Matrix transpose on meshes with buses*

József Békési,Gábor Galambos

Department of Applied Informatics, Gyula Juhász Faculty of Education, University of Szeged, Hungary

Abstract

In this paper we analyze the matrix transpose problem for 2– and 3–dimensional mesh architectures with row and column buses. First we consider the 2-dimensional problem, and we give a lower bound of approximately $0.45n$ for the number of steps required by any matrix transpose algorithm on an $n \times n$ mesh with buses. Next we present an algorithm which solves this problem in less than $0.5n + 9$ steps. Finally, we prove that the given lower bound remains valid for the 3-dimensional case as well.

1. Introduction

The rapidly increasing computational demands of the applied sciences pushed the computer systems progressively towards the higher computational capacity. In the same time they required more effective algorithms. Therefore, recently high performance computing is in the focus of computer science. To make computers more efficient, developments were needed both on the fields of hardware and software. Hardware developments resulted in multicore processors and connected computers with different architectures, while the algorithms became more sophisticated step by step and they have been analysed deeper than ever before. A good architecture or a more efficient algorithm may decrease the processing time strongly in a parallel computational enviroment.

*The paper was supported by the Austrian-Hungarian Action Foundation (Project number: 91öu2)

Email address: {bekesi,galambos}@jgypk.szte.hu (József Békési,Gábor Galambos)

On the hardware side the effectiveness of any parallel computation effort vigorously depends on how fast we can send data from a source processor to a destination one. Therefore different architectures were developed in the last two decades. Hypercubes, tori and meshes are the architectures that have been intensively studied. See eg. [8], [14], [17], and [22].

Routing, sorting, merging, and matrix transpose are the problems that were investigated already in the early ages of parallel computation (see eg. [2], [11], [12], [19]). These problems are among the basic ones, that often appear in numerical computations. For example matrix transpose is one of the basic operations in linear algebra. The speed of such computations can be critical in some real time practical applications, like digital signal processing, image processing, radar systems, etc (see [23], [3]). To exploit the increased computational capacity on the software side parallel algorithms have been developed, so in the last decade parallel processing has been further improved by leaps and bounds. All of the investigated algorithms were accomodated to a given architecture. The effectiveness of certain algorithms were investigated extensively, see eg. [4], [6], [9], [13], [18], [21] and [20].

The effectiveness of a parallel computation effort strongly depends on how fast we can send data from a source processor to a destination one. Meshes are the architectures that are flexible, the processors can be connected in different ways, and they are suitable to implement different algorithms in an efficient way. Therefore among the different architectures the most extensively studied ones are the mesh architectures. In the simplest, one dimensional (1D) case a mesh is a linear array where the elements are the processors and each processor is connected by a full duplex line with its neighbors. In higher dimensions (2D, 3D) processors form an array, and they are connected by communication links. Figure 1. shows some mesh architectures.

Execution of any algorithm is performed in *steps*. In one step two connected processors can change data. Normally, only the connected processors can communicate with each other. There are different communication modes which influence the speed of data transfer. MIMD, SPMD, SIMD and the Weak SIMD are some examples for communication. (More details see in [7], [15], and [16]).

In this paper we suppose MIMD communication among the processors, i.e. processors choose their communication directions independently, and they can communicate with all their neighbors in one step.

The efficiency of an algorithm is measured by the number of steps needed

to fulfill the given task. While routing from a source processor to a destination one, data may pile up at a processor. This may cause a bottleneck effect if we do not have enough memory for storing these data. In this paper we assume that all processors have sufficiently large memory to store the waiting data – sometimes called as *messages* or *items* – in separate queues.

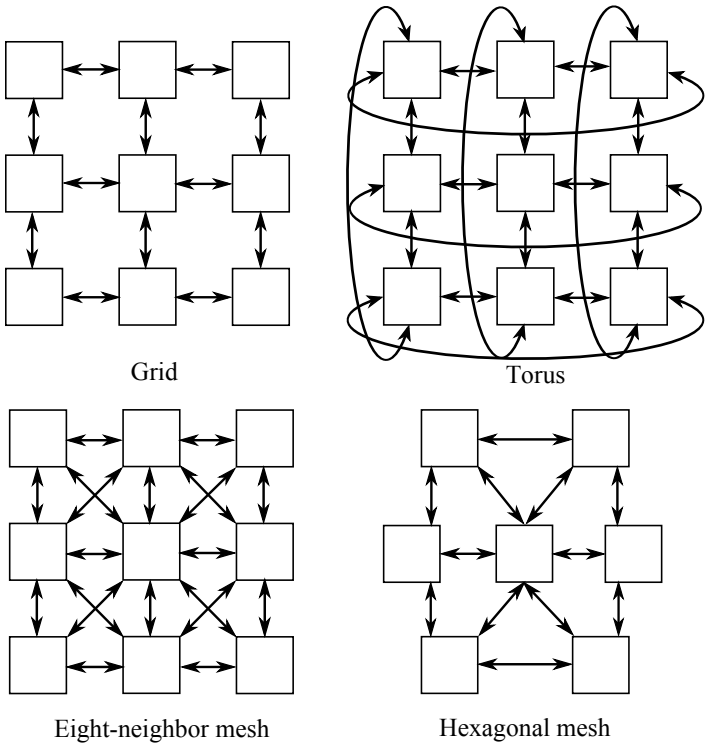

Figure 1: Some mesh architectures

If the processors can communicate with only their neighbors, then sending data from a processor to a far one may take many steps. There are different ways to avoid this situation. In [24] the so-called *wormhole switched meshes* are considered. In case of whormhole routing the data transfer has two steps. In the first one a circuit is established between the source and destination processors facilitating a quick data transfer between the nodes, and in the second step packets are sent over different paths independently from each other. The advantage of this communication lies in the first step: although it takes more time than the second one, it builds up a direct connection between the processors, and the second step allows to send packages between

the processors saving much more time.

