GENERAL CONSTRUCTION AND ANALYSIS OF PARALLEL MATRIX MULTIPLICATION ALGORITHMS

WENJING CUN, LIHUA PEI, AND YUEFAN DENG

Abstract. Algorithms of parallel matrix multiplication for distributed-memory systems are analyzed and a map of the universal parallel matrix multiplication algorithms, based on 3-dimensional hypercube with computational complexity of $O(N^3)$ are presembled. Numerical experiments of several algorithms on clusters were carried out to compare the performace.

1. Introduction

High-performance computing (HPC) algorithms for linear algebra are critical for many application in science, engineering and commerce. Matrix multiplication (MM) is the heart of many linear algebra problems. Much efforts have been devoted to discuss the general method for MM and the parallelization of it. Several classical methods have attracted wide acceptance. It is desirable to design a map for organizing the family of parallel matrix multiplication (PMM) algorithms. With the metric of parallel efficiency of the lattice, it is now possible to evaluate the upper bound of the algorithms depending on any given dimensions of the matrices.

The rest of the manuscript is organized as follows: Section 2 presents the general idea of MM on 3D hypercube, based the maps between sets of coordinates. Section 3 constructs the communication forest among the computation nodes, while proceeding the parallel algorithms, and organizing the family of the parallel algorithms based on 3D hypercube. Section 4 analyzes several classical PMM algorithms and propose a general algorithm for universal MM. Finally, Section 5 discusses the performance and the reduction of the computational complexity, and summarizes the construction of algorithms.

2. Construction of matrix multiplication in 3D hypercube

Firstly, we shall review the naïve algorithm of MM.

Naïve algorithm is the direct definition of MM s for any given pair of matrices: $A \in R^{(m\times l)}, B \in R^{(l\times n)}$, where m, l and n are positive integers. Focusing on one final product $C = A \cdot B$, namely C_{ij}, we have:

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

Date: March 29, 2018.
Key words and phrases. Parallel Computation, Matrix Theory.

Algorithm 1 (Naive 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 $i = 1 : l$ **do**
    for $i = 1 : n$ **do**
      $C_{ik} = C_{ik} + A_{ij} \cdot B_{jk}$
Return C

which dependes on the matrix elements at three indices i,j and k, namely (i, j, k). Thus, grouping all (i, j, k) in A and B may produce a well-structured map from the set (i, j, k) to the set (i, k) for C and this map is expected to be independent of the actual values of the matrix element.

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**:

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

For convenience, we denote such an operation map f as: $f_{C,\tau} : A \to B$, and name L_1, L_2 as the ***coordinate maps***. Here A and B are the topologies from which the sets of values in C abstracted into, and sourcing from the most intuitive concept in geometry, we call A and B the corresponding ***coordinate maps***. Among the many properties, we list the first one for the operation map:

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

This is the stability property of an operation map, i.e., the operation map is stable regardless of the values when the inputs in conservative of the algebraic operation.

For a series of operation maps, **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 at least n operation maps:*

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

Following the chain rule in the opposite direction, if an abstract operation map $f_(G, \tau) : C_1 \to C_2$ can be decomposed into a series of operation maps, then $f_(G, \tau)$ is ***separable***.

The third property shows the flexibility, or the scalability in computational engineering, if the operation map can be conservative in topology:

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 operation 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 operation map.*

We will next abstract the process of MM , based Algorithm 2.1, into a separable operation map composed by several independent stages. We also introduce the parallelization of the algorithm.

2.2. **Algorithms Based on 3-Dimensional Hypercube.** To abstract Algorithm 2.1 into the operation maps into operations maps, we first split all the multiplications between a pair of scalars. Then, for the 3D-hypercube algorithm, we construct an operation map from a coordinate set C_{11} to the other one C_{12}. Since the original coordinate set represents the positions of the elements of A and B, whose coordinate are

$$C_A = \mathbb{Z}/m\mathbb{Z} \times \mathbb{Z}/l\mathbb{Z}, C_B = \mathbb{Z}/l\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z} \tag{2.2}$$

The operation map is

$$(f_1)_{\mathbb{R},\times} : C_A \times C_B \to C_{mult3D} \tag{2.3}$$

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

$$(f_1)_{\mathbb{R},\times}((r_a, c_a), (r_b, c_b)) = \begin{cases} (r_a, c_b, c_a), & \text{if } c_a = r_b \\ \varnothing, & \text{otherwise} \end{cases} \tag{2.4}$$

This operation map represents the process of $A_{ij} \cdot B_{jk}$, and from C_{mult}, it can be illustrated in Fig.1:

The next operation is that of addition. Since only the scalar operations remain, the result is a matrix $C \in \mathbb{R}^{m\times n}$, thus we define the third coordinate set as

$$C_C = \mathbb{Z}/m\mathbb{Z} \times \mathbb{Z}/n\mathbb{Z} \tag{2.5}$$

and the corresponding abstract operation map is

$$\begin{aligned} &(f_2)_{R,+} : C_{mult3D} \to C_C \\ &(f_2)_{R,+}((r_a, c_b, c_a)) = (r_a, c_b) \end{aligned} \tag{2.6}$$

Here "+" is the scalar addition.

Equation (2.7), reveals that all the 3D "cubes" in the Figure 1 with the same "z-value" are reduced in to one in the "xy-plane", illustrated in Figure 2.

Now the whole process has been abstracted into a chain of abstract operation maps, for the MM operation: $F^{m\times l} \times F^{l\times n} \to F^{m\times n}$, we can define the chain of abstract operation maps as the following:

$$f_{\mathbb{R}^{m\times l}\times\mathbb{R}^{l\times n},\text{matrix multiply}} = (f_1)_{\mathbb{R},\times} \circ (f_2)_{\mathbb{R},+} \tag{2.7}$$

where we define $f_{\mathbb{R}^{m\times l}\times\mathbb{R}^{l\times n},\text{matrix multiply}}(x) = x$, is a trivial abstract operation map directly representing that: $C = AB$.

