a general adaptive PMM algorithm for the naïve MM with complexity $O(N^3)$ and beyond for algorithms with complexity lower than $O(N^3)$.

The rest of the manuscript is organized as follows: Section 2 presents the general idea of MM on 3D-supercube, based the maps between sets of coordinates. Section 3 constructs the communication forest among the computation nodes and while proceeding the parallel algorithms, organizing the family of the PMM algorithms based on 3D-supercube. Section 4 proposed the optimization of a new PMM algorithm we call BAMMA. Section 5 analyzes several classical PMM algorithms and proposes a general one. Finally, Section 6 discusses the performance and the reduction of the computational complexity, and summarizes the construction of the algorithms.

2. Construction of matrix multiplication in 3D-supercube

The widely used naïve MM algorithm is

Algorithm 1 (Naive MM Algorithm)

Input: $A \in \mathbb{R}^{m\times l}, B \in \mathbb{R}^{l\times n}$
Output: $C \in \mathbb{R}^{m\times n}$
for $i = 1 : m$ **do**
  for $j = 1 : l$ **do**
    for $k = 1 : n$ **do**
      $C_{ik} = C_{ik} + A_{ij} \cdot B_{jk}$
Return C

The Naïve MM algorithm directly implements the definition of MM for any given pair of matrices $A \in \mathbb{R}^{m\times l}, B \in \mathbb{R}^{l\times n}$ where m, l and $n \in \mathbb{N}_+$. The (i, k) element of the product $C = A \cdot B$

$$C_{ik} = \sum_{j=1}^{l} A_{ij} \cdot B_{jk} \tag{2.1}$$

depends on the matrix elements at (i, j, k). Thus, grouping all (i, j, k) in A and B produce a well-structured map from the set (i, j, k) to the set (i, k) for C and this map does not depend on the values of the matrices' elements.

2.1. Coordinate Set and Operation Map.

Maps between sets of values with associate coordinates are often independent of the values involved, but more like the relationship among the indices of coordinates. So to abstract such relationships, we define the following **operation map (OP-map)**

Definition 2.1. An **operation map** is a map from one set A to another set B with two injective maps $L_1 : C \to A, L_2 : C \to B$ where C is a set with a well-defined operation τ.

For convenience, we introduce an OP-map $f_{C,\tau} : A \to B$, and name L_1, L_2 the ***coordinate maps***, where A and B are the topologies from which the sets of values in C abstracted. Among the many properties of the OP-map, we list one:

Theorem 2.2. *Given an OP-map $f_{G,\tau} : C_1 \to C_2$, if S is closed under a well-defined operation τ, or G is an algebraic group, then for any subset $S \subseteq G$, $f_{S,\tau} : C_1 \to C_2$ is also an OP-map. Vice Versa.*

This is the stability property, *i.e.*, the OP-map is stable regardless of the values when the inputs are conservative of any algebraic operation. The property shows the flexibility, or the scalability in computational engineering, if the OP-map can be conservative in topology

For a series of OP-maps, a **chain rule** applies

Theorem 2.3. *Given a set G that is closed under a series of operations $\tau_1, ..., \tau_n$, a series of sets $C_1, ..., C_n$, also a series of subsets of $G, S_1, ..., S_n$, there exist n OP-maps*

$$(f_1)_{S_1,\tau_1}, ..., (f_n)_{S_n,\tau_n} \tag{2.2}$$

where $(f_1)_{S_1,\tau_1} \circ ... \circ (f_n)_{S_n,\tau_n}$ is also an OP-map.

Conversely, if an OP-map $f_(G, \tau) : C_1 \to C_2$ can be decomposed into a series of OP-maps, then $f_{G,\tau}$ is ***separable***.

Theorem 2.4. *Given two homomorphic algebraic structures G_1 and G_2, closed under two operations τ_1 and τ_2 respectively, if $S_1 \subseteq G_1$ and $S_2 \subseteq G_2$ are two subsets, and an OP-map $f_{S_1,\tau_1} : C_1 \to C_2$, then $f_{S_2,\tau_2} : C_1 \to C_2$ is also an OP-map.*

Implementing the above, we abstract the MM, based Algorithm 1, into a separable OP-map consisting of several independent operation steps, which helps introduce its parallelization.

2.2. **Algorithms Based on the 3D-Supercube.** To abstract Algorithm 1 into the OP-maps, we first gather all multiplications between pairs of scalars in MM. For the 3D-supercube algorithm, we construct an OP-map from a coordinate set C_{11} to another C_{12}. Since the original coordinate set represents the positions of the elements of A and B, whose coordinates are

$$C_A = \mathbb{Z}/m\mathbb{Z} \times \mathbb{Z}/l\mathbb{Z}, C_B = \mathbb{Z}/l\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z} \tag{2.3}$$

The OP-map is

$$(f_1)_{\mathbb{R},\times} : C_A \times C_B \to C_{\text{mult3D}} \tag{2.4}$$

where $C_{\text{mult}} = \mathbb{Z}/m\mathbb{Z} \times \mathbb{Z}/l\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z}$, "$\times$" is the scalar multiplication and

$$(f_1)_{\mathbb{R},\times}((r_a, c_a), (r_b, c_b)) = \begin{cases} (r_a, c_b, c_a), & \text{if } c_a = r_b \\ 0, & \text{o.w.} \end{cases} \tag{2.5}$$

This OP-map represents the operation of $A_{ij} \cdot B_{jk}$, and from C_{mult}, all mapping can be illustrated in Fig.1

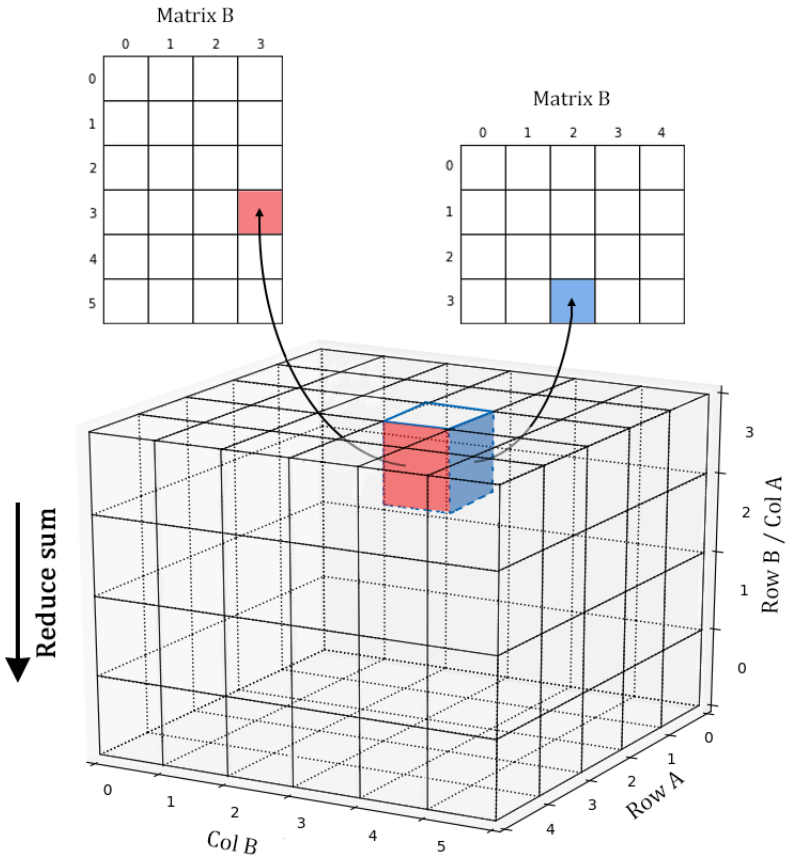

FIGURE 1. The OP-map$(f_1)_(\mathbb{R}, \times) : C_A \times C_B \to C_{\text{mult}}$

Since only the scalar operations remain to compose the result $C \in \mathbb{R}^{m\times n}$, we define the third coordinate set

$$C_C = \mathbb{Z}/m\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z} \tag{2.6}$$

whose corresponding OP-map is

$$\begin{aligned} &(f_2)_{R,+} : C_{\text{mult3D}} \to C_C \\ &(f_2)_{R,+}((r_a, c_b, c_a)) = (r_a, c_b) \end{aligned} \tag{2.7}$$

where "+" indicates the scalar addition. This equation reveals that all the 3D "cubes" in the Fig. 1 with the same "z-value" are reduced to one in the "xy-plane", illustrated in Fig. 2.

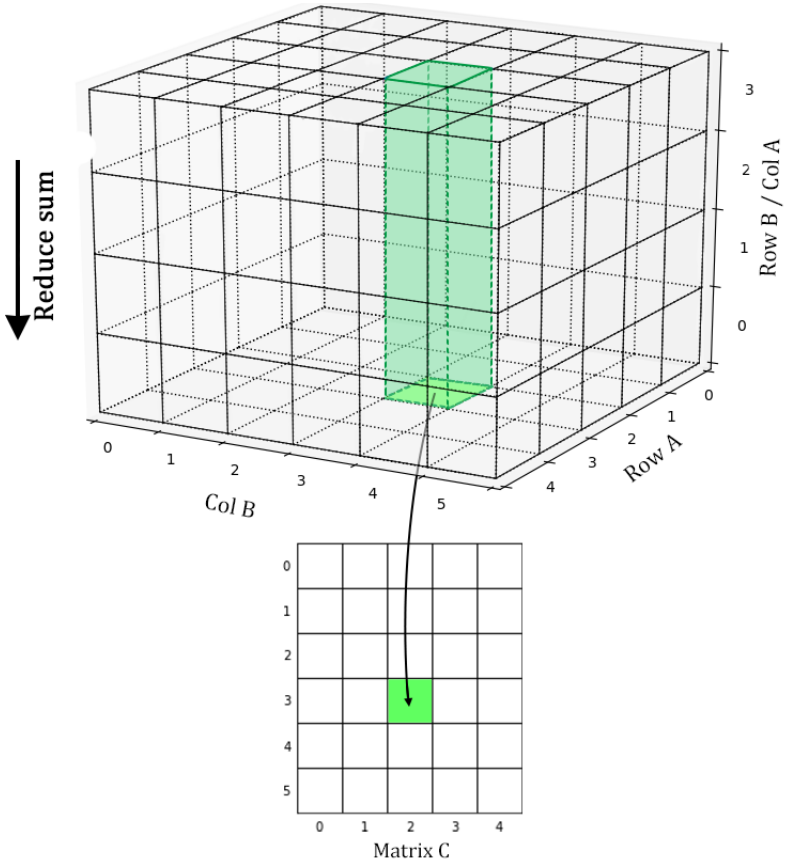

FIGURE 2. OP-map $(f_2)_{F^{m\times l\times n},\text{sum}}$

The MM is then abstracted into a chain of OP-map $F^{m\times l}\times F^{l\times n}\to F^{m\times n}$ defined as

$$f_{\mathbb{R}^{m\times l}\times\mathbb{R}^{l\times n},\text{MM}}=(f_1)_{\mathbb{R},\times}\circ(f_2)_{\mathbb{R},+} \tag{2.8}$$

where $f_{\mathbb{R}^{m\times l}\times\mathbb{R}^{l\times n},\text{MM}}(x)=x$, is an abstract OP-map directly representing $C=AB$.

