Performance analysis of a scalable algorithm for 3D linear transforms

Ivan Lirkov
Institute of Information and Communication Technologies
Bulgarian Academy of Sciences
Acad. G. Bonchev, bl. 25A
1113 Sofia, Bulgaria
ivan@parallel.bas.bg
http://parallel.bas.bg/~ivan/

Marcin Paprzycki
Maria Ganzha
Systems Research Institute, Polish Academy of Sciences
ul. Newelska 6, 01-447 Warsaw, Poland
paprzyck@ibspan.waw.pl, maria.ganzha@ibspan.waw.pl
http://www.ibspan.waw.pl/~paprzyck/
http://inf.ug.edu.pl/~mganzha/

Stanislav Sedukhin
Graduate School of Computer Science and Engineering
The University of Aizu
Tsuruga, Ikki-Machi, Aizu-Wakamatsu City
Fukushima, 965-8580 Japan
sedukhin@u-aizu.ac.jp

Paweł Gepner
Intel Corporation
Pipers Way
Swindon Wiltshire SN3 1RJ
United Kingdom
pawel.gepner@intel.com

***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. Specifically, 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. Recently, a novel, 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-neighbor nodes (directly using the logical arrangements of the nodes within the torus). The proposed 3D orbital algorithms gracefully avoids the, required, 3D data transposition. The aim of this paper is to present a preliminary experimental performance study of the proposed implementation on two different high-performance computer architectures.**

I. Introduction

THREE-DIMENSIONAL (3D) discrete transforms (DT) such as Fourier transform, cosine/sine transform, Hartley transform, Walsh-Hadamard 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 (wall-clock) time is possible only by overlapping these arithmetic operations, i.e. using parallel implementation.

There exists 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, $N \times N \times N$ data is divided into 2D slabs of size $N \times N \times b$, where $b = N/P$ and P is the number of computer nodes. The scalability of the slab-based approach, or the maximum number of nodes that can be used concurrently, 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 $N \times N \times N$ initial data, among a 2D array of $P \times P$ computer nodes. Here, the initial cube is divided into a 1D "pencil" of size $N \times b \times b$, and is assigned to each node (as above, $b = N/P$). This approach increases the maximum number of nodes than can be effectively used in computations, from N to N^2. Parallel 3D FFT implementation with a 2D data decomposition has been discussed, among others, in [1], [3], [4].

In both of these, so-called, "transposed" approaches, the computational part and the inter-node communication part are separated. Moreover, a computational part inside each node is implemented by using either 2D or 1D fast (recursive) algorithm for a "slab"-based or a "pencil"-based decomposition, respectively, *without* any inter-node communication. However, upon completion of each computational part, in order to support contiguity of memory accesses, a transposition of the 3D data array is required, to put data of appropriate dimension(s) into each node. Here, at least one or two transpositions would be needed for the 1D or 2D data decomposition-based approaches, respectively. Each of such transpositions of

3D data is typically implemented by a global "all-to-all" inter-node, message-passing communication.

The last approach is the 3D or "cube" decomposition, which was recently proposed in [5]. The 3D or "cubic" decomposition of an $N \times N \times N$ initial data among $P \times P \times P$ computer nodes, allows a 3D data "cube" of size $b \times b \times b$ to be assigned to each computer node. It is easy to realize that, here, the theoretical scalability is further improved from N^2 to N^3. 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 $P = N/b$ cyclically interconnected (for data reuse) 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.

Te 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 on selected computers. 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 Linux cluster and on the Blue Gene/P supercomputer.

II. 3D Separable Transform

Let us start by introducing basic definitions concerning 3D separable transforms. Let $X = [x(n_1, n_2, n_3)]$, $0 \leq n_1, n_2, n_3 < N$, be an $N \times N \times N$ cubical grid of input data, or a three-way data tensor. A separable forward 3D transform of X is another cubical grid of an $N \times N \times N$ data or a three-way tensor $\ddot{X} = [\ddot{x}(k_1, k_2, k_3)]$, where for all $0 \leq k_1, k_2, k_3 < N$:

$$\ddot{x}(k_1,k_2,k_3) = \sum_{n_3=0}^{N-1} \sum_{n_2=0}^{N-1} \sum_{n_1=0}^{N-1} x(n_1,n_2,n_3) \cdot c(n_1,k_1) \cdot c(n_2,k_2) \cdot c(n_3,k_3) \quad (1)$$

A separable inverse, or backward, 3D transform of a three-way tensor $\ddot{X} = [\ddot{x}(k_1, k_2, k_3)]$ is expressed as:

$$x(n_1,n_2,n_3) = \sum_{k_3=0}^{N-1} \sum_{k_2=0}^{N-1} \sum_{k_1=0}^{N-1} \ddot{x}(k_1,k_2,k_3) \cdot c(n_1,k_1) \cdot c(n_2,k_2) \cdot c(n_3,k_3) \quad (2)$$

where $0 \leq n_1, n_2, n_3 < N$ and $X = [x(n_1, n_2, n_3)]$ is an output $N \times N \times N$ cubical tensor.

We will use the notations from [5] to describe the proposed parallel algorithm. First, we divide the input data $X = [x(n_1, n_2, n_3)]$ into $P_1 \times P_2 \times P_3$ data rectangular cuboid, where each cuboid $X(N_1, N_2, N_3), 0 \leq N_i < P_i$, has the size of $b_1 \times b_2 \times b_3$, i.e. $b_i = N/P_i$. Then, the forward 3D transform can be expressed as a block version of the multi-linear matrix multiplication:

$$\ddot{X}(K_1,K_2,K_3) = \sum_{N_3=0}^{P_3-1} \sum_{N_2=0}^{P_2-1} \sum_{N_1=0}^{P_1-1} X(N_1,N_2,N_3) \times C(N_1,K_1) \times C(N_2,K_2) \times C(N_3,K_3), \quad (3)$$

where $0 \leq K_i < P_i$ and $C(N_s, K_s), s = 1, 2, 3$, is the (N_s, K_s)-th block of the transform matrix C.

