New Parallel Matrix Multiplication Algorithms for Wormhole-Routed All-Port 2D/3D Torus Networks

Cesur Baransel[1], Kayhan İmre[2], and Harun Artuner[2]
[1]Saltus Yazılım Ltd.
Hacettepe University Technopolis, Ankara, TURKEY
cesur@saltus.com.tr

[2]Hacettepe University,
Dept. of Computer Engineering, Ankara, TURKEY
{ki,artuner}@hacettepe.edu.tr

Abstract-*New matrix multiplication algorithms are proposed for massively parallel supercomputers with 2D/3D, all-port torus interconnection networks. The proposed algorithms are based on the traditional row-by-column multiplication matrix product model and employ a special routing pattern for better scalability. They compare favorably to the variants of Cannon's and DNS algorithms since they allow matrices of the same size to be multiplied on a higher number of processors due to lower data communication overhead.*

Keywords-*Fast Matrix Multiplication, Parallel Processing, Torus Interconnection Networks, 2D Torus, 3D Torus.*

1. Introduction

Matrix multiplication is one of the most prevalent operations in scientific computing and its effective parallelization is of utmost importance for being able to harness the processing power offered by massively parallel supercomputers. Matrix multiplication can be formulated according to two general classes of computational models. In the first class, the matrix product can be defined as a *row-by-column multiplication* and for $C=AB$, $C(i,j)$ is the dot product of the i^{th} row vector of A and j^{th} column vector of B. It is also possible to define the matrix product as *sum of a series of column-row products*. Assuming that A and B are of order $n \times n$, both definitions require the same number of multiplications, namely n^3. The second class is comprised of the algorithms[1] aimed to reduce the number of multiplications such as the algorithms proposed by Strassen [6] and Winograd [7]. In this paper, we propose new matrix multiplication algorithms for 2D/3D torus topology where matrix product is defined as a *row-by-column multiplication.* A special routing pattern for 2D/3D tori is integrated into the proposed multiplication algorithms for efficiently exploiting the available bandwidth to provide higher scalability. Torus has proved to be the most popular topology in industry over the years and modern massively parallel supercomputers such as IBM

[1] A review of these methods can be found in Chapter 47, Handbook of Linear Algebra [1].

Blue Gene®/L and CRAY XT3 employ 2D/3D torus interconnection networks to accommodate tens of thousands of processing elements (PE).

This paper is structured as follows. A short review of the previous work is provided in the next section. We will give the details of the proposed algorithms in Section 3. Section 4 provides the performance results. Paper ends with conclusions.

2. Previous Work

Assume two matrices *A* and *B*, both of size $n \times n$, are mapped[2] onto a 2D array of p processing elements (PEs), arranged as a $\sqrt{p} \times \sqrt{p}$ torus with ($p<n^2$). Consequently, each PE stores a separate block of (n^2/p) entries from matrices A, B and C, where $C=AB$. Since, computing $C(i,j)$ requires the i^{th} row vector of A and j^{th} column vector of B, a simple parallel matrix multiplication algorithm can be defined as follows;

1. perform an all-to-all broadcast within each row,
2. perform an all-to-all broadcast within each column,
3. perform required *multiply-and-add* operations.

Under all-port model, column and row broadcasts can be executed in parallel. Broadcasting a block within a row or column takes $(\log_3 \sqrt{p})$ steps to complete and there are $\sqrt{p}$ blocks to broadcast within each row or column. After the broadcasting phase is completed, each PE will have $\left(n^2/\sqrt{p}\right)$ matrix entries and will perform (n^3/p) multiply-and-add operations. Assuming a multiply-and-add operation takes unit time, the parallel cost T_{par} of the algorithm is given in Equation (1),

$$T_{par} = \sqrt{p} \log_3 \sqrt{p} \; (t_s + \frac{n^2}{p} t_w) + \frac{n^3}{p} \tag{1}$$

where t_s and t_w represent the message startup time and the time to transmit a single matrix entry, respectively. Note that it is possible to reduce the cost by properly interleaving and overlapping broadcast and multiply-and-add operations. However, the above algorithm is not memory efficient since it consumes $\sqrt{p}$ times more space compared to the serial implementation.