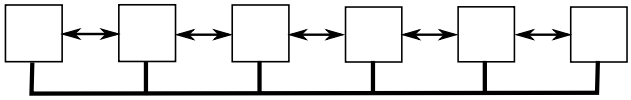

Figure 2: 1D mesh with bus

To speed up the communication between two far processors, it is possible to use buses. Buses can be used in 1D meshes (see Figure 2.) and for 2D meshes as well. We show different bus-configurations in Figure 3. Row and column buses were used in [1]. If we use a bus then the processors connected to the bus can communicate not only with their neighbors but also with the ones that are connected to the same bus. In one step only one processor can send data to a bus and one of the others can accept it in the same step. In case of 2D meshes we can use row and column buses, and all processors in the same row or column are connected to one bus. To a 2D mesh which has both, row and column buses we will refer as *2RCB-mesh*.

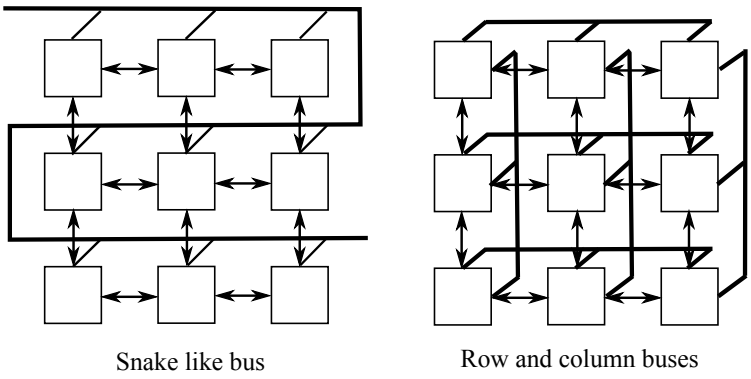

Figure 3: Single (snake-like) bus, row and column buses

Depending on the considered problem we need to move the data in different ways. In case of the *permutation routing problem* each processor should send $k(\geq 1)$ messages to the same processor. We call this the $k-k$ permutation routing problem. We say that the problem is solved, if each message has arrived at its destination. Such problems were considered in [1].

A special permutation routing problem is the *matrix transpose problem* (MTP) on a 2D mesh. In this case a message originally contained by the

processor (i, j) should be routed to the processor (j, i) for all i, j where $1 \leq i, j \leq n$. We will call those two processors *pairs*. For 2D meshes with MIMD processors and without buses Ding, Ho and Tsai [5] analyzed the *MTP*. They denoted by $T_A(k, n)$ the number of steps needed to transpose k pieces of $n \times n$ matrices by the algorithm A. For this $k - k$ version their main result is the following lower bound. For any *MTP* algorithm A,

$$T_A(k, n) \geq (1 - 1/\sqrt{2})kn \approx 0.293kn.$$

Later Kaufmann, Meyer and Sibeyn [10] gave an algorithm which requires $0.301kn + O(n/k)$ steps. So, for any constant k the additive term is proportional to n, therefore the gap is large between the upper and the lower bounds.

A natural generalization of the (2D) matrix transpose problem to d-dimensions ($d \geq 2$) is to consider the permutations

$$(a_1, \ldots, a_d) \rightarrow (a_i, a_{i+1}, \ldots, a_d, a_1, a_2, \ldots, a_{i-1})$$

for some $i, 1 \leq i \leq d$. These permutations are also called *transposes* [10]. There are d transposes, one of them is the identity permutation. Especially in 3D the two non–trivial transposes are $(i, j, k) \rightarrow (j, k, i)$ and $(i, j, k) \rightarrow (k, i, j)$. An architecture of a 3D mesh with buses in each direction will be denoted by *3RCB-mesh*.

In this paper first we will show that in case of a *2RCB-mesh*, we can improve the efficiency of the matrix transpose algorithms. More precisely, if we denote by $T_A^B(1, n)$ the number of steps needed to transpose a matrix by the algorithm A on a *2RCB-mesh*, then in Section 2 we prove that $\lim_{n\rightarrow\infty}(T_A^B(1, n)/n) > 0.4508\ldots$, and we will define an algorithm – denoted by *MTB* – for which $T_{MTB}^B(1, n) \leq \frac{n}{2} + 9$. We also investigate the 3D case, and we prove that any solution of a matrix transpose problem with a *3RCB-mesh* architecture needs at least $0.45n$ steps. In our analysis we consider only the $1 - 1$ version of the problem. Finally, in the conclusion we mention some open problems.

2. Lower Bound in 2D

Let us consider the *MTP* on a *2RCB-mesh* architecture, and let n be the number of processors both in the rows and the columns. In this case the following theorem is true.

