Supercomputing Frontiers and Innovations

2016, Vol. 3, No. 1

Scope

- Enabling technologies for high performance computing
- Future generation supercomputer architectures
- Extreme-scale concepts beyond conventional practices including exascale
- Parallel programming models, interfaces, languages, libraries, and tools
- Supercomputer applications and algorithms
- Distributed operating systems, kernels, supervisors, and virtualization for highly scalable computing
- Scalable runtime systems software
- Methods and means of supercomputer system management, administration, and monitoring
- Mass storage systems, protocols, and allocation
- Energy and power minimization for very large deployed computers
- Resilience, reliability, and fault tolerance for future generation highly parallel computing systems
- Parallel performance and correctness debugging
- Scientific visualization for massive data and computing both external and in situ
- Education in high performance computing and computational science

Editorial Board

Contents

DOI: 10.14529/jsfi160101

NR-MPI: A *N*on-stop and Fault *R*esilient *MPI* Supporting Programmer Defined Data Backup and Restore for E-scale Super Computing Systems

***Guang Suo*[1], *Yutong Lu*[1], *Xiangke Liao*[2], *Min Xie*[1], *Hongjia Cao*[1]**

Fault resilience has became a major issue for HPC systems, particularly, in the perspective of future E-scale systems, which will consist of millions of CPU cores and other components. MPI-level fault tolerant constructs, such as ULFM, are being proposed to support software level fault tolerance. However, there are few systematic evaluations by application programmers using benchmarks or pseudo applications. This paper proposes NR-MPI, a *N*on-stop and Fault *R*esilient *MPI*, supporting programmer defined data backup and restore. To help programmers write fault tolerant programs, NR-MPI provides a set of friendly programming interfaces and a state transition diagram for data backup and restore. This paper focuses on design, implementation and evaluation of NR-MPI. Specifically,this paper puts emphases on failure detection in MPI library, friendly programming interface extending for NR-MPI and examples of fault tolerant programs based NR-MPI. Furthermore, to support failure recovery of applications, NR-MPI implements data backup interfaces based on double in-memory checkpoint/restart. We conduct experiments with both NPB benchmarks and Sweep3D on TH supercomputer in NSCC-TJ. Experimental results show that NR-MPI based fault tolerant programs can recover from failures online without restarting, and the overhead is small even for applications with tens of thousands of cores.

Keywords: Message passing interface, fault tolerant MPI, NR-MPI, Application-level Checkpoint/Restart..

Introduction

Large scale scientific applications have been the main driving force for high-performance computing. Scientists need to analyze ever-larger data set and to run ever-larger simulations, which drives the scale of high-performance computers, growing to millions of processor cores. In the future, extreme scale high-performance computers will consist of even more cores. From the top500 [1] list of November 2015, there are 3120000 cores in the rank of 1 supercomputer. With the expansion of computer systems, the failure rate is also increasing. So the mean time between failures (MTBF) is decreasing. However, many scientific applications need to run for weeks or even months. Therefore, the MTBF of these computers is becoming significantly shorter than the execution time of many current scientific applications. To support the execution of such applications, fault tolerance is imperative [2].

The lack of appropriate resilience solutions is a major problem at exascale. Currently the MPI forum is working on a new fault tolerant MPI standard. Additional MPI-level constructs will be added into the future MPI standard. The most promising way is User Level Failure Migration(ULFM) [3]. However, there is still lack of enough experimental results to proof both the usability and performance of ULFM.

In this paper, we present NR-MPI, a Non-stop and Fault Resilient MPI. The semantics of NR-MPI is derived mainly from FT-MPI and ULFM. The programming interface, which is designed for iteration based on scientific parallel applications, is less complicated than FT-MPI

[1]State Key Laboratory of High Performance Computing, National University of Defense Technology, Changsha, Hunan Province, China
[2]Science and Technology on Parallel and Distributed Processing Laboratory, National University of Defense Technology, Changsha, Hunan Province, China

and ULFM. The interface is more suitable for programmers, who just put emphasis on which data to be backuped and when to backup, to convert a parallel application into a fault tolerant one. The failure detector of NR-MPI is different from ULFM. ULFM detects the process failures in the MPI library, while NR-MPI relies on external failure detector which is usually integrated with process manager or resource manager. We implemented some fault tolerance of ULFM based on MPICH. The implementation of NR-MPI is based on MPICH [4]. NR-MPI has no runtime overhead when there are no failures.

This paper focuses on the design, implementation and evaluation of NR-MPI, which is implemented on top of ULFM. The MPI Forum has not reached a consensus on the principles of a resilient MPI, although ULFM is discussed a lot. However, we think that the following issues are important when implementing a fault tolerant MPI.

- How to detect failures in a MPI library.
- How to recover the state of a MPI library based on ULFM.
- How to recover the lost application data after failures using NR-MPI.
- What programming interfaces are needed in order to reduce the complexity of fault tolerant programming.
- How to convert a non-fault tolerant program into a fault tolerant one by NR-MPI.

The rest of this paper is organized as follows: Section 1 presents the design and implementation of NR-MPI. Section 2 shows the usage of programming interface based an example algorithm. Section 3 evaluates the performance of NR-MPI with NPB benchmarks. Section 4 gives the related work of NR-MPI. In Section 4.2, we conclude this paper and discuss the future work.

1. Design and Implementation of NR-MPI

NR-MPI is designed on top of ULFM and implemented based on MPICH [4]. Traditional ULFM is implemented based on OpenMPI. We use MPICH instead of OpenMPI because the software stack of the high performance network is on top of MPICH. The fault tolerant RMS is modified based on SLURM [5].

1.1. Failure Model

In this paper, we assume a failure model in which fail-stop failures can occur anytime in any process during a parallel execution. There are two types of failure models: fail-stop model and byzantine model. Fail-stop failures can be detected more easily. In fact, byzantine failures can be detected by error checking based on ABFT, which can also be implemented based on NR-MPI.

1.2. The Fault Tolerant RMS of NR-MPI

The Resource Management System, for example SLURM, is developed to manage and monitor parallel applications running on a cluster of computers. It is designed to coordinate a global and consistent system state upon failures. According to roles and locations, the RMS can be divided into two parts: Resource Manager and Process Manager, shown in fig. 1. Furthermore, we add Failure Arbiter (FA) and Failure Detector (FD) to them respectively. The functions of the fault tolerant RMS are: fault tolerant resource management, failure detection and notification.

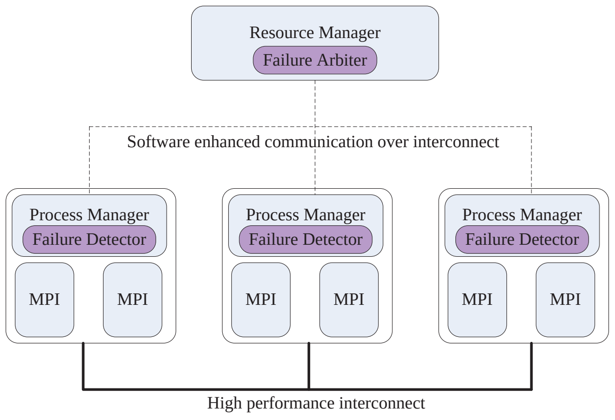


Figure 1. Structure of the resource management system of NR-MPI

For NR-MPI, failures are detected by the fault tolerant RMS. To detect fail-stop failures, we add FDs in Process Managers of RMS in computing nodes and add FA in Resource Manager. FDs can detect process failures of parallel jobs using the `SIGCHLD` signal. FA uses a periodic heartbeat to detect failures from FDs. In this way, FA can detect all process failures of parallel jobs. There are two advantages, if the FDs and FAs are integrated into RMS. Firstly, they can be light weighted so as to not interfere with the performance of jobs. Secondly, FD and FA can make use of the fault tolerant techniques already implemented in the RMS. For example, there are two active Resource Managers on line. One is active while the other one is standing by, so that failures of one Resource Manager don't influence the availability of the system.

Based on the communication topology of Resource Manager and Process Manager, the communication topology of failure detecting system is a tree too. Root of the tree is FA; inter-mediate and leaf nodes are FDs. Shared memory is used for the communication between FD and MPI processes in the same node. Software enhanced communication network for high performance computing used for the communications between FD and FD (or FA), so that the heartbeat and failure notification messages can be unfailingly transferred. In addition, to recover sucessfully, the failure notification messages should be received by all the MPI processes of a parallel job in the same order.

During MPI communications, the NR-MPI library needs to check the failure notification messages by reading the contents of shared memory, when sending or receiving data. For example, based on MPICH, NR-MPI checks the failure notification messages in progress engine.

The contents of the failure notification message are the list of failed ranks of the parallel job. This failure information can help the programmers to recover from process failures, node failures and network failures. When a node crashes, the message contains all the ranks belonging to the program on that node. When a part of network fails, the message contains all the ranks of processes which FA can't communicate with due to the network failure.

1.3. Recovery of NR-MPI Library

Main job of recovering the NR-MPI Library is to recover the communicators. We take MPI_COMM_WORLD as an example, shown in fig. 2, to explain how to recover it inside the library. The world communicator recovery is to repair the corrupted attributes. In fact, old communication context is OK for the repaired communicator. So recovering group and virtual connection table is the main job. It can be done together by using mainly existing MPI procedures.

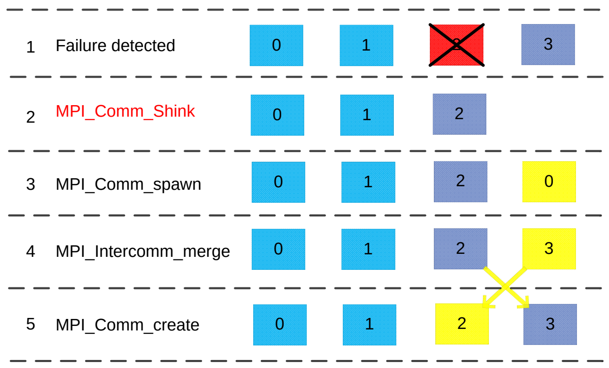


Figure 2. Recovery process of a communicator

The steps of world communicator are as follows.

(1)Failure detection. When failures occur, the other alive ranks enter the failure recovery process as soon as they receive failure information from FDs.

(2)Shinking the communicator. The alive ranks shink the failed ranks from world communicator by calling MPI_Comm_shink. The MPI_Comm_shink operation is defined in ULFM. It will exclude the failed ranks from a failed communicator.

(3)Spawning replacements. The world communicator spawns processes as replacements for the failed ones by calling MPI_Comm_spawn. Note that the spawned processes are in a different process group. They communicate via an inter-communicator.

(4)Merging the inter-communicator. Constructing a new world group from the old world group and the spawned group. Note that the order of new world group is different from the world group before the failures occur.

(5)Reordering rank in a new world group. This can be done by calling MPI_Comm_create.

Most of the steps, except MPI_Comm_shink, of world communicator recovery are based on existing MPI standard version 2. Meanwhile, the recovery process does not manipulate virtual connection table directly. So the complexity of implementing NR-MPI is relatively low.

1.4. State Transition Diagram with Failures

Upon failures, the lost application data also need to be recovered. C/R, ABFT, application level data backup via MPI communications or combination of the above can be used to recover the lost data. In this paper, we don't assume a specific data backup and restore technique. Instead, we define a data backup and recovery protocol for NR-MPI, so that the data backup and recovery techniques defined by programmers can be integrated with NR-MPI as a whole.

During the execution of a NR-MPI parallel program, any processes may fail. In addition, the failures may occur when recovering MPI data, or application data. Meanwhile, a failure is recovered, if and only if all processes recovered their MPI data and application data. The data to recover is different for different kinds of processes at different state. To help programmers implementing NR-MPI parallel programs, we define a state transition diagram of a NR-MPI process, shown in fig. 3. There are 8 states in the diagram, defined in Table 1.

For any MPI parallel programs (including NR-MPI based programs), regardless of failures, they have four states: INIT, RUNNING, FINISH, and ABORT. The other 4 states are the

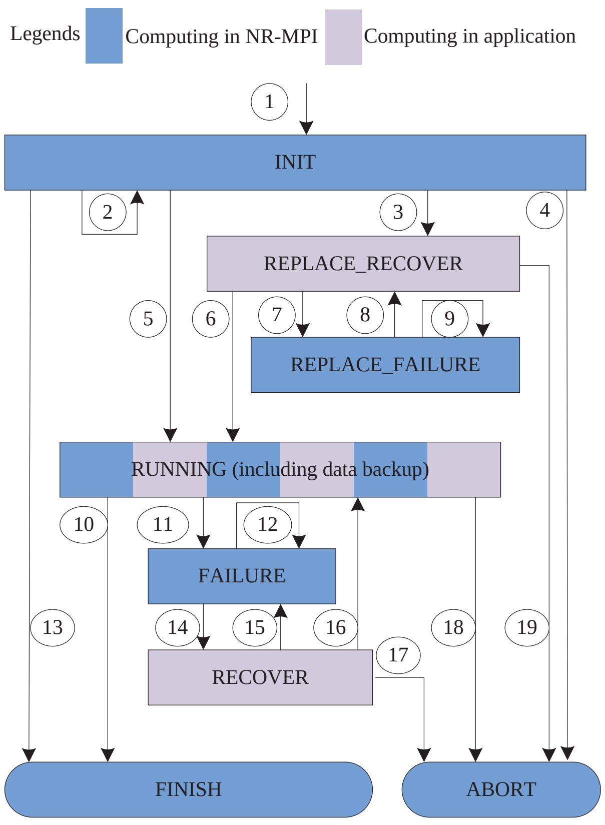


Figure 3. State transition diagram of a NR-MPI process

states to recover MPI core data, MPI extend data and application data for normal processes and replacements. The RUNNING state has two colors because it can be in both MPI library and programmers' codes.

Table 2 summarizes the interactions between the states in fig. 3. State Transfer 2, 9 and 12 are self-to-self, indicating that failures occur in INIT, FAILURE and REPLACE_FAILURE. It means that the stand-by process, the replacements and the normal processes find failures when they recover MPI core data respectively.

When data recovery is via MPI communications, failures can be detected in progress engine (the module that probes upcoming messages and sends queued messages) of MPI library. However, when there is no interaction between processes during data recovery, the processes won't enter REPLACE_RECOVER, so the state transfer 7, 8, 9 and 15 won't be triggered either.

1.5. Extending Interface of NR-MPI

Many parallel applications are iteration based, such as linear solvers and PDE based applications. We focus on fault tolerance for iteration based applications. To help programmers, NR-MPI provides a convenient programming interface which can reduce the burden of programmers, listed in Table 3.