Cannon proposed a memory-efficient matrix algorithm which takes $2\sqrt{p}$ steps to complete, including the initial and final alignment steps in [2]. Since this algorithm is explained elsewhere [3], its details will not be repeated here. Cannon's algorithm has the following phases on 2D torus;

1. the initial alignment; requires at most circular $(\sqrt{p}-1)$-shifts which can be completed in at most $\sqrt{p}/2$ steps, since a circular *q-shift* on a p node ring takes $min\{q, p-q\}$ steps.
2. $\sqrt{p}$ steps of two direct-neighbor shifts followed by a multiply-and-add operation,
3. the final alignment which also can be completed in at most $\sqrt{p}/2$ steps.

[2] Throughout the paper, we assume that the matrix multiplication operation is to be performed such that this initial mapping is preserved at the end of the operation.

The parallel cost T_{par} of Cannon's algorithm is given in Equation (2).

$$T_{par} = 2\sqrt{p}\ (t_s + \frac{n^2}{p} t_w) + \frac{n^3}{p} \tag{2}$$

Here, it is also possible to reduce the cost by properly overlapping transmission and multiply-and-add operations, assuming that the multiplication of two blocks can be completed by the time the next two blocks are received by the PE. In the next section, we will show that it is possible to complete matrix multiplication in $O(\log_5 \sqrt{p})$ time rather than $O(\sqrt{p})$, at the expense of longer messages.

DNS algorithm, proposed by Dekel, Nassimi and Sahni can be employed both in hypercube and 3D torus architectures. This algorithm can use up to n^3 processors to complete the matrix multiplication operation in *O(log n)* time. DNS algorithm has four phases on 3D torus assuming p processors are arranged into a $\sqrt[3]{p} \times \sqrt[3]{p} \times \sqrt[3]{p}$ cube.

1. Assume that the matrices to be multiplied (i.e., A and B) and the result matrix C is to be stored on the front face of the cube. There are $\sqrt[3]{p}$ planes of $\sqrt[3]{p} \times \sqrt[3]{p}$ processors in the cube. Copy column i of the matrix A into the column i of the i^{th} plane and row i of the matrix B into the row i of the i^{th} plane. Regardless of the number of processors, this phase is completed in a single step.
2. row-wise and column-wise propagation within each plane. This phase is completed in $(\log_3 \sqrt[3]{p})$ steps under all-port model since row-wise and column-wise propagation can be executed in parallel.
3. multiply,
4. reduce the result onto the front face of the cube in $(\log_3 \sqrt[3]{p})$ steps by adding relevant terms.

The parallel cost T_{par} of DNS algorithm is given in Equation (3).

$$T_{par} = 2\log_3 \sqrt[3]{p}\ (t_s + \frac{n^2}{p^{2/3}} t_w) + \frac{n^3}{p} \tag{3}$$

Note that, it is not possible to drop the (n^3/p) term from the cost since multiply and add operations are not to be interleaved and therefore it is not possible to overlap communication, multiply and add operations.

3. The Proposed Algorithms

In this section, we propose two algorithms for performing matrix multiplication on 2D torus architecture with up to n^2 processors and on 3D torus architecture with up to n^3 processors. The time complexities of both algorithms are logarithmic.