For a local-memory system, there seems nothing to modify (2.7) for such a fixed chain, but to a distributed-memory cluster, we can add one more abstract operation map to each computing node p, $g_{R,=}$, where "=" simply means the identity operation, but can filter the coordinates for each node, and we define it as:

$$\begin{aligned} &g_{\mathbb{R},=} : C_{mult3D} \to C_{mult3D} \\ &g_{\mathbb{R},=}((r_a, c_b, c_a)) = \begin{cases} (r_a, c_b, c_a), & \text{if } (r_a, c_b, c_a) \in U_p \\ \varnothing, & \text{otherwise} \end{cases} \end{aligned} \tag{2.8}$$

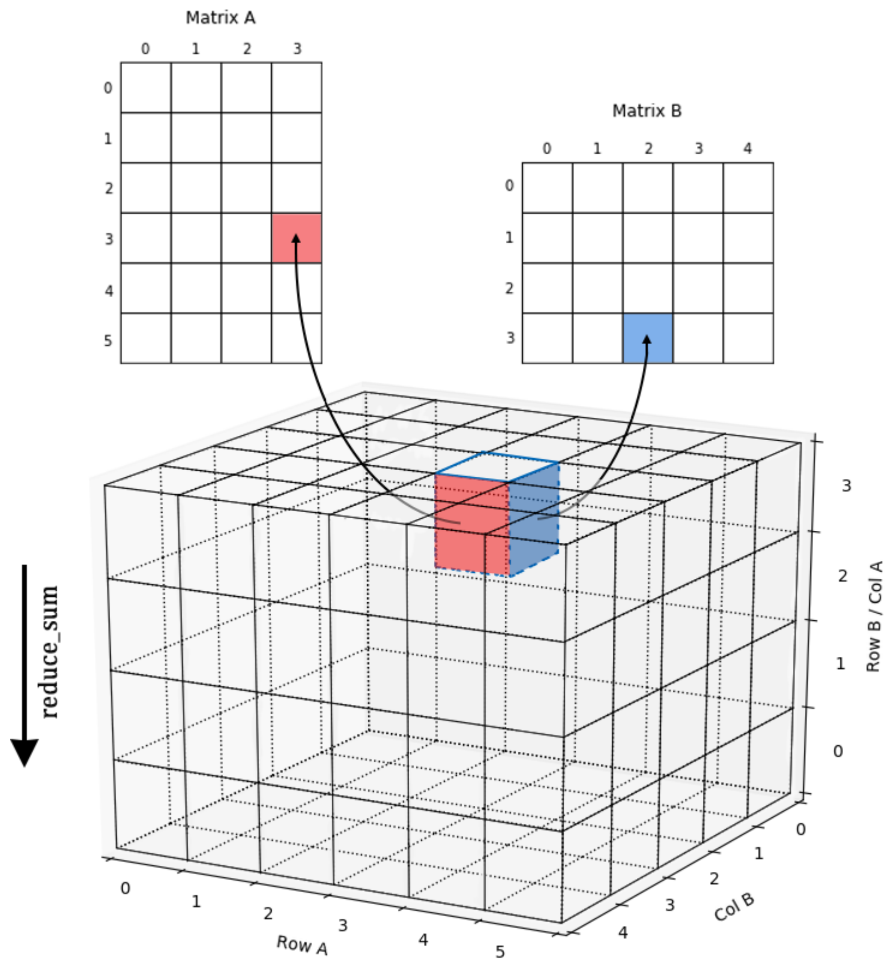

FIGURE 1. The operation map $(f_1)_(\mathbb{R}, \times) : C_A \times C_B \to C_{\text{mult}}$

Here U_p is a custom set chosen for each node p, called a ***filtering set***, represents the "blocks" chosen for node p to calculate. Thus, the chain of abstract operation maps for each node p is written as:

$$(f_p)_{\mathbb{R}^{m\times l}\times\mathbb{R}^{l\times n},\text{matrix multiply}} = (f_1)_{mathbbR,\times} \circ g_{mathbbR,=} \circ (f_2)_{mathbbR,+} \tag{2.9}$$

This formula represents all the value-independent parallel algorithms based on Naïve Algorithm.

Now we take such a case for example: if there are totally 4 nodes for computing $A \cdot B$, where $A \in \mathbb{R}^{6\times 4}, B \in \mathbb{R}^{4\times 6}$, then choose the four filtering sets as:

$$\begin{aligned} U_1 &= \{4,5\} \times \{2,3,4\} \times (\mathbb{Z}/4\mathbb{Z}) \\ U_2 &= \{3\} \times \{2,3,4\} \times (\mathbb{Z}/4\mathbb{Z}) \\ U_3 &= \{3,4,5\} \times \{0,1\} \times (\mathbb{Z}/4\mathbb{Z}) \\ U_4 &= \{0,1,2\} \times \{0,1,2,3,4\} \times (\mathbb{Z}/4\mathbb{Z}) \end{aligned} \tag{2.10}$$