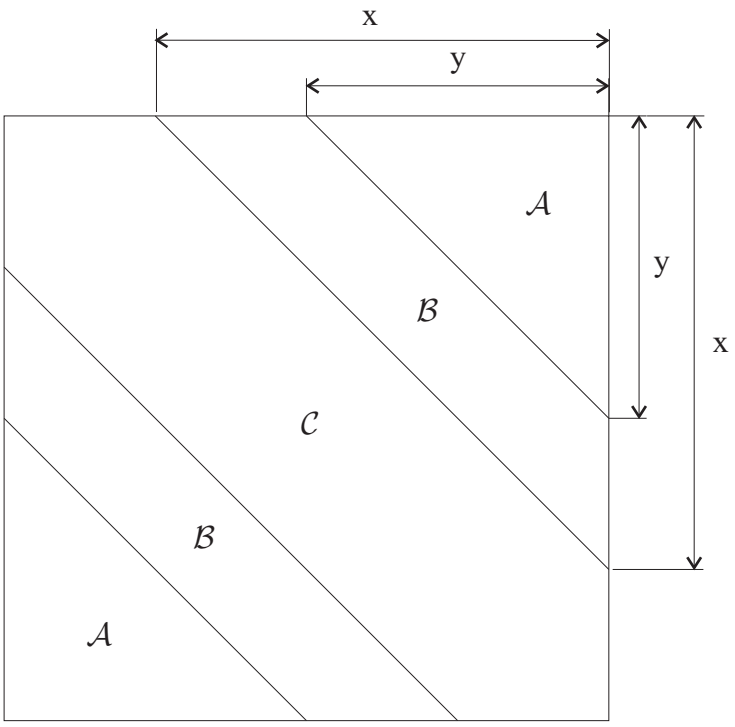

Figure 4: Division of the $n \times n$ mesh

Theorem 2.1. *Let A be an arbitrary algorithm and let $T_A^B(1,n)$ be the number of steps needed to solve the* MTP *on a* 2RCB-mesh *with $n \times n$ processors. Then*

$$\lim_{n\to\infty} \frac{T_A^B(1,n)}{n} \geq 2 - \frac{2}{5}\sqrt{15} \approx 0.450806\ldots \tag{1}$$

Proof. The idea of the proof is that we compare the walking distances to the total number of bus opearations required to send data to far processors and calculate the optimal number of steps. Let us divide the $n \times n$ processors into 5 diagonal regions as shown on Figure 4. The regions denoted by the same letters ($\mathcal{A}$ and $\mathcal{B}$) contain equal number of processors in the corresponding rows and in the columns as well. Furthermore, let us choose x so that $\frac{n}{2} < x < n$, and let $y = 2x - n + 1$. So we get that $0 < y \leq x$. Let $P(\mathcal{A}_i)$ and $P(\mathcal{B}_i)$ be the number of processors in the i-th row in a region $\mathcal{A}$ and $\mathcal{B}$, respectively ($1 \leq i \leq n$). The regions are chosen so that $x = P(\mathcal{B}_1) + P(\mathcal{A}_1)$, and $y = P(\mathcal{A}_1)$.

Let us denote the distance of a pair by $|p_{i,j}|$, which means the minimum

number of steps while a message moves to a destination processor through the connection lines. We will call a route that uses only communication lines to reach the destination processor as *walk*. Then $|p_{i,j}| = 2|i-j|$. We will investigate the regions $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ separately.

In the region $\mathcal{C}$ processors are close to each other, and

$$\max_{p_{i,j} \in C} |p_{i,j}| = |p_{(n-x),1}| = 2(n-x-1) \tag{2}$$

The distance between any pair in the regions $\mathcal{A}$ and $\mathcal{B}$ is always longer than $2(n-x-1)$, which means that if we want to get a better result, then each message must use a bus at least once in these regions. For the pairs in regions $\mathcal{A}$ we get that

$$\min_{p_{i,j} \in A} |p_{i,j}| = |p_{n,y}| = 2(n-y),$$

so we get that

$$2(n-y) = 2(2n-2x-1) > 4(n-x-1). \tag{3}$$

From (3) it follows that the messages in the region $\mathcal{A}$ must use bus twice to reach their destinations in at most $2(n-x-1)$ steps. This means that the total number of steps that use bus operations while routing all the messages is at least

$$4P(\mathcal{A}) + 2P(\mathcal{B}).$$

We get that

$$P(\mathcal{A}) = \sum_{i=1}^{y} P(\mathcal{A}_i) = 1 + 2 + ... + y = \frac{(2x-n+1)(2x-n+2)}{2}.$$

Similarly,

$$\begin{aligned} P(\mathcal{B}) &= \sum_{i=y+1}^{x} P(\mathcal{B}_i) = \ 1 + 2 + ... + x - P(\mathcal{A}) = \\ &= \frac{x(x+1) - (2x-n+1)(2x-n+2)}{2}. \end{aligned}$$

Since the total number of buses is $2n$, routing of all the $2(P(\mathcal{A}) + P(\mathcal{B}))$ messages requires at least

$$T_A^B(1,n,x) = \frac{4P(\mathcal{A}) + 2P(\mathcal{B})}{2n} = \frac{x(x+1) + (2x-n+1)(2x-n+2)}{2n}.$$

steps. Since $T_A^B(1, n, x)$ is an increasing function, while $2(n - x - 1)$ is a decreasing function of x, we get the best possible choice for x by solving the equation

$$T_A^B(1, n, x) = 2(n - x - 1). \tag{4}$$

The solution is

$$x = -\frac{7}{10} + \frac{1}{10}\sqrt{60n^2 - 20n + 9}.$$

Substituting this into (2) and using equation (4) we get the desired result.

