BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **20**, No 6
Special Issue on New Developments in Scalable Computing
Sofia • 2020 Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.2478/cait-2020-0064

Performance Analysis of a Scalable Algorithm for 3D Linear Transforms on Supercomputer with Intel Processors/Co-Processors

Ivan Lirkov

Institute of Information and Communication, Technologies, Bulgarian Academy of Sciences, 1113 Sofia, Bulgaria
E-mail: ivan@parallel.bas.bg

***Abstract*:** *Practical realizations of 3D forward/inverse separable discrete transforms, such as Fourier transform, cosine/sine transform, etc. are frequently the principal limiters that prevent many practical applications from scaling to a large number of processors. Existing approaches, which are based primarily on 1D or 2D data decompositions, prevent the 3D transforms from effectively scaling to the maximum* (*possible/available*) *number of computer nodes. A highly scalable approach to realize forward/inverse 3D transforms has been proposed. It is based on a 3D decomposition of data and geared towards a torus network of computer nodes. The proposed algorithms requires compute-and-roll time-steps, where each step consists of an execution of multiple GEMM operations and concurrent movement of cubical data blocks between nearest neighbors. The aim of this paper is to present an experimental performance study of an implementation on high performance computer architecture.*

***Keywords*:** *3D linear transforms, parallel implementation, Intel processors/ co-processors.*

1. Introduction

Three-Dimensional (3D) Discrete Transforms (DT) such as Fourier transform, cosine/sine transform, etc., are known to play a fundamental role in many application areas, such as spectral analysis, digital filtering, signal and image processing, data compression, medical diagnostics, etc. Continuously increasing demands for high speed computing in a constantly increasing number of many real-world applications have stimulated the development of a number of "fast algorithms", such as the Fast Fourier Transform (FFT), characterized by dramatic reduction of arithmetic complexity. However, further reduction of execution time is possible only by using parallel implementation.

There exist three different approaches to parallel implementation of the 3D forward/inverse discrete transforms. Two of them are particularly well suited for the Fourier transform. The first one is the 1D or "slab" decomposition of the initial 3D data. In this approach, the data is divided into 2D slabs. The scalability of the slab-based approach is limited by the number of data elements along a single dimension of the 3D transform. The second approach is the 2D or "pencil" decomposition, of a 3D initial data, among a 2D array of computer nodes. This approach increases the maximum number of nodes that can be effectively used in computations.

The last approach is the 3D or "cube" decomposition, which was recently proposed in [1]. The 3D or "cubic" decomposition of an initial data among computer nodes, allows a 3D data "cube" to be assigned to each computer node. It is easy to realize that the theoretical scalability is further improved. In this approach, blocked GEMM-based algorithms are used to compute the basic one-dimensional N-size transform, not on a single but on the cyclically interconnected nodes of a 3D torus network. In this way, the proposed algorithm integrates local intra-node computation with a nearest-neighbour inter-node communication, at each step of the three-dimensional processing. It is important to observe that the proposed algorithm, with its 3D data decomposition, and the torus-oriented communication scheme, completely eliminates global communication. In addition, computation and local communication can be overlapped. Finally, note that in the considered approach, the 3D transform is represented as three chained sets of cubical tensor-by-matrix or matrix-by-tensor multiplications, which are executed in a 3D torus network of computer nodes by the fastest and extremely scalable orbital algorithms.

The main contribution of this paper is to experimentally evaluate the performance of the latter algorithm. To do this, we have implemented overlapping of computation and communication for the 3D data decomposition and used GEMM kernels available. The experimental performance of the 3D Discrete Cosine Transform (DCT) and Discrete Fourier Transform (DFT), with the 3D data decomposition, has been evaluated on a supercomputer cluster.