NR_Register, NR_Backup, NR_Recover and NR_Need_-backup are used to backup and restore the application data based on the double in memory checkpoint [6]. Moreover, to prevent the damages of failures during the data backup and restore, we use ping-pong buffers to backup application data. NR_Backup and NR_Recover are simple examples for application level data backup and restore. There can be any data backup and restore techniques, and programmers can implement their own methods by overriding the 4 functions. NR_Get_state and NR_Set_state are

Table 1. STATE DEFINATION

States	Descriptions
INIT	When a process calls MPI_Init, it enters INIT state. In INIT state, the processes, including standby processes, initialize their internal MPI data. At the end of INIT, the normal processes can exit, while the standby processes are waiting and processing failure notification messages. The standby processes exit INIT when they are selected as replacements upon failures or the program exits.
RUNNING	If the return status of MPI_Init indicates that initialization is OK, or if a replacement process recovers its lost data successfully, the process enters RUNNING state. A process can do user computation, user defined MPI communication, and backup application data (via C/R, ABFT, or MPI communication) in RUNNING state.
FINISH	When a process calls MPI_Finalize, it enters FINISH state. FINISH state is an absorbing state.
ABORT	Whenever a process finds unrecoverable failures, it enters ABORT state. To simplify the diagram, we only draw 4 state transfers to ABORT. ABORT is also an absorbing state.
FAILURE	When a normal process in RUNNING state finds failures in MPI library, it enters FAILURE state. This state is hidden in MPI library; the work in this state is to recover MPI core data
RECOVER	After recovering MPI core data, a process enters RECOVER state. The work in this state is to recover MPI extend data and application data by programmers. If successful, the programmers should alter the state of the local process to RUNNING explicitly.
REPLACE_ RECOVER	When a replacement process exits from INIT, it needs to recover MPI extend data and the lost application data. If recovering successfully, programmers need to alter the state of the local process to RUNNING explicitly.
REPLACE_ FAILURE	When failure occurs during REPLACE_RECOVER, the process enters REPLACE_FAILURE. This state is hidden in MPI library; the work in this state is to recover the MPI core data.

used to get and set the state of the local process. NR_Get_failure_ranks is used to query failed rank set in NR_Recover.

1.6. Implementation of the NR-MPI

The structure of NR-MPI is shown in fig. 4. Clearly, NR-MPI is in the middle of the software stack. The fault tolerant RMS is modified based on slurm-2.4.0-rc1[XX]. Many different MPI implementations have been developed to support different supercomputers efficiently. NR-MPI can integrate with a wide range of existing MPI implementations. In this paper, NR-MPI is modified based on mpich2-1.4.1p1[21], integrated with state management module, data backup and restore module, failure detecting module, and failure recovery module. Their functions are as follows: state management module provides a state managing interface for programmers. From state management module, programmers can query or set its own fault tolerant state, which is essential for NR-MPI. Data backup and restore module provide the programming interface based on mutual data backup to save the application data. Failure detecting module can query failures

Table 2. STATE TRANSFER DEFINATION

ID	State Transfer Descriptions
1	When a progress calls MPI_Init.
2	When failures occur during the creation of MPI core data for a replacement process. Or when stand-by processes detect failures.
3	When a replacement exits from MPI_Init, it enters REPLACE_RECOVER.
4,17,18,19	When an unrecoverable failure occurs.
5	When a normal process exits from MPI_Init, it enters RUNNING.
6	When a replacement finishes recovering its app data.
7	When a replacement encounters failures during the process of recovering app data, it enters REPLACE_FAILURE.
8	When a replacement recovers its MPI core data.
9	When a replacement encounters failure during the process of recovering of its MPI library data
10	When a normal process calls MPI_Finalize.
11	When a normal process encounters a failure.
12	When a failed normal process encounters failure during the process of recovering of MPI core data
13	When all normal processes enter FINISH, the stand-by processes enter FINISH.
14	When a failed normal process finishes recovering its MPI library data
15	When a normal process encounters failures during the process of recovering of app data
16	When a normal process finishes recovering the app data.

Table 3. PROGRAMMING INTERFACE

Interface	Descriptions
NR_Register	register the data to be backed up
NR_Backup	backup the registered data
NR_Recover	recover the lost data which had been backed up
NR_Need_backup	determine whether data backup is needed
NR_Get_state	get the state of the caller process
NR_Set_state	set the state of the caller process
NR_Get_failure_ranks	get the set of ranks failed in last failure

of the system efficiently from the external failure detector. When the processes are spawned by process manager, the process manager creates a shared memory used to pass failure notification messages and adds FD_SHMID=shmid into the environmental variables of spawned processes. Failure recovery module can recover the corrupted MPI core data into a consistent state.

2. NR-MPI USAGE EXAMPLE

Based on NR-MPI, the programmers just add two sections to a non-fault tolerate program to make it fault tolerant, without changing other codes of the program. The function of data backup segment is to save the application data periodically in case of failures, while the function of data restore segment is to restore the lost application data of failed processes.

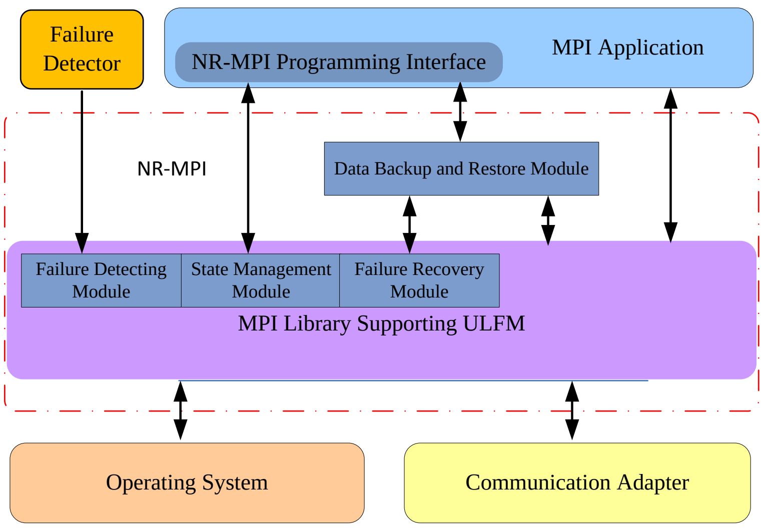


Figure 4. Structure of NR-MPI

We take the conjugate gradient (CG) algorithm as an example to illustrate the usage of the programming interface, shown in fig. 5. NR-CG is the fault tolerance version of CG algorithm based on NR-MPI. To support fault tolerance, we added data backup and restored codes. Lines of 19∼29(data restore segment) and 39∼42(data backup segment) are the additional codes.

From the algorithm of NR-CG (lines 33∼38), we can see that vector x and iteration index j are the data to be backed up. So after initialization of MPI library, NR-CG registers x and j to NR-MPI (line 11∼12). Then, NR-CG sets the return position env, so that NR-CG can switch from MPI library to the position user defined in the program after the world communicator has been recovered in a failure (line 13∼14). The callback function *cg_callback* is called by NR-MPI after recovering the MPI core data. In line 15, NR-CG gets the current state of itself. Then, it sets x and *j_start* based on the program state. If the state is RECOVER or REPLACE_RECOVER, the program needs to restore the application data. For the two states, the codes are the same. However, they may be different for ABFT. Lines 39∼42 are used to back up data. Usually it is not necessary to back up data per iteration, so NR_Need_backup is called to determine whether the backup is needed.

In this example, NR-CG uses longjmp to get to the user-defined position in the program. In fact, NR-MPI can also return FAILURE status like the interfaces defined in FT-MPI, so that setjmp and longjmp can be omitted. We didn't follow the style of returning error codes, because more modifications are needed. Furthermore, programmers can call any MPI routines after recovering MPI core data. For example, MPI_Comm_create can be used to create a new communicator, which doesn't contain the recovered ranks, like shrink operation in FT-MPI and OpenMPI.

After receiving notification from FDs, the survival processes recover MPI core data before calling *cg_callback*. In *cg_callback*, the default action of *cg_callback* is jumping to line 13. *longjmp* can clear the calling stack of MPI library. At this moment, the state of the program is RECOVER. So lines 26∼29 are executed by the survival processes, they will help the failed processes to restore their lost application data, when they find their partner is failed. Meanwhile, the replacements exit from INIT and run to line 15. Their states are REPLACE_RECOVER, so they get x and *j_start* from their partners (lines 22∼25). At last, they set their states to RUNNING.

```
void cg_callback(jmp_buf env){
    if (env) longjmp(env, 1);
    else MPI_Abort();
}
int main(){
    int j, j_start, j_end, rank, jhandle, xhandle, error, state;
    float a, b, c;
    float x[L], r[L], b[L], p[L], A[L,L];
    MPI_Init();
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    jhandle = NR_Register(&j, partner);
    xhandle= NR_Register(&x, partner);
    error = setjmp(env);
    NR_Set_callback(cg_callback, &env);
    NR_Get_state(&state);
    if (state == RUNNING){
        j_start=0;
        x=initial value; }
    if (state == ABORT) {
        MPI_Abort(MPI_COMM_WORLD, error);
        return 0;}
    if (state == REPLACE_RECOVER) {
        NR_Recover(jhandle, &j_start);
        NR_Recover (xhandle, &x);                Data Restore
        NR_Set_state(RUNNING); }                   Section
    if (state == RECOVER) {
        NR_Recover (jhandle, &j_start);
        NR_Recover (xhandle, &x);
        NR_Set_state(RUNNING); }
    r=b - A x
    p= r
    for (j = j_start,…, j_end) {
        c=(r, r)                     sum = 0;
        a=c/(A p, p)                 for(i=1; i<L; i++)
        x= x +a p                      sum+=r[i]*r[i];
        r= r - a p                   MPI_Allreduce(&sum, &c,
        b=(r, r)/c                         MPI_FLOAT, 1, MPI_SUM,
        p= r + b p                         MPI_COMM_WORLD);
        if (NR_Need_backup(j)){
            NR_Backup( jhandle);                Data Backup
            NR_Backup(xhandle);                   Section
        }
    }
    MPI_Finalize();
    return 0;
}
```

Figure 5. Algorithm of NR-CG based on NR-MPI

Figure 6 shows an execution process of NR-CG upon a failure. There are 5 processes in NR-CG, 4 of them are normal process and the other one is the stand-by process. During execution, P0 and P2 backup the application data of each other, and P1 and P3 backup the application data of each other. At phase 1, the 4 processes backup their application data successfully. At phase 2, after backing up their application data, P2 crashed. Then the event, which P2 is crashed,

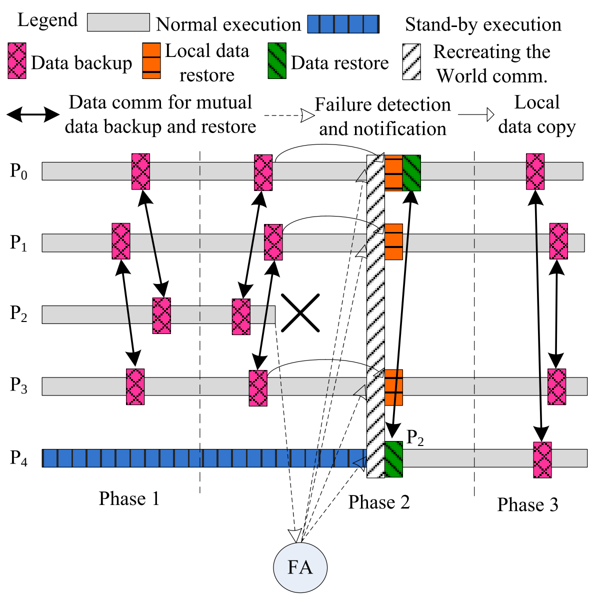


Figure 6. Execution example of NR-CG

is detected by FA. FA notifies the other 3 processes of the failure. The recovery process is as follows:

(0)P0, P1, P3 spawn a new process, which is named P4.

(1)P4 finds that the replacement of P2 is itself. Then P0, P1, P4 and P3 recover their MPI core data (creating a new world communicator).

(2)After recovering MPI core data, the 4 processes begin to recover the application data. P0, P1 and P3 recover their local application data to the last backing up version.

(3)P0 finds that its partner (P2) has failed by calling NR_Get_failure_ranks, so it sends application data of P2 to help it to recover. Meanwhile, the new P2 receives its lost application data.

(4)All of the four processes recover their data to the latest backing up version. And the NR-MPI parallel program recovers from a failure.

3. EXPERIMENTAL EVALUATION

To use NR-MPI, we have modified benchmarks from NAS Parallel Benchmarks [7] (version 3.3) and Sweep3D [8]. Our experimental platform, configuration of which is shown in Table 4, is TH-1A [9, 10], deployed in National Supercomputer Center in Tianjin.

Table 4. CONFIGURATION OF EXPERIMENT PLATFORM

Component	Configuration
CPU	X5670 CPU, 2.93GHz, 6 core, 2 CPUs per node
Memory	48GB
Compiler	Intel C++ Compiler 11.1
Interconnect	bi-bandwidth 160Gbps, MPI bandwidth 6340MB/s

There are two steps to modify a non-fault tolerant program. Firstly, to indentify main iteration and application data to backup. Secondly, to add data restore segment and data backup segment before and inside the main iteration, if necessary. Moreover, if a program has several phases, that is to say, it has several main iterations. For each one of the main iterations, the programmers need to identify application data, to add data restore and to backup segments respectively. So the modification complexity is very low. For example, we have added only a few additional codes to CG in NPB-3.3 to enable fault tolerant.