3. Upper Bound in 2D

Now, we define an algorithm for solving the matrix transpose problem in 2–dimensions. During the construction we will follow the ideas used in the proof of the lower bound: those messages which are close to their destinations will walk. We call these messages *W-messages.* For a longer distance a message will be routed using one bus transfer and some walk. These are the *BW-messages.* Those messages which are very far from their destinations will be scheduled to use buses twice. We call them *BB-messages.* There are two basic problems:

- How can we determine the regions which define the message-type during the execution?
- Those pairs which are very far from each other are placed in the lower left and upper right corners. This induce that they must be defined as BB-messages. But, in this case message transfers for these pairs require a heavy usage of the outer buses, while the center buses remain idle. So, to make our algorithm more efficient we change the schedule of some parts of these messages: first they will walk to reach one of the buses which are closer to the center, and some steps later they will "catch" one of the center buses. So, they become BW-messages. This modification results in load-balancing among the buses.

We will call our algorithm the Load-Balancing Algorithm (*LBA*).

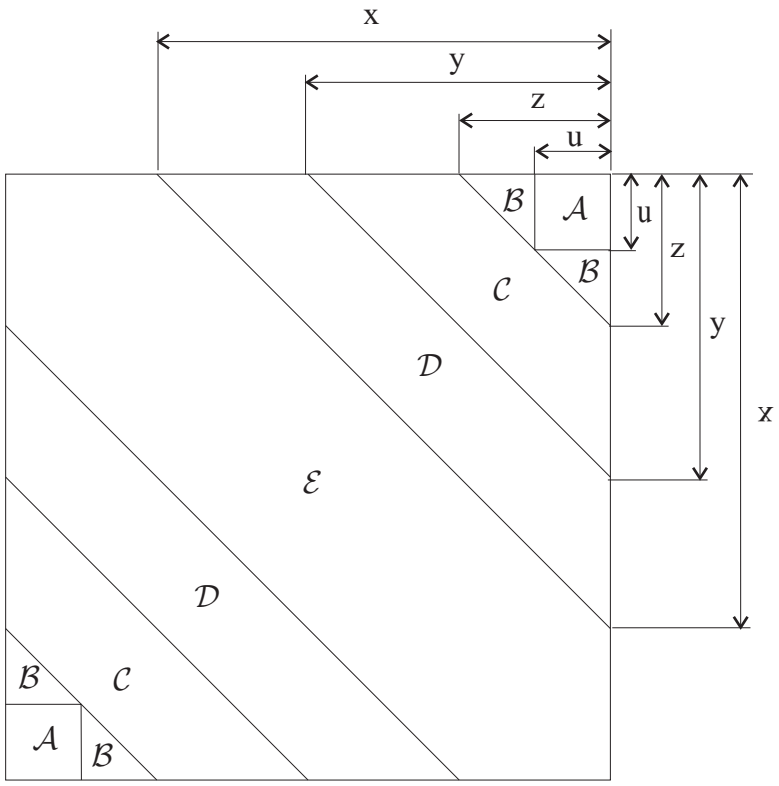

Figure 5: Division of the $n \times n$ mesh by algorithm LBA

3.1. Construction of LBA

Preparation step: Let

$$x = n - \left\lfloor \frac{\lfloor \frac{n}{2} \rfloor}{2} \right\rfloor - 1, \quad y = \left\lceil \frac{n}{2} \right\rceil,$$
$$z = \left\lfloor \frac{\lfloor \frac{n}{2} \rfloor}{2} \right\rfloor + 1, \quad u = \left\lceil \frac{\left\lfloor \frac{\lfloor \frac{n}{2} \rfloor}{2} \right\rfloor + 1}{2} \right\rceil$$

and divide the processors into five diagonally symmetric disjoint sets $\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D}, \mathcal{E}$ as seen on Figure 5.

Step 1: Label the processors in the regions $\mathcal{C}$ and $\mathcal{D}$ by R and C as seen on Figure 3. During the execution, processors labeled by R and C will use a row bus or a column bus first, respectively. A message originated from a *C-labeled* or a *R-labeled* processor will be called *C-item* or *R-item*, respectively. Notice that if a processor is C-labeled then its pair is R-labeled.