2. 3D separable transform

Let $X = [x(n_1, n_2, n_3)]$, $0 \le n_1, n_2, n_3 < N$, be an $N \times N \times N$ cubical grid of input data. A separable forward 3D transform of X is another cubical grid of an $N \times N \times N$ data $Y = [y(k_1, k_2, k_3)]$, where for all $0 \le k_1, k_2, k_3 < N$:

$$y(k_1,k_2,k_3) = \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} \sum_{n_3=0}^{N-1} x(n_1,n_2,n_3) \times c(n_1,k_1) \times c(n_2,k_2) \times c(n_3,k_3).$$

A separable inverse, or backward, 3D transform of a tensor $Y = [y(k_1, k_2, k_3)]$ is expressed as:

$$x(n_1,n_2,n_3) = \sum_{k_1=0}^{N-1} \sum_{k_2=0}^{N-1} \sum_{k_3=0}^{N-1} y(k_1,k_2,k_3) \times c(n_1,k_1) \times c(n_2,k_2) \times c(n_3,k_3).$$

The 3D transform can be split into three data dependent sets of 1D transforms. At the first stage, the 1D transform of $x(n_1, n_2, :)$ is performed for all (n_1, n_2) pairs, as a block tensor-by-matrix multiplication. At the second stage, the 1D transform of

$v(:, n_2, k_3)$ is implemented for all (n_2, k_3) pairs, as the second block tensor-by-matrix multiplication. At the third stage, the 1D transform of $w(k_1, :, k_3)$ is implemented for all (k_1, k_3) pairs, as the third block tensor-by-matrix multiplication.

3. Parallel implementation of the algorithm

The parallel implementation of the proposed algorithm is described in [1]. Our implementation of the parallel algorithm is described in [2]. It should be noted that our implementation is a modification of the parallel algorithms proposed in [1]. The main differences between our implementation and the original algorithm are:

1. The implemented parallel algorithm works only for the 3D DCT and the 3D DFT;
2. The proposed implementation uses additional arrays to store elements of the coefficient matrix C. In the case of the DCT, we use one array with $4N$ elements; while for the DFT two arrays with N elements each. In this way, we avoid rolling the coefficient matrix. In other words, we simplify the communication, while paying the price of somewhat increasing the total memory utilization.

Since the tensor-by-matrix, or the matrix-by-tensor, multiplications can be expressed as the set of matrix-by-matrix multiplications, we can use existing GEMM subroutines, from the BLAS library [3], to compute the 3D transform.

There exist two possible ways to compute the tensor-by-matrix multiplication on computers with multi-core processors. The first one is to use a multi-threaded library, such as the Intel Math Kernel Library (MKL, see [4]). Here, each slice of the tensor is computed by multiple threads. The other possible approach is to use OpenMP. In the current implementation, we have linked our code to the multi-threaded library for the parallelization on a single (multi-core) node of the computer system.

4. Experimental results

A portable parallel code was designed and implemented in C. The parallelization was based on the MPI standard [5, 6]. In the code, we used the BLAS subroutines SGEMM, DGEMM, CGEMM, and ZGEMM to perform matrix-by-matrix multiplication. In order to obtain a better mapping of the processors to the physical interconnect topology of computers actually used in experiments, functions MPI_Dims_create and MPI_Cart_create were used to create a logical 3D Cartesian grid of processors.

The parallel code has been tested on the following system: the supercomputer Avitohol at IICT-BAS (see [7] for details). The computer system Avitohol is constructed with HP Cluster Platform SL250S GEN8. It has 150 servers, and two 8-core Intel Xeon E5-2650 v2 8C processors and two Intel Xeon Phi 7120P coprocessors per node. Each processor runs at 2.6 GHz. Processors within each node share 64 GB of memory. Each Intel Xeon Phi has 61 cores, runs at 1.238 GHz, and has 16 GB of memory. Nodes are interconnected with a high-speed InfiniBand FDR network. We used the Intel C compiler, and compiled the code using the following

options: “-O3 -qopenmp” for the processors and “-O3 -qopenmp -mmic” for the coprocessors. To use the BLAS subroutines, we linked our code to the optimized multi-threaded Intel MKL library. Intel MPI was used to execute the code on the Avitohol computer system.

In our experiments, times have been collected using the MPI provided timer, and we report the best results from multiple runs. In the following tables, we report the elapsed (wall-clock) time in seconds.