3.1. Runtime Overhead of Fault Resilient NPB Benchmarks

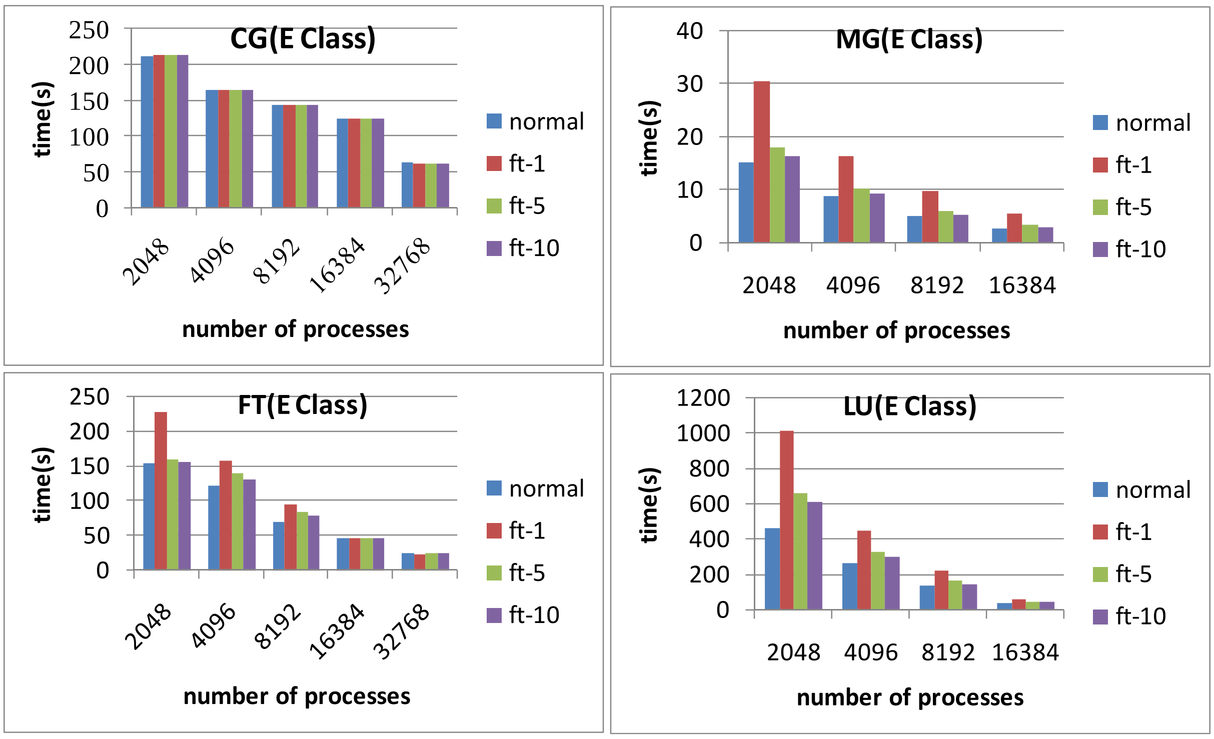


Figure 7. Runtime overheads of E class NPB benchmarks without failures

When there is no failure, the overhead of a MPI program without data backup and restore is the failure detection overhead. In our implementation, the failure detection overhead is almost zero based on experiments. So we don't provide the results of failure detection overhead. fig. 7 presents the execution time of non-fault tolerant and fault tolerant NPB benchmarks with NR-MPI. For fault tolerant version of the benchmarks, the application data is backed up every 1,5 and 10 iterations, using double in memory checkpoint. Normal results are the experimental results without data backup. ft-1 is the experimental result, in which the application data is backed up every 1 iteration. ft-5 is the experimental result, in which the application data is backed up every 5 iteration. ft-10 is the experimental results in which the application data is backed up every 10 iteration. The iteration intervals are different for different benchmarks, so time interval of data backup is different for different benchmarks.

From fig. 7, it can be found that the runtime overheads of NR-MPI parallel programs are different from different benchmarks. For NR-CG, the runtime is almost independent from the backup interval. The reason is that the data amount to be backed up is really small compared with the communications during the execution. However, for NR-MG, NR-FT and NR-LU, the overheads are higher than NR-CG. Take NR-LU as an example, the runtime of ft-1 is 122% higher than normal execution for 2048 processes. However, when the backup interval increases, the overhead due to the data backup is decreasing. For NR-LU, the runtime of ft-10 is by 33 higher than the normal execution for 2048 processes. NR-MG, NR-FT and NR-LU have higher runtime overhead, because the data to be backed up is larger. For example, when normal execution, the LU benchmark only exchanges the surface data of a data cube. However, NR-LU needs to backup the entire cube every 1, 5 or 10 iterations. So the runtime overhead of data backup is relatively higher than NR-CG. When the number of processes increases, the runtime overhead also decreases. For example, the runtime overhead of NR-LU is 15%, when the backup interval is 10 iterations for 16384 processes. The runtime overhead of NR-FT is 2%, when the backup interval is 10 iterations for 32768 processes.

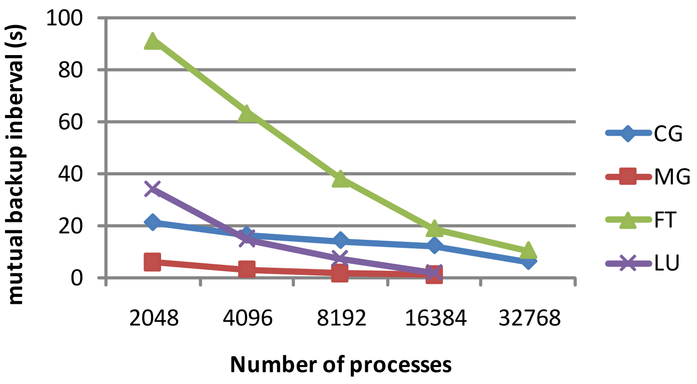


Figure 8. Backup intervals of E class NPB benchmarks

The backup interval (similar to the checkpoint intervals) is another important criterion for NR-MPI parallel programs, because it can be used to measure the expected lost computation due to one failure. Fig. 8 shows the backup of ft-10 intervals of E class NPB benchmarks. It can be found that the intervals are really small. For example, the highest interval is 90 seconds, for FT.E.2048, which means the average lost of computation is 45 seconds. For the other benchmarks, except NR-FT, the highest interval is 20 seconds.

3.2. Overall Time Overhead due to One Failure

NR-MPI is similar to SCR in data backup and restore. For example, SCR also supports mutual file backup when checkpoint files are stored in local disk or memory disk. In addition, when using SCR, the programmers also need to specify the variables to be saved, just like backing up key application data, when programmers use NR-MPI.

However, there are still two differences between SCR and NR-MPI. Firstly, the data backup and restore efficiency of NR-MPI is higher than SCR, because SCR uses file system interface to save and restore data, which needs additional copies between programs and the operating systems. Meanwhile, SCR usually uses TCP/IP-based communication channels. It cant use fast interfaces, such as RDMA, to accelerate communication directly. Secondly, in a failure, SCR needs restarting jobs, which is a time consuming operation in large parallel systems.

Table 5 gives the overall timing analysis of SCR and NR-MPI based on Sweep3d. We have modified two versions of Sweep3d: SCR-Sweep3d and NR-Sweep3d. The two versions save the same data during execution using different interfaces. The grid size of Sweep3d is 1024x1024x 16384 and the parallelism is 16384. The iteration count is 10, and data is saved or checkpointed per iteration.

Table 5. OVERAL TIMING ALALYSIS OF SCR AND NR-MPI

Time (seconds)	SCR- Sweep3d	NR-Sweep3d
Time to detect failure	60	60
Time to backup data	3	2.5
Rebuild or route data (SCR)	1	0
Time to re-launch job(SCR)	10	0
Time to recover MPI library.(NR-MPI)	0	0.1
Time to read application data	5	2
Average lost computation	11	11
Total	90	75.6

The time to detect a failure is two times of the heartbeat interval. In the experimental system, it is about 60s. The way to backup application data is different for the two versions. One is via SCR interfaces, while the other one is via MPI communications. So NR-Sweep3d is faster than SCR-Sweep3d for backing up data. Rebuilding or routing data is specific for SCR, so do restarting jobs. While recovering MPI library is only needed by NR-MPI. The time to read application data after failures is different for the two versions. All processes of SCR-Sweep3d need to read some application data from parallel file systems except the application data in local memory disk, while only the replacement processes of NR-Sweeep3d need to receive lost application data and read lost application data from file systems. The average lost of computation is the same for the two versions, because they all backup data per iteration. In conclusion, NR-Sweep3d is better than SCR-Sweep3d.

4. RELATED WORK

4.1. Fault Tolerant Techniques

In our previous work [11], we all also present NR-MPI, which is implemented based on MPICH. The failure recovery of MPI library is based on our own fault tolerant MPI constructs. In this paper, the failure recovery of MPI library is based on ULFM. In fact, there is few performance difference between this work and previous work.

There are several fault tolerant techniques, such as Checkpoint/Restart [12] (C/R for short), Message logging [13, 14], process-level replication [15] and forward recovery [16]. C/R, which has been used widely in previous high performance computers, periodically saves the state of a computation to stable storages, such as parallel file systems. However, C/R requires a restart of the entire parallel job even when only one process of the job failed. In a restart, all processes of a parallel job need to be reloaded. Then, all processes read the latest checkpoint to recover a consistent state. Both the overheads of checkpoint file I/O and restart overhead are unbearable for the current large scale systems, letting alone for the future extreme large scale systems. Log based methods, such as MPICH-V [17], can recover the process to its initial state and roll it forward by re-playing the messages before failures in the same order they were delivered before the crash. However, the main limitation is the necessity to log all messages of the execution. Process-level replication of parallel executions, such as MrMPI [18] and RedMPI [19], employs replicated processes performing the same task. If a process fails its a replication can take over its execution. Thus, redundant copies can decrease the overall failure rate. One major replication overhead comes from the management of extra messages required for replication. For a double-replication execution, when a process sends a message to another process, four communications of that message take place. Forward recovery, such as FT-MPI (and OpenMPI [20, 21]) and User-Level Failure Mitigation (ULFM), allows applications continue running after a failure, while standard MPI does not provide any specification of the behavior of an MPI application after a failure. FT-MPI allows the semantics and associated failure modes to be completely controlled by the applications. FT-MPI required programmers to recover all the application state after failures. In addition, Harness [22], a distributed virtual machine, is used as the runtime system, in the initial implementation of FT-MPI. Both the programming interface and runtime system of FT-MPI are too complicated to use. ULFM allows the application to get notifications of errors and to use specific functions to reorganize the execution for forward recovery. However, ULFM provides only the basic interface and new semantics to enable applications and libraries to repair the state of MPI and tolerate failures. It is just like an assemble language for fault tolerance and is a little complicated for programmers to write fault tolerant applications.

In addition, there are some other hybrid ways. M^3 [23] is a user-transparent checkpoint system for fault tolerant MPI without restarting jobs. Coordinated checkpoint is used to recover lost data upon failures. Job pause service [24], Adaptive MPI [25], StarFish [26] and LAM/MPI [27] are also similar with M^3, using system level checkpoint to recover MPI runtime system. User-Directed Fault Tolerance (UDFT) [28, 29], on top of standard MPI, provides the user directed support for application level algorithmic fault tolerance.

4.2. Data Recovery for HPC

C/R is the most typical technique for fault tolerance in previous researches. There are two types of C/R. Traditional C/R can tolerant the whole system failures by writing checkpoint data periodically to stable storages, such as parallel file systems, and restarting from the latest checkpoint. While diskless C/R [30] saves the states of the processes into the memory directly, eliminating the overhead of writing data to stable storages. Diskless C/R is faster, but traditional C/R can recover from more serious failures. C/R can be implemented at two levels: application-level checkpoint [31] and system-level checkpoint [32]. SCR [33] reduces the checkpoint and restart overheads by caching checkpoint data in the memory or local storage of the compute node. However, the spatial overhead of SCR is triple size of the checkpoints at least. This is a huge overhead for future supercomputers which have a lower memory/processor ratio. In addition, SCR also needs a restart of the entire parallel job. Usually, the overheads of restarting jobs are directly proportional to the parallelism of jobs. Checkpoint-on-Failure Protocol [34] can reduce checkpoint overhead by eliminating the overhead of customary periodic checkpointing. Using NR-MPI, Checkpoint-on-Failure can reduce restart overhead furthermore.

Algorithm-based Fault Tolerance (ABFT) can also be used to recover lost data due to failures. Huang and Abraham [35] developed the ABFT technique to detect, locate and correct soft failures. For many matrix operations, the checksum relationship in the input checksum matrices is still held in the final computation results. Therefore, the soft failures can be detected by checking the checksum relationship in the final computation results. If some processes fail during computation, the data is lost. Based on the checksum relationship, the lost data can be rebuilt using the data of the survival processes. For example, Davies [36] proposed an algorithm-based recovery scheme for the HPL benchmark, based on the checksum relationship of the right-looking LU factorization algorithm. The checksum is maintained at every step of the computation. Chen [37] found that, for many iterative methods, if the data partitioning scheme satisfies certain conditions, the iterative methods will maintain enough inherent redundant information for the accurate recovery of the lost data. Yang [38] proposed a new application-level fault-tolerant approach for parallel applications called the Fault-Tolerant Parallel Algorithm, which provides fast self-recovery upon failures. In a failure, all survival processes re-compute the workload of the crashed processes in parallel. However, it requires programmers to redesign algorithms.

Conclusion

This paper proposes a convenient, scalable and efficient fault tolerant MPI, named NR-MPI. By the notifications from fault tolerant RMS, failures are internally and automatically recovered by the NR-MPI runtime system. On the one hand, NR-MPI eliminates the restarting job overhead by automatic online recovering MPI communication states after failures for the survival processes. On the other hand, NR-MPI reduces the complexity of fault tolerant MPI by designing new semantics of MPI. For example, duplicate messages and termination of programs do not to be detected any more. We carried detailed experiments to evaluate NR-MPI. The experimental results, from 2048 processes to 32768 processes, show that NR-MPI could be scalable soundly.

NR-MPI also has limitations. Firstly, not all failures can be recovered. Not enough communicator contexts, not enough replacements, or the death of the two partners backing up data each other can cause of unsuccessful recovery. Secondly, programmers need to modify programs to using NR-MPI. Thirdly, upon failures, programmers have to rollback to a consistent position in the NR-MPI parallel programs. However, the lost computation due to rollback is controlled by programmers. Fourthly, NR-MPI assumes that the crashed processes are necessary for the parallel job, so it reinitializes replacements upon failures. In fact, if replacements are not needed, they can be excluded from the recovered world communicator by calling MPI_Comm_create.

In the future, our work focuses on: 1)lazy allocation-based failure recovery, which requires the survival processes to spawn replacements when necessary. 2)more flexible data recovery algorithms to reduce the memory overheads of double in memory checkpoint. 3)combining NR-MPI with the new de facto message passing standard, to support fault tolerance for a broad range of extreme scale applications.

Acknowledgment

The authors thank the engineers in NSCC-TJ for their efforts on this work.

This work is supported in part by the National High Technology Research and Development 863 Program of China under grant 2012AA01A301 and 2012AA01A309, and the National Natural Science Foundation of China under grants 61120106005, 61272140, 61272141 and 61025009.

References

1. Top500, Top500 lists. http://www.top500.org, 2012.

2. F. Cappello, A. Geist, W. Gropp, S. Kale, B. Kramer and M. Snir. Toward Exascale Resilience: 2014 update. *Supercomputing Frontiers and Innovations*, Vol. 1 no. 1, 2014

3. W. Bland. User Level Failure Mitigation in MPI, *Euro-Par 2012: Parallel Processing Workshops Lecture Notes in Computer Science,* vol.7640, p.499-504, 2013.

4. W. Gropp, MPICH2: A New Start for MPI Implementations, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, p.37–42, 2002.