3.1 Matrix Multiplication on 2D Torus with up to n^2 Processors

Our 2D parallel matrix multiplication requires no alignment steps and completes in five phases. We introduce the algorithm assuming single matrix element per processor assign-

ment; its extension to matrix blocks is trivial. For a $\sqrt{p} \times \sqrt{p}$ processor array, we define $\sqrt{p}$ *concentration processors* (CP) with no two CP are being in the same row or column. First, the *CP(i,j)* gathers the elements of the i^{th} row of *A* and broadcasts it to the processors on the i^{th} row in two consecutive phases. Then, *CP(i,j)* gathers the elements of the j^{th} column of *B* and broadcasts it to the processors on the j^{th} column, also in two consecutive phases. Since, computing *C(i,j)* requires the i^{th} row vector of *A* and j^{th} column vector of *B*, all processors acquire the required data in four phases. The last phase is the *multiply-and-add* phase after which no data alignment is required.

The proposed algorithm differs from other proposals mainly in how row-wise and column-wise gathers and broadcasts are performed. Using the communication pattern given in Figure 1, it is possible to complete each gather or broadcast in $\left(\log_5 \sqrt{p}\right)$ steps on a wormhole-routed, all-port torus[3].

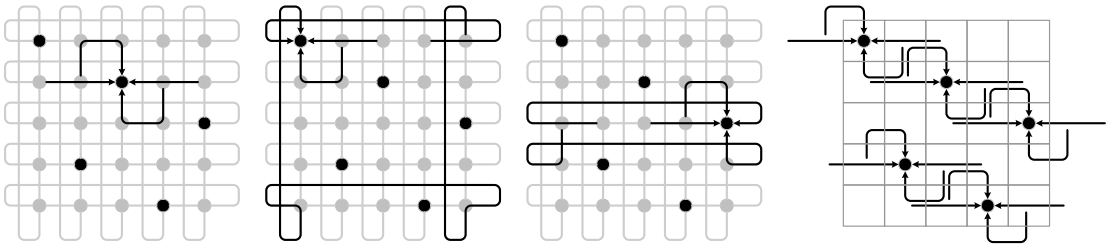
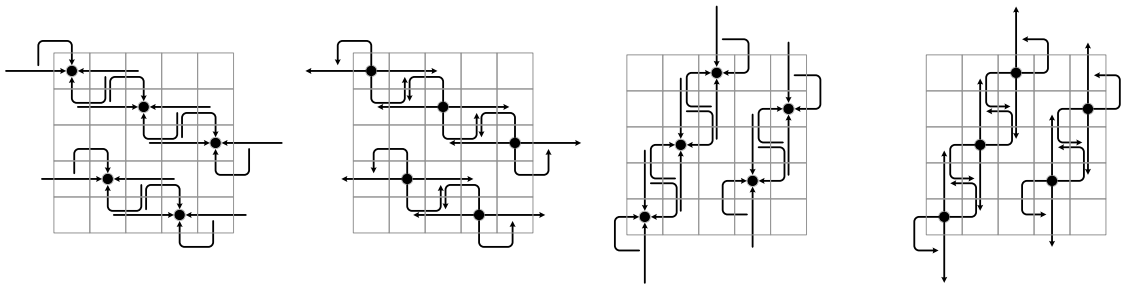


Figure 1. Gather/Broadcast Pattern

Although the proposed communication may seem somewhat complicated, the basic rule is quite simple on a *5×5* torus, *CP(i,j)* has two neighbors located 2 hops away and two immediate neighbors on row *i*. To communicate with the CP, the nodes located 2-hops away use the links within the row while immediate neighbors take a detour via row *(i-1)* and row *(i+1)*. Since detour links are arranged to lay on the opposite direction to the within-row links, the communication pattern is contention free. Column-wise communication is arranged similarly and consequently all CPs can perform row or column gathers or broadcasts in parallel. Matrix Multiplication on 5×5 torus, using this broadcast pattern is illustrated in Figure 2.



(a) (b) (c) (d)

Figure 2. Matrix Multiplication on 5×5 torus; (a) Row-wise Gather (b) Row-wise Broadcast (c) Column-wise Gather (d) Column-wise Broadcast; Local multiply-and-add phase is not shown;

[3] For a good introduction to routing in general and wormhole routing in particular, see [5].