Due to the separability of the linear transforms, 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:

$$\dot{X}(N_1,N_2,K_3) = \sum_{N_3=0}^{P_3-1} X(N_1,N_2,N_3) \times C(N_3,K_3).$$

At the *second stage*, the 1D transform of $\dot{X}(:, N_2, K_3)$ is implemented for all (N_2, K_3) pairs, as the second block tensor-by-matrix multiplication:

$$\ddot{X}(K_1,N_2,K_3) = \sum_{N_1=0}^{P_1-1} \dot{X}(N_1,N_2,K_3) \times C(N_1,K_1).$$

At the *third stage*, the 1D transform of $\ddot{X}(K_1, :, K_3)$ is implemented for all (K_1, K_3) pairs, as the third block tensor-by-matrix multiplication:

$$\dddot{X}(K_1,K_2,K_3) = \sum_{N_2=0}^{P_2-1} \ddot{X}(K_1,N_2,K_3) \times C(N_2,K_2).$$

By slicing the cubical data, i.e. representing the three-way tensors as the set of matrices, it is possible to formulate the 3D transform as a conventional block *matrix-by-matrix multiplication* with its transpose/nontranspose versions. In this case, the initial data grid $X(N_1, N_2, N_3)$, is divided into 1D "slices" along one axis. Then, the 3D transform can also be computed in three data-dependent stages as chaining sets of block matrix-by-matrix products.

III. Algorithm Description

A. Multi-node Implementation

In the proposed approach it is assumed that each computer node CN(Q,R,S) has six bi-directional links labeled as $\pm Q$, $\pm R$ and $\pm S$. These nodes are toroidally interconnected. During processing, some blocks of tensor data are rolled, i.e. cyclically shifted, along (+) or opposite (-) axis (orbit). The first two stages implement the set of space-independent 2D forward transforms, in parallel, along the R-axis (orbit) slabs.

Note that, each stage of both forward and inverse transforms, with 3D data decomposition, has a common structure, i.e. steps of "compute-and-roll".

A three-stage orbital implementation of the 3D forward transform in a 3-dimensional network of toroidally interconnected nodes CN(Q,R,S) proceeds as follows.

Stage I.
$\dot{X}(N_1, N_2, K_3) = \sum_{0 \le N_3 < P_3} X(N_1, N_2, N_3) \times C(N_3, K_3)$:

- **for all** CN(Q, R, S) **do** P_3 times:
  1) **compute**: $\dot{X} \leftarrow X \times C + \dot{X}$
  2) **data roll**: $\overset{+S}{\Longleftarrow} X \overset{-S}{\Longleftarrow}$

Stage II. $\ddot{X}(K_1, N_2, K_3) =$
$\sum_{0 \le N_1 < P_1} C(N_1, K_1)^T \times \dot{X}(N_1, N_2, K_3)$:

- **for all** CN(Q, R, S) **do** P_1 times:
  1) **compute**: $\ddot{X} \leftarrow C^T \times \dot{X} + \ddot{X}$
  2) **data roll**: $\overset{+Q}{\Longleftarrow} \dot{X} \overset{-Q}{\Longleftarrow}$

Stage III.
$\dddot{X}(K_1, K_2, K_3) = \sum_{0 \le N_2 < P_2} \ddot{X}(K_1, N_2, K_3) \times C(N_2, K_2)$:

- **for all** CN(Q, R, S) **do** P_2 times:
  1) **compute**: $\dddot{X} \leftarrow \ddot{X} \times C + \dddot{X}$
  2) **data roll**: $\overset{+R}{\Longleftarrow} \ddot{X} \overset{-R}{\Longleftarrow}$

For more details, see [5].

It should be noted that the implementation described here is a modification of the parallel algorithms proposed in [5]. 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 (by $O(N)$ elements) 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 an existing GEMM subroutines, from the BLAS library [2], to compute the 3D transform.

B. Multi-thread Implementation

There exists two possible ways to compute the tensor-by-matrix multiplication on computers with multi-core processors. The first one is to use the multi-threaded library, such as the Engineering and Scientific Subroutine Library (ESSL, see http://www-03.ibm.com/systems/software/essl/index.html) or the Intel Math Kernel Library (MKL, see http://software.intel.com/en-us/articles/intel-mkl/). 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.

IV. Experimental Results

A portable parallel code was designed and implemented in C. The parallelization was based on the MPI standard [6], [7]. 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. Let us also note that we used one MPI process per computer node.

The parallel code has been tested on the following systems: (1) a cluster computer *Galera*, located in the Polish Informatics Center TASK, and (2) two IBM Blue Gene/P machines, one at the Bulgarian Supercomputing Center, and one at the HPC Center of the West University of Timisoara (UVT).

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 T_p, in seconds, using p MPI processes, and the parallel speed-up $S_p = T_1/T_p$.

Tables I and II show the results collected on the Galera. It is a Linux cluster with 336 nodes, and two Intel Xeon quad core processors per node. Each processor runs at 2.33 GHz. Processors within each node share 8, 16, or 32 GB of memory. Nodes are interconnected with a high-speed InfiniBand network (see also http://www.task.gda.pl/kdm/sprzet/Galera). When running our code on Galera, we used the Intel C compiler, and compiled the code with the options "-O3 -openmp". To use the BLAS subroutines, we linked our code to the optimized multi-threaded Intel MKL library.

The symbol * in the tables denotes that, in the given case, the memory of p nodes was not large enough to compute the 3D transform for data of size $N \times N \times N$.

The reported execution time for $N = 100$ shows that the problem is "small" and can be executed on one node of the cluster (no need for parallelization). Here, there is no significant improvement from using two or more nodes. However, already for the problems of size $N = 600$ a significant performance gain can be observed (see, also, below). Considering the fact that some of the applications that need 3D transforms involve "real-time processing of data," it is worthy noting that, using the proposed method, similar time is required to find the solution on a single node for the problem of size $N = 600$ as finding solution using 256 nodes for the problem of size $2000 < N < 2400$.