5. M. Jette and M. Grondona, SLURM: Simple Linux Utility for Resource Management, in *Proceedings of ClusterWorld Conference and Expo*, San Jose, California, 2003.

6. G. Zheng, L. Shi and L.V. Kal, FTC-Charm++: An In-Memory Checkpoint-Based Fault Tolerant Runtime for Charm++ and MPI, *Proc. Sixth IEEE Int',l Conf. Cluster Computing* (Cluster ',04), p.93-103, Sept. 2004.

7. D. Bailey, T. Harris, W. Saphir, R. Van Der Wijngaart, and A. Woo, The NAS Parallel Benchmarks 2.0, NASA Ames Research Center, Moffett Field, CA 2002.

8. A. Hoisie, O. Lubeck, and H. Wasserman, Performance and Scalability Analysis of Teraflop-Scale Parallel Architectures Using Multidimensional Wavefront Applications, *International Journal of High Performance Computing Applications*, vol.14, p.330-346, 2000.

9. M. Xie, Y. Lu, K. Wang, L. Liu, H. Cao, and X. Yang, Tianhe-1A Interconnect and Message-Passing Services, *IEEE Micro*, vol.32, p.8-20, 2012.

10. X. Yang, X. Liao, K. Lu, Q. Hu, J. Song, and J. Su, The TianHe-1A Supercomputer: Its Hardware and Software, *Journal of Computer Science and Technology*, vol. 26, p.344-351, 2011.

11. G. Suo, Y. Lu, X. Liao, M. Xie, and H. Cao, NR-MPI: a Non-stop and Fault Resilient MPI, In *Proceedings of the 19th IEEE International Conference on Parallel and Distributed Systems(ICPADS 2013)*, Seoul, Korea, p.190-199, 2013

12. R. Koo and S. Toueg, Checkpointing and Rollback-Recovery for Disitributed Systems, *IEEE Transactions on Software Engineering*, vol.13, p.23–31, 1987.

13. A. Bouteiller, P. Lemarinier, G. Krawezik and F. Cappello, Coordinated Checkpoint versus Message Log for Fault Tolerant MPI, in *Proc. Fifth IEEE Int',l Conf. Cluster Computing* (Cluster ',03), p.242, 2003.

14. G. Bosilca, A. Bouteiller, F. Cappello, S. Djilali, G. Fedak, C. Germain, T. Herault, P. Lemarinier, O. Lodygensky, F. Magniette, V. Neri, and A. Selikhov. Toward a scalable fault tolerant mpi for volatile nodes, In *Proceedings of SC 2002.* IEEE, 2002.

15. K. Ferreira, J. Stearley, I. J. H. Laros, R. Oldfield, K. Pedretti, R. Brightwell, R. Riesen, P. G. Bridges, and D. Arnold, Evaluating the viability of process replication reliability for exascale systems, in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, New York, NY, USA, 2011, p.1–12.

16. G. E. Fagg and J. Dongarra, FT-MPI: Fault Tolerant MPI, Supporting Dynamic Applications in a Dynamic World, in *Proceedings of the 7th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, London, UK, UK, 2000, p.346–353.

17. A. Bouteiler, T. Herault, G. Krawezik, P. Lemarinier, and F. Cappello. MPICH-V project: A multiprotocol automatic fault tolerant MPI, *The International Journal of High Performance Computing Applications*, vol.20, p.319-333, 2006.

18. C. Engelmann and S. Bohm. Redundant execution of HPC applications with MR-MPI. In *Proceedings of the 10th IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN)*, p.31C38, 2011.

19. D. Fiala, F. Mueller, C. Engelmann, and R. Riesen, Detection and correction of silent data corruption for large-scale high-performance computing. In *Parallel & Distributed Processing Workshops & Phd Forum IEEE International Sympos*, 7196(5), p.2069-2072, 2011.

20. W. Bland, Enabling Application Resilience with and without the MPI Standard, in *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (ccgrid 2012), Washington, DC, USA, 2012, p.746–751.

21. J. Hursey and R. Graham, Building a Fault Tolerant MPI Application: A Ring Communication Example, in *16th International Workshop on Dependable Parallel, Distributed and Network-Centric Systems* (DPDNS) held in conjunction with *the 25th IEEE International Parallel and Distributed Processing Symposium* (IPDPS), Anchorage, Alaska, 2011.

22. M. Beck, J. J. Dongarra, G. E. Fagg, G. A. Geist, P. Gray, J.s Kohl, M. Migliardi, K. Moore, T. Moore, P. Papadopoulous, S. L. Scott, and V. Sunderam. HARNESS: A Next Generation Distributed Virtual Machine, *Future Generation Computer Systems*, 15(5-6):571-582, 1999.

23. H. Jung, D. Shin, H. Han, J. W. Kim, H. Y. Yeom, and J. Lee, Design and Implementation of Multiple Fault-Tolerant MPI over Myrinet(M^3), in *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, Washington, DC, USA, 2005, p.32–46.

24. C. Wang. Transparent Fault Tolerance for Job Healing in HPC Environments, PhD thesis, North Carolina State University, 2009.

25. C. Huang, O. Lawlor, and L. V. Kalé. Adaptive MPI, In *Proceedings of the 16th International Workshop on Languages and Compilers for Parallel Computing* (LCPC 2003), LNCS 2958, p.306-322, College Station, Texas, October 2003.

26. A. Agbaria and R. Friedman, Starfish: Fault-tolerant dynamic MPI programs on clusters of workstations, In *8th IEEE International Symposium on High Performance Distributed Computing,* 1999.

27. S. Sankaran, J. M. Squyres, B. Barrett, A. Lumsdaine, J. Duell, P. Hargrove, and E. Roman. The LAM/MPI checkpoint/restart framework: System-initiated checkpointing, *International Journal of High Performance Computing Applications,* 19(4):479-493, Winter 2005.

28. R. Wang, E. Yao, P. Balaji, D. Buntinas, M. Chen and G. Tan. Building Algorithmically Nonstop Fault Tolerant MPI Programs, In *Proceedings of the 18th IEEE International Conference on High Performance Computing.* (HiPC 2011), December 2011, Bangalore, India.

29. Z. Wu, R. Wang, W. Xu, M. Chen, E. Yao, Supporting User-directed Fault Tolerance over Standard MPI, in *2012 IEEE 18th International Conference on Parallel and Distributed Systems* (ICPADS), p.696-697, 17-19 Dec. 2012.

30. J. S. Plank, K. Li, and M. A. Puening, Diskless Checkpointing, *IEEE Trans. Parallel Distrib. Syst.,* vol. 9, p.972–986, 1998.

31. J. P. Walters and V. Chaudhary, Application-Level checkpointing techniques for parallel programs, in *ICDCIT'06,* Berlin, Heidelberg, 2006, p. 221–234.

32. J. S. Plank, M. Beck, G. Kingsley, and K. Li, Libckpt: transparent checkpointing under Unix, in *TCON'95,* Berkeley, CA, USA, 1995, p.18–18.

33. A. Moody, G. Bronevetsky, K. Mohror, and B. R. de Supinski, Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System, in *SC '10*, Washington, DC, USA, 2010, p.1–11.

34. W. Bland, P. Du, A. Bouteiller, T. Herault, G. Bosilca, J. Dongarra. A Checkpoint-on-Failure protocol for algorithm-based recovery in standard MPI, In *18th Euro-Par,* LNCS, vol. 7484, p.477-489. 2012.

35. K. Huang and J. A. Abraham, Algorithm-Based Fault Tolerance for Matrix Operations, *IEEE Trans. Comput.,* vol. 33, p.518–528, 1984.

36. D. Fiala, Detection and Correction of Silent Data Corruption for Large-Scale High-Performance Computing, in *IPDPSW '11*, Washington, DC, USA, 2011, p.2069–2072.

37. Z. Chen, Algorithm-based recovery for iterative methods without checkpointing, in *HPDC '11,* New York, NY, USA, 2011, p.73–84.

38. X. Yang, Y. Du, P. Wang, H. Fu, and J. Jia, FTPA: Supporting Fault-Tolerant Parallel Computing through Parallel Recomputing, *IEEE Trans. Parallel Distrib. Syst.,* vol.20, p.1471–1486, 2009.

DOI: 10.14529/jsfi160102

Reconfigurable computer systems: from the first FPGAs towards liquid cooling systems

***Ilya I. Levin*[1], *Alexey I. Dordopulo*[1], *Alexander M. Fedorov*[1], *Igor A. Kalyaev*[2]**

The paper covers the history of development of design technologies of reconfigurable computer systems based on FPGAs of various families. Five generations of reconfigurable computer systems with high placement density, designed on the base of various FPGA families, from Xilinx Virtex-E to modern Virtex UltraScale, are described. The last achievements in the domain of design of energetic effective reconfigurable computer systems with high real performance are presented. One of such achievements is the developed liquid cooling system for Virtex UltraScale FPGAs. It provides independent circulation of the cooling liquid in the 3U computational module with the 19 height for cooling of 96-128 FPGA chips that in total generate 9.6-12.8 kWatt of heat. The distinctive features of the designed immersion liquid cooling system are high cooling efficiency with power reserve for the designed perspective FPGA families, resistance to leaks and their consequences, and compatibility with traditional water cooling systems based on industrial chillers.

Keywords: FPGA, reconfigurable computer systems, immersion liquid cooling system.

Introduction

One of perspective approaches to achieve high real performance of a computer system is the adaptation of its architecture to the structure of a solving task, and creation of a special-purpose computer device which hardwarily implements all computational operation of the information graph of the task with the minimum delays. A natural requirement for a modern computer system is hardware support of modification of both the algorithm of the solving task and the task itself, that is why FPGAs are used as a principal computational resource of reconfigurable computer systems [1].

The main advantages of programmable logic devices (PLD) are:

- possibility of implementation of complicated parallel algorithms;
- availability of CAD-tools for complete system simulation;
- possibility of programming or modification of in-system configuration;
- compatibility of various design projects when they are converted in a VHDL, AHDL, Verilog descriptions or in any other hardware description language.

1. The history of development of FPGA and FPGA-based reconfigurable computer systems

The history of the PLD architectures started at the end of the 70s, when the first PLDs with programmable-AND and programmable-OR arrays appeared. Such architectures were called FPLAs (Field Programmable Logic Array) and FPLSs (Field Programmable Logic Sequencers)[2]. Their main disadvantage is weak use of programmable-OR array.

At the end of the 80s PLD developers suggested new architectures that were simpler and cheaper: PALs (Programmable Array Logic) and GALs (Gate Array Logic). Such architectures

[1]Scientific Research Centre of Supercomputers and Neurocomputers, Taganrog, Russia
[2]A.V. Kalyaev Scientific Research Institute of Multiprocessor Computer Systems at Southern Federal University, Taganrog, Russia

were used in the PLDs of Intel, Altera, AMD, Lattice, etc. The PLDs had rather low integration density, a programmable-AND array and a fixed-OR array [3]. Another approach to reduction of programmable-OR array redundancy was a programmable macro-logic. Macro-logic-based chips contained either a programmable-NAND array or a programmable-NOR array, but it was possible to form complicated logic functions owing to numerous feedbacks.

At the beginning of the 80s there were three leading PLD vendors on the programmable logic device world market. Altera Corporation [4] was founded in June 1983, Xilinx Inc. [5] in February 1984, Actel Corporation [6] in 1985. At present these three companies hold about 80% of PLD world market and determine the ideology of PLD applications. In the past the PLD chips were only one product among various products of such enterprises as Intel, AMD, etc. But in the middle of the 80s leading positions on the PLD market were taken by enterprises specializing solely in design and production of the PLDs.

New vendors offered new PLD architectures, such as CPLDs (Complex Programmable Logic Devices) [7]. The CPLD has rather high integration density and contains several functional blocks connected by a switch matrix. Each functional block contains a programmable-AND array and a fixed-OR array. This PLD class includes Altera MAX7000, Xilinx XC9500 and various PLDs of other vendors, such as Atmel, Vantis, Lucent, etc.

Introduction of FPGAs (Field Programmable Gate Array) [8] ignited revolution of devices with programmable logic. The FPGA class includes Xilinx XC2000, XC3000, XC4000 and Spartan, Actel ACT1, ACT2, Altera FLEX8000 family and some Atmel and Vantis PLDs.

The FPGA configurable logic blocks (CLBs) are connected by a programmable switch matrix. The logic blocks consist of one or several rather simple logic cells based on a 4-input look-up table (LUT), a program-controlled multiplexer, a D-flip-flop. Input/output blocks (IOB) that provide bidirectional input/output, tri-state, etc., are typical for the FPGA-architectures. The FPGA chips have the following advanced features:

– a JTAG port that supports all mandatory boundary-scan instructions specified according to the IEEE 1149.1 standard;

– a master configuration mode (that required a build-in oscillator).

The FPGAs with a dedicated block RAM were the result of further development of the FPGA architecture, owing to which the FPGAs can be used with no external memory devices. The FPGAs have a high logic capacity, an easy-to-use architecture, a quite high reliability and an optimal ratio price/logic capacity, therefore they match various requirements, claimed by circuit engineers.

During the last years, the FPGAs along with custom VLSIs have become the basis of systems-on-chip. IP-cores of such systems have been designed separately and can be used in various projects. The final structure of an FPGA-based SoC is implemented by means of CAD tools.

The ideology of SoC design spurred the leading FPGA vendors on, and at the end of 1998 at the beginning of 1999 they introduced products with an equivalent logic capacity about a million equivalent gates and more. The Altera ApEX20K family is an example of new FPGA families for SoC design.

The Xilinx Virtex [9] family has a similar architecture with a great variety of high speed routing resources, a dedicated block RAM, an optimal high-speed carry logic. The Virtex family provides a high speed of data exchange between the chips up to 200 MHz (HSTL IV standard). The chips of the Virtex family have a relatively low price (not more than 40% from the equivalent price of XC4000XL series), owing to the advanced production technique and perfected verification methods.

In 1998-1999 the augmentation of FPGA equivalent logic capacity changed the attitude to CAD-tools of both software developers and users. Till the end of the 90s the main tool of project description and schematic entry was a graphic editor and libraries of standard primitives and macros, such as logic elements, elementary combinational and sequential functional units, analogues of standard integrated circuits of small-scale and medium-scale integration. At present, circuit engineers use widely the hardware description languages for FPGA-based implementation of algorithms. Besides, up-to-date CAD-tools support both standard hardware description languages (such as VHDL, Verilog) and specialized hardware description languages developed by FPGA vendors specially for their own needs, CAD-tools and FPGA families with special architecture features. Such example is AHDL (Altera Hardware Description Language), which is supported by the Altera CAD-tools MAX PLUS II and Quartus.