For a system with local-memory, operation (2.5) does not need any change while for a parallel system with distributed memories, we add one more OP-map to each computing node p, $g_{R,=}$, where "=" is the identity operation, but can filter the coordinates for each node, and we define it as

$$\begin{aligned} &g_{\mathbb{R},=}:C_{\text{mult3D}}\to C_{\text{mult3D}}\\ &g_{\mathbb{R},=}((r_a,c_b,c_a))=\begin{cases}(r_a,c_b,c_a), & \text{if } (r_a,c_b,c_a)\in U_p\\ 0, & \text{o.w.}\end{cases}\end{aligned} \tag{2.9}$$

where U_p is a custom set chosen for each node p, called a ***filtering set***, and it represents the "blocks" chosen for node p to calculate. Thus, the chain of OP-maps for each node p is written as

$$(f_p)_{\mathbb{R}^{m\times l}\times\mathbb{R}^{l\times n},\mathrm{MM}} = (f_1)_{\mathbb{R},\times} \circ g_{\mathbb{R},=} \circ (f_2)_{\mathbb{R},+} \tag{2.10}$$

expressing all PMM algorithms based on the Naïve MM.

For example, a computer with 4 nodes computes $A \cdot B$, where $A \in \mathbb{R}^{6\times 4}$, $B \in \mathbb{R}^{4\times 5}$. The four filtering sets are

$$\begin{aligned} U_1 &= \{4,5\} \times \{2,3,4\} \times (\mathbb{Z}/4\mathbb{Z}) \\ U_2 &= \{3\} \times \{2,3,4\} \times (\mathbb{Z}/4\mathbb{Z}) \\ U_3 &= \{3,4,5\} \times \{0,1\} \times (\mathbb{Z}/4\mathbb{Z}) \\ U_4 &= \{0,1,2\} \times \{0,1,2,3,4\} \times (\mathbb{Z}/4\mathbb{Z}) \end{aligned} \tag{2.11}$$

The "blocks" in $C_\times$ allotted to each node are shown in Fig. 3

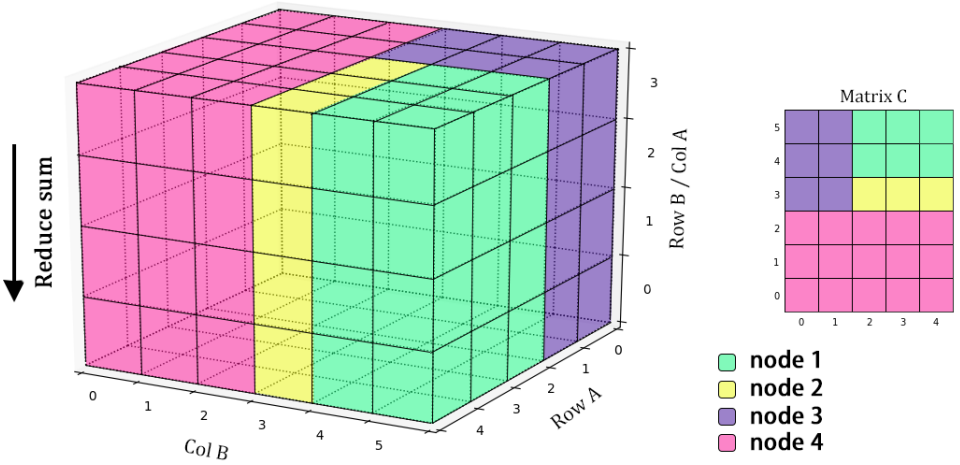

FIGURE 3. Parallel 3D-Hypercube

The strategy of choosing the filtering sets as single "cubes" that all span over the z-axis is similar to the BMR method [1] without considering communications among nodes. Several variations of the strategy will result in the other well-known PMM algorithms including the Cannon's algorithm [2] and SUMMA [3] if one step is needed.

When considering only the computational part of the MM, all PMM algorithms based on the naïve MM Algorithm 1 can be deduced by selecting the filtering sets U_p and expressed by the following pseudocode

Algorithm 2 (**3D-Supercube MM**)

Input: $A \in \mathbb{R}^{m\times l}, B \in \mathbb{R}^{l\times n}$
Output: $C \in \mathbb{R}^{m\times n}$
if at rank p **do**
    choose $U_p = \{(i_1,j_1,k_1),(i_2,j_2,k_2),...,(i_K,j_K,k_K)\}$
    for i,j,k in U_p **do**
        $C_{ik} = C_{ik} + A_{ij} \cdot B_{jk}$
Return C

An appropriate arrangement of the lattices in the 3D-supercube often helps minimize communication, although small buffer can district such optimization for large matrices. Therefore, minimizing communication requires deploying the best subset $V_p \subseteq U_p$ and finding the best communication tree for data sharing.

We will design a general map for PMM and analyze the impact of buffer sizes on the communication cost and the overall PMM performance.

3. Communication in 3D-supercube

Allocation of blocks of operations to each node is straightforward for 3D-supercube PMM while dealing with large matrices, small buffer will limit the size of data stored on each node, necessitating substantial data motions. The key of optimizing the PMM is to minimize such data movement.

Properly initiating data distribution will help lower communication. For PMM, Algorithm 2 infers that only part of U_p is assigned to node p before following the communication tree to enable computation tasks with its own local data.

3.1. **Communications.** For convenience, a "cube" assigned to node p can be labeled by m_p, l_p, n_p, the lengths of the lattice assigned to node p, along with the x-, y and z-axis, respectively;

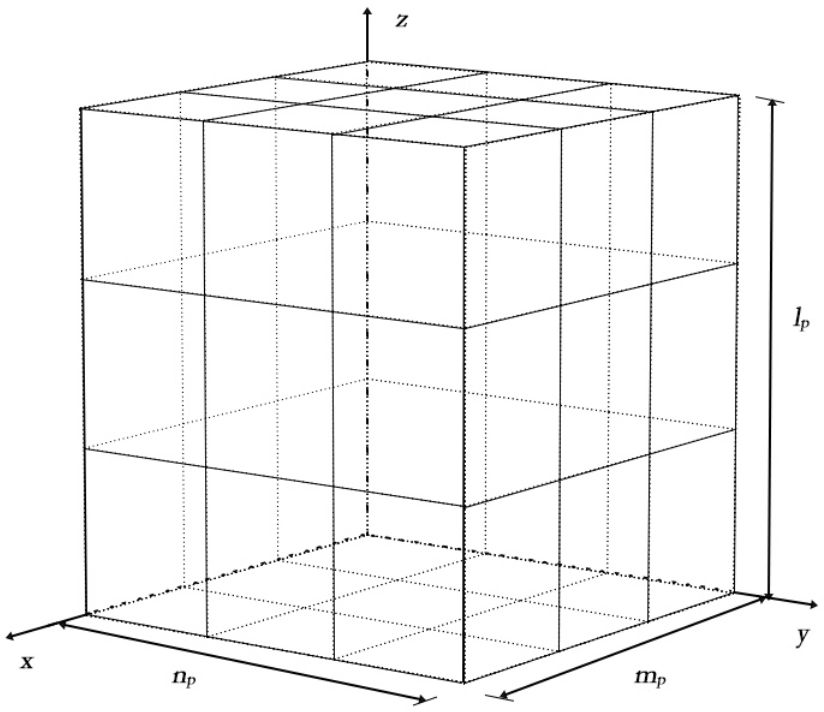

Figure 4. Cube onside for the one node p

For many algorithms in 3D-supercube family, such P communication trees are similar in the following sense:

- Volume, *i.e.*, the number of multiplication between entries;

- Total projection area onto xz and yz-planes, *i.e.*, the total size of data needed for computations;
- Maximum area projected to xz and yz-planes can be covered at each communication step, *i.e.*, the maximum buffer assigned to each core, denoted as B_p.
- Total projection area onto xy-plane, *i.e.*, the size of data to be reduced onto one node.

To design the PMM, we follow the following rules

Rule 3.1 The total volume of the P lattices is exactly mln, *i.e.*, the total volume of the 3D-supercube. The total non-repetitive computations among the nodes is

$$\sum_{p=1}^{P} m_p l_p n_p = mln \tag{3.1}$$

Rule 3.2 The entries of two matrices can be covered by the data stored in the P nodes at each communication step and the total required buffer is

$$\sum_{p=1}^{P} \mathrm{B}_p \geq ml + ln \tag{3.2}$$

Rule 3.3 Any entry (or block) stored local to a node can only be released if and only if all the scalar (block) multiplications involving it have been finished. The total cost of receiving data on node p is

$$T_{p,\mathrm{recv}} = C \cdot (m_p l_p + l_p n_p - \mathrm{B}_p) \tag{3.3}$$

This rule shows that, since the total amount of data sent and received, $T_{\mathrm{comm,sendrecv}}$, is equal, the average cost of data communication for each node is

$$\begin{cases} T_{\mathrm{comm,sendrecv}} \leq \sum_{p=1}^{P} T_{p,\mathrm{recv}} & \text{in serial} \\ T_{\mathrm{comm,sendrecv}} \geq \max\limits_{p} \{T_{p,\mathrm{recv}}\} & \text{Communications in parallel} \end{cases} \tag{3.4}$$

where the lower bound indicates the strategies for minimizing waiting, where data are passes in parallel; the largest data determines the true total communication time.

