ANALYSIS OF PARALLEL MATRIX MULTIPLICATION ALGORITHMS

WENJING CUN, LIHUA PEI, AND YUEFAN DENG

ABSTRACT. This paper proposes a general vision onto the construction of the algorithms of parallel matrix multiplication in distributed-memory system. Via the analysis of the communication forest of the algorithms, this paper presents a map of the parallel algorithms for universal matrix multiplication, based on 3-dimensional hypercube algorithm with computational complexity of $\mathrm{O(N^3)}, and proposes an open problem of promoting the computational complexity in 4-dimensional hypercube.$

1. INTRODUCTION

High-performance computing (HPC) algorithms of linear operations algebra are critical for many of the computing tasksapplications in engineering and science., Tand to promote the performance of computing in computers, the topology-based algorithms of operations on specific fields are more valuable for research, in order to promote the flexibility and robustness more regardless of the values involved.

And Aas one of the most essential linear operationsalgebra problems, matrix multiplication is a good start point to launch our research on abstract operation. This paper will mainly discuss about the general method of constructing algorithms for matrix multiplication, and the parallelization of it, with several classical methods as well as analysis of their performances for instance.

In this research, we have discovered a general map to organize the family of the algorithms, and with the metric of parallel efficiency of the lattice, it is now possible for mathematicians to evaluate the upper bound of the algorithms depending on any given dimensions of the matrices,

Furtherly, this paper proposes a formulated problem to reduce the computational complexity of the algorithm of matrix multiplication, which is still not resolved, yet we will figure it as a future work to research on, to lead the way to optimize the distributed algorithms with computational complexity lower than $\mathrm{O(N^3)}$.

The organization rest of this manuscriptpaper is organized as the followsing: Section 2 will presents the general idea of designing the algorithms of matrix multiplication on 3D hypercube, based the maps between sets of coordinates. In Section 3, we will propose the problem of describes the construction ofng the communication forest among the computation nodes, while proceeding the parallel algorithms, and organize the family of the parallel algorithms based on 3D hypercube. Then Section 4 will present theperforms analysis of several classical parallel algorithms for matrix multiplication algorithms and proposes an universal matrix multiplication that encapsulates all published algorithms.general algorithm for constructing the distributed plan for universal matrix multiplication. Finally Tthis paper will propose a problem of reducing the computational complexity, and summarize the construction of algorithms. Finally, Section 5 gives the conclusions and outlines the future work.

2. CONSTRUCTION OF MATRIX MULTIPLICATION IN 3D HYPERCUBE

Firstly, we shall review the naïve algorithm of matrix multiplication.

Naïve algorithm is the direct definition of matrix multiplications for any given pair of matrices: AR$^(ml), BR^(ln), wherem, landnaresomepositiveintegers.Andifwejustfocusononefinalresult$
$AB, namelyC_ij, thenwehave :$

Then regardless of the actual values of A$_ijandB_jk, formula(1)revealstheinformationthatC_ikonlyresultsfro$
$definedmapfromsuchasetof(i, j, k)tothatof(i, k)inC.$

Sourced from such an idea, and in order to make the algorithms totally independent of the values involved, so that can be flexible and robust, we will define such a map from a set of coordinates to another one in the first subsection.

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 proposed the concept called operation map, of which the definition is as the following:

Definition 2.1. An operation map is a map from one set A to another set B, where there are two injective maps L$_1$ $: CA, L_2 : CB, andCisasetwithawell -$
$definedoperationfunction.Forconvenience, wedenotesuchanoperationmapfas : f_(C,) :$
$AB, andnameL_1, L_2asthecoordinatemaps.HereAandBarethetopologiesfromwhichthesetsofvaluesinCabstrac$

And an abstract operation map has a property sourced from the abstraction of the value set to coordinate sets:

Property 2.2. Given an operation map f$_(G,) : C_1C_2, ifSisclosedunderawell -$
$definedoperation, orsayGisanalgebraicgroup, thenforanysubsetSG, f_(S,) : C_1C_2isalsoanoperationmap.ViceV$

Such a property shows the stability of an operation map regardless of the values when the inputs in conservative of the algebraic operation.