The parallel cost T_{par} of the proposed algorithm is given in Equation (4).

$$T_{par} = 4\log_5 \sqrt{p}\ t_s + \left(\frac{5^{(1+\log_5 \sqrt{p})} - 1}{2} \right) \frac{n^2}{p} t_w + \frac{n^3}{p} \tag{4}$$

Here, it is not possible to drop the (n^3/p) term from the cost since multiply and add operations are not to be interleaved and therefore it is not possible to overlap communication, multiply and add operations. Also note that message lengths are different for gather and broadcast phases.

The given seed broadcast pattern can be recursively extended to the powers of 5 by replacing each node by a *5×5* torus. Figure 3 shows the communication pattern for a *25×25* torus. Other seeds are also be defined for *2×2, 3×3, 4×4* and *5×5* tori (Figure 4). These seed patterns can be used in combination to support virtually all practical matrix sizes [4].

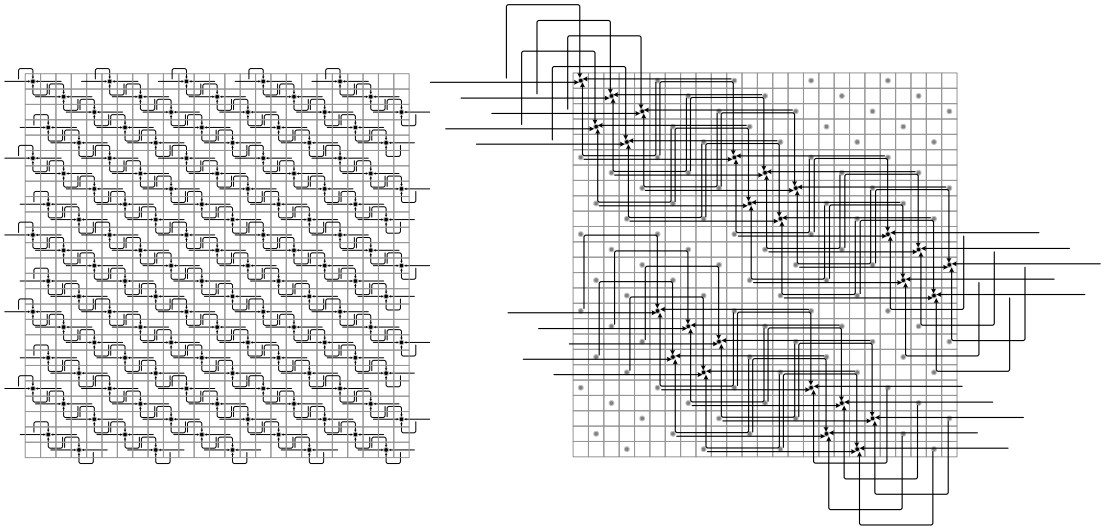


Figure 3. Communication pattern for *25×25* torus

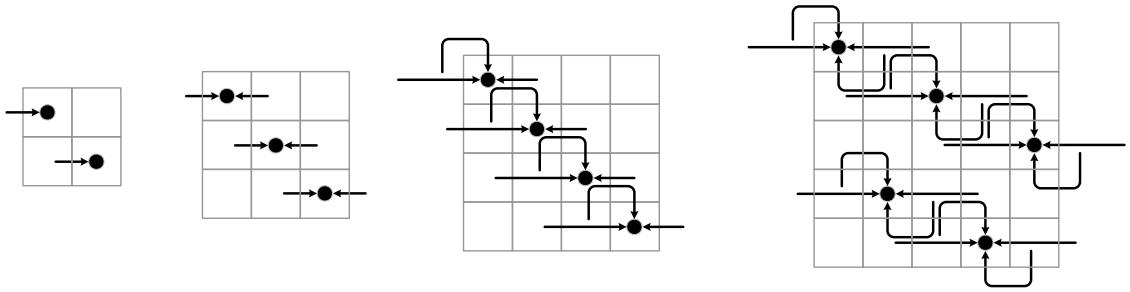


Figure 4. Seed communication patterns for *2×2, 3×3, 4×4* and *5×5* tori.