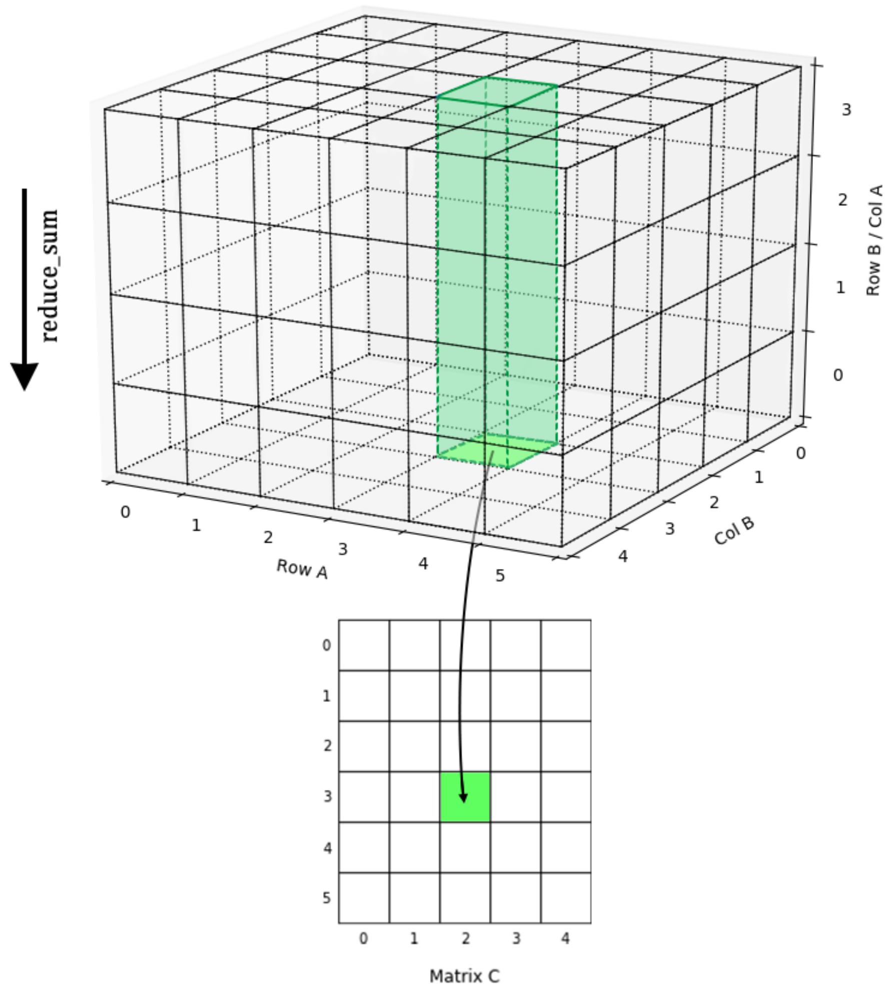

FIGURE 2. Operation map $(f_2)_{F^{m\times l\times n},\mathrm{sum}}$

So the "blocks" in $C_m ult$ allotted to each node are shown as the following figure:

In fact, the strategy to choose filtering sets as single "cubes" that all span over z-axis, is similar to one of the popular general algorithm for parallel universal MM, called BMR[1], if not consider the communication among the nodes. Specially, if the blocks are not finished computing at one communicating step, it can deduce many interesting algorithms, where Cannon's algorithm[2] and SUMMA[3] are two of the famous ones.

So only consider the computational part of the whole process of MM, all the algorithms based on Naive Algorithm can be deduced by selecting the filtering sets $\{U_p\}$, so that the algorithm is presented as the following pseudocode:

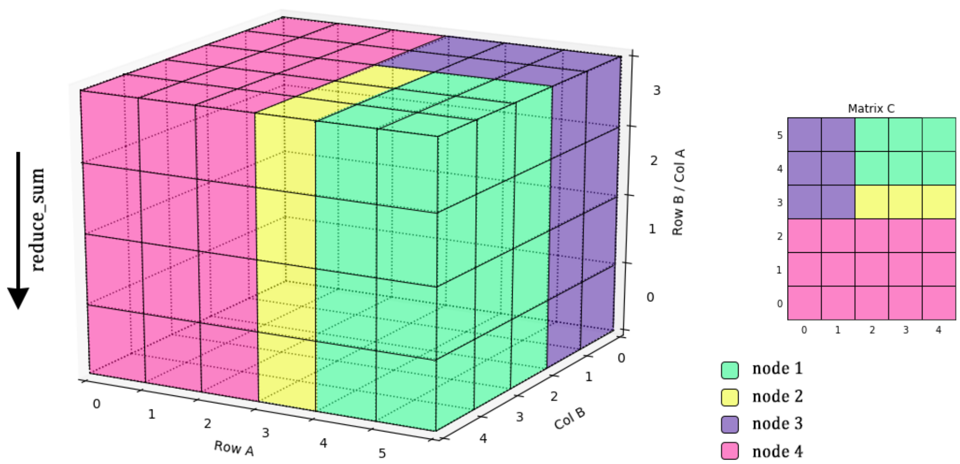

FIGURE 3. Parallel 3D-Hypercube

Algorithm 2 (3D-Hyperblock Matrix Multiplication)

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

The members in the 3D-Hyperblock family often have the best potential in minimizing the time in communication, however, while dealing with large-scale matrices, the buffer size often becomes a problem for the cluster to limit the total size of data to be deployed onto each computation node. Therefore, each such optimization in communication involves deploying the best subset $V_p \subseteq U_p$, and find the best communication tree to pass the data among the nodes, in order to finish the tasks undertaken by U_p efficiently, which is sophisticated while dealing with such P cubes simultaneously.

In the next section, we will discuss the influence of limiting buffer size on designing the communication part for a parallel algorithm, and find out a general map for designing the algorithm for PMM.

3. Communication in 3D-hypercube

Allocation of blocks of operations to each computational node is straightforward for 3D hypercube algorithms, however, while dealing with large-scale MMs, the restricted buffer size will limit the size of data stored onside each node.

Generally, when a parallel program allows all the nodes to own all the necessary data already, in terms of the algorithm itself, the cost of communication can be minimized. Otherwise, one node lacking necessary data needs to fetch those needed from the other ones. So for PMM, referring to algorithm 2.2, often we can only deploy part of U_p onto node p, then following the communication tree, at each step, one need to finish as much as computation tasks with its data owing, release or send the data no longer needed, and require and receive the data needed for next step of computation.