Table III contains the speed-up obtained on the Galera. For the largest problem, which can be executed on a single node, the parallel efficiency is above 50% for the number of nodes up to 16 for the DCT and up to 32 for the DFT. We note that the main advantage of the parallel algorithm is that the code allows performing the 3D transform for very large data. Taking into account the largest cases reported in Tables I and II, we

TABLE I
EXECUTION TIME FOR THE 3D DISCRETE COSINE TRANSFORM ON GALERA.

N	nodes								
	1	2	4	8	16	32	64	128	256
	single precision								
	forward transform								
100	0.08	0.06	0.06	0.18	0.07	0.14	0.20	0.19	0.20
200	0.23	0.18	0.15	0.10	0.09	0.10	0.11	0.14	0.18
300	0.86	0.54	0.31	0.21	0.14	0.15	0.17	0.12	0.21
400	2.18	1.26	0.71	0.41	0.32	0.25	0.23	0.16	0.22
600	9.59	5.58	3.06	1.80	0.98	0.68	0.45	0.36	0.35
800	*	14.51	7.70	4.29	2.51	1.43	0.85	0.72	0.52
1000	*	*	17.91	10.12	5.47	3.07	1.77	1.28	0.99
1200	*	*	*	18.88	10.71	5.80	3.31	2.48	1.56
1400	*	*	*	34.11	19.02	10.23	5.67	3.97	2.76
1600	*	*	*	*	27.45	14.83	8.11	5.11	3.74
2000	*	*	*	*	*	35.00	18.50	11.16	7.48
2400	*	*	*	*	*	75.50	35.10	22.12	13.56
2800	*	*	*	*	*	*	63.67	36.11	23.61
3200	*	*	*	*	*	*	*	53.51	34.92
	backward transform								
100	0.03	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02
200	0.20	0.13	0.08	0.05	0.03	0.02	0.01	0.01	0.03
300	0.83	0.49	0.29	0.17	0.11	0.08	0.05	0.06	0.04
400	2.08	1.16	0.65	0.38	0.25	0.18	0.09	0.09	0.07
600	9.38	5.44	2.96	1.61	0.98	0.56	0.35	0.29	0.20
800	*	14.00	7.36	4.13	2.36	1.24	0.72	0.58	0.43
1000	*	*	17.65	9.56	5.14	3.00	1.75	1.33	0.88
1200	*	*	*	18.04	10.73	5.82	3.19	2.10	1.58
1400	*	*	*	33.53	18.49	9.85	5.48	3.45	2.46
1600	*	*	*	*	26.68	14.36	7.68	5.57	3.56
2000	*	*	*	*	*	34.52	18.50	12.05	7.77
2400	*	*	*	*	*	70.40	37.74	21.43	13.99
2800	*	*	*	*	*	*	65.15	38.27	23.74
3200	*	*	*	*	*	*	*	58.55	33.11
	double precision								
	forward transform								
100	0.07	0.05	0.07	0.14	0.08	0.15	0.19	0.20	0.48
200	0.39	0.27	0.20	0.16	0.15	0.12	0.16	0.16	0.20
300	1.51	0.89	0.51	0.37	0.31	0.21	0.19	0.22	0.20
400	4.09	2.26	1.27	0.78	0.49	0.34	0.25	0.27	0.24
600	18.49	9.63	5.29	2.90	1.77	1.04	0.62	0.60	0.44
800	*	*	14.72	8.00	4.36	2.56	1.41	1.15	0.92
1000	*	*	*	18.11	9.69	5.40	3.12	2.36	1.72
1200	*	*	*	36.27	18.10	10.00	5.68	4.11	2.95
1400	*	*	*	*	32.85	18.05	9.78	6.59	4.55
1600	*	*	*	*	*	30.76	15.28	9.17	6.89
2000	*	*	*	*	*	*	*	20.80	14.69
2400	*	*	*	*	*	*	*	39.98	26.38
2800	*	*	*	*	*	*	*	68.79	47.75
	backward transform								
100	0.04	0.02	0.02	0.01	0.01	0.01	0.01	0.01	0.02
200	0.36	0.21	0.13	0.07	0.05	0.04	0.04	0.02	0.03
300	1.43	0.82	0.48	0.32	0.17	0.13	0.08	0.06	0.07
400	3.94	2.09	1.24	0.75	0.41	0.28	0.17	0.14	0.13
600	18.10	9.06	4.88	2.74	1.60	1.02	0.57	0.45	0.28
800	*	*	13.87	7.39	4.13	2.30	1.37	1.14	0.71
1000	*	*	*	16.92	9.17	5.02	2.90	2.54	1.68
1200	*	*	*	34.43	17.00	9.43	4.78	3.88	2.76
1400	*	*	*	*	31.85	16.97	8.90	7.09	4.76
1600	*	*	*	*	*	27.07	14.07	10.75	7.14
2000	*	*	*	*	*	*	*	22.22	15.28
2400	*	*	*	*	*	*	*	42.08	25.26
2800	*	*	*	*	*	*	*	71.30	44.40

TABLE II
EXECUTION TIME FOR THE 3D DISCRETE FOURIER TRANSFORM ON GALERA.