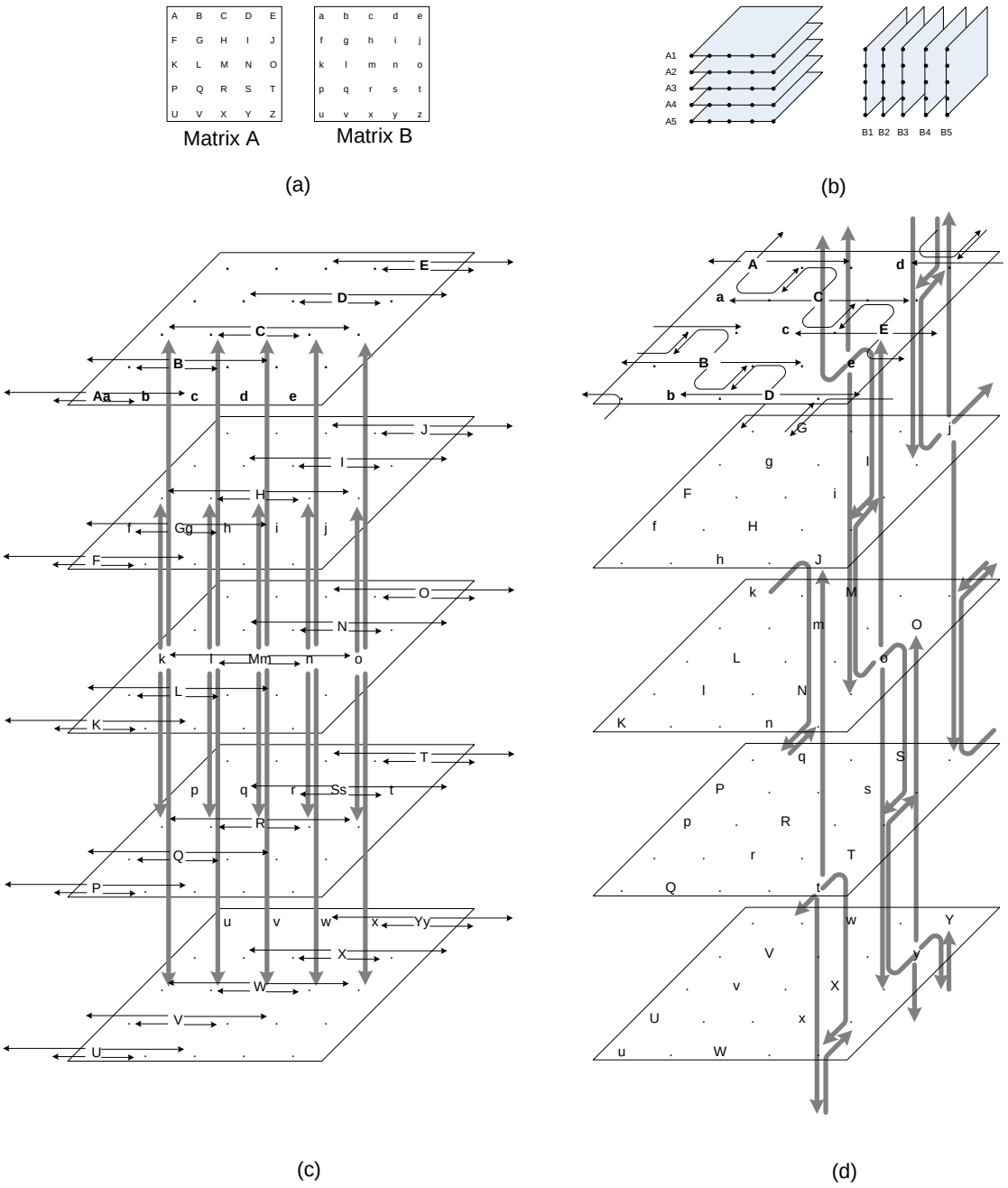


Figure 5.Communication patterns on a 5×5×5 torus for
(c) DNS algorithm (d) the proposed algorithm