3.1. **Communication Rules and Cost.** For convenience, we describe a "cube" assigned to node p with:

- m_p, l_p, n_p, the lengths of the cube assigned to node p, along with x-axis, y-axis and z-axis respectively;

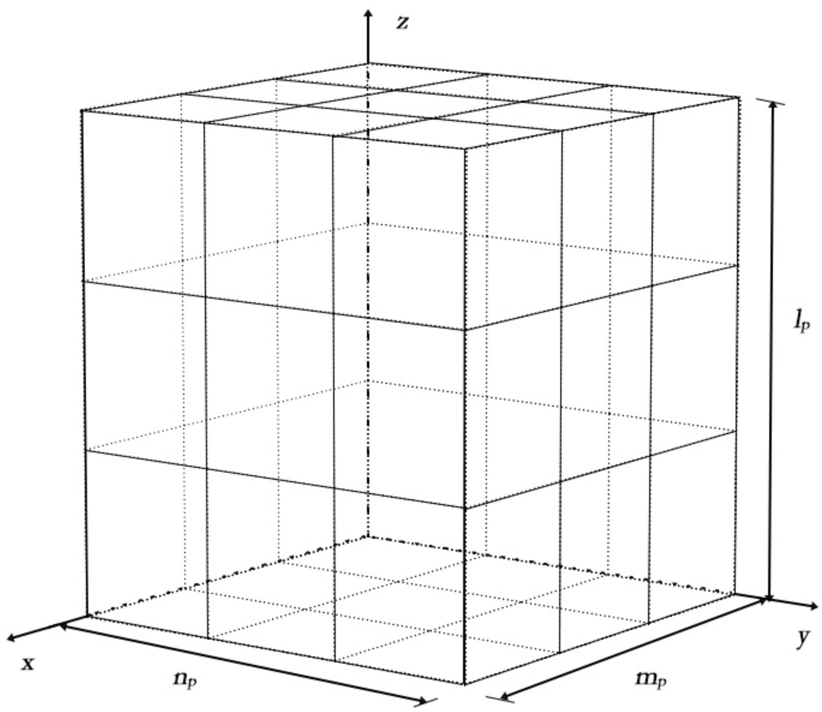

FIGURE 4. Cube Onside Node p

For many of the algorithms in 3D-hypercube family, we may assume that such P communication trees are similar, or out of the geometric vision, such P "cubes" are similar in the following properties:

- Volume, i.e. the number of multiplication between entries;
- Total projection area onto xz-plane and yz-plane, i.e. the total size of data needed for the computation tasks;
- Maximum area projective onto xz-plane and yz-plane can be covered at each communication step, i.e. the maximum buffer size assigned to each core, denoted as Buffer_p.

The three properties above in fact determined the performance of the part of scalar (or even block) multiplication. As for the part of scalar (or even block) addition, there is only one additional property needed to be considered:

- Total projection area onto xy-plane, i.e. the size of data to be reduced onto one node.

Then to design a smart algorithm, we may design following the rules below:
Rule 3.1 The total volume of the P cubes is exactly mln, i.e. the total volume of the whole 3D-hypercube. i.e. there would be no repetitive computational tasks among the nodes:

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

Rule 3.2 The entries of two matrices can be covered by the data stored in the P nodes at each communication step, so that it requires:

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

Rule 3.3 Any entry (or block) stored onside one node can only be released if and only if all the scalar (block) multiplications involving it have been finished. So that the total cost of receiving data on node p is:

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

From Rule 3.3, since the total number of sending is always conservative with that of receiving, so that we may estimate the average cost of communication on send-receive for each node would follow:

$$\begin{cases} T_{comm,send-recv} \leq \sum_{p=1}^{P} T_{p,recv} & \text{send/require one by one} \\ T_{comm,send-recv} \geq \max\limits_{p}\{T_{p,recv}\} & \text{communication parallelized} \end{cases} \tag{3.4}$$

where the lower bound comes from the strategy that minimizing the waiting time, where we let the one with the largest demand to send nothing out.

Next, based on the analysis of cost on send-recv, we are going to discuss the strategy of designing the cubes and communication tree for each node.

3.2. **Algorithm for Parallel Matrix Multiplication.** In practical cases, since where all the computation nodes are in the similar conditions (in CPU, GPU, RAM, etc.), we often divide the whole 3D hypercube into P similar (not necessary to be identical) cubes, and assigned with similar limits of buffer sizes.

For each cube assigned to each node, due to the restriction to buffer size, our communication objective is to minimizing the time cost for communication among the nodes.

So numerate $U_p = \{(x_{pi}, y_{pi}, z_{pi})\}$ by taking the operation map (2.9) mentioned in subsection 2.1, which is source from $A_{pk} \times B_{pk}$ at each communication step k, where it should satisfy that:

$$|C_{pAk}| + |C_{pBk}| \leq \text{Buffer}_p \tag{3.5}$$

So that we can see, to parallelize the MM by adding the communication part in 3D-hypercube, the form of the operation maps can be totally conserved, and so construct a union of a series of operation maps in the same forms:

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

Now to minimize the communication cost on each node with fixed limit of buffer size, we want to prove that:

Theorem 3.1. *To numerate $U_p = \{(x_{pi}, y_{pi}, z_{pi})\}$, for $\{C_{pAk}\}$ and $\{C_pBk\}$ satisfying (3.5), there can exist $\{C_{pAk}\}$ and $\{C_{pBk}\}$ so that:*

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

if given that $Buffer_p \geq \min\{m_p, n_p\}$.

This theorem states the availability of Rule 3.3, so that no data (entry or block) would be sent to any one node redundantly. And the given condition can propose a general strategy for send-receive and so prove the existence.