N	nodes								
	1	2	4	8	16	32	64	128	256
	single precision								
	forward transform								
100	0.13	0.07	0.15	0.14	0.12	0.14	0.21	0.19	0.23
200	0.61	0.39	0.26	0.24	0.12	0.19	0.20	0.21	0.22
300	2.52	1.56	0.93	0.55	0.42	0.37	0.32	0.27	0.27
400	7.08	3.93	2.16	1.24	0.75	0.54	0.36	0.36	0.36
600	32.02	17.42	9.34	5.15	3.20	1.72	1.01	0.77	0.57
800	*	*	26.21	14.13	7.99	4.40	2.46	1.49	1.13
1000	*	*	*	31.98	17.89	9.75	5.33	3.12	2.05
1200	*	*	*	63.48	34.49	18.41	9.79	6.19	3.78
1400	*	*	*	*	61.93	32.31	17.73	10.15	6.09
1600	*	*	*	*	*	51.22	27.99	15.03	8.85
2000	*	*	*	*	*	*	61.28	34.67	18.51
2400	*	*	*	*	*	*	153.02	66.21	37.28
2800	*	*	*	*	*	*	*	124.36	67.65
	backward transform								
100	0.06	0.04	0.03	0.02	0.01	0.01	0.01	0.01	0.01
200	0.58	0.33	0.20	0.11	0.08	0.09	0.06	0.04	0.03
300	2.44	1.43	0.83	0.47	0.31	0.20	0.11	0.08	0.06
400	6.93	3.80	2.04	1.13	0.64	0.44	0.26	0.22	0.17
600	31.49	16.84	8.99	4.92	2.90	1.77	0.94	0.68	0.43
800	*	*	25.66	13.67	7.44	4.39	2.32	1.67	1.00
1000	*	*	*	31.46	17.56	9.25	5.03	3.20	2.31
1200	*	*	*	64.16	33.55	17.38	10.00	5.88	3.92
1400	*	*	*	*	60.57	32.48	17.19	10.54	6.77
1600	*	*	*	*	*	51.21	26.56	15.28	9.66
2000	*	*	*	*	*	*	62.10	36.40	22.04
2400	*	*	*	*	*	*	173.82	66.95	37.35
2800	*	*	*	*	*	*	*	136.07	66.60
	double precision								
	forward transform								
100	0.17	0.14	0.09	0.17	0.06	0.20	0.19	0.20	0.29
200	1.23	0.73	0.42	0.37	0.19	0.17	0.22	0.24	0.22
300	5.24	2.96	1.57	1.02	0.66	0.47	0.34	0.32	0.33
400	14.63	8.21	4.30	2.43	1.36	0.88	0.60	0.47	0.43
600	74.84	36.03	18.82	10.47	5.95	3.24	1.88	1.30	0.98
800	*	*	*	28.70	15.78	8.34	5.05	2.96	1.92
1000	*	*	*	*	36.49	18.98	10.29	6.13	3.89
1200	*	*	*	*	71.56	36.18	19.53	11.37	6.88
1400	*	*	*	*	*	64.20	34.80	19.71	11.65
1600	*	*	*	*	*	*	58.76	29.84	18.58
2000	*	*	*	*	*	*	*	70.18	37.90
2400	*	*	*	*	*	*	*	204.81	69.83
	backward transform								
100	0.11	0.07	0.04	0.03	0.02	0.04	0.01	0.01	0.01
200	1.20	0.67	0.39	0.22	0.16	0.10	0.07	0.06	0.11
300	5.09	2.81	1.51	0.86	0.57	0.31	0.21	0.12	0.11
400	14.32	7.81	4.16	2.37	1.30	0.78	0.45	0.35	0.33
600	85.48	35.01	18.05	9.79	5.25	2.99	1.81	1.07	0.78
800	*	*	*	28.14	14.88	8.13	4.33	3.01	2.18
1000	*	*	*	*	35.58	17.76	9.97	6.14	4.40
1200	*	*	*	*	72.70	36.49	18.90	11.06	7.65
1400	*	*	*	*	*	64.13	33.69	19.83	11.94
1600	*	*	*	*	*	*	54.91	30.35	18.84
2000	*	*	*	*	*	*	*	71.07	39.38
2400	*	*	*	*	*	*	*	136.01	74.78

TABLE III
Speed-up on Galera.

N	nodes							
	2	4	8	16	32	64	128	256
	single precision DCT							
	forward transform							
100	1.26	1.28	0.46	1.20	0.58	0.40	0.43	0.41
200	1.32	1.57	2.31	2.52	2.38	2.17	1.65	1.29
300	1.61	2.74	4.16	6.03	5.75	5.17	7.13	4.08
400	1.73	3.06	5.25	6.74	8.55	9.46	14.01	9.77
600	1.72	3.14	5.34	9.82	14.10	21.26	26.72	27.47
	backward transform							
100	1.75	2.31	2.83	5.07	3.37	3.56	9.85	1.21
200	1.54	2.59	4.29	6.83	9.73	15.57	15.37	7.95
300	1.70	2.91	4.91	7.62	10.44	17.70	13.50	20.04
400	1.79	3.19	5.43	8.19	11.66	22.21	22.79	31.49
600	1.72	3.17	5.83	9.58	16.85	26.89	32.52	47.97
	double precision DCT							
	forward transform							
100	1.36	1.02	0.50	0.86	0.45	0.36	0.34	0.15
200	1.45	1.92	2.38	2.63	3.36	2.49	2.46	1.94
300	1.69	2.93	4.08	4.93	7.21	7.77	6.95	7.60
400	1.81	3.23	5.24	8.41	11.98	16.25	15.15	17.22
600	1.92	3.50	6.38	10.48	17.83	29.61	32.63	41.63
	backward transform							
100	1.60	2.30	3.76	5.36	6.55	4.05	4.37	1.66
200	1.73	2.79	4.79	7.18	9.10	8.76	19.52	13.77
300	1.74	3.00	4.46	8.24	10.64	18.67	23.22	20.88
400	1.89	3.18	5.24	9.60	14.23	22.51	27.85	30.30
600	2.00	3.71	6.60	11.31	17.73	31.83	39.98	64.69
	single precision DFT							
	forward transform							
100	1.85	0.85	0.94	1.09	0.94	0.61	0.66	0.55
200	1.56	2.34	2.57	4.91	3.20	3.08	2.89	2.73
300	1.62	2.72	4.42	5.98	6.90	7.88	9.21	9.25
400	1.80	3.28	5.72	9.41	13.06	19.86	19.51	19.50
600	1.84	3.43	6.21	10.02	18.65	31.72	41.76	56.01
	backward transform							
100	1.66	2.31	3.62	5.10	5.26	10.08	9.24	6.03
200	1.76	2.95	5.08	7.56	6.79	10.56	15.38	22.71
300	1.71	2.94	5.22	7.95	12.19	22.02	30.82	38.79
400	1.82	3.39	6.13	10.88	15.83	26.44	31.29	41.30
600	1.87	3.50	6.41	10.85	17.79	33.33	46.24	73.48
	double precision DFT							
	forward transform							
100	1.16	1.94	0.98	2.79	0.84	0.87	0.84	0.57
200	1.68	2.95	3.36	6.56	7.36	5.68	5.17	5.50
300	1.77	3.34	5.12	7.93	11.23	15.32	16.52	15.72
400	1.78	3.40	6.01	10.77	16.60	24.59	31.07	33.90
600	2.08	3.98	7.15	12.57	23.09	39.85	57.58	76.71
	backward transform							
100	1.63	2.64	3.50	5.65	2.82	9.63	9.58	13.64
200	1.79	3.06	5.32	7.59	12.19	17.53	20.07	10.50
300	1.81	3.36	5.95	8.99	16.44	24.28	42.88	44.84
400	1.83	3.44	6.03	11.02	18.38	31.52	40.92	43.24
600	2.44	4.74	8.73	16.29	28.59	47.11	79.60	110.08