Xilinx corporation designs IP-cores of frequently used functional blocks, including blocks of digital processing, bus interfaces, processors, etc. Owing to design of the IP-cores, based on the use of the Xilinx LogiCORE^{TM} development package and on the use of some software analogues developed by second-party vendors, it is possible to reduce the time of project design, minimize risks, and achieve the highest performance. In addition, with the help of the CORE Generator^{TM} the circuit engineer can design his own IP-cores with predictable and repeatable timing characteristics. The CORE Generator^{TM} has a quite simple user interface for generation of parameterized IP-cores specially perfected for use in the Xilinx FPGAs.

2. Research and design of FPGA-based computational systems

At the same time appearance of high-performance FPGA chips on the market gives wide perspectives for their use as principal components of high-performance computer systems (supercomputers).

At present, there are two kinds of high-performance computer systems which use FPGAs as principal components. The first kind is so called hybrid computer systems, i.e. classic cluster computers which contain FPGAs in their microprocessor nodes and use them as accelerators of calculations. The examples of such hybrid supercomputers are XT4 by Cray and RASC by SiliconGraphics. In these systems blocks of programmable co-processors are implemented in FPGAs, interconnected with themselves and the principal processors by high-speed busses.

The second kind of computer systems, which use FPGAs as principal components, is reconfigurable computer systems (RCS). In RCSs, FPGAs are principal computational components, whereas general-purpose processors are minor components, which control the operation of the reconfigurable part of the system.

The acknowledged leader in the domain of design of reconfigurable computer systems based on FPGA computational fields is Taganrog scientific school, founded by academician A.V. Kalyaev. Today it is represented by various RCSs of the supercomputer class designed in Scientific Research Institute of multiprocessor computer systems at Taganrog State University of Radio-Engineering (now at Southern Federal University) and Scientific Research Centre of Supercomputers and Neurocomputers. The principal computational resource of such systems is not microprocessors, but a set of FPGA chips united into computational fields by high-speed data transfer channels. The spectrum of produced and designed products is rather wide: from completely stand-alone small-size reconfigurable accelerators (computational blocks), computational modules of desktop and rack design (based on Xilinx Virtex-6, Virtex-7, and UltraScale FPGAs) to computer systems which consist of several computer racks placed in a specially equipped computer room.

Since 2001 four generations of FPGA-based reconfigurable computer systems have changed one another owing to the production of new FPGA families and growth of computational complexity of problems that require continuous increasing of RCS performance.

RCSs with macroprocessor architecture (RCS MPA) were the first generation of RCSs. They consisted of a number of basic modules implemented on FPGAs and a personal computer. Each basic module (BM) is a reconfigurable computational device designed according to the same architectural principles that as the whole system. Such approach provided natural implementation of structural procedural parallel programs for different granularity of parallelism and piping of calculations. First single-board RCS MPA basic module was designed and created in 2001. Its principal components were Xilinx Virtex- FPGAs. The board of the BM was produced according to 12-layer technology with double-side mounting of components. There are six signal layers and six layers of potentials: two ground layers, two layers for the power of 1.8 V, and two layers for the power 3.3 V. Fig. 1 shows the front and the back sides of the board of the RCS MPA basic module.

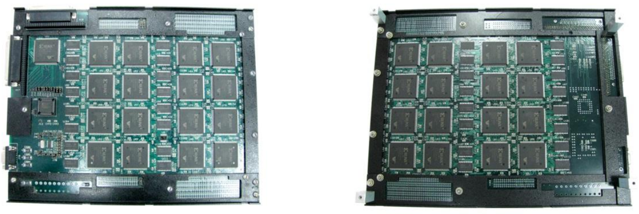


Figure 1. Front and back sides of the board of the RCS MPA basic module

The BM printed circuit board contained 32 FPGAs and 32 RAM chips, mounted on its both sides. Placement of the components on the BM printed circuit board provided the minimum length of connections between them. The performance of the RCS MPA basic module was 2.5x10^{10} op/sec, 64 processing elements, the power consumption was 30 Watt. On the basis of the BM printed circuit board of the RCS MPA a number of modular-scalable multiprocessor computer systems were designed and created [1].

RCS MPAs gave way to reconfigurable computer systems of the second generation, such as RCS with macroobject architecture. Using the RCS with macroobject architecture, the developer has two levels of its architecture programming [10].

On the first level (the lower one) the developer creates functional objects, which are necessary for implementation of a certain problem and which are physically placed in the FPGAs of the computational field. As a rule, the place and the function of the object remain constant not only for implementation of a certain task, but also for implementation of tasks of a certain class. In other words, on the first level the RCS is adjusted to some special-purpose architecture.

On the second level, with the help of the communication system the functional objects are united into computing structures, similar to the information graph of the solving task. Owing to such programming of RCS architecture, it is possible to increase the efficiency of the computational process in ten times in comparison with RCS MPA.

The first representative of RCS of the second generation with macroobject architecture was a modular-scalable RCS Bear (see its basic module in Fig. 2).

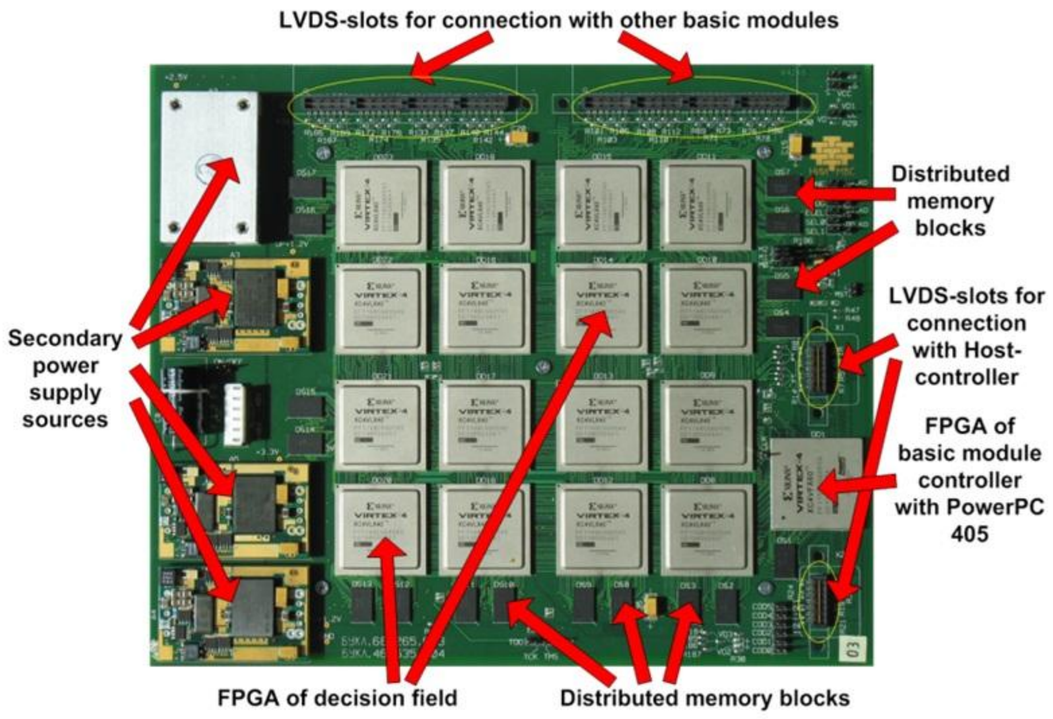

LVDS-slots for connection with other basic modules
Distributed memory blocks
LVDS-slots for connection with Host-controller
FPGA of basic module controller with PowerPC 405
Secondary power supply sources
FPGA of decision field
Distributed memory blocks

Figure 2. A general view of the basic module of the RCS Bear

The base of the basic module is a 20-layer printed circuit board with a double-side mounting of components, a computational field of 16 Xilinx XC4VLX40-10FF1148 FPGAs, a basic module controller, 17 chips of dynamic memory SDRAM, a programmable clock generator, four small-size DC-DC voltage transducers, LVDS-connectors for connection of basic modules via data channels, and other elements of surface mounting. The performance of the basic module of the RCS Bear was 50 GFlops, the frequency of the basic module was 160 MHz, and power consumption did not exceed 150 Watt.

The principles of macroobject architecture were used during the design of a small-size reconfigurable accelerator of the personal computer, intended for implementation of computationally laborious fragments of tasks of various problem domains. Fig. 3 shows the small-size reconfigurable accelerator of the personal computer.

The base of the accelerator was a basic module 4V4-25, implemented on a 18-layer printed circuit board of 150190 mm with double-side mounting of components. The performance of the basic module 4V4-25 was 25 GFlops, the frequency of the basic module was 160 MHz, and its power consumption did not exceed 145 Watt.

Owing to design experience of implementation of the RCS MPA and the RCS Bear, it became possible to implement RCSs of the third generation. The first representatives are RCSs

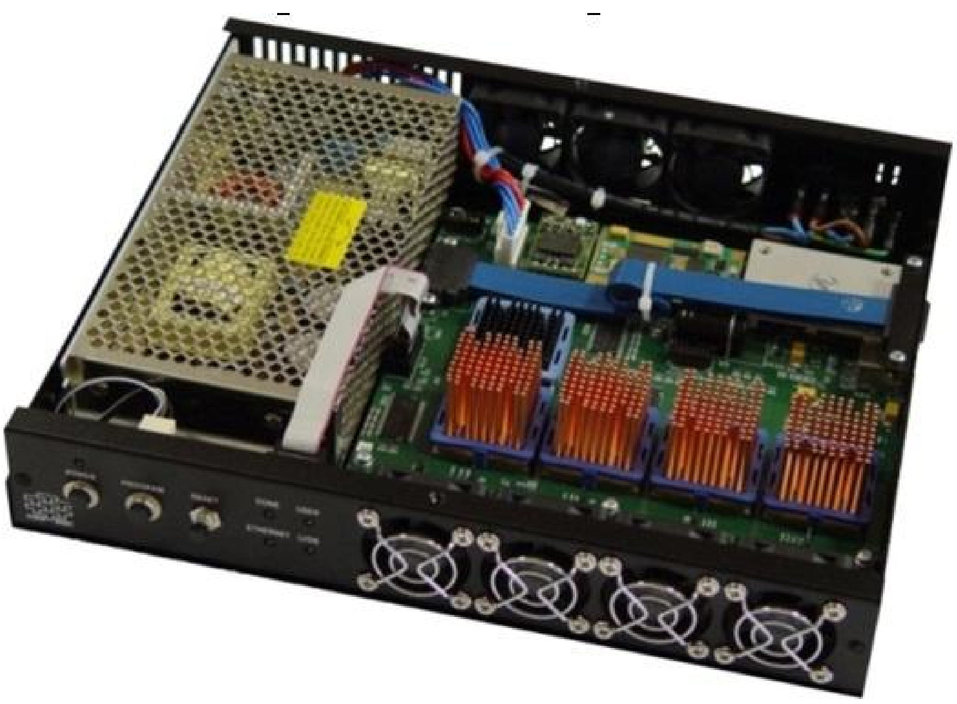


Figure 3. The small-size reconfigurable accelerator of the personal computer

of a family Ursa Major, which were designed on the basis of three types of basic modules, such as 16V5-75, 16V5-50, and 16S3-25. The basic module 16V5-75 (the most high-performance one) was used in such computer systems of the family as RCS-5, RCS-1R, and RCS-0.2-WS. The basic modules 16V5-50 and 16S3-25 were the components of such accelerators of the personal computer as RASC-50 and RASC-25.

The computational field of the basic module 16V5-75 consists of 16 Xilinx Virtex-5 XC5VLX110-2FF1153 FPGAs, which contain 11 million equivalent gates. The FPGAs are placed in the nodes of a 44 2D-lattice and are interconnected by a lattice-like communication system. Such communication system simplifies considerably the printed circuit board and improves its frequency characteristics because connections between the adjacent chips do not exceed 4 centimeters. Between remote FPGAs the data are transferred via transit channels through transit chips according to the lattice-like connections. The FPGAs placed on the borders of the 44 2D-lattice of the computation field are connected with 20 SDRAM DDR2 chips, which form distributed memory with the total volume of 1.25 GByte. Fig. 4 shows the printed circuit board of the basic module 16V5-75 with mounted electronic elements and the assembled basic module with its cooling system.

For heat dissipation and supporting all required temperature modes of the chips of the basic module, a combined cooling system is designed, which contains radiators, placed on FPGAs of the computation field, and air-fans for blowing them round. In general, the basic module 16V5-75 is a high-performance computational node with the performance above 75 (140) GFlops.

The basic module 16V5-75 is the base of the workstation RCS-0.2-WS and the computational block RCS-0.2-CB with the performance 300 GFlops.

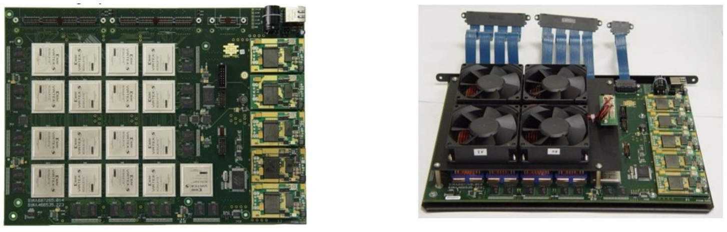


Figure 4. The printed circuit board of the basic module 16V5-75

Fig. 5 shows the workstation RCS-0.2-WS without its top cover and the computational block RCS-0.2-CB.

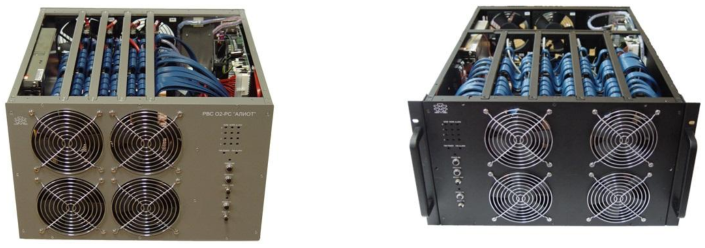


Figure 5. The workstation RCS-0.2-WS and the computational block RCS-0.2-CB

The computational blocks RCS-0.2-CB are the base of the reconfigurable computer system RCS-1R with the peak performance of 1200 GFlops. This computer system is intended for scientific centres, which deal with research in such domains as physics, chemistry, biology, space, design of information-control systems for control of potentially dangerous industry, air-space industry, automobile industry, energetics, etc.

The high-end representative of the RCSs of the third generation was the system RCS-5 (see Fig. 6) with the peak performance of 6000 GFlops, which consisted of five racks RCS-1R, interconnected by Ethernet-switchers with general control.