This strategy with the condition $\text{Buffer}_p \geq \min\{m_p, n_p\}$ can be stated as the following:

Algorithm 3 (3D-Hyperblock Algorithm with Communication)

Input: $A \in \mathbb{R}^{m \times l}, B \in \mathbb{R}^{l \times n}$ $, m_p, l_p, n_p, \text{Buffer}_p \in N^+, (m_p \leq n_p)$
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 $\text{StepSize}_A = \lfloor \text{Buffer}_p / m_p \rfloor$,and initialize the buffer as:

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

    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},
            $k_B = k_B + 1$.
    Gather the results onto the root node.
Return C

This algorithm designs a communication tree for each node, with which we can construct an algorithm with only the following two given conditions:

- m_p, l_p, n_p,i.e. a cube assigned to each node p;
- Buffer_p.

And the algorithm designed based on algorithm 3.1 is called the **Buffer Adaptive Matrix Multiplication Algorithm (BAMMA)**, which can fit to pair of matrices with random dimensions and make the best usage of buffer onside each node.

Next we will evaluate the performance of BAMMA, based on the variables raised in section 3.1, and show the general prediction of parallel efficiency of the algorithms for PMM in 3D-Hypercube family.

3.3. **Evaluation of Performance.** To evaluate the performance, or say the total cost of time of a PMM algorithm, we need to evaluate both the computational cost and the communication cost, i.e.:

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

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

If we assigned similar cubes to p nodes with similar limits of buffer sizes, then we may assume that there is a way to minimize $T_{p,idle}$ by building static routines for the whole communication forest. So that we may remove the maximum in (3.7) to estimate the total cost:

$$T_{total,P} \approx T_{p,comp} + T_{p,idle} \tag{3.8}$$

And considering the practical cases in computational clusters, we assume that the average time cost on one pair of send-receive process on one floating number (notated as $\hat{t}_{send-recv}$), and that of one arithmetical operation, (notated as $\hat{t}_{op}$), will satisfy that:

$$\gamma := \frac{\hat{t}_{send-recv}}{\hat{t}_{op}} \tag{3.9}$$

and for practical cases, we assume τ is a constant value associated with particular machine, cluster topology, to be used to estimate the real fraction while running the program.

Now firstly, for the computational cost $T_{p,comp}$, by the operation map defined as (2.10), it sources from two parts: scalar (or block) multiplication and addition, so that we have:

$$T_{p,comp} = T_{p,mult} + T_{p,add} = (2 \cdot m_p l_p n_p - m_p n_p) \cdot \hat{t}_{op} \approx 2 m_p l_p n_p \cdot \hat{t}_{op} \tag{3.10}$$

And for those similar p cubes:

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

where P is the total number of nodes. So in total:

$$T_{comp} = T_{p,comp} \approx \frac{2mln}{P} \cdot \hat{t}_{op} \tag{3.12}$$

Now consider the next part of communication, it consists of two parts: send-receive in order to complete all the computations onside each cube, and the reduction with summation:

$$T_{comm} = T_{send-recv} + T_{reduce} \tag{3.13}$$

For reduction, since the total cost is just the total number of entries (or blocks) need to be sent to the root node, along the z-axis, or say it is just the total projection area onto the xy-plane from the P nodes. And since for each step, it involves one addition and one send/receive action, thus:

$$T_{send-recv} \geq C \cdot \max_p \{m_p l_p + n_p l_p - \text{Buffer}_p\} \cdot \hat{t}_{send-recv} \tag{3.14}$$

where generally C is a constant, usually equals or larger than 2 for machines assigned with similar conditions, since one node can only do send or receive action at one single step.

So in summary, we add all of the costs of parts respectively together, and it deduces the prediction formula of the total cost of time:

(3.15)

$$T_{total,P} \geq \frac{2mln}{P} \cdot \hat{t}_{op} + (\hat{t}_{send-recv} + \hat{t}_{op}) \cdot \sum_{p=1}^{P} m_p n_p + \hat{t}_{send-recv} \cdot 2 \cdot \max_p \{m_p l_p + n_p l_p - \text{Buffer}_p\}$$

For the convenience to construct the space to include all the algorithms in 3D-Hypercube family, we define two new variables: **Z-variance** and **Maximum Saturabiliity**, denoted by A_z and $\bar{S}$ respectively, as the following:

$$A_z := \frac{\sum_{p=1}^{P} m_p n_p}{mn}, \bar{S} := 1 - \max p\{\frac{m_p l_p + n_p l_p - \text{Buffer}_p}{l(m+n)}\} \tag{3.16}$$

here $A_z \geq 1$ and $S \in (0, 1]$.

And by parallelizing the reducing process, the whole process to "compress" the cubes can be optimized into a flipped binary tree, so that it would take much shorter time than that in (3.13):

$$A_z := \frac{\sum_{p=1}^{P} m_p n_p}{mn}, T_{reduce} = (\hat{t}_{send-recv} + \hat{t}_{op}) \cdot \log_2 A_z \cdot mn \cdot \frac{A_z}{P} \tag{3.17}$$

with considering that the columns involving different cores can reduce parallelly.

And based on the assumptions of similar cubes we made above, we may deduce that:

$$\bar{S} := 1 - \frac{\bar{m}_p \bar{l}_p + \bar{n}_p \bar{l}_p - \bar{\text{Buffer}}_p}{l(m+n)} \tag{3.18}$$

where $\bar{m}_p, \bar{l}_p, \bar{n}_p$ and $\bar{\text{Buffer}}_p$ are set to be the similar for every node.