3.2. **PMM Algorithm.** The parallel system is assumed homogeneous for which all nodes contain similar CPU, GPU, RAM. This allows division of the 3D-supercube into P similar lattices with each assigning similar amount of buffer. Each such logical lattice is assigned to a physical node.

So numerate $U_p = \{(x_{pi}, y_{pi}, z_{pi})\}$ by taking the OP-map from $A_{pk} \times B_{pk}$ at each communication step k, where it should satisfy

$$|C_{pAk}| + |C_{pBk}| \leq \mathrm{B}_p \tag{3.5}$$

In this PMM, adding the communication costs in 3D-supercube enables the conservation of the form of the OP-maps and it also allows the construction of a union of a series of OP-maps

$$\begin{aligned} & (f_{p1})_{\mathbb{R},\times} : C_{pA} \times C_{pB} \to C_{p,mult3D} \\ \Longrightarrow \quad & C_{p,mult3D} = \bigcup_{k=1}^{\text{total steps } K} C_{p,mult3D,k} = \bigcup_{k=1}^{K} C_{pAk} \times C_{pBk} \end{aligned} \tag{3.6}$$

Now, minimizing communication for each node with fixed buffer size can be handled by

Theorem 3.1. *To numerate $U_p = \{(x_{pi}, y_{pi}, z_{pi})\}$, for $\{C_{pAk}\}$ and $\{C_{pBk}\}$ satisfying (14), there exist $\{C_{pAk}\}$ and $\{C_{pBk}\}$ so that, for increment $\triangle k > 1$,*

- *$d \in C_{pAk} \cap C_{pA,(k+\triangle k)}$ only if $d \in C_{pAk} \cap C_{pA,(k+\triangle k-1)}$.*
- *$d \in C_{pBk} \cap C_{pB,(k+\triangle k)}$ only if $d \in C_{pBk} \cap C_{pB,(k+\triangle k-1)}$.*

given $\mathrm{B}_p \geq \min\{m_p, n_p\}$.

This theorem states the fact of Rule 3.3 that no redundant data movement exists. The given condition permits a general strategy for data movement.

This strategy with the condition $\mathrm{B}_p \geq \min\{m_p, n_p\}$ can be stated as

Algorithm 3 (3D-Supercube Algorithm with Communications)
$(m_p, l_p, n_p, \mathrm{B}_p \in \mathbb{N}^+, m_p \leq n_p)$

Input: $A \in \mathbb{R}^{m \times l}, B \in \mathbb{R}^{l \times n}$, $m_p, l_p, n_p, \mathrm{B}_p \in N^+, (m_p \leq n_p)$
Output: $C \in \mathbb{R}^{m \times n}$
if at rank p **do**
Choose $U_p = \{(i_1, j_1, k_1), (i_2, j_2, k_2), ..., (i_K, j_K, k_K)\}$
$\subset (\mathbb{Z}_{m_p} \times \mathbb{Z}_{l_p} \times \mathbb{Z}_{n_p}) \bigoplus (i_{row}, j_{row}, k_{row})$
Let $C_{pA} = (Z_(m_p) \times Z_(l_p)) \bigoplus (i_{low}, j_{low})$ and
$C_{pB} = (Z_(l_p) \times Z_(n_p)) \bigoplus (j_{low}, k_{low})$.
Sort U_p, C_{pA} and C_{pB} increasingly with j.
Let $\mathrm{StepSize}_A = \lfloor \mathrm{B}_p / m_p \rfloor$
and initialize the buffer as

$$C_{pA,0} = C_{pA}[0 : \mathrm{StepSize}_A], C_{pB,0} = C_{pB}[0 : \mathrm{B}_p - \mathrm{StepSize}_A]$$

Set numbers of steps associated to two sets as $k_A = k_B = 0$.
for i, j, k in U_p,
if $(i, j) \in C_{pA,k_A}$ and $(j, k) \in C_{pB,k_B}$ **do**
$C_{ik} = C_{ik} + A_{ij} \cdot B_{jk}$
ready to send $(i, j) \in C_{pA,k_A}$ and $(j, k) \in C_{pB,k_B}$
else if $(i, j) \notin C_{pA,k_A}$**do**
require (i, j) from another node,
pop the front of C_{pA,k_A},
push (i, j) into the back of C_{pA,k_A},
$k_A = k_A + 1$.
else if $(j, k) \notin C_{pB,k_B}$**do**
require (j, k) from another node,
pop the front of C_{pB,k_B},
push (j, k) into the back of C_{pB,k_B},
$k_B = k_B + 1$.
If rank = root, gather results;
Return C

which designs a communication tree for each node and, with the tree, we construct an algorithm with two conditions

- m_p, l_p, n_p, a lattice can be assigned to each node p;

- B_p is known.

The algorithms based on Algorithm 3, which we call the **Buffer Adaptive Matrix Multiplication Algorithm (BAMMA)**, can handle matrices with arbitrary dimensions for optimal utilization of given buffer on each node.

3.3. **Performance Evaluation.** To evaluate the BAMMA performance, we compute three costs of computation, communication, and idle due to data waiting

$$T_P = \max_p \{T_{p,\text{comm}} + T_{p,\text{comp}} + T_{p,\text{idle}}\} \tag{3.7}$$

where the $T_{p,\text{idle}}$ is the waiting time on node p.

If we assign similar number of lattices and similar buffer to each of the P nodes and we assume a minimal $T_{p,\text{idle}}$ is attainable by building static routines for the communication forest, total cost can be estimated by approximating (3.7) as

$$T_P \approx T_{p,\text{comm}} + T_{p,\text{comp}} \tag{3.8}$$

Considering the practical cases in clusters, the average time for communicating one floating number as t_{comm} and that for performing one floating-point operation as t_{comp}, we define

$$\gamma = \frac{t_{\text{comm}}}{t_{\text{comp}}} \tag{3.9}$$

which is an elementary constant determined by the given hardware.

The computational cost $T_{p,comp}$, estimated by the OP-map defined as (2.8), contains two parts: multiplication and addition and it is

$$T_{p,\text{comp}} = T_{p,\text{mult}} + T_{p,\text{add}} = (m_p l_p n_p + m_p l_p n_p - m_p n_p) \cdot t_{\text{comp}} \approx 2 m_p l_p n_p \cdot t_{\text{comp}} \tag{3.10}$$

Give that each node p containing similar cubes, we have

$$m_p l_p n_p \approx \frac{mln}{P} \tag{3.11}$$

So the computational cost is

$$T_{p,\text{comp}} \approx \frac{2mln}{P} \cdot t_{\text{comp}} \tag{3.12}$$

The communication cost consists of two parts: send-receive to enable computations on each lattice and the reduction for the final

$$T_{\text{comm}} = T_{\text{SR}} + T_{\text{reduce}} \tag{3.13}$$

where T_{SR} denotes the time for sending and receiving and T_{reduce} denotes the reduction operations. This reduction step may be ignored for practical applications of PMM in which a scattered resulting matrix may be more useful for the next steps than an assembled matrix.

For reduction, the total cost is that of sending the total number of entries to the root node, along the z-axis, which is also the total projection onto the xy-plane from the P nodes. For each step, it involves one addition and one communication, thus:

$$T_{\text{reduce,total}} = (t_{\text{SR}} + t_{\text{op}}) \sum_{p=1}^{P} Area_{p,xy} = t_{\text{SR}} + t_{\text{op}}) \sum_{p=1}^{P} m_p n_p \tag{3.14}$$

where t denotes the time for unit communication among nodes to finish only the multiplication, and t_{reduce} denotes the time for a unit reduction with summations.

For the communication part, each node can only send or receive data at a single step and these two operations cost the same. By (3.14) and the assumption that each node performs similar numbers of sending and receiving, we estimate the lower bound of the communication cost

$$T_{\text{SR}} \geq C \cdot \max_p\{m_p l_p + n_p l_p - \text{B}_p\} \cdot t_{\text{SR}} \tag{3.15}$$

where $C_{\text{SR}} \sim 2$ is an hardware-dependent constant.

The total cost is

$$T_{\text{total},P} \geq \frac{2mln}{P} \cdot t_{\text{op}} + (t_{\text{SR}} + t_{\text{op}}) \cdot \sum_{p=1}^{P} m_p n_p + t_{\text{SR}} \cdot 2 \cdot \max_p\{m_p l_p + n_p l_p - \text{B}_p\} \tag{3.16}$$

For analysis convenience of the general PMM algorithms in the 3D-supercube, we define two new variables: Z**-variance** k_z and **maximum saturability** $\bar{S}$ as

$$\kappa_z := \frac{\sum_{p=1}^{P} m_p n_p}{mn}, \quad \bar{S} := 1 - \max_p \left\{ \frac{m_p l_p + n_p l_p - \text{B}_p}{l(m+n)} \right\} \tag{3.17}$$

where $\kappa_z \geq 1$ and $S \in (0, 1]$.