For a series of operation maps, they will follow the chain rule if they follow property 2.1:

Property 2.3. Given a set G that is closed under a series of operations $_1, \ldots,_n, aseriesofsetsC_1, \ldots, C_n, alsoase$
$(f_1)_(S_{1,1}), \ldots, (f_n)_(S_{n,n})$

And (f$_1)_(S_{1,1})\ldots(f_n)_(S_{n,n})isalsoanoperationmap.$

Following the chain rule in the opposite direction, if an abstract operation map f$_(G,) : C_1C_2canbedecomposedintoaseriesofabstractoperationmaps, thenwesayf_(G,)isseparable.$

The third property we raise here shows one additional great value of abstract operations, that is the flexibility, aka the scalability in computational engineering, if the operation map can be conservative in topology:

Property 2.4. Given two homomorphic algebraic structures G$_1andG_2, closedundertwooperations_1and_2respecti$
$C_1C_2, thenf_(S_{2,2}) : C_1C_2isalsoanoperationmap.$

In the next subsection, we will present the ideas of abstracting the process of matrix multiplication, based on the naïve algorithm 2.1, into a separable operation map composed by several independent stages, and shows the methodology to parallelize the algorithm.

2.2. **Algorithms Based on 3-Dimensional Hypercube.** To abstract the naïve algorithm 3.1 into the operation maps into operations maps mentioned in the previous subsection, the first step is to split all the multiplications between a pair of scalars out, then for the 3D-hypercube algorithm, this step means an abstract operation map from a coordinate set C$_1 1 to the other one C_1 2. Since the original coordinate set represents the positions of al$

$$(C_A = ZmZZlZ, C_B = ZlZZnZ)$$

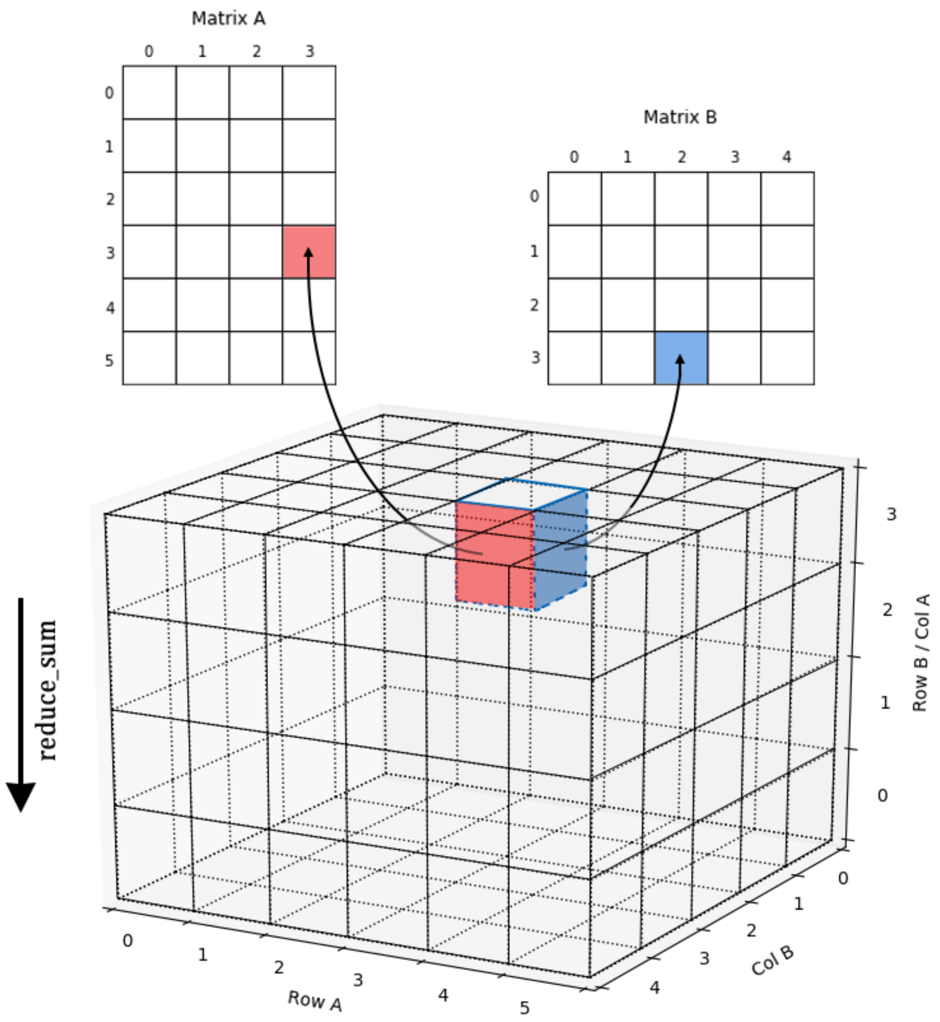
And define the following abstract operation map:

$$(2)(f_1)_(R,) : C_A C_B C_m ult3D$$

where C$_m ult = ZmZZlZZnZ, "" is the scalar multiplication and :$

$$(3)(f_1)_(R,)((r_a, c_a), (r_b, c_b)) = ((r_a, c_b, c_a) if c_a = r_b @ ohterwise)$$