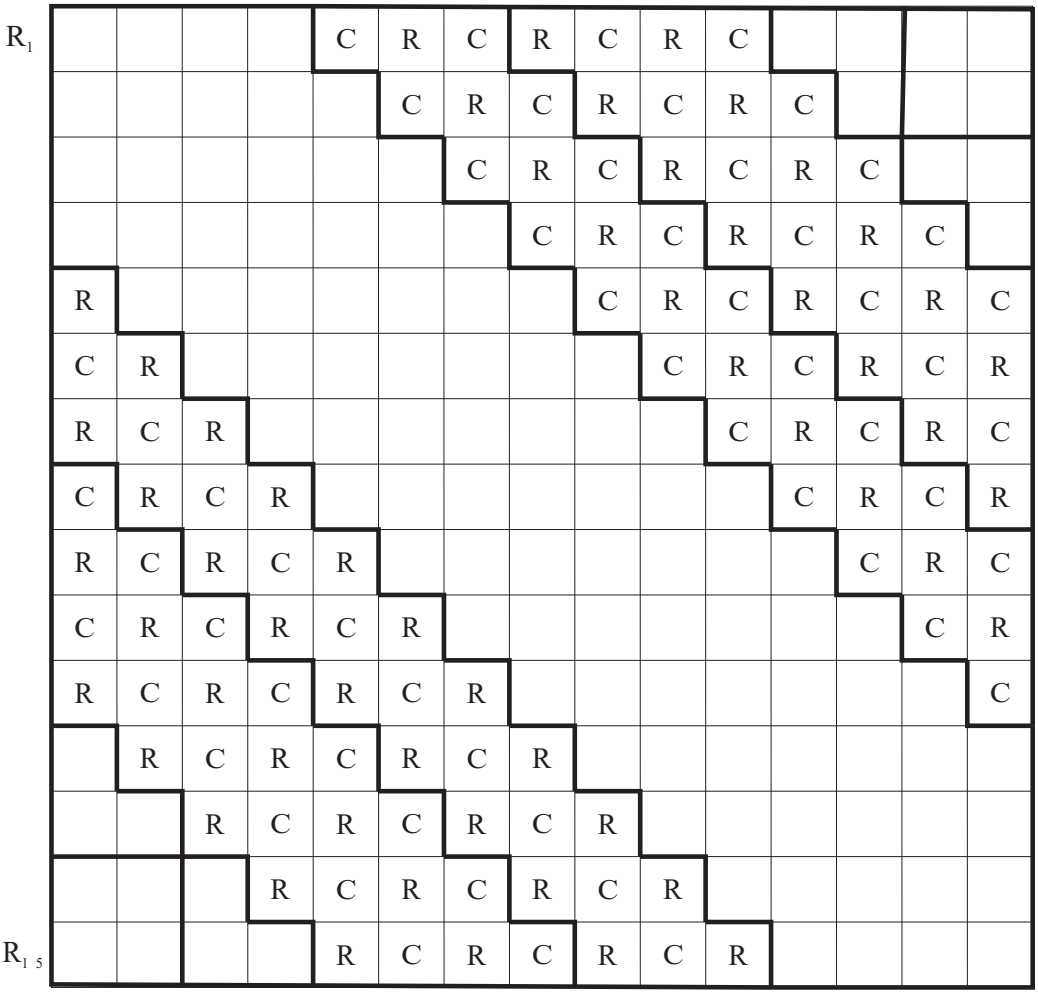

Figure 6: Schedule of the row and column buses on a 15×15 bused mesh

The next four steps (Step 2.1 - Step 2.4) are scheduled in parallel:

Step 2.1: Pairs in the region $\mathcal{E}$ walk by a greedy algorithm.

- Each element under the main diagonal first moves to its destination column, turns up, and moves step by step to its destination processor.
- Each element above the main diagonal first moves to its destination row, turns left and moves to its destination processor.

Step 2.2: Schedule the pairs of the regions $\mathcal{C}$ and $\mathcal{D}$ according to their labels assigned in Step 1. A row bus transfers messages belonging to a R-labeled processors using farthest first strategy, starting with the elements of the regions $\mathcal{D}$.

- Messages originated from the region $\mathcal{D}$ will walk along their destination column.
- Messages from the region $\mathcal{C}$ will use column buses,so their transfers require always only two steps.

The transfer of the items belonging to C-labeled processors is similar.

Step 2.3: Items of regions $\mathcal{B}$ in the positions $(1, n-z+1)$, (z, n), $(n-z+1, 1)$, and (n, z) walk to the direction of the center until they arrive at positions $(1, y+1), (n-y, n), (y+1, 1)$, and $(n, n-y)$, respectively.

Step 2.4: Schedule the newly arrived items (originated from the regions $\mathcal{B}$) into the region $\mathcal{C}$ to the corresponding row and column buses when the buses become available. The items have been moved up or down will be scheduled to row buses, the items moving left or right will be scheduled to column buses. Whenever such an item is transferred by a bus, we always move a new item to its former position to become transferable in the next step for the bus. After the first bus transfer, these items will wait for the second bus transfer in the processor buffers. To shorten the processor queues these items even can move in the direction of their destinations.

Step 3: When the above steps are finished, the algorithm transfers the items of regions $\mathcal{A}$ and those ones which are waiting in a buffer from Step 3.4 using row and column buses. Since they do not require common buses, the transfer of these two groups can be done in parallel.

3.2. Analysis of LBA

Lemma 3.1. *Items of the region $\mathcal{E}$ can be routed to their destination in at most $\frac{n}{2}$ steps.*

Proof. Since $\max_{p_{i,j}\in E} |p_{i,j}| = 2(n-x-1)$, using the definition of x we get that

$$\max_{p_{i,j}\in E} |p_{i,j}| = 2(n-x-1) = 2\left\lfloor \frac{\lfloor \frac{n}{2} \rfloor}{2} \right\rfloor \leq \left\lfloor \frac{n}{2} \right\rfloor \leq \frac{n}{2}.$$

Since all messages turn at the main diagonal, no items can delay each other. ∎

Lemma 3.2. *Items of the region $\mathcal{D}$ can be routed to their destination in at most $\frac{n}{2}$ steps.*

Proof. We give the proof only for the R-items. Similar argument can be used for C-items. Step 2.2 uses farthest first strategy in the region $\mathcal{D}$ for the R-items. Consider an arbitrary row of the mesh. Order the R-items of the region $\mathcal{D}$ in this row by the distances from their pairs in decreasing order. The distance of the i-th item in this order from its pair is $2(n-y-i)$. By the schedule, the i-th item uses the bus in the i-th step, so for the total number of steps S required we get that

$$S = (n-y-i) + (i-1) + 1 = n - y = n - \left\lceil \frac{n}{2} \right\rceil \leq \frac{n}{2}.$$

∎

Consider now those rows from the region $\mathcal{C}$ which are indexed by $n-x+1, ..., x$, i.e. the central rows of the mesh and denote them by $R_{n-x+1}, ..., R_x$. Let $|R_i|$ be the total number of bus operations required to route all the R-items in row R_i, $n-x+1 \leq i \leq x$, including also those row bus operations which route the C-items destined to the row R_i in the region $\mathcal{C}$.