By parallelizing the reduction, the process to "compress" the lattices can be optimized as a flipped binary tree and the cost is much lower than that by (3.14) as

$$T_{\text{reduce}} = (t_{\text{SR}} + t_{\text{op}}) \cdot \log_2 \kappa_z \cdot mn \cdot \frac{\kappa_z}{P} \tag{3.18}$$

Similarly, we found

$$\bar{S} := 1 - \frac{\bar{m}_p \bar{l}_p + \bar{n}_p \bar{l}_p - \bar{\text{B}}_p}{l(m+n)} \tag{3.19}$$

where $\bar{m}_p, \bar{l}_p, \bar{n}_p$ and $\bar{\text{B}}_p$ are set equal for every node. Considering the total cost of the MM on a single node

$$T_{\text{single}} = (2mln - mn) \cdot t_{\text{op}} \tag{3.20}$$

we obtain the parallel efficiency

$$\begin{aligned} E(m,l,n,P) \leq & \frac{\frac{1}{P} \cdot (2mln - mn) \cdot t_{\text{op}}}{\frac{2mln - mn}{P} \cdot t_{\text{op}} + (t_{\text{SR}} + t_{\text{op}}) \cdot \log_2 A_z \cdot mn \cdot \frac{\kappa_z}{P} + t_{\text{SR}} \cdot 2 \cdot l(m+n)(1-\bar{S})} \\ \approx & \frac{mln}{mln + (\frac{1+\gamma}{2} \cdot \log_2 \kappa_z \cdot mn \cdot \frac{\kappa_z}{+} P \cdot \gamma \cdot l(m+n)(1-\bar{S}))} \end{aligned} \tag{3.21}$$

In practice, $l \gg 1$, which helps simplify (3.21) as

$$E(m,l,n,P) \leq \frac{1}{1 + \frac{1+\gamma}{2l} \cdot \kappa_z \cdot \log_2 \kappa_z + P \cdot \gamma \cdot (\frac{1}{m} + \frac{1}{n})(1-\bar{S})} \tag{3.22}$$

where $\bar{\kappa}_z = 1 - \frac{\log_2 \kappa_z}{\log_2 P} = 1 - \log_P k_z \in (0, 1]$, each node can at most host a projection area mn on the xy-plane for any one column along the z-axis, at most P segments can be placed to different nodes. Thus,

$$E(m,l,n,P) \leq \frac{1}{1 + \frac{1+\gamma}{2l} \cdot \log_2 P \cdot P^{1-\bar{\kappa}_z}(1-\bar{\kappa}_z) + P \cdot \gamma \cdot \left(\frac{1}{m} + \frac{1}{n}\right)\left(1 - \bar{S}\right)} \tag{3.23}$$

(3.23) establishes the relationship between the parallel efficiency of a PMM algorithm and the dimensions of matrices, buffer size, and the number of nodes, with which we can compute the upper bound of the parallel efficiency with given m, l, n and P. And we can also conveniently identify the overhead as

$$h(m,l,n,P) \approx \frac{1+\gamma}{2l} \cdot \log_2 P \cdot P^{(1-\bar{\kappa_z})} \cdot (1-\bar{\kappa_z}) + P \cdot \gamma \cdot \left(\frac{1}{m} + \frac{1}{n}\right)(1 - \bar{S}) \tag{3.24}$$

More importantly, we can identify the PMM algorithms in the phase diagram by their parallel efficiencies and parameters $\bar{S}$ and $\bar{\kappa_z}$. In formal analysis, maximizing $E(m,l,n,P)$ is equivalent to minimizing $h(m,l,n,P)$, whichever more convenient is used for analysis.

3.4. **3D-Supercube.** It is difficult to use (3.23) to choose m_p, l_p, n_p for given m, l, n and B_p, but the PMM performances for matrices with given dimensions can be estimated.

The well-known PMM algorithms including Cannon's method [2], Fox method [1], and SUMMA [3] are special cases of our general method by particular divisions of the supercube as shown in Fig. 5.

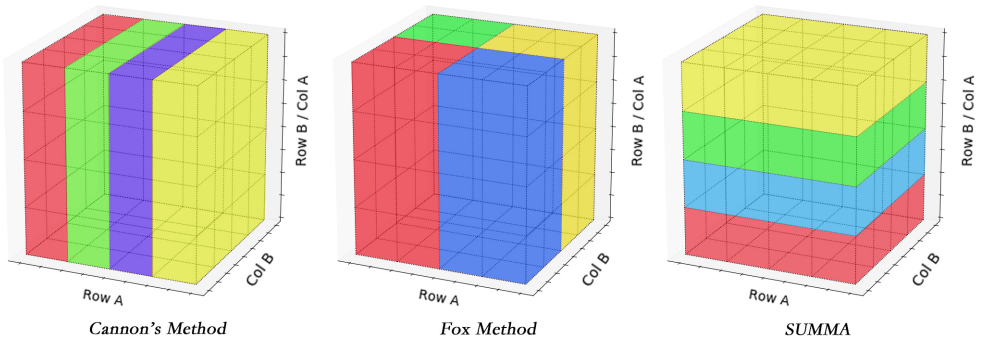

FIGURE 5. Some Classical Methods

The dimensions of matrices dicatate the design of the lattices that lead to different performances. We plot three different color maps to show the distributions of 3D-supercube for PMM algorithms on computers with $P = 8, 32, 64$ cores, with the metric of parallel efficiency $E(m,l,n,P)$ defined in (3.22), or similarly in (3.23), under 3 different sets of dimensions of m, l and n, with the assumption that $\gamma = 1$

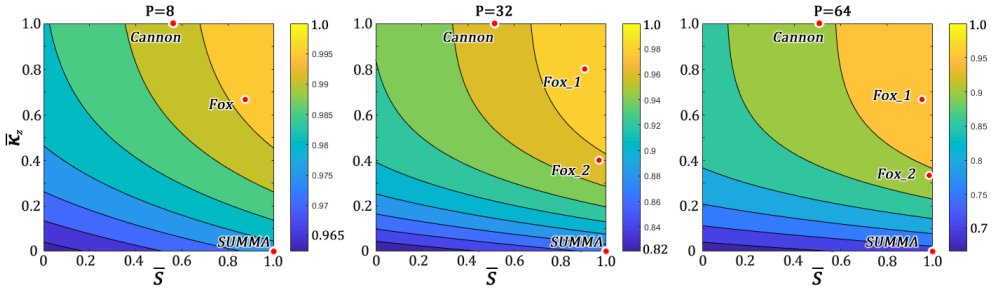

FIGURE 6-a. Color Maps of $E(m, l, n, P)$ for Square Matrices ($m = l = n = 1024$)

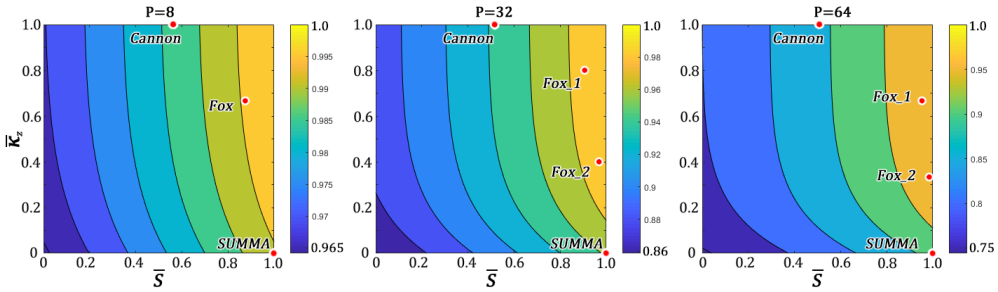

FIGURE 6-b. Color Maps of $E(m, l, n, P)$ for Tall Matrices ($m = n = 512, l = 4096$)

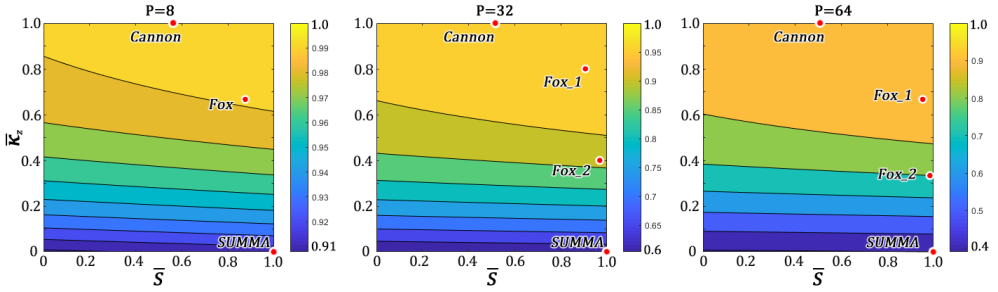

FIGURE 6-c. Color Maps of $E(m, l, n, P)$ for Flat Matrices ($m = n = 2048, l = 256$)

These phase diagrams reveal the general ideas of constructing, and the properties of, the PMMs using the 3D-supercube. We observe

- It is hard to increase $\bar{\kappa}_z$ and $\bar{S}$ with limited buffer size;
- When $l \gg m, n$, it is more efficient to slice the supercube along z-axis, which means smaller $\bar{S}$ can compensate for the smaller $\bar{\kappa}_z$;
- When $l \ll m, n$, it is more efficient to let each lattice cover more blocks along the z-axis;

We also observe

- if a lattice on node P is more cubic, *i.e.*, $m_p \approx l_p \approx n_p$ then with fixed volume $m_p l_p n_p, m_p l_p + n_p l_p$ can be smaller for increasing $\bar{S}$;
- the larger the B_p, the bigger both $\bar{\kappa}_z$ and $\bar{S}$;
- the smaller the communication-to-computation ratios, the larger the efficiency, as expected.

For given matrices dimensions, buffer size and other hardware parameters, our strategies can help divide the 3D-supercube appropriately for PMM to achieve optimal parallel efficiency.

4. Optimization using the 3D-supercube

Using (3.23), we can optimize the BAMMA parallel efficiency by constructing the 3D-supercubes for matrices with given dimensions.

The quantized features of the performance of a 3D-supercube PMM algorithm are valuable features for identifying the existing PMM's, and searching for new PMM algorithms that are optimal. To formalize the optimization, we introduce

Definition 4.1. Partitioning a 3D-supercube is a set of partitions Ω_i, $\forall i \leq P$, where each partition Ω_i is a set of coordinates, namely $\Omega_i = \{(x_{ij}, y_{ij}, z_{ij})\}_{j=1}^{N_i}$.

4.1. **One-Time-I/O Group.** The on-time-I/O, data in and out, only occurs once but it dose require careful planning to minimize the I/O and to make the best use of RAM and caches. We want to search for a group of **One-time-I/O algorithms**, among all possibilities that BAMMA can reach with given buffer sizes, to scatter the two matrices. The heatmaps (Fig. 6-abc) show the one-time-I/O PMM algorithms for a given set of dimensions. We fit a curve with the discrete points of the One-time-I/O curves vs. the matrices dimensions (m, l, n) on P nodes with total buffer sizes. Such a series of curves form an isogram each of which corresponds to a given buffer size.