can see that increasing the number of nodes from 128 to 256 results in efficiency of 60-69% for the DCT, and 40-60% for the DFT (depending if the transform forward or backward and if it runs in single or double precision).

Tables IV and V present times collected on the IBM Blue Gene/P supercomputers. For our experiments we used the BG/P machine located at the Bulgarian Supercomputing Center and a slightly different one located at the HPC Center of the West University of Timisoara (UVT). The supercomputer in Bulgaria has two BG/P racks, while the supercomputer in Romania has one BG/P rack. One BG/P rack consists of 1024 compute nodes with quad core PowerPC 450 processors (running at 850 MHz). Each node of the Bulgarian rack has 2 GB of RAM, while each node of the Romanian rack has 4 GB of RAM. For the point-to-point communications a 3.4 Gb 3D mesh network is used (for more details, see http://www.scc.acad.bg/ and http://hpc.uvt.ro/infrastructure/bluegenep/). In our experiments, to compile the code we have used the IBM XL C compiler and compiled the code with the following options: "-O5 -qstrict -qarch=450d -qtune=450 -qsmp=omp". To use the BLAS subroutines, we linked our code to the multi-threaded ESSL library.

TABLE IV
EXECUTION TIME FOR 3D DISCRETE COSINE TRANSFORM ON IBM BLUE GENE/P.

N	nodes										
	1	2	4	8	16	32	64	128	256	512	1024
	single precision										
	forward transform										
100	0.09	0.06	0.05	0.04	0.03	0.02	0.02	0.02	0.01	0.01	0.01
200	1.01	0.62	0.38	0.24	0.17	0.10	0.06	0.05	0.04	0.03	0.03
300	4.45	2.43	1.73	1.05	0.59	0.32	0.18	0.12	0.08	0.06	0.05
400	14.40	7.69	4.77	2.64	1.47	0.77	0.42	0.27	0.17	0.10	0.08
600	70.59	36.30	19.09	10.87	6.29	3.33	1.79	1.13	0.54	0.31	0.19
800	*	117.36	57.54	33.19	18.21	9.39	4.91	2.83	1.35	0.75	0.43
1000	*	*	140.94	75.20	44.04	22.93	11.94	7.15	3.46	1.88	1.09
1200	*	*	*	139.73	76.14	40.04	20.30	11.95	5.93	3.20	1.95
1400	*	*	*	*	135.50	70.74	38.77	22.47	11.29	6.03	3.23
1600	*	*	*	*	229.10	120.66	63.27	35.68	17.12	9.10	5.04
	backward transform										
100	0.09	0.05	0.04	0.03	0.02	0.01	0.01	0.01	0.01	0.01	0.01
200	1.01	0.63	0.38	0.24	0.17	0.09	0.05	0.04	0.03	0.02	0.02
300	4.48	2.48	1.83	1.06	0.60	0.32	0.17	0.11	0.07	0.05	0.03
400	14.49	7.92	4.87	2.72	1.51	0.78	0.41	0.26	0.16	0.09	0.07
600	70.91	36.59	20.04	11.26	6.44	3.34	1.77	1.14	0.55	0.31	0.18
800	*	118.42	59.93	34.29	18.77	9.44	5.00	2.89	1.35	0.76	0.44
1000	*	*	146.71	77.66	44.82	22.49	11.88	7.27	3.48	1.93	1.11
1200	*	*	*	143.08	78.76	40.51	20.29	12.18	5.94	3.22	1.98
1400	*	*	*	*	140.26	72.08	38.59	22.61	11.37	6.04	3.28
1600	*	*	*	*	236.68	120.59	63.96	36.31	17.45	9.18	5.20
	double precision										
	forward transform										
100	0.10	0.07	0.05	0.05	0.04	0.03	0.02	0.02	0.01	0.01	0.01
200	1.15	0.72	0.42	0.26	0.18	0.10	0.07	0.06	0.05	0.03	0.03
300	4.97	2.80	1.84	1.07	0.60	0.35	0.21	0.13	0.11	0.08	0.06
400	16.27	8.89	5.02	2.77	1.50	0.84	0.47	0.28	0.20	0.13	0.11
600	*	39.37	20.28	11.34	6.53	3.48	1.91	1.17	0.66	0.39	0.25
800	*	*	66.34	35.33	18.89	10.00	5.34	2.94	1.60	0.91	0.56
1000	*	*	*	85.23	47.80	25.34	13.45	7.65	4.05	2.28	1.30
1200	*	*	*	*	78.03	41.22	22.15	12.31	6.94	3.80	2.32
1400	*	*	*	*	*	85.32	44.44	25.87	13.90	7.54	3.82
1600	*	*	*	*	*	147.61	70.00	36.61	19.86	10.74	5.88
	backward transform										
100	0.10	0.06	0.05	0.03	0.02	0.01	0.01	0.01	0.01	0.01	0.01
200	1.16	0.73	0.43	0.26	0.16	0.09	0.05	0.05	0.03	0.02	0.02
300	5.01	2.83	1.87	1.09	0.61	0.33	0.19	0.12	0.10	0.06	0.04
400	16.33	9.01	5.12	2.87	1.52	0.84	0.46	0.28	0.17	0.10	0.09
600	*	39.78	20.65	11.70	6.65	3.53	1.93	1.18	0.66	0.39	0.25
800	*	*	68.21	36.05	19.00	10.20	5.50	3.05	1.60	0.93	0.57
1000	*	*	*	87.68	48.31	24.94	13.41	7.71	4.11	2.27	1.30
1200	*	*	*	*	80.46	42.39	22.17	12.32	7.12	3.86	2.35
1400	*	*	*	*	*	86.05	45.25	25.79	14.12	7.74	3.84
1600	*	*	*	*	*	135.85	70.88	37.47	20.28	10.86	6.04