Table 1. Execution time (in seconds) for the 3D discrete cosine transform using only processors of the Avitohol

N	Number of nodes						
	1	2	4	8	16	32	64
Single precision							
Forward transform							
200	0.0760	0.0539	0.0377	0.0383	0.0229	0.0382	0.0430
400	0.7918	0.4543	0.2868	0.1728	0.1300	0.0953	0.0858
600	3.5642	2.0189	1.1939	0.6426	0.4223	0.2949	0.2033
800	10.5593	5.7550	3.1507	1.7097	1.0306	0.6472	0.4305
1000	24.4799	13.2777	7.2777	3.9019	2.3329	1.4249	0.8147
1200	49.9908	26.1017	14.1655	7.5324	4.4960	2.6249	1.4916
1600		78.8522	41.6059	21.7961	12.3199	6.8936	3.7470
2000			99.5842	50.8931	28.2158	15.5731	8.6936
Backward transform							
200	0.0578	0.0365	0.0216	0.0132	0.0093	0.0109	0.0068
400	0.7034	0.4019	0.2266	0.1300	0.0880	0.0568	0.0473
600	3.3402	1.8719	1.0193	0.5534	0.3539	0.2301	0.1477
800	9.9892	5.3489	2.8409	1.5194	0.8993	0.5489	0.3376
1000	23.2535	12.5482	6.6254	3.5717	2.1258	1.2804	0.7387
1200	47.9282	24.7840	13.0700	6.9643	4.1524	2.3548	1.3040
1600		75.7462	38.6973	20.3578	11.5351	6.5202	3.6148
2000			94.4404	47.9034	26.8596	14.7392	8.1383
Double precision							
Forward transform							
200	0.1449	0.0930	0.0646	0.0482	0.0396	0.0451	0.0540
400	1.6184	0.9230	0.5462	0.3293	0.2197	0.1373	0.1230
600	7.2401	3.9339	2.2233	1.2521	0.7916	0.4978	0.3228
800	21.3262	11.4433	6.3369	3.4622	2.1033	1.2351	0.7732
1000	51.5733	26.4074	14.4434	7.7706	4.6434	2.6952	1.6234
1200	110.0560	52.9195	28.4283	15.1895	8.6458	4.9744	2.7664
1600			87.1884	44.3340	24.5492	13.5643	7.7001
2000					56.4890	30.6572	16.9175
Backward transform							
200	0.1190	0.0698	0.0393	0.0218	0.0156	0.0119	0.0104
400	1.4522	0.8111	0.4456	0.2619	0.1689	0.1058	0.0713
600	6.7475	3.6326	1.9578	1.0926	0.6862	0.4143	0.2865
800	20.2844	10.7186	5.6413	3.0990	1.8295	1.0747	0.6505
1000	49.1480	25.0125	13.0928	7.0599	4.1540	2.4528	1.4205
1200	101.8070	50.5865	26.1044	13.9867	7.9348	4.5263	2.5333
1600			82.1612	41.9168	23.0593	12.7839	7.2690
2000					53.7557	29.2964	16.0420

Tables 1 and 2 show the results collected on the Avitohol using only Intel Xeon processors. The main memory on one node is 64 GB and allows execution of the DCT algorithm for $N = 200, 400, ..., 1200$. The DFT algorithm requires more memory and the limit for the execution on one node is $N = 1000$. For larger problems we used the distributed memory on more nodes. For example, for DFT with $N = 2000$ the algorithm requires the memory of at least 16 nodes. The reported execution time for $N = 200$ shows that the problem is small and can be executed on one node (no need for parallelization). Here, there is no significant improvement from using two or more nodes. However, for the problems of size $N = 800, 1000, 1200$ a significant performance gain can be observed. Moreover, for 3D DFT with double precision super-linear speed-up is observed for $N = 1000$ on up to 8 nodes.

Table 2. Execution time (in seconds) for the 3D discrete Fourier transform using only processors of the Avitohol