Lemma 3.3. $\max_i |R_i| < \frac{3n}{8} + 4$ *where* $n-x+1 \leq i \leq n-y$.

Proof. By a simple calculation we get the following formula for $|R_i|$.

$$\begin{aligned} |R_i| &= \left\lceil \frac{i-n+x}{2} \right\rceil + \left\lfloor \frac{x-y}{2} \right\rfloor + 2\left\lceil \frac{y-i+1}{2} \right\rceil \leq \\ &\leq \frac{i-n+x}{2} + \frac{x-y}{2} + y - i + 3 = \\ &= x + \frac{y}{2} - \frac{n}{2} - \frac{i}{2} + 3. \end{aligned}$$

From this it follows that

$$\begin{aligned}
\max_i |R_i| &= x + \frac{y}{2} - \frac{n}{2} - \frac{n-x+1}{2} + 3 = \\
&= \frac{3x}{2} + \frac{y}{2} - n + \frac{5}{2} = \\
&= \frac{3\left(n - \left\lfloor \frac{\lfloor \frac{n}{2} \rfloor}{2} \right\rfloor - 1\right) + \lceil \frac{n}{2} \rceil - 2n}{2} + \frac{5}{2} \leq \\
&\leq \frac{n - \frac{3\lfloor \frac{n}{2} \rfloor}{2} + \frac{n}{2} + 1}{2} + \frac{5}{2} \leq \\
&\leq \frac{n - \frac{3n}{4} + \frac{n}{2} + \frac{5}{2}}{2} + \frac{5}{2} < \\
&< \frac{3n}{8} + 4.
\end{aligned}$$

∎

Lemma 3.4. $|R_{n-i+1}| \leq |R_i| + 1$ *where* $n - x + 1 \leq i \leq n - y$. *If* n *is odd then* $|R_{n-y+1}| = |R_{n-y}|$

Proof.

$$\begin{aligned}
|R_{n-i+1}| &= \left\lfloor \frac{i-n+x}{2} \right\rfloor + \left\lceil \frac{x-y}{2} \right\rceil + 2\left\lfloor \frac{y-i+1}{2} \right\rfloor \leq \\
&\leq \left\lceil \frac{i-n+x}{2} \right\rceil + \left\lceil \frac{x-y}{2} \right\rceil + 2\left\lceil \frac{y-i+1}{2} \right\rceil \leq |R_i| + 1
\end{aligned}$$

If n is odd then

$$|R_{n-y+1}| = \left\lfloor \frac{x-y}{2} \right\rfloor + \left\lceil \frac{x-y}{2} \right\rceil + 2 = |R_{n-y}|$$

∎

Corollary 3.1. $\max_i |R_i| < \frac{3n}{8} + 5$ *where* $n - x + 1 \leq i \leq x$.

Lemma 3.5. $|R_{n-y}| > x - y$.

Proof.

$$|R_{n-y}| = \left\lceil \frac{x-y}{2} \right\rceil + \left\lfloor \frac{x-y}{2} \right\rfloor + 2 \left\lceil \frac{2y-n+1}{2} \right\rceil > x - y.$$

∎

Corollary 3.2. *The items at positions* $(1, n-z+1), (z, n), (n-z+1, 1), (n, z)$ *arrive at positions* $(1, y+1), (n-y, n), (y+1, 1), (n, n-y)$ *in Step 2.3 of LBA before the row and column buses in rows* R_{n-y} *and* R_{y+1} *and in columns* C_{y+1} *and* C_{n-y} *finish their task defined in Step 2.2.*

Lemma 3.6. *Step 2.4 needs at most* $\frac{3n}{8} + 6$ *steps.*

Proof. From Corollary (3.2) it follows that the buses in rows R_{n-y-i}, where $1 \leq i \leq z - u$ can continue their work with Step 2.4 immediately after finishing Step 2.2. We calculate the total number of operations defined in Steps 2.2 and 2.4 for the buses in rows R_{n-y-i}. Denote again this number by $|R_{n-y-i}|$. We get the following:

$$\begin{aligned} |R_{n-y-i}| &= \left\lceil \frac{x-y-i}{2} \right\rceil + \left\lceil \frac{x-y}{2} \right\rceil + 2 \left\lceil \frac{2y-n+i+1}{2} \right\rceil \\ &+ z - u - i + 1 \leq \\ &\leq x + y + z - n - u - \frac{i}{2} + 5. \end{aligned}$$

From this

$$\max_i |R_{n-y-i}| = x + y + z - n - u + \frac{9}{2} = y - u + \frac{9}{2} < \frac{3n}{8} + 6. \qquad (5)$$

From Corollary (3.1) and (5) the statement of the lemma follows.