Here, again the execution time for $N = 100$ shows that the code can be executed on one node and it is not necessary to use the parallel algorithm. Note that the memory of a single node of the IBM supercomputer is substantially smaller than that on the Galera cluster and is not sufficient for solving large problems. While both BG/P machines have the same processors, the one located in Romania has larger memory (with 4 GB memory per node). This is thus the machine used to run experiments with larger data sets. Due to the lack of space, and relative similarity of results, we do not report results obtained on both machines separately (when running problems of the same size). Note that individual processors on supercomputer are slower than these on the Galera cluster. For the double precision DFT the Blue Gene is approximately three times slower than the Galera.

Let us also observe that almost the same time was spent solving the problem of size $N = 600$ on a single node as it was spent when solving problem of size $N = 1600$ on 64 nodes. This indicates that the BG/P is more efficient in supporting parallel computing than the Galera cluster.

Table VI shows the speed-up obtained on the Blue Gene. Because of smaller memory per node we calculated the actual speed-up only for $N = 100, 200, 300, 400$. Furthermore, only for the single precision DCT the speed-up for $N = 600$ is reported. For $N = 400$ the parallel efficiency is more than 50% on up to 64 nodes for the DCT and on up to 512 nodes for the DFT.

An interesting observation comes from comparing results reported in Tables VI and VI, as well as those found in Tables II and V. For instance, in the most complex problem (where such

TABLE V
EXECUTION TIME FOR 3D DISCRETE FOURIER TRANSFORM ON IBM BLUE GENE/P.

N	nodes										
	1	2	4	8	16	32	64	128	256	512	
	single precision										
	forward transform										
100	0.25	0.15	0.09	0.06	0.05	0.03	0.02	0.02	0.02	0.01	0.01
200	3.65	1.94	1.05	0.58	0.33	0.18	0.11	0.07	0.06	0.04	0.03
300	17.95	9.47	5.28	2.83	1.52	0.80	0.45	0.25	0.17	0.10	0.08
400	55.63	28.70	14.60	7.69	4.06	2.13	1.13	0.62	0.34	0.20	0.14
600	*	140.97	72.54	37.82	19.51	10.20	5.27	2.97	1.58	0.89	0.49
800	*	*	223.40	113.48	57.70	29.60	15.04	7.99	4.18	2.22	1.22
1000	*	*	*	276.03	268.76	72.45	36.94	20.16	11.15	6.01	3.26
1200	*	*	*	*	287.60	146.05	74.36	38.99	20.12	10.52	5.90
1400	*	*	*	*	*	270.33	137.73	74.55	39.40	21.26	10.33
1600	*	*	*	*	*	446.59	226.00	115.32	58.73	30.08	15.87
	backward transform										
100	0.25	0.15	0.08	0.05	0.04	0.02	0.02	0.01	0.01	0.01	0.01
200	3.66	1.95	1.09	0.60	0.32	0.17	0.10	0.06	0.05	0.03	0.03
300	18.03	9.54	5.33	2.88	1.54	0.81	0.44	0.25	0.16	0.09	0.07
400	55.79	28.83	14.76	7.79	4.14	2.13	1.13	0.63	0.34	0.19	0.13
600	*	141.86	73.07	38.26	19.81	10.23	5.32	2.96	1.60	0.88	0.49
800	*	*	224.63	114.12	58.22	29.86	15.28	8.10	4.28	2.25	1.25
1000	*	*	*	279.21	234.82	72.90	37.02	20.33	11.23	6.02	3.32
1200	*	*	*	*	289.29	146.65	75.11	39.30	20.49	10.59	5.98
1400	*	*	*	*	*	271.09	139.31	74.81	39.44	21.30	10.61
1600	*	*	*	*	*	446.08	228.34	116.13	58.99	30.17	16.20
	double precision										
	forward transform										
100	0.29	0.18	0.11	0.07	0.05	0.04	0.03	0.02	0.02	0.02	0.02
200	3.98	2.19	1.17	0.67	0.37	0.21	0.13	0.08	0.06	0.04	0.04
300	19.06	10.39	5.62	3.08	1.68	0.91	0.52	0.30	0.18	0.11	0.09
400	61.59	32.04	16.59	8.77	4.53	2.43	1.35	0.73	0.40	0.24	0.15
600	*	*	79.35	41.38	21.69	11.25	6.09	3.22	1.78	1.01	0.59
800	*	*	*	125.18	63.35	33.33	17.23	9.00	4.80	2.63	1.40
1000	*	*	*	*	152.19	78.01	41.13	21.59	11.52	6.21	3.48
1200	*	*	*	*	*	157.86	81.49	42.20	22.05	11.80	6.46
1400	*	*	*	*	*	*	152.59	78.99	41.33	22.14	10.87
1600	*	*	*	*	*	*	253.92	128.75	65.41	33.98	17.91
	backward transform										
100	0.28	0.17	0.11	0.06	0.04	0.03	0.02	0.01	0.02	0.01	0.02
200	3.99	2.21	1.19	0.67	0.37	0.21	0.12	0.07	0.05	0.03	0.03
300	19.13	10.44	5.69	3.13	1.66	0.91	0.51	0.30	0.17	0.10	0.07
400	61.82	32.24	17.12	8.92	4.63	2.46	1.34	0.72	0.39	0.23	0.16
600	*	*	80.27	41.79	21.96	11.39	6.01	3.34	1.80	1.01	0.58
800	*	*	*	127.55	64.07	33.35	17.19	9.12	4.77	2.66	1.43
1000	*	*	*	*	153.40	78.62	41.46	21.56	11.57	6.19	3.47
1200	*	*	*	*	*	156.72	81.52	43.23	22.79	12.18	6.55
1400	*	*	*	*	*	*	152.42	79.23	41.22	22.18	11.00
1600	*	*	*	*	*	*	257.88	131.82	66.12	34.10	18.32