This abstract operation map represents the process: A$_i j B_j k, and from C_m ult, it can be observed that all the mappi$



The next process to be considered is the addition part, here since it is known that only the scalar operation is remained, and the result is a matrix CR$^(mn), thuswedefinethethirdcoordinatesetas:$

$$C_C = ZmZZnZ$$

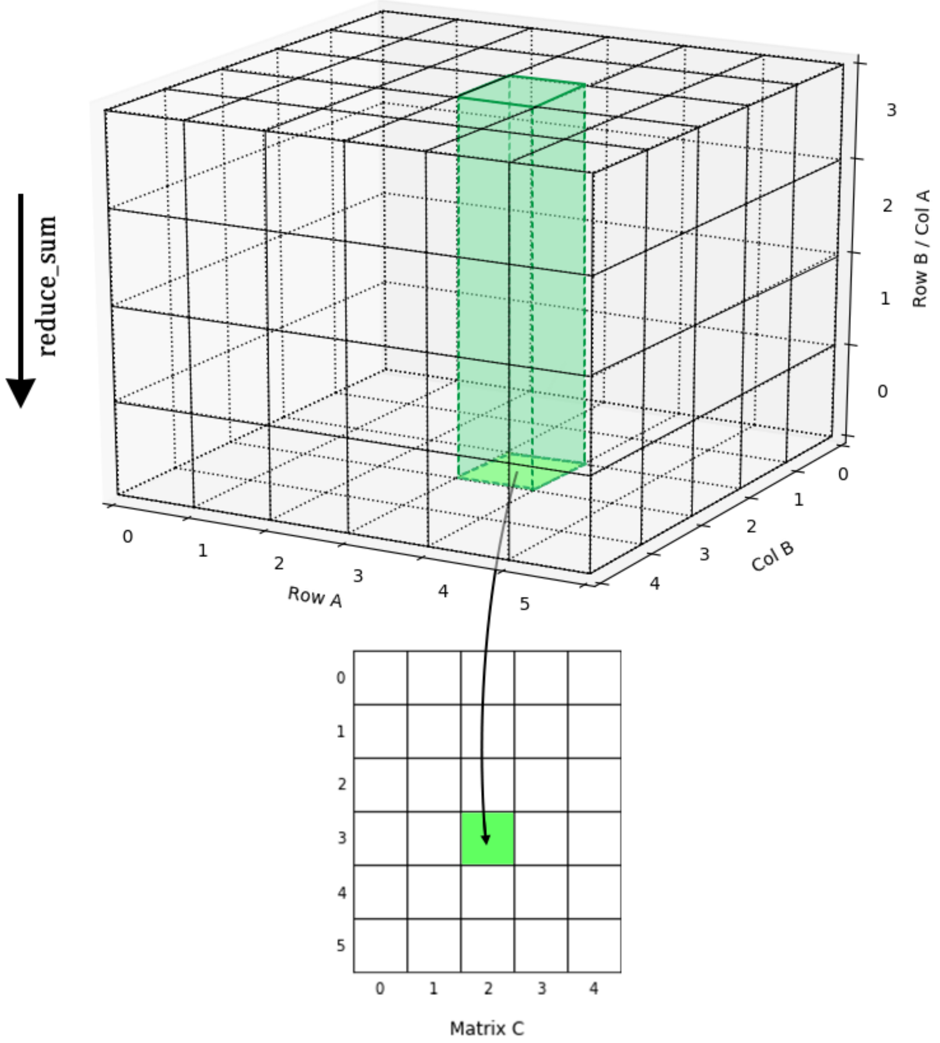
And the corresponding abstract operation map is defined as:

$$(4)(f_2)_(R,):C_mult3DC_C$$

$$(5)(f_2)_(R,+)((r_a,c_b,c_a)) = (r_a,c_b)$$

Here "+" is the scalar addition.

Observed from (4), all the 3D "cubes" in the Figure 2.1 with the same "z-value" are reduced in to one in the "xy-plane", which can be plotted as the following:



Now the whole process has been abstracted into a chain of abstract operation maps, for the matrix multiplication operation: F$^(ml)F^(ln)F^(mn), wecandefinethechainofabstractoperationmap$

$$(6) f_(R^(ml)R^(ln), matrixmultiply) = (f_1)_(R,)(f_2)_(R,+)$$

where we define f$_(R^(ml)R^(ln), matrixmultiply) : 11, isatrivialabstractoperationmapdirectlyrepresentingtha$
$C = AB$.