N	Number of nodes						
	1	2	4	8	16	32	64
Single precision							
Forward transform							
200	0.2067	0.1258	0.0897	0.0646	0.0493	0.0588	0.0886
400	2.6787	1.4327	0.8654	0.4731	0.3066	0.1958	0.1228
600	12.8205	6.6810	3.6997	1.8912	1.1365	0.6946	0.4571
800	37.6882	19.9276	10.5935	5.5964	3.1616	1.8203	1.1089
1000	89.3692	46.4069	24.6894	12.9602	7.2873	4.1502	2.3694
1200		96.4621	50.2716	26.4370	14.3301	8.0371	4.2275
1600			152.9335	77.4458	41.4072	22.2047	12.1978
2000					96.3475	51.1302	27.6908
Backward transform							
200	0.1776	0.1025	0.0579	0.0325	0.0214	0.0215	0.0158
400	2.5317	1.3308	0.7188	0.3885	0.2352	0.1483	0.1085
600	12.4051	6.5596	3.4169	1.7331	1.0117	0.5914	0.3761
800	36.8481	19.4619	10.0323	5.2546	2.8643	1.6462	0.8828
1000	87.4686	45.3237	23.2665	12.3150	6.8326	3.7944	2.2520
1200		93.3873	48.3290	25.3269	13.8271	7.5106	4.1530
1600			148.0450	75.8290	40.7702	21.6635	11.8671
2000					94.6919	49.9192	26.6086
Double precision							
Forward transform							
200	0.4485	0.2545	0.1686	0.0971	0.0897	0.0655	0.0672
400	5.9599	3.1502	1.7757	0.9420	0.5871	0.3953	0.2616
600	28.5156	14.7995	7.7479	3.9093	2.2731	1.3496	0.7991
800	86.9170	45.2880	23.6875	12.2694	6.7581	3.8764	2.1842
1000	238.4780	107.3896	56.2608	29.2851	15.6836	8.7244	4.8547
1200			115.2575	58.9024	30.9047	16.3460	8.6596
1600					94.2719	48.8668	26.2570
Backward transform							
200	0.3976	0.2212	0.1356	0.0649	0.0455	0.0288	0.0243
400	5.6193	2.9700	1.5695	0.8475	0.4922	0.2879	0.1865
600	27.3639	14.1340	7.2135	3.6298	2.1130	1.2073	0.6771
800	84.5136	43.6104	22.6527	11.4513	6.2700	3.5147	2.1582
1000	216.1370	104.4850	53.0580	27.9994	14.9497	8.3255	4.6314
1200			110.3050	56.8326	29.5519	15.6329	8.2157
1600					91.7581	47.6514	25.0236

Tables 3 and 4 contain the execution time using only Intel Xeon Phi. Here one can observe that the algorithms run faster using one coprocessor than using 2, 4, 8 coprocessors. It is clear that the communication time between coprocessors is large and there is no improvement using the parallel implementation of the algorithms.

Table 3. Execution time (in seconds) for the 3D discrete cosine transform using only coprocessors of the Avitohol

N	Number of coprocessors						
	1	2	4	8	16	32	64
Single precision							
Forward transform							
200	0.5124	0.6388	0.6370	0.4128	0.3938	0.3576	0.2495
400	1.2582	1.9077	1.9876	1.4857	1.3592	1.1430	0.7298
600	3.7672	5.4232	5.4707	3.8514	3.5044	2.7911	1.8514
800	8.5344	13.5098	11.9203	8.0540	7.5502	5.7663	3.8314
1000		26.7401	23.8716	16.2240	14.3935	10.6471	6.9524
1200			41.1496	29.6708	24.2277	18.2094	11.7408
1600				69.3336	59.1269	41.4999	26.6460
2000					115.3765	81.7667	52.4187
Backward transform							
200	0.0642	0.1746	0.2181	0.1598	0.1425	0.1228	0.0836
400	0.4447	1.3052	1.4283	1.0150	0.9230	0.7501	0.4711
600	1.8362	4.4320	4.5303	3.3942	3.0188	2.3530	1.4862
800	4.8155	11.4296	10.4895	7.7898	6.9923	5.3358	3.3693
1000		23.5476	21.2399	15.2939	13.6295	10.2196	6.4541
1200			36.8365	27.3000	23.6922	17.5493	11.2333
1600				64.4491	57.4970	40.9009	26.0027
2000					112.4650	80.4906	51.5106
Double precision							
Forward transform							
200	0.5582	0.7781	0.8064	0.5260	0.5054	0.4507	0.3163
400	2.5085	3.3958	3.2535	2.4080	2.2024	1.7706	1.1271
600	7.7727	11.1460	10.3057	7.0143	6.3357	4.7157	3.1787
800		27.1808	24.2012	16.1103	14.4354	10.5462	6.9842
1000			48.4785	34.2117	28.2551	20.3837	13.1819
1200				59.5258	49.7826	35.1730	22.3118
1600					118.5665	83.4042	53.2028
2000							105.3077
Backward transform							
200	0.0929	0.3182	0.3482	0.2501	0.2369	0.2030	0.1229
400	0.9738	2.5964	2.6336	1.9374	1.7742	1.3468	0.8450
600	4.4812	9.3757	8.9153	6.5473	5.8316	4.3498	2.7440
800		24.3902	21.5345	15.2626	13.9939	10.1905	6.5200
1000			42.7577	32.2109	27.9494	19.8042	12.6917
1200				56.4205	48.3041	34.6965	21.8462
1600					117.8540	82.2662	52.1192
2000							104.8230