For analysis of the partitioning that similar-shaped supercubes for each node, we introduce two new variables

– right variance (or x variance) $\kappa_x = \frac{1}{ml} \cdot \sum_{p=1}^{P} m_p l_p$

– left variance (or y variance) $\kappa_y = \frac{1}{ln} \cdot \sum_{p=1}^{P} l_p n_p$

which represent the average numbers of nodes, the entries of matrices A or B need to be transferred. If we only want to assign all the entries onto all the nodes, without requiring extra times for I/O, or assigning any entry of the matrices onto more than one node, we make such "curves", *i.e.*, the minimal-buffer curves, to link all such algorithms on the κ_z-$\bar{S}$ plane, Here is a list of observations

$$\begin{aligned} \text{Area of } B &= ln = \kappa_x \cdot \tfrac{1}{P} \cdot \textstyle\sum_p \mathrm{B}_{p,B} \\ \text{Area of } A &= ml = \kappa_y \cdot \tfrac{1}{P} \cdot \textstyle\sum_p \mathrm{B}_{p,A} \\ \text{Total Area} &= l(m+n) = \textstyle\sum_p \mathrm{B}_p \end{aligned} \tag{4.1}$$

where $B_{p,B}$ and $B_{p,A}$ represent the sizes of buffer on node p used to store the data from B and A respectively. Since the supercubes are all made cubic, we can assume

$$P = \kappa_x \kappa_y \kappa_z \tag{4.2}$$

Using (3.19) for the maximum saturability $\bar{S}$

$$\bar{S}_p = 1 - \frac{\bar{m}_p \bar{l}_p + \bar{n}_p \bar{l}_p - \bar{\mathrm{B}}_p}{l(m+n)} \tag{4.3}$$

We get

$$
\begin{aligned}
P \cdot \bar{S}_p &= P - \frac{P \cdot \bar{m}_p \bar{l}_p + P \cdot \bar{n}_p \bar{l}_p - P \cdot \bar{\mathrm{B}}_p}{l(m+n)} \\
&= P - \frac{\kappa_y \cdot ml + \kappa_x \cdot ln - l(m+n)}{l(m+n)} \\
&= P + 1 - \frac{\kappa_y \cdot ml + \kappa_x \cdot ln}{l(m+n)} \\
&= P + 1 - \frac{\frac{P}{\kappa_x \kappa_z} \cdot ml + \kappa_x \cdot ln}{l(m+n)}
\end{aligned} \tag{4.4}
$$

Thus,

$$
\bar{S}_p = 1 + \frac{1}{P} - \frac{\frac{P}{\kappa_x \kappa_z} \cdot m + \kappa_x \cdot n}{P \cdot (m+n)} \tag{4.5}
$$

Using the normalized $\bar{\kappa_z} = 1 - \log_P \kappa_z$, we get

$$
\bar{S} = 1 + \frac{1}{P} - \frac{\kappa_x \cdot n + \frac{P}{\kappa_x P^{1-\bar{\kappa}_z}} \cdot m}{P \cdot (m+n)} = 1 + \frac{1}{P} - \frac{\frac{P^{\bar{\kappa}_z}}{\kappa_x} \cdot m + \kappa_x \cdot n}{P \cdot (m+n)} \tag{4.6}
$$

The one-time-I/O curve is non-unique, due to different strategies for cutting along the x-axis or y-axis. The group of curves can distribute on the $\bar{S} - \bar{\kappa}_z$ plane, subject to the following constraint of

$$
\begin{aligned}
P = \kappa_x \kappa_y \kappa_z \quad &\Longrightarrow \quad P = \kappa_x \kappa_y P^{1-\bar{\kappa}_z} \\
&\Longrightarrow \quad \kappa_x \kappa_y = P^{\bar{\kappa}_z} \\
&\Longrightarrow \quad \kappa_x \in [1, \bar{\kappa}_z]
\end{aligned} \tag{4.7}
$$

Slicing along the $\bar{\kappa}_z$-axis, we can plot all the theoretical **minimal-buffer** curves on the heatmap. Fig. 7 is one example of $m = l = n = 1024$.

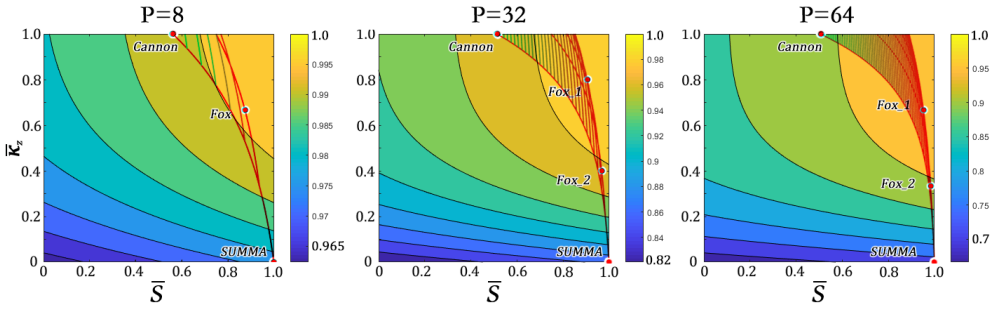

FIGURE 7. Minimal-Buffer Curves of $(1024, 1024, 1024)$ on 8/32/64 with $\mathrm{B}_p \geq l(m+n)$

For larger buffers, we may describe the total buffer using the minimal total buffer size $l(m+n)$ as

$$
\sum_p \mathrm{B}_p = \mu \cdot l(m+n) \tag{4.8}
$$

where $\mu \geq 1$ is the multiplicity of over-capacity.

The one-time-I/O curve with mu in the first equation in (4.1) gives

$$\bar{S} = 1 + \frac{\mu}{P} - \frac{\frac{P^{\bar{\kappa}_z}}{\kappa_x} \cdot m + \kappa_x \cdot n}{P \cdot (m+n)} \tag{4.9}$$

which forms one-time-I/O curve $C(\bar{S}, \bar{\kappa_z}, \kappa_x, \mu)$ with four parameters: $\bar{S}, \bar{\kappa}_z, \kappa_x$ and μ.

On these curves, all PMM algorithms can be located and their performances can be estimated with the heatmap of (3.23). Additionally, optimal PMM algorithms under the constraints and other given environment conditions can also be derived.

4.2. **Optimal BAMMA Partitions.** To optimize the PMM efficiency described by overhead (3.24), we formulate the following objective function

(4.10) Overhead objective function

$$\begin{aligned}
\textbf{given} \quad & P \geq 1, \mu \geq 1, \gamma > 0 \\
\textbf{minimize} \quad & h(m,l,n,p) = \tfrac{1+\gamma}{2l} \cdot \log_2 P \cdot P^{(1-\bar{k_z})} \cdot (1-\bar{k_z}) + P \cdot \gamma \cdot (\tfrac{1}{m} + \tfrac{1}{n})(1-\bar{S}) \\
\textbf{subject to} \quad & g(\bar{\kappa}_z, \bar{S}) = 1 + \tfrac{\mu}{P} - \tfrac{\frac{P^{\bar{\kappa}_z}}{\kappa_x} \cdot m + \kappa_x \cdot n}{P \cdot (m+n)} - \bar{S} = 0 \\
\text{and} \quad & \bar{\kappa}_z, \bar{S} \in [0,1]
\end{aligned}$$

Given $\kappa_x \in [1, P^{\bar{\kappa}_z}]$, we have

$$\bar{S} = 1 + \frac{\mu}{P} - \frac{1}{P \cdot (m+n)} \cdot \left(\frac{P^{\bar{\kappa}_z}}{\kappa_x} \cdot m + \kappa_x \cdot n \right) \tag{4.11}$$

Substituting κ_z by $1 - \log_P \kappa_z$ and also $\bar{S}$ in $h(\bar{\kappa}_z, \bar{S})$

$$h(\kappa_x, \kappa_z) = \frac{1+\gamma}{2l} \cdot \log_2 P \cdot \kappa_z \cdot \log_P \kappa_z + \gamma \cdot \left(\frac{\frac{P}{\kappa_x \kappa_z} \cdot m + \kappa_x \cdot n}{mn} - \mu \right) \tag{4.12}$$

resulting in a modified objective function in terms of κ_x and κ_z subjective to $\kappa_x, \kappa_z \in [1, P]$.

To minimize $h(\kappa_x, \kappa_z)$, we set

$$\frac{\partial h(\kappa_x, \kappa_z)}{\partial \kappa_z} = \frac{\partial h(\kappa_x, \kappa_z)}{\partial \kappa_x} = 0$$

or

$$\frac{\partial h(\kappa_x, \kappa_z)}{\partial \kappa_z} = \frac{1+\gamma}{2l} \cdot \log_2 P \cdot \left(\log_P \kappa_z + \frac{1}{\ln P} \right) - \gamma \cdot \frac{P}{n \cdot \kappa_x \kappa_z^2} = 0 \tag{4.13}$$

$$\frac{\partial h(\kappa_x, \kappa_z)}{\partial \kappa_x} = \gamma \cdot \frac{n - \frac{P}{\kappa_x^2 \kappa_z}}{mn} = 0 \tag{4.14}$$

whose solutions are

$$\begin{cases} \log_P \kappa_z = \dfrac{2\gamma l}{\log_2 P \cdot (1+\gamma)} \cdot \dfrac{P}{n \cdot \kappa_z \kappa_z^2} - \dfrac{1}{\ln P} \\ \kappa_x \kappa_z^2 = \dfrac{Pm}{n} \end{cases} \tag{4.15}$$

Using (4.2), we have $\kappa_x \kappa_z = \frac{P}{\kappa_y}$, so

$$\begin{aligned} & \log_P \kappa_z = \frac{2\gamma l}{\log_2 P \cdot (1+\gamma)} \cdot \frac{\kappa_y}{n \cdot \kappa_z \kappa_z^2} - \frac{1}{\ln P} \\ \Rightarrow \quad & \kappa_y = n \cdot \frac{\log_2 P \cdot (1+\gamma) \cdot (\log_P \kappa_z + \frac{1}{\ln P})}{2\gamma l} \end{aligned} \tag{4.16}$$

Thus, the optimizing division of the supercube of dimensions (m, l, n) for the obvious conditions $P \geq 1, \mu \geq 1$ is

$$\begin{cases} \kappa_y = n \cdot \dfrac{\log_2 P \cdot (1+\gamma) \cdot (\log_P \kappa_z + \frac{1}{\ln P})}{2\gamma l} \\ \kappa_z = \dfrac{Pm}{n\kappa_x^2} = \dfrac{Pm}{n(P/(\kappa_x \kappa_y))^2} = \dfrac{m(\kappa_x \kappa_y)^2}{nP} = \dfrac{Pn}{m\kappa_y^2} = \dfrac{Pn}{m\kappa_x^2} \end{cases} \tag{4.17}$$

where we used (4.2).