∎

Notice that when Step 2.4 is finished, then all the items in the region $\mathcal{C}$ have arrived at their pair.

Lemma 3.7. *Step 2.2 finishes in at most* $\frac{n}{8} + 3$ *steps.*

Proof. Since $u < \frac{n}{8} + 2$, the items in the region $\mathcal{A}$ and the waiting items from Step 2.4 can be routed in at most $\frac{n}{8} + 3$ steps using bus operations.

∎

Combining the statements of Lemmas (3.1),(3.2),(3.6),(3.7) we get the following theorem.

Theorem 3.1. *Algorithm LBA solves the matrix transpose problem in less than* $\frac{n}{2} + 9$ *steps.*

The next table shows on which step the specific elements arrive at their destinations on a 10x10 mesh. The second code presents the way of transfer in the last step, which can be walking (W), row bus (R) and column bus(C).

0	2W	4W	5W	5W	5C	5R	5C	4R	4C
4W	0	2W	5W	5W	5W	3C	5R	4C	4R
4W	4W	0	2W	5W	5W	5W	3C	3R	4R
5W	4W	4W	0	2W	5W	4W	5W	5C	4R
5W	5W	4W	4W	0	2W	4W	4W	5W	5C
5R	5W	5W	4W	4W	0	4W	4W	4W	5W
5C	3R	5W	4W	4W	4W	0	4W	4W	4W
5R	5C	3R	5W	4W	4W	2W	0	4W	4W
4C	4R	3C	5R	5W	4W	4W	2W	0	4W
4R	4C	4C	4C	5R	5W	4W	4W	2W	0

Finally, we show some values of the upper bound on the required steps and the corresponding lower bounds:

n	10	100	500	1000	4000
LB	5	45	226	451	1803
MTB	14	59	259	509	2009
Ratio	2.8	1.3111	1.1460	1.1286	1.1142

4. Lower Bound in 3D

Consider now the $3D\ MTP$ on a *3RCB–mesh* architecture, and investigate the transpose $(i, j, k) \to (j, k, i)$ for each possible triplet (i, j, k), where $1 \leq i, j, k \leq n$. Because of symmetry the following analysis remain valid for the other non–trivial transpose as well. Let n be the number of processors in each directions. By definition the distance of the processor (i, j, k) from its transpose (j, k, i) is

$$|i - j| + |j - k| + |k - i| \,. \tag{6}$$

It is easy to see that

$$|i-j|+|j-k|+|k-i|=2t \tag{7}$$

for some $0 \leq t \leq n-1$, and $t=0$ iff the tree indices in the triplets are equal.

Lemma 4.1. *The number of triplets for which the sum (6) is equl to* $2t, 1 \leq t \leq n-1$ *is* $6t\,(n-t)$.

Proof. Consider an arbitrary triplet $(i,j,k), 1 \leq i,j,k \leq n$. Let

$$m = \min(i,j,k)$$

and

$$M = \max(i,j,k).$$

Then

$$|i-j|+|j-k|+|k-i| = 2(M-m),$$

Let us count the number of different triplets with maximum M and minimum m and $t = M-m$ for all t, $1 \leq t \leq n-1$. We can choose M and m with this property by $n-t$ different ways. The third component of the triplet can be chosen by t different ways. We can permute these triplets 3! ways, so the statement of the lemma is true.

∎

Theorem 4.1. *Let A be an arbitrary algorithm and let $T_A^B(1,n)$ be the number of steps needed for A to solve the* MTP *on a 3–dimensional $n \times n \times n$ 3RCB–mesh architecture. Then*

$$\lim_{n\to\infty} \frac{T_A^B(1,n)}{n} > 0.45. \tag{8}$$

Proof. The idea is similar to the one we used in Theorem 2.1. First we classify the processors into groups, depending on the distances between a processor and its transpose. From Lemma 4.1 it follows that we get n different groups, each of them having the different values of m. Let us denote the groups by $\mathcal{A}_0, ..., \mathcal{A}_{n-1}$ and the corresponding number of processors by $P(\mathcal{A}_0), ..., P(\mathcal{A}_{n-1})$. Let us choose x so that $0 < x < \frac{n}{2}$, and let $y = 2x$.
As before, the items belonging to close processors will walk, some others with farther transposes will use bus once, and those ones which belong to processor pairs being far from each other will use buses twice. Suppose that

- items in the groups $P(\mathcal{A}_i)$, $0 \leq i \leq x$ will walk,
- items in the groups $P(\mathcal{A}_i)$, $x < i \leq y$ use buses ones, and
- items in the groups $P(\mathcal{A}_i)$, $y < i$ use buses twice.

Based on Lemma 4.1 the total number O_B of bus operations required is

$$O_B = \sum_{i=x+1}^{y} P(\mathcal{A}_i) + 2 \sum_{i=y+1}^{n-1} P(\mathcal{A}_i) = 6 \sum_{i=x+1}^{2x} i(n-i) + 12 \sum_{i=2x+1}^{n-1} i(n-i).$$

Because the total number of buses is $3n^2$, the O_B bus operations requires at least

$$T_A^B(1, n, x) = \frac{O_B}{3n^2} = \frac{2n^3 - n\left(15x^2 + 9x + 2\right) + 3x(6x^2 + 5x + 1)}{3n^2}$$

steps. On the other side, the longest walk in the first group is dominant for the number of steps of any algorithm A, so

$$T_A^B(1, n, x) = 2x. \tag{9}$$

We have to minimize the length of the longest walk while we try to load the buses uniformly. So, we need to solve the equation

$$\frac{2n^3 - n\left(15x^2 + 9x + 2\right) + 3x(6x^2 + 5x + 1)}{3n^2} = 2x. \tag{10}$$

Equation (10) has a root between $0.225n$ and $0.25n$, so the proof is completed.