Table 4. Execution time (in seconds) for the 3D discrete Fourier transform using only coprocessors of the Avitohol

N	Number of coprocessors						
	1	2	4	8	16	32	64
Single precision							
Forward transform							
200	0.6733	0.8424	0.8686	0.5990	0.5611	0.5129	0.3537
400	2.6020	3.6413	3.5441	2.4803	2.2891	1.8462	1.1848
600	8.1777	11.9750	10.3334	7.2245	6.5103	4.9759	3.2558
800		27.8812	24.3975	16.9994	14.7433	10.8582	7.0387
1000			49.5779	35.6633	28.9276	20.9952	13.5672
1200				61.3304	49.5451	35.3441	22.8684
1600					118.7209	83.9047	53.5311
2000							105.9995
Backward transform							
200	0.1423	0.3791	0.4080	0.3115	0.2805	0.2506	0.1830
400	1.1092	2.7578	2.7353	1.9834	1.8473	1.4136	0.9093
600	4.7513	9.6823	9.1412	6.6159	6.0284	4.4975	2.8664
800		24.6317	21.8166	16.1799	14.0978	10.3299	6.5010
1000			44.4229	32.3628	27.8660	19.8807	12.8469
1200				57.0783	48.8767	34.6318	22.2119
1600					116.7550	82.6160	52.6932
2000							104.9290
Double precision							
Forward transform							
200	0.8991	1.1888	1.2154	0.7955	0.7516	0.6587	0.4543
400	4.9907	7.0999	6.4597	4.4377	3.9776	3.0814	1.9723
600		24.1780	21.1313	14.2214	12.7873	9.0109	5.9362
800			51.2982	36.0511	29.5805	21.0108	13.5255
1000				74.3123	58.3255	41.1495	26.7309
1200					102.2205	71.1527	45.5205
Backward transform							
200	0.3142	0.7203	0.7460	0.5714	0.5261	0.4479	0.2994
400	2.6119	5.6940	5.4064	3.9828	3.5879	2.6907	1.7472
600		21.6765	18.6295	13.3133	11.8331	8.7192	5.6127
800			45.0476	32.8309	28.7036	20.6122	13.1516
1000				67.0370	56.7831	40.8836	25.7711
1200					100.3330	70.5225	44.7571

Tables 5 and 6 present times collected on the Avitohol using processors as well as coprocessors. We made the experiments using 4 MPI processes on each node: 2 processes on processors and 2 processes on coprocessors. The results in Table 5 show that again there is no improvement using the parallel implementation of the algorithm. Conversely, we can see from Table 6 the execution time of the parallel implementation of the algorithm for 3D DFT is better than the time of the sequential algorithm.

Table 5. Execution time (in seconds) for the 3D discrete cosine transform using processors and coprocessors of the Avitohol