Then consider the total cost of the MM on a single node is always:

$$T_{single} = (2mln - mn) \cdot \hat{t}_{op} \tag{3.19}$$

So onside the $A_z - \bar{S}$ plane, we can now sketch the parallel efficiency:

(3.20)

$$\begin{aligned} \tau(m, l, n, P) \leq & \quad \frac{1}{P} \cdot \frac{(2mln - mn) \cdot \hat{t}_{op}}{\frac{2mln - mn}{P} \cdot \hat{t}_{op} + (\hat{t}_{send-recv} + \hat{t}_{op}) \cdot \log_2 A_z \cdot mn \cdot \frac{A_z}{P} + \hat{t}_{send-recv} \cdot 2 \cdot l(m+n)(1 - \hat{S})} \\ \approx & \quad \frac{mln}{mln + (\frac{1+\gamma}{2} \cdot \log_2 A_z \cdot mn \cdot \frac{A_z}{+} P \cdot \gamma \cdot l(m+n)(1 - \bar{S}))} \end{aligned}$$

because in most of the cases, $l \gg 1$ and simplify (3.18) we can finally get:

$$\tau(m,l,n,P) \leq \frac{1}{1 + \frac{1+\gamma}{2l} \cdot A_z \cdot \log_2 A_z + P \cdot \gamma \cdot (\frac{1}{m} + \frac{1}{n})(1 - \hat{S})} \tag{3.21}$$

Here to normalize the space, we set $\hat{A}_z = 1 - \frac{\log_2 A_z}{\log_2 P} = 1 - \log_P A_z \in (0, 1]$, since each node can at most own projection area mn onto xy-plane, and for any one column along the z-axis, there are at most P segments belong to different nodes. So that:

$$\tau(m,l,n,P) \leq \frac{1}{1 + \frac{1+\gamma}{2l} \cdot \log_2 P \cdot P^{1-\hat{A}_z} \cdot (1 - \hat{A}_z) + P \cdot \gamma \cdot (\frac{1}{m} + \frac{1}{n})(1 - \hat{S})} \tag{3.22}$$

Now with formula (3.22), we can easily predict the upper bound of the parallel efficiency by given certain $m, l, n, P, \hat{m}_p, \hat{l}_p, \hat{n}_p$ and $hattextBuffer_p$, and we can also locate the algorithms inside the space, with the metric of parallel efficiency.

3.4. **3D-Hypercube Family.** Out of vision of the algorithm designer, it cannot be directly indicated from formula (3.22) that how to choose $\{m_p, l_p, n_p\}$ for given m, l, n and Buffer_p, but with which we may estimate the performance of a parallel algorithm for matrices with certain dimensions, and so support us to promote the algorithm.

For classical method, such as Cannon's method[2], Fox method[1], and SUMMA[3], they all focus on dividing the hypercube into cubes with special shapes following some certain rules, as shown in the following figure for example:

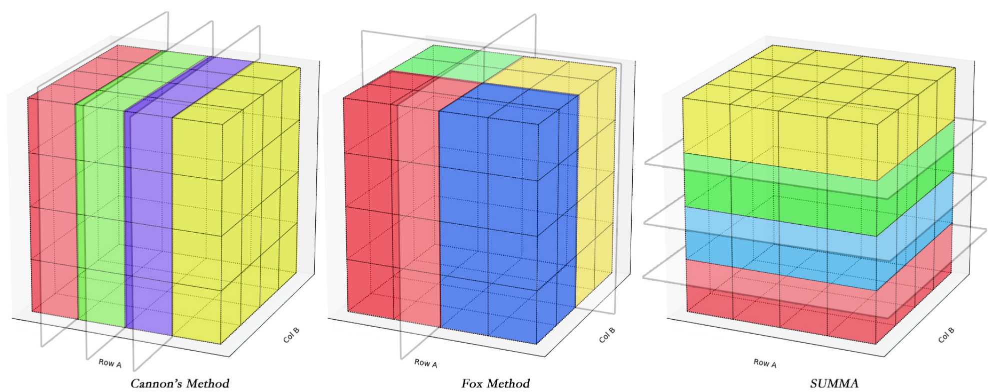

Figure 5. Some Classical Methods

But due to different dimensions of matrices involved in the multiplication, one certain strategy to design the shape of cubes can comparatively result in different performances. Here we may plot three different color maps to show the distributions of 3D-Hypercube algorithms for PMM, with the metric of parallel efficiency $\tau_{nogather}(m, l, n, P)$ defined in (3.22), under 3 different sets of dimensions, namely m, l and n:

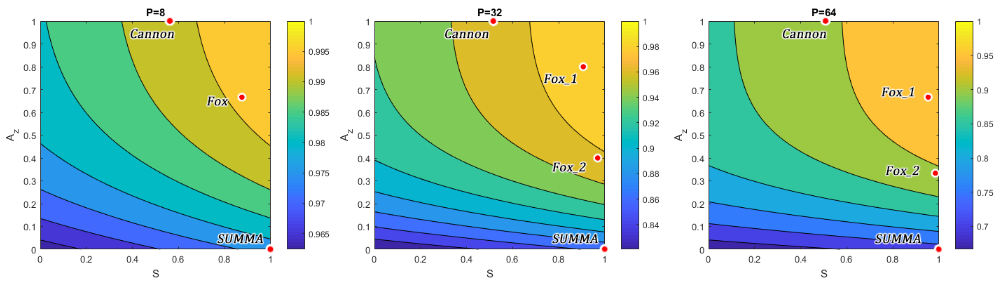

FIGURE 6. Efficiency Distribution: Square Matrices $_{(m=l=n=1024,\tau=1)}$

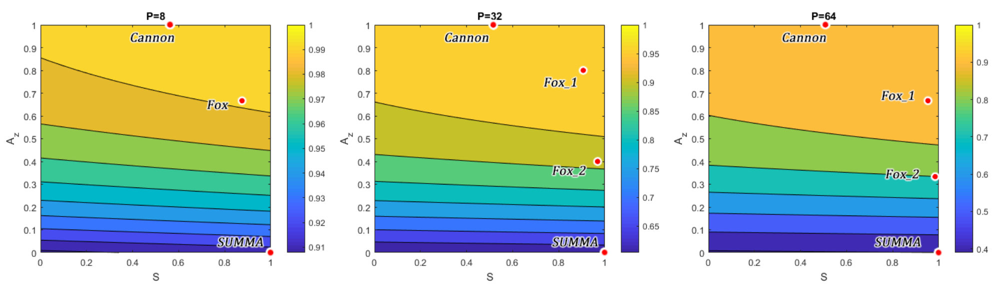

FIGURE 7. Efficiency Distribution: Tall Matrices $_{(m=n=512,l=4096,\tau=1)}$

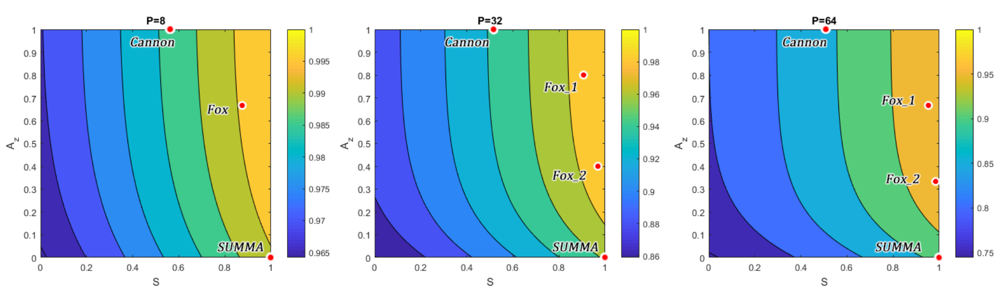

FIGURE 8. Efficiency Distribution: Flat Matrices $_{(m=n=2048,l=256,\tau=1)}$

So from the colormaps above, we may conclude some general ideas for constructing the algorithms for PMM based on 3D-Hypercube:

- It would be hard to increase $\hat{A}_z$ and $\hat{S}$ with limited buffer size;
- When l is larger than m and n, it would be better to be prior to slice the whole hypercube along z-axis, which means smaller $\hat{S}$ can compromise to smaller $\hat{A}_z$;
- When l is smaller than m and n, it would be better to let each cube cover more blocks along the z-axis;

And we have also raised some observations from the defined variables up to now:

- If a cube on node P is more regular, i.e. m_p, l_p and n_p are closer to each other, then with fixed volume $m_p l_p n_p$, $m_p l_p + n_p l_p$ can be smaller so that it can be more possible to increase $\hat{S}$;
- The larger the limit of buffer size Buffer_p is, the more that both $\hat{A}_z$ and $\hat{S}$ can increase;
- For matrices with larger scale, the proportion of computations would be larger, so that the distribution of such colormap would be closer to 1.

Based on those strategies, though we cannot reduce the total cost of computation, we can estimate and help to search out the best 3D-Hypercube algorithm for PMM, adaptive to certain matrices dimensions, limited buffer size and some other specific conditions.

4. Application of 3D-hypercube algorithm

In this section, we designed a series of experiments to verify our estimations of parallel efficiency using formula (3.22).

From formula (3.22), we want to find out the relationship between the parallel efficiency of a constructed 3D-hypercube algorithm and a few variables, namely the dimensions of matrices, buffer size, and the number of nodes.

We designed the experiments consist of two parts: one for **estimating τ** and the other one for **measuring time cost for particular algorithms**, all the experiments were tested in the SeaWulf Clusters[1], and run with the same one testing program, which is developed with MPI for C/C++ and is capable of parallel communication, and PMM of complexity $O(N^3)$.

4.1. Estimating the Value of τ.

As mentioned in Subsection 3.3 and referring to definition (18), τ is presumed as a constant value, associated with particular clusters with P cores. We ran the testing program to test two jobs: one for multiplying two matrices, both of dimensions 1024×1024, and the other ones for broadcasting an 1024×1024 matrix from each of P cores one by one, so that to measuring the average costs, on unit send-receive communication, and floating operation.

For taking the average, we have run two jobs both for 10 times, in 8, 32 and 64 cores assigned by SeaWulf respectively. And the results[2]are shown as below:

Table 1. Statistics of Communication and Computation Tests

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

So with these data, we can calculate γ respective to each P following the deduction below:

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

[2]The detailed results may be referred in Appendix A.

$$\frac{T_{becast}}{T_{mult}} = \frac{P(P-1)\cdot 1024^2 \cdot \hat{t}_{sendrecv}}{(C_{program}\cdot 1024^3 - 1024^2)\cdot \hat{t}_{op}} = \frac{P(P-1)}{C_{program}\cdot 1024 - 1}\cdot \gamma \tag{4.1}$$

Here $C_program$ means the account of floating operations executed per step in a practical program for MM. However, in practical cases, it is tough to measure $C_program$ stably and precisely, so compromising to experimental cases, formula (3.21) is modified as the following:

(4.2)

$$\begin{aligned}\tau(m,l,n,P) = &\ \frac{1}{P}\cdot\frac{mln\cdot T_{mult\_per\_step}}{\frac{mln}{P}\cdot T_{mult\_per\_step}+(\hat{t}_{send-recv}+\hat{t}_{op})\cdot\log_2 A_z\cdot mn\cdot\frac{A_z}{P}+\hat{t}_{sendrecv}\cdot 2\cdot l}\\ = &\ \frac{1}{1+\frac{\hat{t}_{sendrecv}+\hat{t}_{op}}{T_{mult\_per\_step}}\cdot\frac{\log_2\hat{A}_z}{l}\cdot\hat{A}_z+\frac{2\cdot\hat{t}_{send-recv}}{T_{mult\_per\_step}}\cdot P\cdot(\frac{1}{m}+\frac{1}{n})(1-\bar{S})}\end{aligned}$$

Here $T_{mult\_per\_step}$ is the time cost per step in the loop of a MM in programming.
Then comparing to formula (3.22), it is reasonable to obtain the following relationship:

$$\gamma \approx \frac{2\cdot\hat{t}_{sendrecv}}{T_{mult_{per_s}tep}} = 2\cdot\frac{T_{bcast}/P(P-1)\cdot 1024^2}{T_{mult}/(1024^3)} = \frac{2048}{P(P-1)}\cdot\frac{T_{bcast}}{T_{mult}} \tag{4.3}$$

So the estimated value of each γ is calculated as the following:

TABLE 2. γ Values Calculated from the Experiments

P	**8**	**32**	**64**
γ	0.7822	0.8195	0.8170

It has to be admitted that there values are less than the real ones due to those not considered, and the communication performance is usually more complicated in parallel program, than that analyzed in theory so that hard to obtain the correct γ.