∎

5. Conclusions

In this paper we considered the matrix transposition problem for parallel machines, and we proved that using meshes with buses one can expect better results than for those architectures where buses are not available. Although the gap is very small between the lower and upper bound, we are not able to narrow it, so the question remains open for the tight bound. Since we analyzed only the case of $1 - 1$, the $k - k$, $(k \geq 2)$ version is also unsolved for those meshes which can use row and column buses.

References

[1] J. Békési, G. Galambos, P. Hajnal: Analysis of permutation routing algorithms. EJOR, 125(2), pp. 249-256, 2000.

[2] A. Borodin, J. E. Hopcroft: Routing, merging and sorting on parallel models of computation. J. Comput. System Sci., 30 (1985), pp. 130-145.

[3] Y. K. Chan and V. C. Koo: An Introduction to synthetic aperture radar (SAR).Progress In Electromagnetics Research B, Vol. 2, 27–60, 2008

[4] S. Cheung, F.C.M. Lau: A lower bound for permutation routing on two-dimensional bused meshes. Inf. Proc. Lett. 45(1993), pp. 225-228.

[5] K.S. Ding, C.T. Ho, J.J. Tsay: Matrix Transpose on Meshes with Wormhole and XY Routing. Proc. 6th Symposium on Parallel and Distributed Processing, pp. 656-663, IEEE, 1994.

[6] S. Fujita and M. Yamashita: Fast gossiping on mesh-bus computers. IEEE Trans. on Computers, 45(2002), pp. 1326-1330).

[7] A. Grama, A. Gupta, G. Karypis, V. Kumar: Introduction to Parallel Computing. Pearson, 2003. ISBN: 978-0201648652.

[8] K. Iwama, E. Miyano and Y. Kambayashi: Routing problems on mesh of buses, J. Algorithms, 20(1996), pp. 613-631.

[9] K. Iwama, S. Tajima, E. Miyano, and H. Tamaki: Randomized Routing Algorithms on the Two-Dimensional Mesh of Buses. 4th Annual International Conference, COCOO, 2007.

[10] M. Kaufmann, U. Meyer, J.F. Sibeyn: Matrix Transpose on Meshes: Theory and Practice. Computers and Artificial Intelligence 16, pp. 107-140, 1997.

[11] M. Kaufmann, U. Meyer, J.F. Sibeyn: Routing on meshes with buses.Algorithmica 18(1997), pp. 417-444.

[12] M. Kunde: Routing and Sorting on mesh-connected arrays. Proc. 3rd Aegean Workshop on Computing:VLSI Algorithms and Architectures. LNCS, pp. 411-422, 1988.

[13] T. Leighton: Average case analysis of greedy rooting algorithms on arrays. in Proc. 1990 ACM Symp. on Parallel Algorithms and Architectures, pp. 2-10.

[14] T. Nesson and S. L. Johnson: ROMM routing on mesh and torus networks. Proc. 7th Symp.on Parallel Algorithms and Architectures, ACM, pp. 275-287, 1995.

[15] B. Parhami: Introduction to Parallel Processing, Algorithms and Architectures. Series in Computer Science, Kluwer Academic Publishers, ISBN: 978-0-306-45970-2, 2002.

[16] Algorithms and Theory of Computation Handbook, CRC Press LLC, Vreda Pieterse and Paul E. Black, eds. 1999.

[17] A.A. Ravankar, and S. Sedukhin: Mesh-of thory:A novel interconnection network for frontal plane cellular processors. 2013 International Conference on Computing. Networking and Communication (ICNC), pp. 281-284.

[18] A.A. Ravankar, and S. Sedukhin: An $O(n)$ time-complexity matrix transpose on torus array processors. In ICNC, 2011, pp. 242-247.

[19] C.P. Schnorr, A. Schamir: An optimal sorting algorithm for mesh connected computers. Proc. 18th Ann. ACM Symp. on Theory of Computing, pp. 255-263, 1986.

[20] S. G. Sedukhin: An $O(n)$ time-complexity matrix transpose on torus array processor. Second International Conference on Networking and Computing, ICNC 2011, Osaka, Japan

[21] S.G. Sedukin and M. Paprzycki: Generalizing matrix multiplication for efficient computations on modern computers, in Parallel Processing and Applied Mathematics (LNCS), vol. 7203, pp. 225-234, 2012.

[22] Q.F. Stout and B. Wagar: Intensive hypercube communication: Pearranged communication in link-bound machines. J. Parallel and Distrib. Computing, pp. 167-181, 1990.

[23] Texas Instruments. Multicore Fixed and Floating-Point Digital Signal Processor, TMS320C6678. http://www.ti.com/lit/ds/symlink/tms320c6678.pdf

[24] J-J. Tsay, K-S. Ding, and W-T. Wang: Optimal Algorithm for Matrix Transpose on Wormhole-Switched Meshes. J. of Inf. Science and Eng., 19(2003), pp. 167-177.