N	Number of nodes					
	1	2	4	8	16	32
Single precision						
Forward transform						
200	1.0461	0.9174	0.7746	0.6689	0.5356	0.5219
400	5.7840	3.7579	2.5590	1.8448	1.1934	0.9889
600	19.8967	11.1379	7.0881	4.5794	2.9457	2.0704
800	47.7289	27.0011	15.7896	9.7563	5.9931	3.9820
1000	96.2926	54.0093	31.1242	18.5277	10.9908	7.0737
1200	172.6714	94.5210	54.0907	31.5976	18.3560	11.6593
1600		244.7497	134.0510	76.1986	42.2924	26.2381
2000			272.4915	152.2338	83.7523	51.7131
Backward transform						
200	0.3160	0.0430	0.0309	0.0620	0.0742	0.0825
400	1.6914	0.6862	0.6055	0.4975	0.3279	0.2435
600	4.2980	2.6665	2.3685	1.8131	1.1951	0.8605
800	9.2476	6.1460	6.0348	4.4966	2.7682	2.0479
1000	19.5277	13.0295	12.2256	9.0832	5.3651	3.9084
1200	33.4088	22.9409	20.9885	16.0037	9.2687	6.9492
1600		57.2751	51.6913	37.6339	21.8078	16.4330
2000			101.5010	75.0368	43.7860	33.0999
Double precision						
Forward transform						
200	1.1424	0.9993	0.8713	0.7596	0.6030	0.5582
400	6.4779	4.4091	3.2350	2.3677	1.5319	1.2655
600	23.4372	14.2287	9.4932	6.2915	4.0761	2.9181
800	57.5915	34.4666	21.7705	13.9947	8.6594	5.9716
1000	119.0888	69.6325	43.1526	27.0916	16.0252	10.8204
1200		120.6609	76.2304	47.3631	27.6925	18.0780
1600			193.9048	114.2277	65.3929	42.6685
2000					130.9305	86.1398
Backward transform						
200	0.5059	0.0926	0.0493	0.0863	0.0590	0.0650
400	2.8307	1.4440	1.3961	0.9820	0.6129	0.4586
600	8.1704	5.2758	5.0683	3.7362	2.2408	1.6889
800	19.7701	13.2111	12.4784	9.2636	5.4426	3.9528
1000	39.2059	26.1137	24.7823	18.1029	10.6478	7.8551
1200		45.5004	43.6663	31.7628	18.3667	13.7592
1600			104.6020	75.7410	43.5752	32.6848
2000					87.7772	63.9718

Table 6. Execution time (in seconds) for the 3D discrete Fourier transform using processors and coprocessors of the Avitohol

N	Number of nodes					
	1	2	4	8	16	32
Single precision						
Forward transform						
200	0.7257	0.8685	0.8249	0.7840	0.6487	0.6235
400	3.3775	2.8575	2.3905	1.9840	1.4639	1.2543
600	11.8316	8.5432	6.5842	4.8935	3.3939	2.6014
800	31.0851	21.1444	15.2111	10.7736	6.8629	5.1113
1000	67.0818	43.5047	30.1038	20.7041	12.9051	9.3361
1200		79.1054	54.5836	35.8087	22.2026	15.6287
1600			144.7684	87.8005	53.2528	36.7738
2000					107.5775	74.2922
Backward transform						
200	0.6455	0.2580	0.0944	0.2067	0.1816	0.1652
400	2.7533	1.5852	1.5336	1.1868	0.7561	0.5598
600	8.2752	5.4929	5.3142	3.9293	2.3069	1.7960
800	21.0940	14.5966	12.8842	9.4114	5.4024	4.0734
1000	45.6085	29.0226	25.4616	18.8960	10.9276	7.7307
1200		50.1291	41.9014	32.7591	18.8841	13.7003
1600			102.9000	76.8544	44.2353	33.2393
2000					89.3026	65.2866
Double precision						
Forward transform						
200	0.8904	1.0013	0.9526	0.8867	0.7206	0.6537
400	5.9818	4.5343	3.8480	2.9957	2.0940	1.7015
600	22.0847	15.3617	12.3924	8.7832	5.6635	4.2174
800	60.0458	38.9667	29.2292	20.1616	12.8326	9.1372
1000		79.0519	60.2493	40.4552	25.1706	17.5858
1200			107.1120	70.7953	43.2693	31.0853
1600					104.4458	73.8282
Backward transform						
200	1.0100	0.3795	0.3436	0.3092	0.2282	0.2239
400	4.9562	3.2677	2.9362	2.2727	1.4620	1.1017
600	17.4575	11.6983	10.7004	7.9430	4.6395	3.4696
800	45.2816	28.4348	25.3424	18.9955	10.9793	8.1076
1000		59.8817	50.8526	38.1525	21.9127	15.8757
1200			87.3476	66.0734	38.0652	27.1347
1600					90.2499	66.7974

Finally, in Fig. 1, we represent execution time of the code, which performs one forward and one backward DFT. Results are presented for single and double precision, for problems of size $N = 1000$ and $N = 1600$.