The computer system RCS-5, placed in Research Computing Centre at Lomonosov Moscow State University, contains 20 computational blocks RCS-0.2-CB, 80 basic modules 16V5-75, 16384 processing elements (IEEE-754), which perform processing of 64-digit IEEE-754 data at the frequency of 330 MHz with the performance of more than 6000 GFlops. Between the blocks

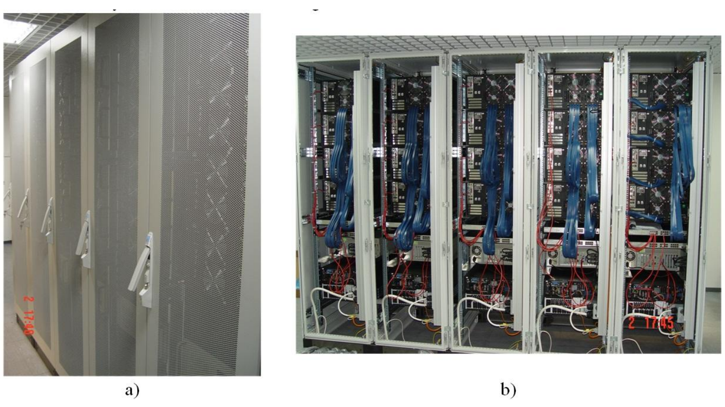


Figure 6. The RCS-5) the front view; b) the back view

the data exchange is performed via external LVDS and Gigabit Ethernet interfaces at frequency of 640 MHz.

Due to growing demands to RCS performance the density of placement of RCS components increased in 4 times. In 2012-2014, on the basis of Xilinx Virtex-7 FPGAs and computational modules (CM) 24V7-750 (Pleiad) and Taygeta [11], the fourth generation of RCS was designed.

The computational module 24V7-750 contains 4 printed circuit boards 6V7-180 (see Fig. 7); a control unit CU-7; a power supply subsystem; a cooling subsystem and other subsystems. The performance of the CM 24V7-750 is 2.58 TFlops for processing of 32-digit floating point data.

The CM 24V7-750 was used for the creation of a reconfigurable computer system RCS-7 (the state contract 14.527.12.0004 from 03.10.2011). The system RCS-7 contains 24 CMs 24V7-750 with a computation field of 576 Xilinx Virtex-7 XC7V585T-FFG1761 FPGAs (58 million equivalent gates each), interconnected and placed in one 47U computer rack with the peak performance of 1015 fixed point operations per second. The performance of the RCS-7, which contains from 24 to 36 24V7-750 CMs is from 62 to 93 TFlops for processing of 32-digit floating point data, and is from 19.4 to 29.4 TFlops for processing of 64-digit floating point data. The application area of the RCS-7 and computer complexes, created on its basis, is digital signal processing and multichannel digital filtration.

The 2U computational module Taygeta (see Fig. 8a) was also designed on the basis of Virtex-7 FPGAs and can be placed in a standard 19” computer rack. The CM Taygeta contains 4 boards of the CM 8V7-200 (see Fig. 8b), connected by high-speed LVDS-channels, an embedded host-computer, a power supply system, a control system, a cooling system and other subsystems.

The computational module Taygeta is based on the board of the CM 8V7-200, which is a 20-layer printed circuit board with double-side mounting of components. The board contains 8 Xilinx XC7VX485T-1FFG1761 FPGAs (48.5 million equivalent gates), 16 chips of distributed memory SDRAM DDR2 with its total volume of 2 GByte, LVDS and Ethernet interfaces, and other components. The performance of the CM Taygeta is 2.66 TFlops for processing of 32-digit floating point data. The performance of an RCS, which is based on the CM Taygeta and which contains 18 CMs Taygeta, is 48 TFlops for processing of 32-digit floating point data and 23 TFlops for processing of 64-digit floating point data.

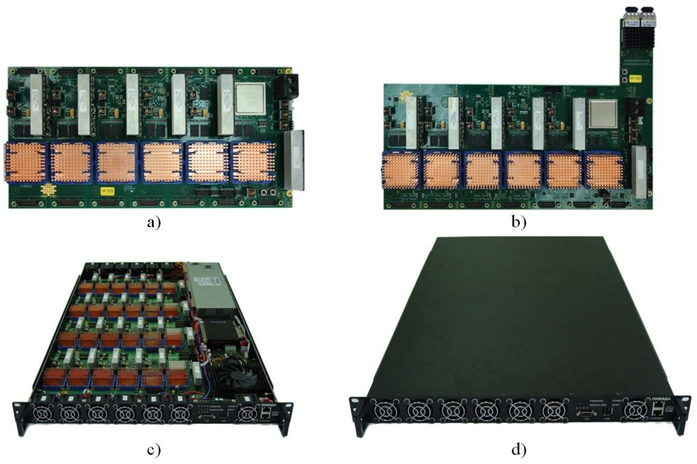


Figure 7. The computational module 24V7-750 (a – the boards 0-2, b – the board 3 with optical connector for connection with other CMs 24V7-750, c – the CM 24V7-750 without its top cover, d – with its top cover)

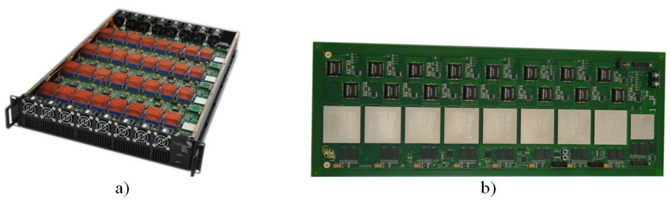


Figure 8. The CM Taygeta (a – without its top cover, b – the board 8V7-200)

Due to the number of problems with the cooling system, the fourth generation of RCS was designed. According to the obtained experimental data, conversion from the FPGA family Virtex-6 to the next family Virtex-7 leads to growth of the FPGA maximum temperature at 1115^o C. Therefore, the further development of FPGA production technologies and conversion to the next FPGA family Virtex Ultra Scale will lead to the growth of FPGA overheat at additional 1015^o C. This will shift the range of their operating temperature to 8085^o C, which means that their operating temperature exceeds the permissible range of the FPGA operating temperature (65...70^o C), and hence, this will have negative influence on their reliability.

Practical experience of maintenance of large computer complexes based on CM RCS proves that air cooling systems have reached their heat limit. That is why use of air cooling for the next generation of FPGAs - Virtex UltraScale, which contain about 100 million equivalent gates and have power consumption not less than 100 Watt per FPGA chip, will not provide stable and reliable operating of the RCS when its chips are filled up to 85-95% of available hardware resource. This circumstance requires a quite different cooling method which provides keeping of growth rates of the RCS performance for promising designed Xilinx FPGA families: Virtex UltraScale, Virtex UltraScale+, Virtex UltraScale 2, etc.

3. Liquid cooling for reconfigurable computer systems

Development of computer technologies leads to the design of computer technique which provides higher performance, and hence, more heat. Dissipation of released heat is provided by a system of electronic element cooling, that transfers heat from the more heated object (the cooled object) to the less heated one (the cooling system). If the cooled object is constantly heated, then the temperature of the cooling system grows and some time will be equal to the temperature of the cooled object. So, heat transfer stops and the cooled object will be overheated. The cooling system is protected from overheat with the help of cooling medium (heat-transfer agent). Cooling efficiency of the heat-transfer agent is characterized by heat capacity and heat dissipation. As a rule, the heat transfer is based on the principles of heat conduction, that require a physical contact of the heat-transfer agent with the cooled object, or on principles of convective heat exchange with the heat-transfer agent, that consists of physical transfer of the freely circulating heat-transfer agent.

To organize the heat transfer to the heat-transfer agent, it is necessary to provide the heat contact between the cooling system and the heat-transfer agent. For this various radiators facilities for heat dissipation in the heat-transfer agent are used. Radiators are set on the most heated components of computer systems. To increase the efficiency of heat transfer from an electronic component to a radiator, a heat interface is set between them. The heat interface is a layer of heat-conducing medium (usually multicomponent) between the cooled surface and the heat dissipating facility, used for reduction of heat resistance between two contacting surfaces. Modern processors and FPGAs need cooling facilities with as low as possible heat resistance, because at present even the most advanced radiators and heat interfaces cannot provide necessary cooling, if an air cooling system is used.

Till 2012 the air cooling systems were used quite successfully for cooling supercomputers. But due to the growth of performance and circuit complexity of microprocessors and FGAs used as components of supercomputer systems the air cooling systems have practically reached their limits for designed perspective supercomputers, including hybrid computer systems. Therefore, the majority of vendors of computer technique consider liquid cooling systems as an alternative decision of the cooling problem. Today liquid cooling systems are the most promising design area for cooling modern high-loaded electronic components of computer systems.

A considerable advantage of all liquid cooling systems is the heat capacity of liquids, which is better than air capacity (from 1500 to 4000 times) and higher than the heat-transfer coefficient (increasing up to 100 times). To cool one modern FPGA chip, 1 m3 of air or 0.00025 m3 (250 ml) of water per minute is required. Transfer of 250 ml of water requires much less of electric energy, than transfer of 1 m3 of air. Heat flow, transferred by similar surfaces with traditional velocity of the heat-transfer agent, is in 70 times more intensive in the case of liquid cooling than in the case of air cooling. Additional advantage is the use of traditional, rather reliable and cheap components such as pumps, heat exchangers, valves, control devices, etc. In fact, for corporations and companies, which deal with equipment with high packing density of components operating at high temperatures, liquid cooling is the only possible solution of the problem of cooling of modern computer systems. Additional possibilities to increase liquid cooling efficiency are improvement of the initial parameters of the heat transfer agent: increasing of velocity, decreasing of temperature, providing of turbulent flow, increasing of heat capacity and reducing of viscosity.

Heat transfer agent in liquid cooling systems of computer technique is liquid such as water or any dielectric liquid. Heated electronic components transfer the heat to the permanently circulating heat transfer agent liquid, which, after its cooling in the external heat exchanger, is used again for cooling of heated electronic components. There are several types of liquid cooling systems. Closed loop liquid cooling systems have no direct contact between liquid and electronic components of printed circuit boards. In open loop cooling systems (liquid immersion cooling systems) the electronic components are immersed directly into the cooling liquid. Each type of the liquid cooling systems has its own advantages and disadvantages.

In the closed loop liquid cooling systems all heat-generating elements of the printed circuit board are closed by one or several flat plates with a channel for liquid pumping. So, for example, cooling of a supercomputer SKIF-Avrora is based on the principle of one cooling plate for one printed circuit board. The plate, of course, had a complex surface relief to provide tight heat contact with each chip. Cooling of a supercomputer IBM Aquasar is based on the principle of one cooling plate for one (heated) chip. In each case the channels of the plates are united by collectors into a single loop connected to a common radiator (or another heat exchanger), usually placed outside the computer case and/or rack or even the computer room. With the help of the pump the heat transfer agent is pumped through the plates and dissipates the heat, generated by the computational elements, by means of the heat exchanger. In such system it is necessary to provide the access of the heat transfer agent to each heat-generating element of the calculator, which means a rather complex piping system and a large number of pressure-tight connections. Besides, if it is necessary to provide the maintenance of the printed circuit boards without any serious demounting, then the cooling system must be equipped with special liquid connectors which provide pressure-tight connections and simple mounting/demounting of the system.

In closed loop liquid cooling systems it is possible to use water or glycol solutions as a heat transfer agent. However, the leak of heat transfer agent can lead to possible ingress of electrically conducting liquid to unprotected contacts of printed circuit boards of the cooled computer, and this, in its turn, can be fatal for both separate electronic components and the whole computer system. To eliminate failures the whole complex must be stopped, and the power supply system must be tested and dried up. Control and monitoring systems of such computers always contain multiple internal humidity and leak sensors. To solve the leak problem a method, based on negative pressure of liquid in cooling system, is frequently used. According to this method, water is not pumped in under pressure, it is pumped out, and this practically excludes the leak of liquid. If air-tightness of the cooling systems is damaged, then the air ingresses the system but no leak of liquid occurs. Special sensors are used for the detection of leaks, and modular design allows the maintenance without stopping of the whole system. However, all these capabilities considerably complicate the design of hydraulic system.

Another problem of closed loop liquid cooling systems is a dew point problem. In the section of data processing the air is in contact with the cooling plates. It means that if any sections of these plates are too cold and the air in the section of data processing is warmer and not very dry, then moisture can condense out of the air on the plates. Consequences of this process are similar to leaks. This problem can be solved ether by hot water cooling, which is not effective, or by control and keeping on the necessary level of the temperature and humidity parameters of the air in the section of data processing, which is complicated and expensive.

The design becomes even more complex, when it is necessary to cool several components with a water flow proportionally to their heat generation. Besides the branched pipes, it is necessary to use complex control devices (simple T-branches and four-ways are not enough). An alternative approach is the use of an industrial device with flow control, but in this case the user cannot considerably change configuration of cooled computational modules.

Advantages of closed loop liquid cooling systems are:

- use water or water solutions as the heat transfer agent, which are available, have perfect thermotechnical properties (heat transfer capacity, heat capacity, viscosity), simple and comparatively safe maintenance;

– a large number of unified mechanisms, nodes and details for water supply systems, which can be used;

– great experience of maintenance of water cooling systems in industry. However, closed loop liquid cooling systems have a number of significant disadvantages, which restrict their widespread use:

– difficulties with detection of the point of water leakage;

– catastrophic consequences that are the result of leakages not detected in time;

– technological problems of leakage elimination (a required power-off of the whole computer rack, that is not always possible and suitable);

– required support of microclimate in the computer room (a dew point problem);

– a problem of cooling of all the rest components of the printed circuit board of the RCS computational module. Even slight modification of the RCS configuration requires a new heat exchanger;

– a problem of galvanic corrosion of aluminum heat exchangers or a problem of mass and dimensions restrictions for more resistant copper heat exchangers (aluminum is three times as lighter than copper);

– air removal from the cooling system that is required before starting-up and adjustment and during maintenance;

– complex placement of the computational modules in the rack with a large number of fittings required for plug-in of every computational module;

– necessity of the use of specialized computer rack with significant mass and dimension characteristics.

In open loop liquid cooling systems the heat transfer agent is the principal component, a dielectric liquid based, as a rule, on a white mineral oil that provides much higher heat storage capacity of the heat transfer agent, than the one of the air in the same volume. According to their design, such system is a bath filed with the heat transfer liquid (also placed into a computer rack) and which contains printed circuit boards and servers of computational equipment. The heat, generated by electronic components, is dissipated by the heat transfer agent that circulates within the whole bath. Advantages of immersion liquid cooling systems are simple design and capability of adaptation to changing geometry of printed circuit boards, simplicity of collectors and liquid connectors, no problems with control of liquid flows, no dew point problem, high reliability and low cost of the product.