We notice an interesting but obvious relationship between κ_x and κ_y

$$\kappa_x : \kappa_y = m : n \tag{4.18}$$

Using (4.17) $\kappa_z = \frac{Pm}{n\kappa_x^2}$, we get

$$\begin{aligned} \kappa_y = \sqrt{\frac{Pn}{m\kappa_z}} = & \quad n \cdot \frac{\log_2 P \cdot (1+\gamma) \cdot \left(\log_P \kappa_z + \frac{1}{\ln P}\right)}{2\gamma l} \\ = & \quad n \cdot \frac{\frac{\ln P}{\ln 2} \cdot (1+\gamma) \cdot \left(\frac{\ln \kappa_z}{\ln P} + \frac{1}{\ln P}\right)}{2\gamma l} \\ = & \quad \frac{n}{l} \cdot \frac{1+\gamma}{2\ln 2 \cdot \gamma} \cdot (\ln \kappa_z + 1) \end{aligned} \tag{4.19}$$

resulting in an implicit solution for κ_z,

$$\kappa_z \cdot (\ln \kappa_z + 1)^2 = \frac{Pl^2}{mn} \cdot \left(\frac{2\ln 2 \cdot \gamma}{1+\gamma}\right)^2 \tag{4.20}$$

Finally, with the solution to κ_z, we found the optimal conditions for κ_x and κ_y

$$\begin{cases} \kappa_x = \sqrt{mP/n\kappa_z} \\ \kappa_y = \sqrt{nP/m\kappa_z} \end{cases} \tag{4.21}$$

In summarize, the optimal partitions for BAMMA are

$$\begin{cases} \kappa_z \cdot (\ln\kappa_z + 1)^2 = \dfrac{Pl^2}{mn} \cdot \left(\dfrac{2\ln 2 \cdot \gamma}{1+\gamma}\right)^2 \\ \kappa_x = \sqrt{mP/n\kappa_z} \\ \kappa_y = \sqrt{nP/m\kappa_z} \end{cases} \tag{4.22}$$

which are the conditions for an optimal PMM algorithm and conditions (4.22) can be further examined by numerical experiments. The practical procedure for constructing optimal PMM algorithms is thus demonstrated.

4.3. **Analysis and Construction of Optimal BAMMA.** Being implicit, conditions (4.22) require us to solve for κ_z as well as κ_x and κ_y numerically.

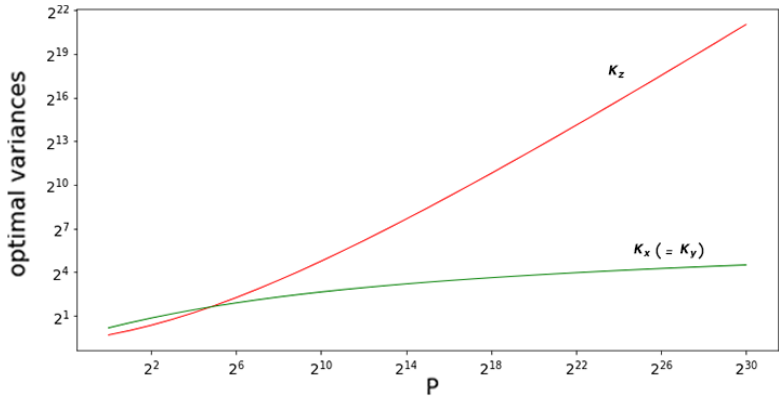

FIGURE 8-a. Solutions of κ_x, κ_y and κ_z as functions of P cores ($m = l = n = 1024$)

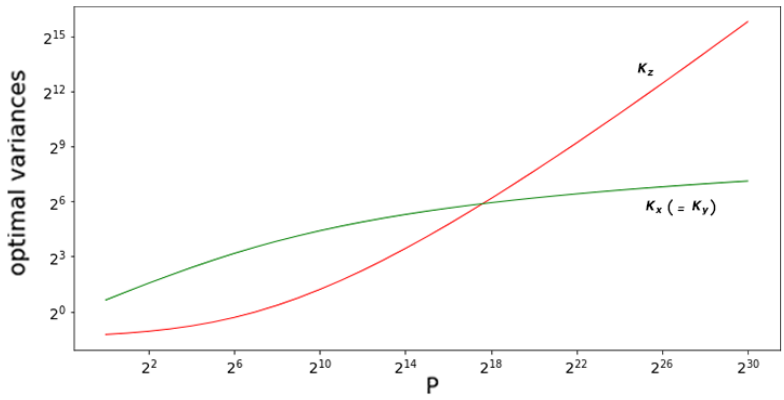

FIGURE 8-b. Solutions of κ_x, κ_y and κ_z as functions of P cores ($m = n = 2048, l = 256$)

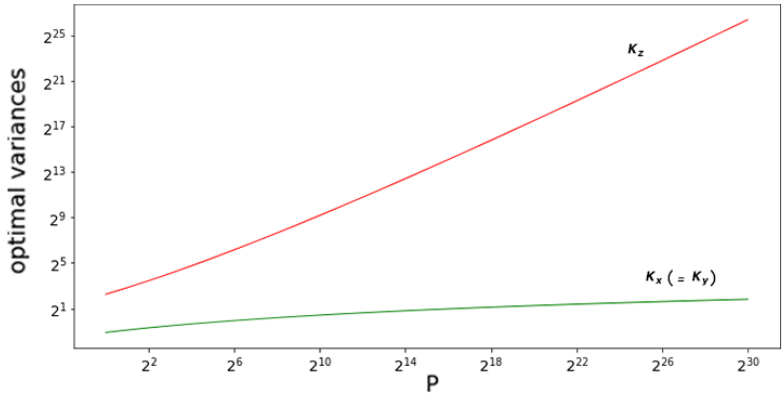

FIGURE 8-c. Solutions of κ_x, κ_y and κ_z as functions of P cores ($m = n = 512, l = 4096$)

For varying P and several different shapes of matrices we obtained those figures, which inspire us to summarize the optimal partitions for BAMMA as the solutions of κ_x, κ_y and κ_z to equation (4.22) and depict them in Fig. 8.

Property 4.3.1 (Conservation of Buffer Size)
Associated with fixed dimensions of matrices and clusters' conditions, the optimal partition will not change with more than maximal buffer sizes on each nodes, if no extra time of I/O is needed.

This property directly implies, by the independency of μ for (4.22), that the optimal BAMMA will not improve having more than the **minimal-buffer curves** required on each node.

Property 4.3.2 (Strictly Monotone Increasing)
The optimal κ_x, κ_y and κ_z increase monotonically with the number of cores.

Proof of monotonicity of κ_z:
Using (4.22), we get

$$\frac{d\kappa_z(P)}{dP}((\ln(\kappa_z(P))+1)^2+2(\ln(\kappa_z(P))+1)) = \frac{l^2}{mn}\cdot\left(\frac{2\ln 2\cdot\gamma}{1+\gamma}\right)^2 \tag{4.23}$$

For convenience, we write

$$\alpha = \alpha(m,l,n,\gamma) = \frac{l^2}{mn}\cdot\left(\frac{2\ln 2\cdot\gamma}{1+\gamma}\right)^2 > 0 \tag{4.24}$$

Equation (4.23) becomes

$$\frac{d\kappa_z(P)}{dP} = \frac{\alpha(m,l,n,\gamma)}{(\ln(\kappa_z(P))+1)(\ln(\kappa_z(P))+3)} \tag{4.25}$$

As long as $\ln\kappa_z + 1 > 0$ for any $P > 0$, or $\kappa_z > e^{-1}$, we can prove $d\kappa_z/dP > 0$.

Proof of monotonicity of κ_z:

Using the second equation of (4.22)

$$(\kappa_x)^2 = \frac{mP}{n\kappa_z} \tag{4.26}$$

and

$$\frac{d\kappa_x}{dP} = \frac{m}{n} \cdot \frac{1}{2\kappa_x\kappa_z}\left(1 - \frac{P}{\kappa_z} \cdot \frac{d\kappa_x}{dP}\right) \tag{4.27}$$

To get $d\kappa_x/dP > 0$ for $P > 0$, we need

$$\frac{P}{\kappa_z} \cdot \frac{d\kappa_x}{dP} < 1 \tag{4.28}$$

or

$$P \cdot \alpha(m, l, n, \gamma) < \kappa_z \cdot (\ln(\kappa_z) + 1)(\ln(\kappa_z) + 3)$$

which is true by the first equation in (4.22)

$$P \cdot \alpha(m, l, n, \gamma) = \kappa_z \cdot (\ln(\kappa_z) + 1)^2 < \kappa_z \cdot (\ln(\kappa_z) + 1)(\ln(\kappa_z) + 3)$$

Those two properties can allow us to introduce two strategies of optimal partitioning for BAMMA

- The optimal partition of BAMMA is adaptive to varying limit of buffer size;
- The proportional partitioning along each axis is optimal.