For a local-memory system, there seems nothing to modify (5) 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$_(R,=), where" = "simplymeanstheidentityoperation, butcanfilterthecoordina$

$$(7) g_(R,=), C_mult3DC_mult3D$$

g$_(R,=)((r_a, c_b, c_a)) = ((r_a, c_b, c_a), if(r_a, c_b, c_a)U_p@, otherwise),$

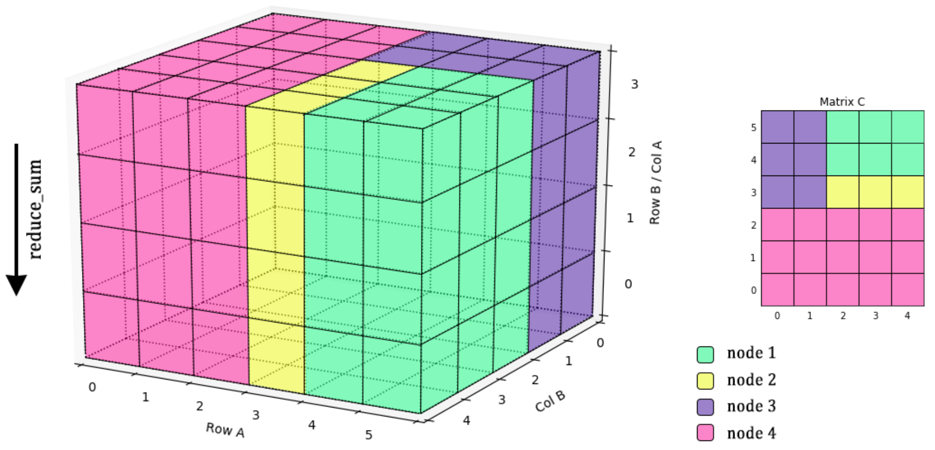
Here U$_p isacustomsetchosenforeachnodep, calledafilteringset, representsthe\blocks" chosenfornodeptocalc$

$$(8)(f_p)_(R^(ml)R^(ln), matrixmultiply) = (f_1)_(R,)g_(R,=)(f_2)_(R,+)$$

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 AB, where AR$^(64), BR^(46), thenchoosethefourfilteringsetsas:$

$$(9)(U_1 = 4,52,3,4(Z4Z), U_2 = 32,3,4(Z4Z)@U_3 = 3,4,50,1(Z4Z), U_4 = 0,1,20,1,2,3,4(Z4Z))$$

So the "blocks" in C$_multallottedtoeachnodeareshownasthefollowingfigure:$



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 matrix multiplication, 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 matrix multiplication, all the algorithms based on naïve algorithm 2.1 can be deduced by selecting the filtering sets U$_p, sothatthealgorithmispresentedasthefollowingpseudocode:$

$Algorithm 2.2 : 3D-Hyperblock Matrix Multiplication(AR^(ml), BR^(ln)) if at rank p, Choose U_p = (i_1, j_1, k_1), (i_2,$

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$_p U_p, and find the best communication tree to pass the data among$

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 parallel matrix multiplication.