comparison was possible), for the backward double precision DFT, for $N = 400$ and 256 nodes, speedup obtained on Galera is 43, while on the BG/P it reaches 157. Furthermore, for the same problem (backward double precision DFT) the execution time on Galera on 256 nodes is 18 seconds, which is almost exactly the time needed to compute the same problem on 1024 nodes of the BG/P. Overall, this indicates that, in the case of the BG/P, somewhat slower nodes have been combined with superior network infrastructure, which is exactly the opposite than in the case of the Galera cluster (where more powerful processors are connected through a slower network).

Finally, in Figure 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 $N = 400$ and $N = 600$. Here, it becomes even clearer that for both problems, using more than 64 nodes on the Galera cluster results in, so called, Amdahl's effect (where adding more resources does not result in a commensurate time reduction). This is not the case for the BG/P machines. Nevertheless, for up to 256 nodes, for $N = 600$, the cluster is faster in completing the task.

V. CONCLUDING REMARKS

The aim of this paper was to describe our attempt at implementing a slightly simplified version of a novel algorithm for 3D forward/inverse discrete transforms, and to report its performance on two different parallel computers. Obtained results show that the proposed approach allows solution of large 3D problems on a supercomputer as well as on a cluster. Furthermore, the initial estimates indicate quite good scalability of the proposed implementation. It should be noted

TABLE VI
SPEED-UP ON IBM BLUE GENE/P.

N	nodes									
	2	4	8	16	32	64	128	256	512	1024
	single precision DCT									
	forward transform									
100	1.57	1.97	2.51	2.76	3.95	4.56	5.13	6.84	16.40	15.43
200	1.62	2.67	4.29	5.89	10.26	16.76	20.42	24.84	29.33	35.17
300	1.83	2.57	4.25	7.53	13.82	24.36	37.75	52.39	78.08	88.86
400	1.87	3.02	5.45	9.78	18.58	34.42	53.36	87.08	140.70	180.09
600	1.94	3.70	6.50	11.22	21.20	39.47	62.53	130.89	228.25	372.52
	backward transform									
100	1.58	2.07	2.97	3.78	6.80	9.29	10.69	13.50	15.14	14.33
200	1.61	2.64	4.18	5.91	11.03	19.81	27.45	35.69	52.30	56.93
300	1.80	2.45	4.24	7.52	13.96	25.90	41.00	66.22	95.26	133.22
400	1.83	2.98	5.33	9.58	18.60	35.57	54.84	89.59	156.33	218.11
600	1.94	3.54	6.30	11.01	21.22	40.11	62.13	129.60	230.57	388.31
	double precision DCT									
	forward transform									
100	1.44	1.99	2.17	2.84	3.75	4.58	5.45	8.54	14.48	15.43
200	1.60	2.72	4.46	6.52	11.00	17.46	20.23	23.82	32.91	39.41
300	1.78	2.70	4.64	8.22	14.38	24.21	37.95	45.96	65.02	86.02
400	1.83	3.24	5.88	10.88	19.29	34.97	58.15	82.13	128.29	154.47
	backward transform									
100	1.52	2.15	2.77	3.94	6.73	10.06	10.26	7.04	13.75	14.75
200	1.60	2.68	4.49	7.38	13.33	22.49	24.98	38.40	53.37	51.18
300	1.77	2.67	4.60	8.22	15.01	26.68	41.23	47.99	80.81	114.23
400	1.81	3.19	5.70	10.74	19.38	35.45	58.13	94.07	160.43	172.13
	single precision DFT									
	forward transform									
100	1.66	2.91	4.06	5.11	7.90	11.60	11.96	12.87	21.17	22.00
200	1.89	3.47	6.25	11.16	20.41	34.17	49.06	61.54	92.30	110.87
300	1.89	3.40	6.34	11.78	22.36	39.70	71.21	104.45	173.67	222.25
400	1.94	3.81	7.23	13.69	26.16	49.44	90.28	162.72	281.33	404.33
	backward transform									
100	1.70	3.17	4.58	6.08	10.38	16.00	22.77	18.99	20.82	20.72
200	1.88	3.37	6.14	11.54	21.50	38.36	56.38	74.31	110.65	141.26
300	1.89	3.38	6.27	11.67	22.20	40.56	73.45	111.63	202.70	263.55
400	1.93	3.78	7.16	13.47	26.13	49.31	88.81	165.38	293.04	427.32
	double precision DFT									
	forward transform									
100	1.62	2.56	4.05	5.75	7.86	11.45	13.54	13.19	11.92	15.16
200	1.82	3.38	5.96	10.77	18.85	31.19	49.06	61.22	89.00	99.92
300	1.83	3.39	6.19	11.34	20.90	36.34	64.60	103.16	168.98	220.23
400	1.92	3.71	7.02	13.61	25.32	45.66	84.68	153.37	260.39	407.95
	backward transform									
100	1.66	2.67	4.37	6.65	11.66	17.58	20.57	18.39	20.99	15.04
200	1.81	3.37	5.93	10.90	19.39	33.92	54.65	77.85	117.11	140.65
300	1.83	3.37	6.11	11.49	20.95	37.25	64.82	111.71	186.97	277.44
400	1.92	3.61	6.93	13.35	25.16	46.24	85.44	157.80	270.35	397.71

that the code was tested on the machines in case of which we deal with a discrepancy between the physical layout of the computing nodes and the layout assumed by the method. Nevertheless, we believe that the initial results are encouraging enough to continue work. Here, the first step will be to perform more involved testing of the performance to establish performance profile (especially for the largest problems). We also plan to investigate the performance on the cluster utilizing Intel Phi coprocessors.

