J. Parallel Distrib. Comput. 96 (2016) 194–201

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

# Matrix transpose on meshes with buses☆

József Békési *, Gábor Galambos

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

## HIGHLIGHTS

- We analyze the matrix transpose problem for 2- and 3-dimensional mesh architectures with row- and column-buses.
- We give a lower bound of approximately 0.45n for the number of steps required by any matrix transpose algorithm on an n x n mesh with buses.
- We present an algorithm which solves this problem in less than $0.5n + 9$ steps.

## ARTICLE INFO

*Article history:*
Received 29 September 2015
Received in revised form
12 May 2016
Accepted 23 May 2016
Available online 3 June 2016

*Keywords:*
Algorithm analysis
Mesh architecture
Matrix transpose

## 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 analyzed deeper than ever before. A good architecture or a more efficient algorithm may decrease the processing time strongly in a parallel computational environment.

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 e.g. [8,14,17,22].

Routing, sorting, merging, and matrix transpose are the problems that were investigated already in the early ages of parallel computation (see e.g. [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 accommodated to a given architecture. The effectiveness of certain algorithms were investigated extensively, see e.g. [4,6,9,13,18,21,20].

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

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

* Corresponding author.
*E-mail addresses:* bekesi@jgypk.szte.hu (J. Békési), galambos@jgypk.szte.hu (G. Galambos).

http://dx.doi.org/10.1016/j.jpdc.2016.05.015
0743-7315/

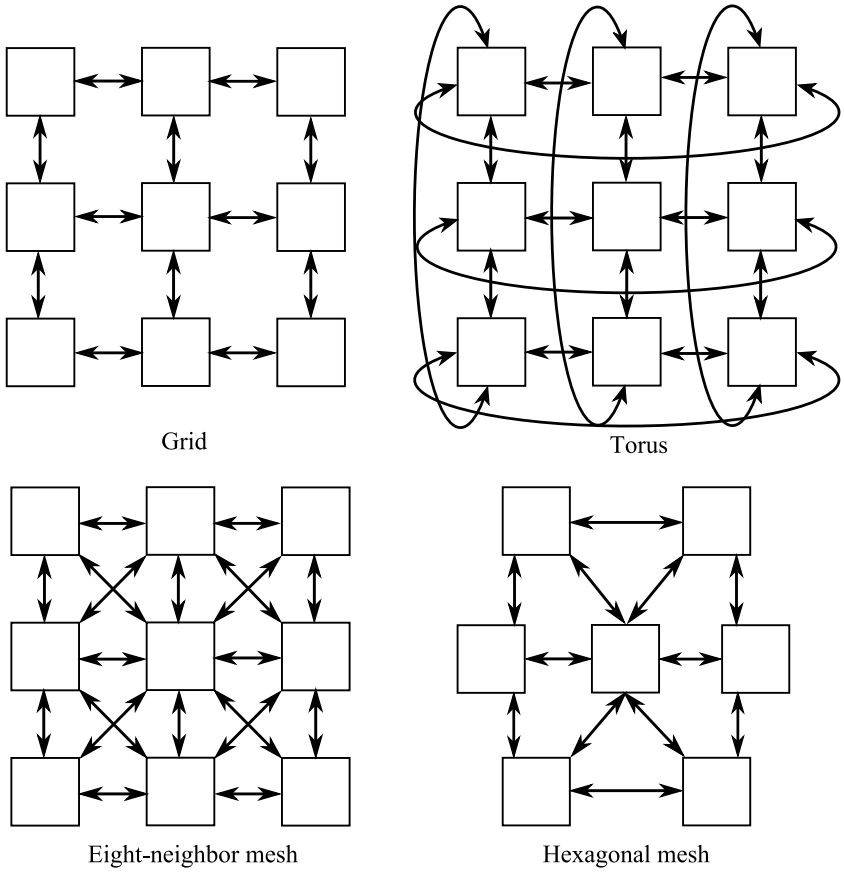


**Fig. 1.** Some mesh architectures.

line with its neighbors. In higher dimensions (2D, 3D) processors form an array, and they are connected by communication links. Fig. 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 [7,15,16]).

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.

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 wormhole 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.

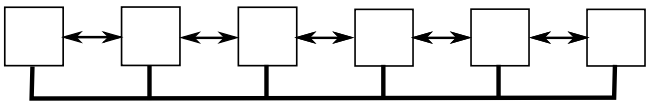

**Fig. 2.** 1D mesh with bus.

To speed up the communication between two far processors, it is possible to use buses. Buses can be used in 1D meshes (see Fig. 2.) and for 2D meshes as well. We show different bus-configurations in Fig. 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*.

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