The main problem of open loop liquid cooling systems is chemical composition of the used heat transfer liquid which must fulfil strict requirements of heat transfer capacity, electrical conduction, viscosity, toxicity, fire safety, stability of the main parameters and reasonable cost of the liquid.

Open loop liquid cooling systems have the following advantages:

– insensibility to the leakages and their consequences, capability of operating even with local leakages of the heat transfer agent;

– insensibility to climate characteristics of the computer room;

– solution of the problem of cooling of all RCS components, because the printed circuit board of the computational module is immersed into the heat transfer agent;

– capability of modification of the configuration of the printed circuit board of the computational module without modification of the cooling system;

– simplicity of hydraulic adjustment of the system owing to the lack of complex system of collectors;

– possibility of the use of unified mechanisms, nodes and details produced for hydraulic systems of machine industry, and know-how of maintenance of electrical equipment that uses dielectric oils;

– increasing of the total reliability of the liquid cooling system.

The disadvantages of open loop liquid cooling systems are the following:

– necessity of additional pump and heat exchange equipment for improvement of thermotechnical properties (heat transfer capacity, heat capacity, viscosity) of the heat transfer agent. Here special dielectric organic liquids are used as the heat transfer agent;

– necessity of training of the maintenance of staff and keeping increased safety precautions for work with the heat transfer agent;

– necessity of more frequent cleaning of the computer room because of high permeability of the heat transfer agent, especially in the case of leakage;

– necessity of special equipment for scheduled and emergency maintenance operations (mounting/demounting of the computational module, loading/unloading of the heat transfer liquid, etc.);

– increasing of the maintenance cost because of the necessity of regular changeout of the heat transfer liquid when its service life is over and necessity of heat transfer agent management (transporting, receipt, accounting, storing, distribution, recovery of the heat transfer agent, etc.) in the corporation.

Estimating the given advantages and disadvantages of the two liquid cooling systems we can note more weighty advantages of open loop cooling systems for electronic components of computer systems. That is why for RCS computational modules designed on the basis of promising FPGA families, it is reasonable to use liquid cooling, particularly immersion of printed circuit boards of computational modules into liquid heat transfer agent based on a mineral oil.

At present the technology of liquid cooling of servers and separate computational modules are developed by many vendors and some of them have achieved success in this direction. However, most of these technologies are intended for cooling computational modules which contain one or two microprocessors. All attempts of its adaptation to cooling computational modules, which contain a large number of heat generating components (an FPGA field of 8 chips), have proved a number of shortcomings of liquid cooling of RCS computational modules [12]. The special feature of the RCS produced in Scientific Research Centre of Supercomputers and Neurocomputers is the number of FPGAs, not less than 6-8 chips on one printed circuit board and high density of placement. This increases considerably the number of heat generating components in comparison with microprocessor modules, complicates application of the technology of direct liquid cooling IMMERS along with the other end solutions of immersion systems, and requires additional technical and design solutions for effective cooling of RCS computational modules.

4. Reconfigurable computer system based on Xilinx UltraScale FPGAs

Since 2013 the scientific team of SRC SC and NC has actively developed the domain of creation of next-generation RCS on the basis of their original liquid cooling system for printed circuit boards with high density of placement and the large number of heat generating electronic components. The basis of design criteria of the computational module (CM) of next-generation RCS with open loop liquid cooling system are the following principles:

– the principal configuration of the computer rack is the computational module with the 3U height and the 19 width and with self-contained circulation of the cooling liquid;

– one standard 47U computer rack can contain not less than 12 computational modules with liquid cooling;

– one computational module can contain 12-16 printed circuit boards with FPGA chips;

– each printed circuit board must contain up to 8 FPGAs with dissipating heat flow of about 100 Watt from each FPGA;

– a standard water cooling system, based on industrial chillers, must be used for cooling the liquid.

The principal element of modular implementation of open loop immersion liquid cooling system for electronic components of computer systems is a reconfigurable computational module of a new generation (see the design in Fig. 9). The CM of a new generation consists of a computational section, a heat exchange section, a casing, a pump, a heat exchanger and a fitting. In the casing, which is the base of the computational section, a hermetic container with dielectric cooling liquid and electronic components with the elements that generate heat during the operation, is placed. The electronic components can be as follows: computational modules (not less than 12-16), control boards, RAM, power supply blocks, storage devices, daughter boards, etc. The computational section is closed with a cover. The computational section adjoins to the heat exchange section, which contains a pump and a heat exchanger. The pump provides circulation of the heat transfer agent in the CM through the closed loop: from the computational module the heated heat-transfer agent passes into the heat exchanger and is cooled there. From the heat exchanger the cooled heat-transfer agent again passes into the computational module and there cools the heated electronic components. As a result of heat dissipation the agent becomes heated and again passes into the heat exchanger, and so on. The heat exchanger is connected to the external heat exchange loop via fittings and is intended for cooling the heat-transfer agent with the help of the secondary cooling liquid. As a heat exchanger it is possible to use a plate heat exchanger in which the first and the second loops are separated. So, as the secondary cooling liquid it is possible to use water cooled by an industrial chiller. The chiller can be placed outside the server room and can be connected with the reconfigurable computational modules by means of the stationary system of engineering services. The design of the computer rack with placed CMs is shown in Fig. 9b.

The computational and the heat exchange sections are mechanically interconnected into a single reconfigurable computational module. Maintenance of the reconfigurable computational module requires its connection to the source of the secondary cooling liquid (by means of valves), to the power supply or to the hub (by means of electrical connectors).

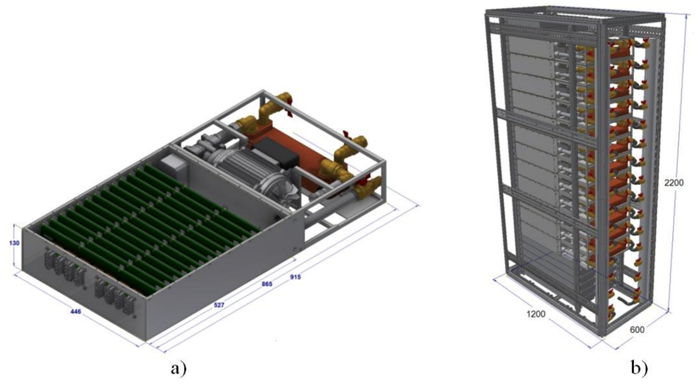


Figure 9. The design of the computer system based on liquid cooling a – the design of the new generation CM, b – the design of the computer rack

In the casing of the computer rack the CMs are placed one over another. Their number is limited by the dimensions of the rack, by technical capabilities of the computer room and by the engineering services. Each CM of the computer rack is connected to the source of the secondary cooling liquid with the help of supply return collectors through fittings (or balanced valves) and flexible pipes; connection to the power supply and the hub is performed via electric connectors. The supply of cold secondary cooling liquid and the extraction of the heated one into the stationary system of engineering services connected to the rack is performed via fittings (or balanced valves). A set of computer racks placed in one or several computer rooms forms a computer complex. To maintain the computer complex, it is connected to the source of the secondary cooling liquid, to the power supply, and to the host computer that controls this computer complex.

Besides the advantages, which are typical for open loop liquid cooling systems, the considered modular implementation of the open loop liquid cooling system for electronic components of computer systems has a number of additional advantages:

– printed circuit boards of computational modules and reconfigurable computational modules are identical, relatively stand-alone and interchangeable. If one of the CMs fails or if technical diagnosis is required, then it is not needed to disconnect completely the computer rack and to stop the execution of a task;

– high placement complexity of FPGAs in the CMs;

– the proposed technical solution allows, if it is necessary, increasing of performance of reconfigurable computational modules without significant increasing of dimensions (a more high-power pump and a heat exchanger can be placed into the selected dimensions). Growth of the number of printed circuit boards of the computational modules will slightly increase the dimensions (depth) of the reconfigurable computational module, but the density of placement will remain unchangeable.

Owing to simplicity of design of the heat exchange section of the reconfigurable computational module, its reliability grows significantly.

The 19 computer rack of the supercomputer (see the design in Fig. 9b) has the following technical characteristics:

– a standard 47U computer rack;

– 12 3U computational modules with liquid cooling;

– each computational module contains 12 printed circuit boards with the power of 800 Watt each;

– each printed circuit board contains 8 Kintex UltraScale XCKU095-1FFVB2104C FPGAs, 95 million equivalent gates (134 400 logic blocks) each;

– the performance of the new generation computational module is 105 TFlops;

– the performance of the computer rack, which contains 12 CMs, is 1 PFlops;

– the power of the computer rack, which contains 12 CMs is 124 kWatt.

The performance of one computer rack with the liquid cooling system, which contains 12 CMs with 12 printed circuit boards each, in 6.55 times exceeds the performance of the similar rack with CMs Taygeta. Here the performance of one CM of a new generation is increased in 8.74 times in comparison with the CM Taygeta. Such qualitative increase of the specific performance of the system is provided by the density of placement, increased more than in three times owing to the original design solutions, and by increasing of the clock rate and the number of gates in one chip.

For testing technical, technological solutions and for determination of expected technical and economical characteristics and service performance of the designed high-performance reconfigurable computer system with liquid cooling, a number of models, experimental and technological prototypes. Fig. 10 shows the technological prototype of the new generation CM. For this CM new designs of printed circuit boards and computational modules with high density of placement were created.

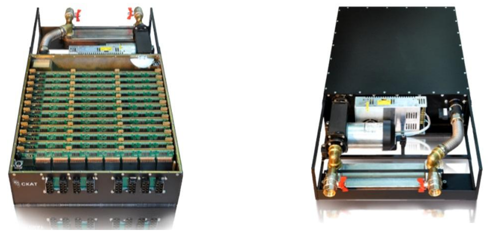


Figure 10. The technological prototype of a new generation CM

The printed circuit board of the promising computational module contains 8 Virtex UltraScale FPGAs of logic capacity of not less than 100 million equivalent gates each. The CM computational section contains 12-16 printed circuit boards of computational modules with the power up to 800 Watt each. Besides, all boards are completely immersed into electrically neutral liquid heat-transfer agent; the heat exchange section contains pump components and the heat exchanger, which provide the flow and cooling of the heat-transfer agent. The design height of the new generation CM is 3U.

For creation of an effective immersion cooling system a dielectric heat-transfer agent was developed. This heat-transfer agent has the best electric strength, high heat transfer capacity, the maximum possible heat capacity and low viscosity. On the basis of the transformer oil, a new oil called Dielectric oil with reduced viscosity MD-4.5 for cooling of electronic components of computers with reduced viscosity was created according to the method of vacuum distillation. For the oil DM-4.5 are determined technical specification TU 38.401-58-421-2015 and recommendations of its use. The oil MD-4.5 was completely tested in the heat engineering laboratory of SRC SC and NC on the technological prototype of the computational module with immersion open loop liquid cooling system. The performed set of laboratory and service tests proved reasonability of use of the oil MD-4.5 for cooling of electronic computer components and of the use of low-power pumps for its circulation (due to its reduced viscosity).

During the design of the new generation CM we have obtained a number of breakthrough technical solutions, such as an immersion power supply block for the voltage of 380 V and a transducer DC/DC 380/12 V, the minimum height of the printed circuit board of the CM of 100 mm is provided, an original immersion control board is designed and produced. For the cooling subsystem of the new generation CM we have determined required components of the cooling system such as an original heat interface, a low-height FPGA radiator of an original design for convective heat exchange, a pump and a heat exchanger optimal for the used heat-transfer agent. Besides, we have determined the design of a volume compensator of the heat-transfer agent and control elements of the cooling subsystem, such as optical level sensors and a flow sensor. The developed implementations of the cooling system design and circulation of the heat-transfer agent provide effective solution of the heat dissipation problem from the most heated components of the CM.

The complex of the developed solutions concerning immersion liquid cooling system will provide the temperature of the heat-transfer agent not more than 33^o C, the power of 91 Watt for each FPGA (8736 Watt for the CM) in the operating mode of the CM. At the same time, the maximum FPGA temperature does not exceed 55^o C. This proves that the designed immersion liquid cooling system has a reserve and can provide effective cooling for promising families of Xilinx FPGAs (UltraScale+, UltraScale 2,etc.).

Conclusion

The use of air cooling systems for the designed supercomputers has practically reached its limit because of the reduction of cooling effectiveness with growing of consumed and dissipated power, caused by the growth of circuit complexity of microprocessors and other chips. That is why the use of liquid cooling in modern computer systems is a priority direction of perfection of cooling systems with wide perspectives of further development. Liquid cooling of RCS computational modules, which contain not less than 8 FPGAs of high circuit complexity, is specific in comparison with the cooling of microprocessors and requires development of a specialized immersion cooling system. The designed original liquid cooling system for a new generation RCS computational module provides high maintenance characteristics, such as the maximum FPGA temperature not more than 55 ^{o}C and the temperature of the heat-transfer agent not more than 33 ^{o}C in the operating mode. Owing to the obtained breakthrough solutions of the immersion liquid cooling system it is possible to place not less than 12 CMs of the new generation with the total performance over 1 PFlops within one 47U computer rack. Power reserve of the liquid cooling system of the new generation CMs provides effective cooling of not only existing but of the developed promising FPGA families Xilinx UltraScale+ and UltraScale 2.

Since FPGAs, such as principal components of reconfigurable supercomputers, provide stable, practically linear growth of RCS performance, it is possible to get specific performance of RCS, based on Xilinx Virtex UltraScale FPGAs, similar to the one of the world best cluster supercomputers, and to find new perspectives of design of super-high performance supercomputers.

This paper was financially supported in part by the Ministry of Education and Science of the Russian Federation under Grant 14.578.21.0006 of 05.06.2014, ID RFMEFI57814X0006

Referenses

1. I.A. Kalyaev, I.I. Levin, E.A. Semernikov, V.I. Shmoilov. Reconfigurable multipipeline computing structures. Nova Science Publishers, New York, USA, 2012.

2. Vicyn N. Sovremennye tendencii razvitija sistem avtomatizirovannogo proektirovanija v oblasti jelektroniki // Chip News, 1, 1997. S. 1215. [N.Vitsyn. Modern trends of development of CAD-systems in electronics // Chip News, 1, 1997, pp. 12–15.]

3. In the Beginning. By Ron Wilson, Editor-in-Chief, Altera Corporation. https://www.altera.com/solutions/technology/system-design/articles/_2013/in-the-beginning.html/ (accessed : 04.04.2016)

4. https://www.altera.com/(accessed: 04.04.2016)

5. www.xilinx.com/(accessed: 04.04.2016)

6. www.actel.com/(accessed: 04.04.2016)

7. Grushvickij R.I., Mursaev A.H., Ugrjumov E.P. Proektirovanie sistem na mikroshemah programmiruemoj logiki. Peterburg: BHV-Peterburg, 2002. 636 s. [R.I. Grushvitskiy, A.Kh. Mursayev, E.P. Ugrumov. System design on chips of programmable logic. St.-Petersburg: BHV-Petersburg, 2002, 636 pp.]