(professor, please concern here)

Based on the optimal partition (4.22), we construct a PMM algorithm, as a generalization of the classical Fox method

Algorithm 4 Optimal Regular BAMMA ($A \in \mathbb{R}^{m\times l}, B \in \mathbb{R}^{l\times n}$)

Input: $A \in \mathbb{R}^{m\times l}, B \in \mathbb{R}^{l\times n}, P, \gamma, \mathrm{B}_1, ..., \mathrm{B}_P$
Output: $C \in \mathbb{R}^{m\times n}$
Solve the first equation in (4.22) to get κ_x, κ_y,and κ_z.
Find integers X, Y and Z by solving

$$\begin{cases} \textbf{minimize} & |X - \kappa_x|^2 + |Y - \kappa_y|^2 + |Z - \kappa_z|^2 \\ \textbf{subject to} & XYZ = P, X, Y, Z \in \mathbb{Z} \end{cases}$$

Evenly splitting $\{0, 1, \ldots, m-1\}$ into X parts as $\{seg_{X_i}\}_{i=1,\ldots,X}$;
Evenly splitting $\{0, 1, \ldots, n-1\}$ into Y parts as $\{seg_{Y_i}\}_{j=1,\ldots,Y}$;
Evenly splitting $\{0, 1, \ldots, l-1\}$ into Z parts as $\{seg_{Z_i}\}_{k=1,\ldots,Z}$;
let $p = 0$
for $i = 1 : X - 1$ **do**
  for $i = 1 : Y - 1$ **do**
    for $i = 1 : Z - 1$ **do**
      $U_p = seg_{X_i} \times seg_{Y_j} \times seg_{Z_k}$
      $p = p + 1$

if rank= p **do**
  for i, j, k in U_p **do**
    if $(i, j) \in C_{pA,k_A}$ and $(j, k) \in C_{pB,k_B}$ **do**
      $C_{ik} = C_{ik} + A_{ij} \cdot B_{jk}$
      ready to send $(i, j) \in C_{pA,k_A}$ and $(j, k) \in C_{pB,k_B}$
    else if $(j, k) \notin C_{pB,k_B}$ **do**
      require (j, k) from another rank
      pop the front of C_{pB,k_B}
      push (j, k) into the back of C_{pB,k_B}
      $k_B = k_B + 1$
    else if $(i, j) \notin C_{pA,k_A}$ **do**
      require (i, j) from another rank
      pop the front of C_{pA,k_A}
      push (i, j) into the back of C_{pA,k_A}
      $k_A = k_A + 1$
  If rank = root, gather results;
Return C

In Algorithm 4, as usual, even divisions of matrices are assumed for convenience. If even divisions are not possible, we will either patch the matrices with zero to make them even or make approximate even divisions. Results reported in this manuscript were obtained by this method for consistency and comparison with those of other methods using the same program. It is however possible to design a general program to decompose the 3D-supercube with floating κ_x, κ_y,and κ_z to allow BAMMA to perform PMM at $O(N^3)$.

5. Application of 3D-supercube algorithm

We designed a series of experiments to verify parallel efficiency (3.23) and to compare them with results from Algorithm 4. The experiments consist of two parts: one for estimating γ and the other one for measuring time for the particular algorithms. All experiments were conducted on a Beowulf Clusters[1]and our testing program is developed using MPI and C/C++.

5.1. Estimating the Value of γ.

The hardware parameter γ defined in (3.9) can be measured fairly accurately by some basic experiments, for example, one for multiplying two matrices, both of dimensions 1024×1024, and the other for broadcasting an 1024×1024 matrix from each of P cores one by one. Obviously, we vary the sizes of matrices and the computing systems, and conduct multiple experiments, to avoid bias. For taking the average, we have run two jobs both for 10 times, in 8, 32 and 64 cores.

[1]Seawulf Clusters: https://it.stonybrook.edu/help/kb/understanding-seawulf

TABLE 1. Statistics of communication and computation tests

P	Individual Multiplication	Broadcast
8	5.2372±0.0014	0.1120±0.0014
32	5.2732±0.0049	2.0944±0.0457
64	5.2729±0.0059	8.4876±0.1488

With data in Table 1, we calculate γ for each P. In practices, measuring γ accurately from (3.9) directly requires some care. We modify (3.22) as

(5.1)
$$\begin{aligned} E(m,l,n,P) &= \frac{1}{P}\cdot\frac{mln\cdot T_{\text{mult\_per\_step}}}{\frac{mln}{P}\cdot T_{mult\_per\_step}+(t_{\text{SR}}+t_{\text{op}})\cdot\log_2 A_z\cdot mn\cdot\frac{A_z}{P}+t_{\text{SR}}\cdot 2l} \\ &= \frac{1}{1+\frac{t_{\text{SR}}+t_{\text{op}}}{T_{\text{mult\_per\_step}}}\cdot\frac{\log_2\bar{\kappa}_z}{l}\cdot\bar{\kappa}_z+\frac{2\cdot t_{\text{SR}}}{T_{\text{mult\_per\_step}}}\cdot P\cdot(\frac{1}{m}+\frac{1}{n})(1-\bar{S})} \end{aligned}$$

where $T_{\text{mult\_per\_step}}$ is the time per step for MM. Combining the above, we obtain the following approximation

$$\gamma \approx \frac{2\cdot t_{\text{SR}}}{T_{\text{mult\_per\_step}}} = 2\cdot\frac{T_{\text{bcast}}/P(P-1)\cdot 1024^2}{T_{\text{mult}}/(1024^3)} = \frac{2048}{P(P-1)}\cdot\frac{T_{\text{bcast}}}{T_{\text{mult}}} \tag{5.2}$$

The estimated value of each γ is shown in Table 2

TABLE 2. γ Values Calculated from the Experiments

P	8	32	64
γ	0.7822	0.8195	0.8170

These γ values independent of P, are fairly consistent with other observations, and manufacturer's published specifications.

5.2. **PMM Experiments.** For 8, 32 and 64 cores, we designed 7, 10 and 10 different strategies to decompose the 3D-supercube and allocate varying buffer sizes, respectively, to assess the performances of various PMM algorithms.

As usual, the parallel efficiency

$$E = \frac{1}{P}\cdot\frac{T_{one\_core}}{T_{P\_Cores}} \tag{5.3}$$

is used for such measurements and a summary of of the results are tabulated while more detailed original data are recorded in Appendix B.

TABLE 3.a Seven sets of PMM experiments for $P = 8$

$(*)$: The best partitions for optimal BAMMA with $\kappa_x = \kappa_y \approx 2.3096, \kappa_z \approx 1.4997$.

Cube Dimension	$m_p = 128$ $l_p = 1024$ $n_p = 1024$			$m_p = 512$ $(*)$ $l_p = 512$ $n_p = 512$			$m_p = 1024$ $l_p = 128$ $n_p = 1024$
Buffer size on A	128 ×1024	128 ×1024	128 ×1024	512 ×256	512 ×256	512 ×512	1024 × 128
Buffer size on B	128 ×1024	256 ×1024	1024 ×1024	512 ×256	512 ×512	512 ×512	1024 × 128
$\bar{S}$	0.5625	0.625	1	0.875	0.9375	1	1
$\bar{A}_z$	1			0.6667			0
Efficiency	0.9787	0.9899	1.0007	0.9942	0.9948	0.9951	0.9712
Expected (sec)	0.9947	0.9954	1.0000	0.9967	0.9975	0.9983	0.9795
Error (%)	1.5963	0.5575	0.0678	0.2491	0.2662	0.3151	0.8346

TABLE 3.b Ten sets of PMM experiments for $P = 32$

$(*)$: The best partitions for optimal BAMMA with $\kappa_x = \kappa_y \approx 3.3142, \kappa_z \approx 2.9134$.

Cube Dimension	$m_p = 32$ $l_p = 1024$ $n_p = 1024$				$m_p = 256$ $(*)$ $l_p = 512$ $n_p = 256$			$m_p = 512$ $l_p = 128$ $n_p = 512$		$m_p = 1024$ $l_p = 32$ $n_p = 1024$
Buffer size on A	32 ×1024	32 ×1024	32 ×1024	32 ×1024	256 ×128	256 ×256	256 ×512	512 × 64	512 ×128	1024 × 32
Buffer size on B	32 ×1024	256 ×1024	512 ×1024	1024 ×1024	256 ×128	256 ×256	256 ×512	512 × 64	512 ×128	1024 × 32
$\bar{S}$	0.515625	0.625	0.75	1	0.90625	0.9375	1	0.96875	1	1
$\bar{A}_z$	1				0.8			0.4		0
Efficiency	0.9514	0.9809	0.9880	1.0054	0.9872	0.9894	0.9902	0.9729	0.9743	0.8438
Expected (sec)	0.9761	0.9814	0.9875	1.0000	0.9951	0.9982	0.9817	0.9777	0.9792	0.8762
Error (%)	2.4684	0.0546	0.0447	0.5386	0.7892	0.8836	0.8493	0.4849	0.4980	3.2378

TABLE 3.c Ten sets of PMM experiments for $P = 64$

$(*)$: The best partitions for optimal BAMMA with $\kappa_x = \kappa_y \approx 3.9052, \kappa_z \approx 4.1965$.

Cube Dimension	$m_p = 16$ $l_p = 1024$ $n_p = 1024$				$m_p = 256$ $(*)$ $l_p = 256$ $n_p = 256$			$m_p = 512$ $l_p = 64$ $n_p = 512$		$m_p = 1024$ $l_p = 16$ $n_p = 1024$
Buffer size on A	16 ×1024	16 ×1024	16 ×1024	16 ×1024	256 × 64	256 ×128	256 ×256	512 × 32	512 × 64	1024 × 16
Buffer size on B	16 ×1024	64 ×1024	256 ×1024	1024 ×1024	256 × 64	256 ×128	256 ×256	512 × 32	512 × 64	1024 × 16
$\bar{S}$	0.5078125	0.53125	0.625	1	0.953125	0.96875	1	0.984375	1	1
$\bar{A}_z$	1				0.6667			0.3333		0
Efficiency	0.9015	0.9141	0.9539	0.9959	0.9732	0.9746	0.9748	0.9329	0.9342	0.7002
Expected (sec)	0.9525	0.9547	0.9634	1.0000	0.9883	0.9899	0.9930	0.9450	0.9464	0.7465
Error (%)	5.1028	4.0593	0.9562	0.4088	1.5095	1.5263	1.8142	1.2093	1.2239	4.6290

For these experiments, we plot the efficiency in dots as a function of $\bar{S}$ and $\bar{\kappa}_z$, together with the expected theoretical results in surfaces sketched from (3.23).

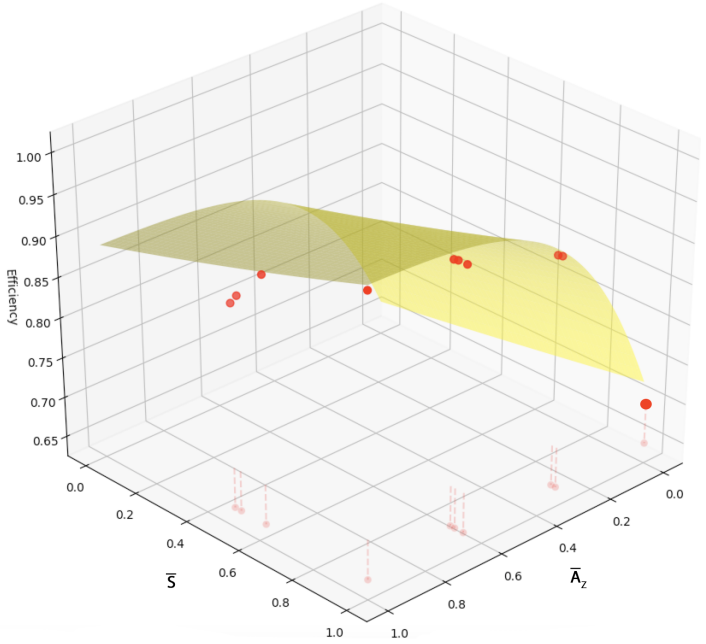

FIGURE 9. The Parallel Efficiency Distribution for Experiments at $P = 64$

We noticed, besides the consistency between theoretical and experiment results, the communication overhead of the algorithm impacted the efficiency while the communication itself is affected by numbers of factors.