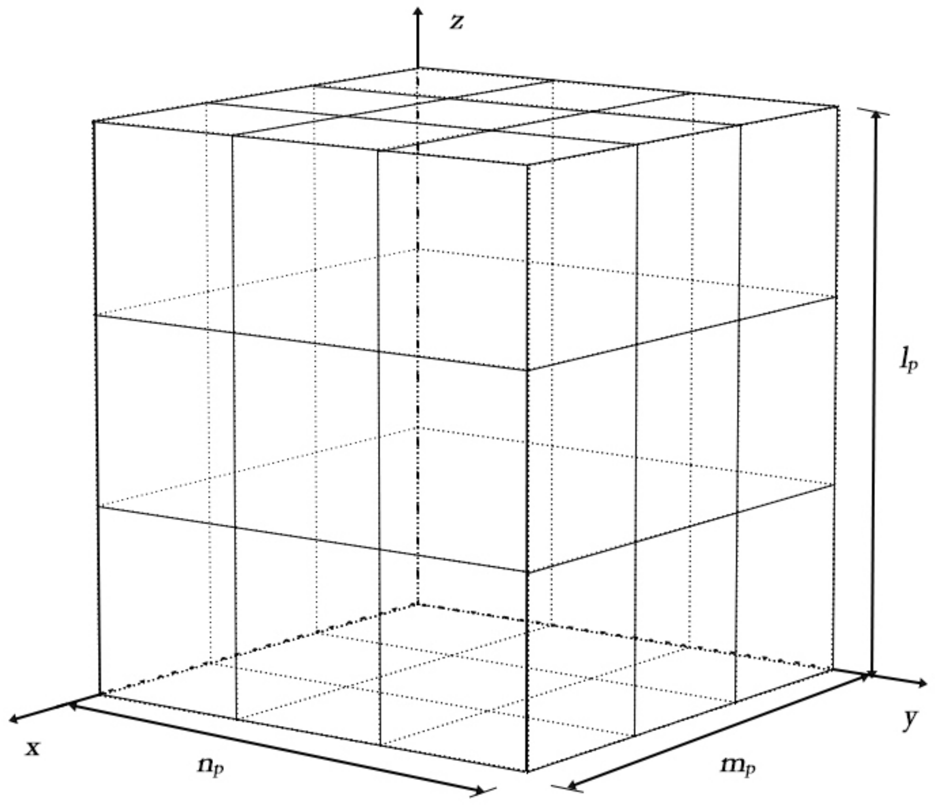
3. COMMUNICATION IN 3D-HYPERCUBE

Allocation of blocks of operations to each computational node is straightforward for 3D hypercube algorithms, however, while dealing with large-scale matrix multiplications, 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 parallel matrix multiplication, referring to algorithm 2.2, often we can only deploy part of U$_p onto node p, then following the communication tree, at each step, one need to finish$

3.1. **Communication Rules Cost.**

For convenience, we describe a "cube" assigned to node p with: m$_p, l_p, n_p, the lengths of the cube assigned to node p, along with x-axis, y-axis and z-axis respectively;$



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:

$$(10)_(p = 1)^P m_p l_p n_p = mln$$

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:

$$(11)_(p=1)^P Buffer_p ml + ln$$

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:

$$(12)T_(p, recv) = C(m_p l_p + l_p n_p - Buffer_p)$$

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:

$$(13)(T_(comm, send - recv)_(p=1)^P T_(p, recv) send/require one by one@T_(comm, send - recv) maxp T_(p, recv) comm$$

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. **Communication Rules Cost.** 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:

Numerate U$_p = (x_p i, y_p i, z_p i) by taking the operation map (3) mentioned in section 2.1, which is source from A_p k B_p$

$$(14)|C_p Ak| + |C_p Bk| Buffer_p$$

So that we can see, to parallelize the matrix multiplication 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:

$$(15)(f_p 1)_(R,) : C_p A C_p B C_(p, mult3D)(parallelized) C_(p, mult3D) =_(k=1)^(total steps K) C_(p, mult3D, k) =_(k =$$

Now to minimize the communication cost on each node with fixed limit of buffer size, we want to prove that: To numerate U$_p = (x_p i, y_p i, z_p i), for C_p Ak and C_p Bk satisfying (14), there can exist C$
$For k > 1, d C_p Ak C_(pA, (k+k)) only if d C_p Ak C_(pA, (k+k-1)); For k > 1, d C_p Bk C_(pB, (k+ k)) only if d C_p Bk C_(pB, (k+k-1));$

if given that Buffer$_p min m_p, n_p$.

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 Buffer$_p min m_p, n_p can be stated as the following :$

$Algorithm 3.1 : 3D{-}Hyperblock Algorithm with Communication(m_p, l_p, n_p, Buffer_p N^+, m_p n_p)$

if at rank p, Choose U$_p = (i_1, j_1, k_1), (i_2, j_2, k_2), \ldots, (i_K, j_K, k_K) = (Z_(m_p) Z_(l_p) Z_(n_p))(i_l ow, j_l ow, k_l ow), Let C$
$(Z_(m_p) Z_(l_p))(i_l ow, j_l ow) and C_p B = (Z_(l_p) Z_(n_p))(j_l ow, k_l ow). Sort U_p, C_p A and C_p B increasingly with j. Let Step Si$

$Buffer_p m_p. And initialize the buffer as : (C_(pA, 0) = C_p A[0 : StepSize_A], C_(pB, 0) = C_p B[0 : Buffer_p - StepSize_A]) Set numbers of steps associated to two sets as k_A = k_B = 0.$

for i,j,k in U$_p, if (i, j) C_(pA, k_A) and (j, k) C_(pB, k_B) : C_i k = C_i k + A_i j B_j k ready to send (i, j) C_(pA, k_A) and (j, k) C_($
$require (j, k) from another node pop the front of C_(pB, k_B) push (j, k) into the back of C_(pB, k_B) k_B = k_B + 1 else if (i, j) C_(pA, k_A) : require (j, k) from another node pop the front of C_(pA, k_A) push (i, j) into the back of C_(p$
$k_A + 1$

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 parallel matrix multiplication in 3D Hypercube family.

3.3. **Performance Analysis.**