ACKNOWLEDGMENTS

Computer time grants from the TASK computing center in Gdansk, Poland, the Bulgarian Supercomputing Center, and the HPC Center from West University of Timisoara are kindly acknowledged. This research was partially supported by grants DCVP 02/1 and I01/5 from the Bulgarian NSF. Work presented here is a part of the Poland-Bulgaria collaborative grant "Parallel and distributed computing practices." Work of Marcin Paprzycki was completed in part, while he was visiting University of Aizu.

REFERENCES

[1] O. Ayala and L.P. Wang. Parallel implementation and scalability analysis of 3D fast Fourier transform using 2D domain decomposition. *Parallel Computing*, 2012. DOI: 10.1016/j.parco.2012.12.002

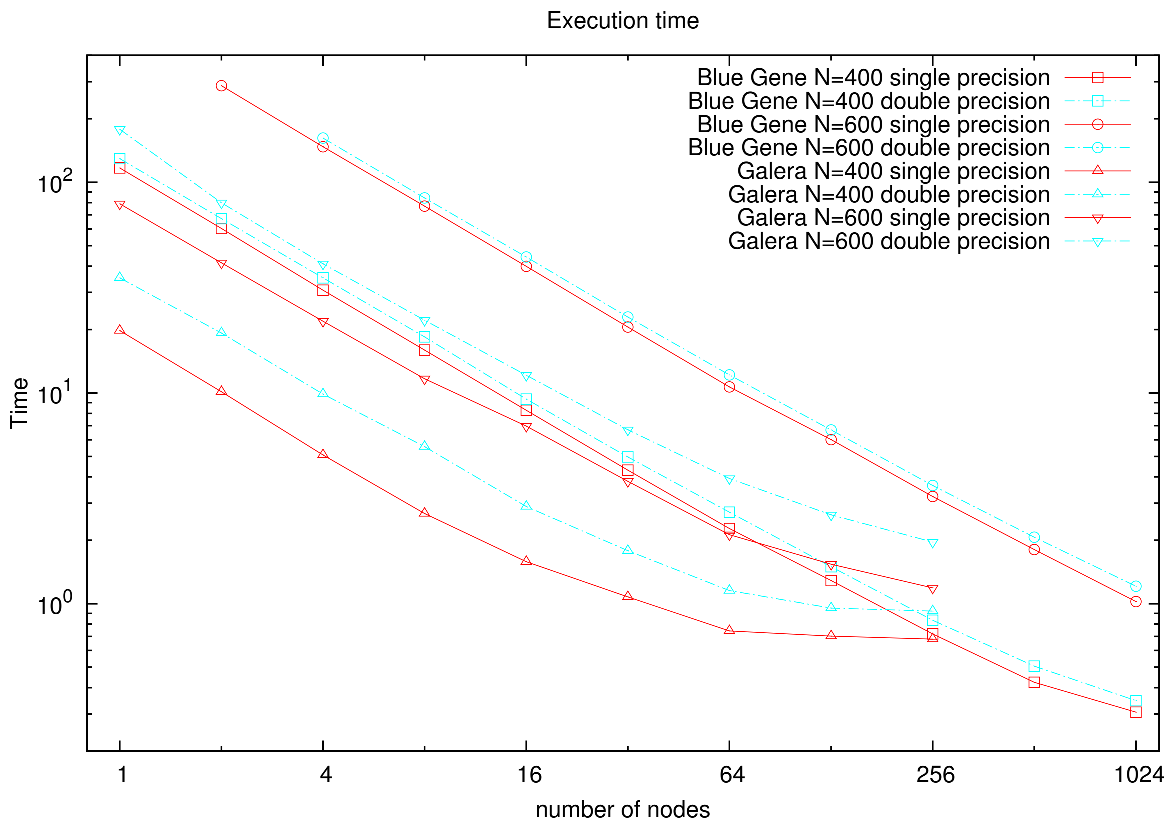

Fig. 1. Execution time of code which performs forward and backward DFT for $N = 400, 600$.

[2] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Iain S. Duff. A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software (TOMS)*, 16(1):1–17, 1990. DOI: 10.1145/77626.79170

[3] Maria Eleftheriou, José E. Moreira, Blake G. Fitch, and Robert S. Germain. A volumetric FFT for Blue Gene/L. In Timothy Mark Pinkston and Viktor K. Prasanna, editors, *High Performance Computing - HiPC 2003*, volume 2913 of *Lecture Notes in Computer Science*, pages 194–203. Springer Berlin Heidelberg, 2003. DOI: 10.1007/978-3-540-24596-4_21

[4] Ning Li and Sylvain Laizet. 2DECOMP&FFT - A Highly Scalable 2D Decomposition: Library and FFT Interface. In *Cray User Group 2010 conference*, pages 1–13, 2010.

[5] Stanislav G. Sedukhin, Co-design of Extremely Scalable Algorithms/Architecture for 3-Dimensional Linear Transforms, Technical Report TR2012-001, The University of Aizu, July 2012.

[6] Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J.: MPI: The Complete Reference, second edition, Volume 1, The MPI Core. Scientific and engineering computation series, The MIT Press, Cambridge, Massachusetts (1998), ISBN: 9780262692151

[7] Walker, D., Dongarra, J.: MPI: a standard Message Passing Interface. Supercomputer, 12 (1), 56–68 (1996), ISSN 0168-7875