4.2. **Experiments of Particular Algorithms.** For each cases with 8, 32 and 64 cores respectively, we designed 7, 10 and 10 different strategies to split the 3D-hypercube and allocate the buffers in different way, so that constructed various algorithms for testing the performances of each.

To evaluate the performance, the formula for calculating the parallel efficiency is applied here:

$$\tau = \frac{1}{P}\cdot\frac{T_{one\_core}}{T_{P\_Cores}} \tag{4.4}$$

The plans with results[3]are shown as the following subsections:

[3]The detailed results may be referred in Appendix B.

4.2.1. $P = 8$. (total 7 cases)

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

4.2.2. $P = 32$. (total 10 cases)

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

4.2.3. $P = 64$. (total 10 cases)

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

Therefore, finally we can plot the experimental results inside a 3D space together with the expected surfaces sketched from formula (3.22), which is shown as the following:

Comparing to the analytic surfaces, we found that the communication parts of the algorithms are indeed affected by more other elements not discussed in this paper, which to be searched out is part of the future work on BAMMA.

5. Conclusions and future work

In this paper, we have discussed about the construction of algorithms for PMM, based on 3D-Hypercube so that with total computational complexity $O(N^3)$. We have observed that the construction of such algorithm is influenced by the number of computational nodes, the dimensions of matrices and the limit of the buffer size.

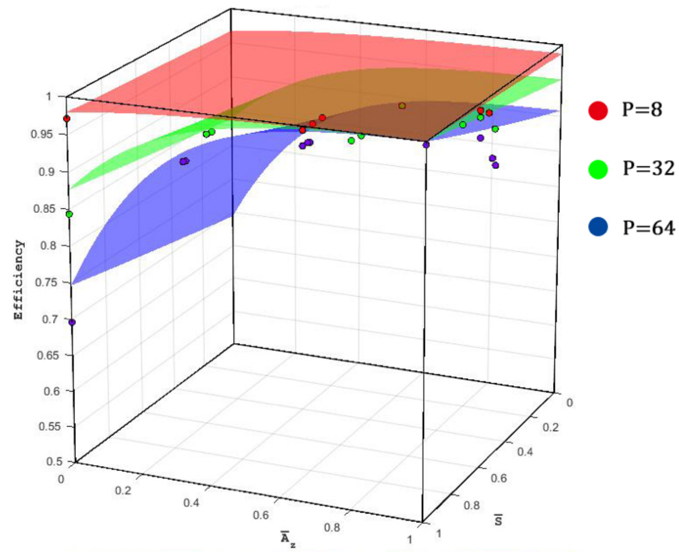

FIGURE 9. Efficiency Distribution with Experiments Results

Concluded from both the theoretical analysis and the experiments, it is learnt that often it is better to construct the parallel algorithms considering adaptive to certain cases, described by the three variables mentioned above, rather than primly follow the rules defined by classical methods.

All of the theoretical analysis is based on the assumption that all the cubes and communication trees onside the P nodes are designed similarly. So as another part of our future work on PMM, we are going to dig out the possibility to design and evaluate the algorithm, with more flexibility on each specific node with specific condition respectively, so that it can adapt to more practical cases with the best performance. As one of the most latest coming work, we are going to optimize the algorithms on clusters with particular or even special topologies, using the ideas raised in this paper.

And as mentioned in Section 3.2, another one of our research plans in the future is to optimize and finish the program of BAMMA, so that it can satisfy the requirement for parallel universal MM adaptive to specific conditions, expected with better performance comparable to the classical method. And referring to a popular branch of algorithms of PMM, the recursive algorithms, for instance CARMA[4] raised by the team led by James Demmel in 2013, we may need to furtherly research on the influence of sequential recursion, on the performance of a parallel algorithm of PMM.

Furtherly expanding the algorithm into higher dimensions, series of productions, and special matrices, is also worthwhile. Obtaining a reliable method of parallel matrix multiplication may provide a way to predict and promote the speedup and efficiency of tensor production [7]. And to follow current great researches on sparse tensor space[8], 3D-hypercube algorithm may be helpful with assiting constructing a denser space for faster computitation, with avoiding the uncessary and redundant calculations.

Finally, as the final target to optimize the algorithm for general matrix multiplications, our group has proposed a formulated problem in 4D Hypercube, with the concepts of operation map mentioned in Section 2.1, to reduce the computational complexity of matrix multiplication, of which is lower than $O(N^3)$, inspired by the ideas[5], raised by Strassen in 1969as well as Coppersmith and Winograd[6] in 1987. And now there has been some great work on that, for instance the research on parallel Strassen on MIMD system[9]. Our next target is to find a general strategy to solve the problem so that find the limit of recursive algorithms to reduce the total complexity of matrix multiplications, and furtherly construct stable and scalable parallel algorithm for universal matrix multiplication, with lower computational complexity.

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. Chung-Chiang Chou, Yuefan Deng, Gang Li, and Yuan Wang. *Parallelizing Strassen's Method for Matrix Multiplication on Distributed Memory MIMD architectures*.

Appendix A: Measurement of the values of γ

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

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

$P = 8, m = l = n = 1024$: (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

$P = 32, m = l = n = 1024$: (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

$P = 64, m = l = n = 1024$: (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, Stonybrook Univ. SUNY, Stony Brook, New York 11790

E-mail address: wenjing.cun@stonybrook.edu

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

E-mail address: lihua.pei@stonybrook.edu

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

E-mail address: yuefan.deng@stonybrook.edu