6. Conclusions and future work

We have discussed the construction of PMM algorithms, based on decomposition of the 3D-supercube for the naïve MM with complexity $O(N^3)$. As usual, the construction and performance of such algorithms are influenced by the number of nodes, buffer sizes, and the dimensions of matrices as well as the strategies of decomposing the matrices. Our strategies allow us to design all published PMM algorithms in a unified framework.

Given the restrictions in the matrix sizes and assumption of homogeneous computer systems with fairly equal core performance and equal core-to-core communication latency and bandwidth, our BAMMA algorithms and their analysis can be further extended to consider general conditions without such limitations. Many advances including the recursive algorithm CARMA [4] by Demmel may be incorporated in our algorithms.

Finally, inspired by [9], our PMM algorithms can be extended to 4D-supercube with OP-map introduced in Section 2.1 to address families of MM algorithms with complexities lower than $O(N^3)$ including Strassen [5] and Coppersmith and Winograd [6].

References

1. Fox, G.C., et al., *Solving problems on concurrent processors*. Vol. 1: General techniques and regular problems. 1988: Prentice-Hall, Inc.

2. Cannon, L.E., *A cellular computation to implement the kalman filter algorithm*. 1969, Montana State University Bozeman Engineering Research Labs.
3. Van De Geijn, R.A. and J. Watts, *SUMMA: Scalable universal matrix multiplication algorithm*. Concurrency-Practice and Experience, 1997. 9(4): p. 255-274.
4. Demmel, J., et al. *Communication-optimal parallel recursive rectangular matrix multiplication*. in Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on. 2013. IEEE.
5. Strassen, V., *Gaussian elimination is not optimal*. Numerische mathematik, 1969. 13(4): p. 354-356.
6. Coppersmith, D. and S. Winograd. *Matrix multiplication via arithmetic progressions*. in Proceedings of the nineteenth annual ACM symposium on Theory of computing. 1987. ACM.
7. M. Griebel and H. Harbrecht., *On the construction of sparse tensor product spaces*, Math. Comp. 82 (2013), 975-994.
8. F. Johansson. *A fast algorithm for reversion of power series*. Math. Comp., 84:475–484, 2015.
9. C.-C. Chou, Deng, Y., Li G., and Wang, Y. *Parallelizing Strassen's Method for Matrix Multiplication on Distributed Memory MIMD architectures*.
10. Sadayappan, Ponnuswamy, Fikret Ercal, and J. Ramanujam. *Cluster partitioning approaches to mapping parallel programs onto a hypercube.* Parallel computing 13.1 (1990): 1-16.

Appendix A: Measurement of the values of γ

Table A.1 Results of Experiments of Individual Computation

P	Test1	Test2	Test3	Test4	Test5	Test6	Test7	Test8	Test9	Test10
8	5.2368	5.2352	5.2354	5.2396	5.2383	5.2362	5.2357	5.2368	5.2355	5.2372
32	5.2768	5.2717	5.2759	5.2862	5.2733	5.2842	5.2752	5.2728	5.2766	5.2732
64	5.2709	5.2760	5.2852	5.2748	5.2739	5.2712	5.2891	5.2773	5.2777	5.2729

Table A.2 Results of Experiments of Broadcasting

P	Test1	Test2	Test3	Test4	Test5	Test6	Test7	Test8	Test9	Test10
8	5.2368	5.2352	5.2354	5.2396	5.2383	5.2362	5.2357	5.2368	5.2355	5.2372
32	5.2768	5.2717	5.2759	5.2862	5.2733	5.2842	5.2752	5.2728	5.2766	5.2732
64	5.2709	5.2760	5.2852	5.2748	5.2739	5.2712	5.2891	5.2773	5.2777	5.2729

Appendix B: Experiments of 3D-hypercube algorithms

Table B.1 Experimental data for $P = 8, m = l = n = 1024$ (in sec)

Cube Shape	Buffer on A^T	Buffer on B	Test1	Test2	Test3	Test4	Test5	Test6	Test7	Test8	Test9	Test10
m 1024 l 1024 n 128	1024 × 128	1024 × 128	0.6738	0.6738	0.6741	0.6739	0.6743	0.6738	0.6739	0.6742	0.6741	0.6742
m 128 l 1024 n 1024	128 ×1024	128 ×1024	0.6705	0.6676	0.668	0.6692	0.6693	0.6692	0.6680	0.6677	0.6680	0.6707
	128 ×1024	256 ×1024	0.6616	0.6615	0.6604	0.6613	0.6603	0.6615	0.6618	0.662	0.6608	0.6617
	128 ×1024	1024 ×1024	0.6543	0.6543	0.6541	0.6551	0.6539	0.6539	0.6537	0.6537	0.6538	0.6536
m 512 l 512 n 512	512 ×256	512 ×256	0.6585	0.6587	0.6588	0.6582	0.6581	0.6583	0.6580	0.6586	0.6581	0.6584
	512 ×256	512 ×512	0.6581	0.6579	0.6583	0.6578	0.6580	0.6579	0.6574	0.6579	0.6583	0.6582
	512 ×512	512 ×512	0.6574	0.6575	0.6577	0.6575	0.6582	0.658	0.6583	0.6576	0.6576	0.6582

Table B.2 Experimental data for $P = 32, m = l = n = 1024$ (in sec)

Cube Shape	Buffer on A^T	Buffer on B	Test1	Test2	Test3	Test4	Test5	Test6	Test7	Test8	Test9	Test10
m 1024 l 1024 n 32	1024 × 32	1024 × 32	0.1952	0.1954	0.1963	0.1954	0.1954	0.1953	0.1952	0.1953	0.1953	0.1954
m 32 l 1024 n 1024	32 ×1024	32 ×1024	0.1727	0.1727	0.1728	0.1721	0.1739	0.1728	0.1736	0.1728	0.1760	0.1737
	32 ×1024	256 ×1024	0.1682	0.1686	0.1676	0.1677	0.1683	0.1686	0.1686	0.1671	0.1681	0.1683
	32 ×1024	512 ×1024	0.1666	0.1665	0.1669	0.1685	0.1663	0.1670	0.1667	0.1667	0.1668	0.1670
	32 ×1024	1024 ×1024	0.1637	0.1639	0.1641	0.1638	0.1638	0.1642	0.1645	0.1642	0.1639	0.1640
m 256 l 256 n 512	512 × 256	512 × 256	0.1668	0.1669	0.1672	0.1670	0.1676	0.1667	0.1668	0.1669	0.1669	0.1675
	512 × 256	512 × 512	0.1666	0.1669	0.1666	0.1667	0.1666	0.1669	0.1665	0.1667	0.1666	0.1665
	512 × 512	512 × 512	0.1664	0.1665	0.1667	0.1666	0.1665	0.1664	0.1665	0.1666	0.1665	0.1665
m 512 l 512 n 128	512 × 64	512 × 64	0.1692	0.1694	0.1693	0.1713	0.1693	0.1694	0.1691	0.1692	0.1693	0.1694
	512 × 128	512 × 128	0.1694	0.1691	0.1693	0.1692	0.1693	0.1692	0.1693	0.1692	0.1692	0.1693

Table B.3 Experimental data for $P = 64, m = l = n = 1024$ (in sec)

Cube Shape	Buffer on A^T	Buffer on B	Test1	Test2	Test3	Test4	Test5	Test6	Test7	Test8	Test9	Test10
m 1024 l 1024 n 16	1024 × 16	1024 × 16	0.1186	0.1182	0.1184	0.1183	0.1182	0.1183	0.1181	0.1188	0.1182	0.1195
m 16 l 1024 n 1024	16 ×1024	16 ×1024	0.0973	0.0897	0.0928	0.0890	0.0888	0.0954	0.0931	0.0890	0.0901	0.0894
	16 ×1024	64 ×1024	0.0900	0.0906	0.0899	0.0898	0.0905	0.0901	0.0899	0.0905	0.0909	0.0898
	16 ×1024	256 ×1024	0.0858	0.0857	0.0854	0.0948	0.0856	0.0858	0.0854	0.0854	0.0854	0.0851
	16 ×1024	1024 ×1024	0.0822	0.0823	0.0822	0.0822	0.0823	0.0824	0.0822	0.0874	0.0822	0.0825
m 256 l 256 n 256	256 × 64	256 × 64	0.0847	0.0848	0.0847	0.0847	0.0846	0.0848	0.0847	0.0846	0.0848	0.0848
	256 × 128	256 × 128	0.0845	0.0846	0.0846	0.0846	0.0846	0.0847	0.0845	0.0845	0.0849	0.0845
	256 × 256	256 × 256	0.0845	0.0847	0.0844	0.0844	0.0850	0.0844	0.0844	0.0847	0.0847	0.0846
m 512 l 512 n 64	512 × 64	512 × 64	0.0883	0.0882	0.0884	0.0884	0.0883	0.0883	0.0884	0.0883	0.0883	0.0889
	512 × 128	512 × 128	0.0883	0.0885	0.0883	0.0883	0.0883	0.0884	0.0883	0.0880	0.0883	0.0879

Department of Applied Mathematics, Stony Brook Univ., Stony Brook, New York 11790
E-mail address: wenjing.cun.sbu@cvoidcreate.com

Department of Applied Mathematics, Stony Brook Univ., Stony Brook, New York 11790
E-mail address: lihua.peidata@gmail.com

Department of Applied Mathematics, Stony Brook Univ., Stony Brook, New York 11790

Division of Mathematical Sciences School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore, 637371
E-mail address: yuefan.deng@stonybrook.edu