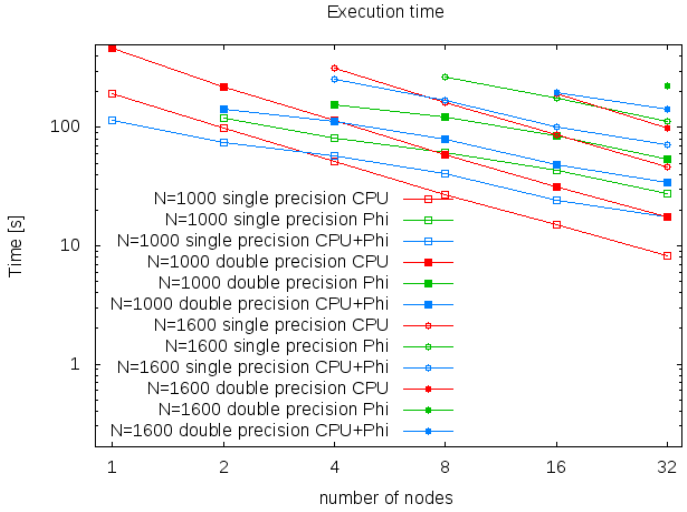

Fig. 1. Execution time of code which performs forward and backward DFT for N = 1000, 1600

5. Conclusion

The aim of this paper was to analyze an implementation of a slightly simplified version of an algorithm for 3D forward/inverse discrete transforms on supercomputer with Intel processors/co-processors. The algorithm was designed for a 3D torus network. The results on the supercomputer Avitohol show that a fast communication network is needed in order to obtain parallel efficiency of the algorithm. On the basis of the obtained results we can conclude that the proposed approach allows solution of large 3D problems on a supercomputer.

***Acknowledgments*:** This research was partially supported by grant DNTS/Russia 02/7 from the Bulgarian NSF. This work has been accomplished with the partial support by the Grant No. BG05M2OP001-1.001-0003, financed by the Science and Education for Smart Growth Operational Program (2014-2020) and co-financed by the European Union through the European structural and Investment funds.

References

1. Sedukhin, S. G. Co-Design of Extremely Scalable Algorithms/Architecture for 3-Dimensional Linear Transforms. Technical Report TR2012-001. The University of Aizu, July 2012.
2. Lirkov, I., M. Paprzycki, M. Ganzha, S. Sedukhin, P. Gepner. Performance Analysis of Scalable Algorithms for 3D Linear Transforms. – In: Proc. of M. Ganzha, L. Maciaszek, M. Paprzycki, Eds. 2014 Federated Conference on Computer Science and Information Systems, Annals of Computer Science and Information Systems, Vol. **2**, IEEE, 2014, pp. 613-622. DOI: 10.15439/2014F374.
3. Dongarra, J. J., J. Du Croz, S. Hammarling, I. S. Duff. A Set of Level 3 Basic Linear Algebra Subprograms. – ACM Transactions On Mathematical Software (TOMS), Vol. **16**, 1990, No 1, pp. 1-17. DOI: 10.1145/77626.79170.

4. Intel Math Kernel Library.
http://software.intel.com/en-us/articles/intel-mkl/
5. S n i r, M., S. O t t o, S. H u s s-L e d e r m a n, D. W a l k e r, J. D o n g a r r a. MPI: The Complete Reference. Second Edition. Vol. **1**. The MPI Core. Scientific and Engineering Computation Series, Cambridge, Massachusetts, MIT Press, 1998. ISBN: 9780262692151.
6. W a l k e r, D., J. D o n g a r r a. MPI: A Standard Message Passing Interface. – Supercomputer, Vol. **12**, 1996, No 1, pp. 56-68. ISSN 0168-7875.
7. High-Performance Computing System – AVITOHOL.
http://www.hpc.acad.bg/avitohol/

Received: 15.09.2020; Second Version: 19.10.2020; Accepted: 23.10.2020