3.2 Matrix Multiplication on 3D Torus with up to n^3 Processors

The proposed 3D multiplication algorithm is basically an extension of our 2D multiplication algorithm onto the third dimension. On 3D torus, each processing element has six links compared to those four on 2D torus, and proper use of these extra two links allow the broadcast of elements of matrices A and B on the third dimension to be completed in the same phase and in $O\left(\log_5 \sqrt[3]{p}\right)$ steps on a $\sqrt[3]{p} \times \sqrt[3]{p} \times \sqrt[3]{p}$ torus.

Figure 5 shows the communication patterns on a 5×5×5 torus for DNS algorithm and the proposed algorithm. In the figure, the elements of matrix A and matrix B are indicated with upper-case and lower-case letters, respectively. The elements of matrix A are broadcasted along the horizontal planes and the elements of matrix B along the vertical planes. The broadcast pattern of the proposed algorithm is more efficient compared to the broadcast pattern DNS algorithm since its time complexity is $O\left(\log_5 \sqrt[3]{p}\right)$ steps, rather than $\left(\log_3 \sqrt[3]{p}\right)$ of DNS on a $\sqrt[3]{p} \times \sqrt[3]{p} \times \sqrt[3]{p}$ torus. The parallel cost T_{par} of the proposed algorithm is given in Equation (5).

$$T_{par} = 2\left(1 + \log_5 \sqrt[3]{p}\right)\left(t_s + \frac{n^2}{p^{2/3}} t_w\right) + \frac{n^3}{p} \tag{5}$$

4. Performance Analysis

Performance results are provided in Table 1 and Table 2. In computing speedups, Strassen's algorithm with the cost of $n^{2.807}$ is taken as the best available serial implementation. The proposed 2D algorithm yields a speedup similar to Cannon's when the matrix block size is large (e.g., *753.76* vs. *726.31* when block size is *125* for *625×625* matrices). The proposed algorithm performs better as the block size gets smaller (e.g., 373.38 vs. *15,364.08* when block size is *1* for *625×625* matrices). The difference between the DNS and the proposed 3D algorithms is not as great compared to 2D case. However, the proposed algorithm is never slower than DNS and can provide up to *30%* more speedups in some cases. The results also indicate that as the t_s/t_w ratio of the systems gets smaller, the difference between the algorithms also gets somewhat smaller both in 2D and 3D cases, but not significantly.

5. Conclusions

In the last two decades, the number of processors in massively parallel supercomputers have been increased from tens of processors to tens of thousands of processors and, as progressively larger number of processors became available, the size of the matrix sub-blocks in matrix multiplication grew to be smaller for a given problem size. Consequently, new algorithms which can work efficiently with smaller blocks are required to exploit the processing power offered by the modern massively parallel processors. At the same time, torus interconnection networks gained wide-spread popularity in the industry. In this paper, we proposed a new parallel matrix multiplication algorithm for 2D and 3D torus architectures which performs better than the competitive algorithms especially as the size of the matrix sub-blocks gets smaller.

Matrix Size	p	Block Size (n^2/p)	t_s/t_w=150/1		t_s/t_w =450/1	
			Speed up for Cannon	Speed up for 2D Proposed	Speed up for Cannon	Speed up for 2D Proposed
25×25	25	25	4.80	5.50	1.77	3.08
	125	5	2.42	7.22	0.83	2.83
	625	1	1.11	6.52	0.37	2.28
125×125	25	625	9.38	8.92	9.21	8.80
	125	125	41.13	38.54	34.88	35.35
	625	25	87.91	130.92	32.39	92.95
	3125	5	44.39	272.63	15.12	132.13
	15625	1	20.38	343.84	6.82	131.78
625×625	25	15625	7.16	7.08	7.16	7.08
	125	3125	35.42	34.55	35.36	34.52
	625	625	171.89	163.68	168.80	162.77
	3125	125	753.76	726.31	639.12	704.53
	15625	25	1,610.86	2,793.86	593.47	2,444.93
	78125	5	813.35	8,085.38	277.08	5,456.26
	390625	1	373.38	15,364.08	125.01	7,507.73
3125×3125	25	390625	5.28	5.27	5.28	5.27
	125	78125	26.35	26.22	26.35	26.22
	625	15625	131.18	129.67	131.16	129.66
	3125	3125	649.05	632.92	647.96	632.73
	15625	625	3,149.85	3,003.45	3,093.27	2,998.43
	78125	125	13,812.49	13,452.24	11,711.77	13,335.56
	390625	25	29,518.61	54,001.23	10,875.28	51,917.16
	1953125	5	14,904.50	180,410.04	5,077.36	156,759.35
	9765625	1	6,842.06	463,313.24	2,290.80	323,880.05

Table 1. Performance Results for 2D Algorithm

Matrix Size	p	Block Size ($n^2/p^{2/3}$)	t_s/t_w=150/1		t_s/t_w =450/1	
			Speed up for DNS	Speed up for 3D Proposed	Speed up for DNS	Speed up for 3D Proposed
25×25	125	25	10.33	10.18	4.21	4.15
	15625	1	8.10	9.26	2.71	3.10
125×125	125	625	41.20	41.08	38.75	38.60
	15625	25	580.30	654.62	227.34	258.55
	1953125	1	519.97	636.21	174.17	213.13
625×625	125	15,625	34.97	34.95	34.95	34.93
	15625	625	3,365.34	3,475.96	3,064.21	3,192.53
	1953125	25	38,338.68	46,213.13	14,758.63	17,955.42
	244140625	1	36,673.50	46,641.31	12,282.96	15,622.93
3125×3125	125	390,625	26.28	26.28	26.28	26.28
	15625	15,625	3,132.52	3,153.27	3,129.40	3,150.51
	1953125	625	278,182.03	295,862.39	246,937.82	266,550.94
	244140625	25	2,746,617.80	3,443,838.20	1,047,073.45	1,324,553.15
	30517578125	1	2,7313,43.54	3,561,608.73	914,742.69	1,192,905.34

Table 2. Performance Results for 3D Algorithm

References

[1] D. A. Bini. Fast Matrix Multiplication. In: Leslie Hogben, Editor, *Handbook of Linear Algebra*, Chapman & Hall/CRC press, Boca Raton (2007) (Chapter 47).

[2] L. E. Cannon. *A cellular computer to implement the kalman filter algorithm*. Ph.D. Thesis, Montana State University, 1969.

[3] A. Gramma, A. Gupta, G. Karypis and V. Kumar. *Introduction to Parallel Computing*, Second Edition, Addison Wesley, 2003.

[4] K. M. İmre, C. Baransel and H. Artuner. Efficient and Scalable Routing Algorithms for Collective Communication Operations on 2D All-Port Torus Networks. Submitted for publication, 2010.

[5] L.M. Ni and P.K. McKinley. A survey of wormhole routing techniques in direct networks. *Computer* , vol.26, no.2, pp.62-76, Feb 1993.

[6] V. Strassen. Gaussian Elimination is not Optimal. Numer. Math. 13, p.354-356, 1969.

[7] S. Winograd. On multiplication of 2 × 2 matrices. *Linear Algebra and Its Applications*, 4:381–388, 1971.