8. Steshenko V., Shipulin S., Hrapov V. Tendencii i perspektivy razvitija PLIS i ih primenenie pri proektirovanii apparatury COS // Komponenty i tehnologii, 2000, 8. [V. Steshenko, S. Shipulin, V. Khrapov. Trends and perspectives of FPGA development and their application for design of DSP devices // Components and technologies, 2000, 8.]

9. Tarasov I. Jevoljucija PLIS serii Virtex // Komponenty i tehnologii, 2005, 1. [I. Tarasov. Evolution of Virtex FPGAs // Components and technologies, 2005, 1.]

10. V.A. Gudkov, A.A. Gulenok, V.B. Kovalenko, L.M. Slasten. Multi-level Programming of FPGA-based Computer Systems with Reconfigurable Macroobject Architecture // IFAC Proceedings Volumes (ISSN 14746670), Programmable Devices and Embedded Systems, Volume 12, part 1, 2013, pp. 204–209.

11. Kalyaev I.A., Levin I.I., Dordopulo A.I., Slasten L.M. Reconfigurable Computer Systems Based on Virtex-6 and Virtex-7 FPGAs // IFAC Proceedings Volumes, Programmable Devices and Embedded Systems. vol.12, 1, 2013, pp. 210–214.

12. I.I. Levin, A.I. Dordopulo, Y.I. Doronchenko, M.K. Raskladkin. Reconfigurable computer system on the base of Virtex UltraScale FPGAs with liquid cooling // Proceedings of international scientific conference Parallel computer technologies (PaCT2016), 2016 pp. 221–230.

DOI: 10.14529/jsfi160103

Supercomputer technologies in tomographic imaging applications

***Alexander V. Goncharsky*[1]*, Sergey Y. Romanov*[1]*, Sergey Y. Seryozhnikov*[1]**

Currently, tomographic imaging is widely used in medical and industrial non-destructive testing applications. X-ray tomography is the prevalent imaging technology. Modern medical X-ray CT scanners provide up to 1 mm of spatial resolution. The disadvantage of X-ray tomography is that it cannot be used for regular medical examinations. Early breast cancer diagnosis is one of the most pressing issues in modern healthcare. Ultrasound tomography devices are being developed in the USA, Germany and Russia to address this problem. One of the main challenges in ultrasound tomographic imaging is the development of efficient algorithms for solving inverse problems of wave tomography, which are nonlinear three-dimensional coefficient inverse problems for a hyperbolic differential equation. Solving such computationally-expensive problems requires the use of supercomputers.

Keywords: supercomputer, GPU cluster, wave-tomography imaging, coefficient inverse problems, scalar wave equation, breast cancer.

Introduction

Tomographic imaging methods are widely used in medicine, non-destructive testing in industry, microscopy, civil engineering, hydrolocation, and seismic studies [1-5]. In Greek, "tomography" means "slice". Tomographic methods can significantly increase resolution for studying the internal structure of various objects. A typical example would be the comparison of the photofluorographic imaging methods and X-ray tomography. As you know, Röntgen had received his Nobel Prize at the beginning of the twentieth century for the discovery of X-rays. Almost immediately, X-ray examination was applied in medicine. Usually, one radiation source and one detector were used. The detector was a film sensitive to X-rays. Despite the high resolution of the film, resolving power remained low. At the end of this article, we will return to the concept of resolving power as the ability to distinguish fine details of the object.

Tomographic imaging suggests solving inverse problems. Surprisingly, mathematical foundations of computed tomography were developed at the same time, at the beginning of the twentieth century. The fundamental works in this area are studies in integral geometry (G. Minkovsky, 1905, P. Funk 1913, Y. Radon 1917). It took humankind half a century to invent first tomographs in the early 60s of the last century. This is primarily due to the fact that the tomographic methods cannot be implemented without using computers. The first tomographs were X-ray tomographs focused on medical research.

Modern X-ray tomographs are very complex machines, and solving mathematical problems is not the most difficult issue. From the mathematical point of view, the inverse problem of X-ray tomography is a linear operator equation that is reduced to two-dimensional integral equations that depend on the difference of the arguments. Solving such 2D problems numerically on a 1000×1000-point grid takes about few seconds using a regular personal computer. The use of graphics processing units (GPUs) allows performing complex image processing procedures, isolation of various internal structures, building 3D models, etc. [6, 7]. The volume of data used in modern X-ray tomography is about 1 GB, and the resolution of medical CT scanners is about 1 mm.

[1]Lomonosov Moscow State University, Moscow, Russia

In this article, the term “tomographic examination” is understood in a broad sense. It would be easier to formulate, what is not a tomographic examination in our understanding. Tomographic studies do not include the methods where one fixed detector and one fixed source are used. If the source and the detector change their position, then such examination may be considered tomographic, from the extended point of view. As a rule, in X-ray tomography the source and the detector run around the scanned object.

In some applications we cannot place sources and detectors around the object. For example, in Synthetic Aperture Radars (SAR), used for studying the surface of the Earth from the space or from an aircraft, sources and detectors may be positioned on a straight line, at a great distance from the examined object. However, such a measurement scheme provides amazing results and may be classified as a tomographic method. The synthetic aperture method allows increasing the image resolution up to hundreds of times. The problems of interpreting the data of such an experiment are inverse problems. It is solving the inverse problems, which makes it possible to obtain high resolution.

In this article, an attempt is made to assess the possibility to use supercomputer technologies for improving and developing new tomographic methods. One of the most promising medical diagnostic methods is a ultrasonic method. All the ultrasonic methods currently used in medicine are not tomographic. Development of ultrasound tomographic equipment at the model level is performed extensively in the United States, Germany, and Russia [1, 8–10]. One of the main problems in ultrasonic tomography is creating efficient methods for solving inverse problems of wave tomography. Unlike X-ray tomography, the inverse problems of wave tomography are nonlinear, and cannot be solved without using powerful supercomputers. Supercomputer technologies open new possibilities for designing wave-based tomography devices. This area is discussed in detail in the article. The developed wave tomography methods can be used for medical examinations, non-destructive testing in industry and for seismic studies as well. One of the most promising applications is developing medical ultrasonic tomography devices for differential diagnosis of breast cancer.

1. Linear tomographic models

1.1. Formulation and methods of solving inverse problems of X-ray tomography

X-ray tomography is one of the most used diagnostic technologies both in medicine and in non-destructive testing of industrial products. Resolution of modern medical tomographs reaches 1–2 mm; CT scanners with resolution of about 10 μm have been developed for industrial diagnostics at the micro-level.

The mathematical models used in X-ray tomography are the models of geometric optics [11]. The layer-by-layer approach is used for imaging three-dimensional objects. A 3D object is examined in layers, where in each of the layers the X-ray the absorption coefficient $f(x,y)$ is reconstructed as a two-dimensional function of Cartesian coordinates x, y. The possibility of solving inverse problems of X-ray tomography separately in each layer is associated with the unique properties of X-ray radiation, which is easy to absorb, but is very difficult to deflect from the straight line. The refractive index of any material differs from unity by not more than hundredths of percent for X-ray radiation. The inverse problems of X-ray tomography are therefore linear.

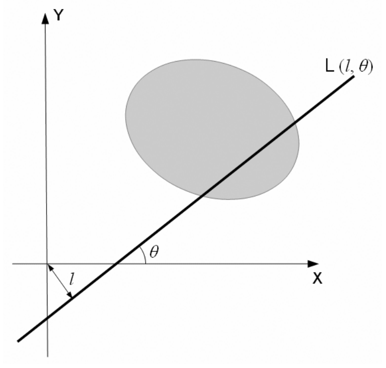


Figure 1. The scheme of X-ray tomography

Fig. 1 shows the scheme of X-ray tomography. The detectors measure the integral absorption along the lines L connecting each detector and a fixed source. Let us introduce a family of lines L(l, θ) in the plane Oxy. The parameter l varies between -∞ and +∞, θ varies between -$\pi/2$ and $\pi/2$. If l and θ can take all the values in the specified ranges, we call it a "full-range" tomography problem.

In reality, the studied object is usually compact, and it is sufficient to vary the parameter l in the specified limits. In the tomographic experiment, the integral absorption $p(l,\ \theta)$ along the line L(l, θ) is measured:

$$p(l, \theta) = \int_{L(l,\theta)} f(x, y)\, \mathrm{d}l. \tag{1}$$

If the function $f(x,y)$ is specified, calculation of $p(l,\ \theta)$ by formula (1) is a forward problem described by the Radon transform [12]. The inverse problem of X-ray tomography may be formulated as a problem of solving an operator equation

$$Af = \int_{L(l,\theta)} f(x, y)\, \mathrm{d}l = p(l, \theta). \tag{2}$$

For the sake of simplicity, we will consider a case where the functions $f(x,y)$ and $p(l,\ \theta)$ are square integrable. In this case, the operator A acts from the $\mathbf{L_2}$ space of the square integrable functions into the same $\mathbf{L_2}$ space. The inverse problem of X-ray tomography consists of finding a function $f(x,y)$, which is the solution of the equation (2), given a specified $p(l,\theta)$. The function $p(l,\theta)$ is obtained from the experiment, therefore it is specified with some error δ.

The distinctive feature of X-ray tomography problems is not only the linearity of the inverse problem (2), but the possibility of direct inversion of the operator A as well, i.e., the possibility of calculating the inverse operator A^{-1} in the domain of its definition $D_A \subset \mathbf{L_2}$. In the Hilbert space, the operator A has an adjoint operator A^*, which is determined by the relation $(Af, p) = (f, A^*p)$, which remains valid for any f and p from $\mathbf{L_2}$. Let's multiply the left and the right part of equation (2) by A^*. Equation (2) can then be re-written as follows:

$$A^*Af = \int\int \frac{f(\xi, \eta)\, \mathrm{d}\xi\, \mathrm{d}\eta}{\sqrt{(x-\xi)^2 + (y-\eta)^2}} = u(x, y), \tag{3}$$

where the function $u(x,y)$ is defined by the following relation

$$u(x, y) = A^*p = \int_\theta p(x\cos(\theta) + y\sin(\theta), \theta)\, \mathrm{d}\theta. \tag{4}$$

In the equation (3), the function $u(x, y)$ is obtained from the experiment according to the formula (4). The equation (3) depends only on the difference of its arguments. Applying the Fourier transform, we obtain

$$\frac{\tilde{F}(\omega_1, \omega_2)}{\sqrt{\omega_1^2 + \omega_2^2}} = \tilde{U}(\omega_1, \omega_2). \tag{5}$$

Here $\tilde{F}(\omega_1, \omega_2)$ represents the Fourier image of the function $f(x, y)$, and $\tilde{U}(\omega_1, \omega_2)$ represents the Fourier image of $u(x, y)$. Let's rewrite the equation (5) as

$$\tilde{F}(\omega_1, \omega_2) = \sqrt{\omega_1^2 + \omega_2^2}\, \tilde{U}(\omega_1, \omega_2). \tag{6}$$

The formula (6), in fact, inverts the operator A and provides an opportunity to write the solution of the original operator equation (2) explicitly in the case where full-range data are provided. Note that if a solution to the problem exists, then, as we have shown, it can be represented as (6), and is therefore the only solution. Applying the inverse Fourier transform to $\tilde{F}(\omega_1, \omega_2)$ in the representation (6), we can obtain the unknown function $f(x,y)$. From the computational perspective, solving the problem (6) is an easy task. The algorithm can be implemented using an ordinary personal computer.

Modern medical CT scanners examine three-dimensional objects. The data collection scheme is designed so that it makes it possible to reconstruct the image in many layers simultaneously. A 1000×1000×1000-point cubical grid is typically used to represent the data for 3D X-ray imaging. The number of two-dimensional sections may reach 1000 for a single patient. Since each of these sections can be reconstructed independently, it is reasonable to use GPUs, which provide at least 100 times speedup when performing the Fourier transform, compared to a single-processor PC. A powerful workstation equipped with several GPU devices is sufficient not only for reconstruction of cross-sections, but also for preliminary image processing, presenting 3D information, etc. [7, 13]. In particular, Digisens Company [6] has developed software for 3D X-ray tomography for GPU clusters.

1.2. The problem of reconstructing the image of the Earth's surface from the SAR (Synthetic Aperture Radar) data

The methods of synthesized aperture are used in various fields of wave sensing. One can distinguish two large groups of synthetic aperture radars: aviation-based radars with the resolution up to centimeters, and space-based radars with the resolution of about several meters [14-15]. Such systems are used for monitoring hard-to-reach areas of economic activity (monitoring, classification, assessment of vegetative cover biomass, monitoring of underground pipelines), etc. Synthetic aperture methods are used in medicine as well [16].

The SAR is one of the promising methods of remote examination of the Earth's surface from an aircraft. The use of electromagnetic radiation with the wavelength of about 0.1 m makes it possible to perform the monitoring virtually at any time of day and in all atmospheric conditions.

The main problem in using aircraft-based radars is their low resolving power. Significant remoteness of the radar carrier from the examined area (in case of using spacecraft, the distance may be hundreds of kilometers), natural limits of the size (aperture) of the antenna, sufficiently large wavelength of the probing radiation — all these factors result in an extremely large diameter of the area of interaction of radar radiation with the examined surface, up to several kilometers.

Radar resolution may be improved by using specialized methods of observation based on the principles of synthesized aperture. In remote sensing by synthetic aperture radars, the radar moves along the predetermined path, emits sounding pulses and received reflected signals. By synthesizing a virtual antenna hundreds and thousands of times larger in length than the actual size of the locator, one can improve the resolution by hundreds of times [14, 15, 17].

Processing the data from SAR systems is often performed in real-time. The peculiarity of the SAR image reconstruction problems is the huge volume of input data. It is advisable to use graphics processors [18, 19] to accelerate and parallelize the computations.

The scheme of observations and data recording in a SAR system is shown in fig. 2. A side looking radar is installed on the aircraft and is rigidly connected with the carrier. The radar emits probing signals $s(t)$ (usually, short pulse signals). The aircraft moves along a rectilinear trajectory above the area being mapped. The reflected signals $u(t)$ are recorded. The radar equipment allows recording both the amplitude and the phase of reflected signal.

Let's build a mathematical model of the SAR imaging process using the linear approximation. We will use the standard start-stop approximation [14], i.e., we neglect any changes in the geometrical parameters of the "carrier – surface" system during the emission-reception cycle. For the sake of simplicity, we assume that the carrier moves along a straight line parallel to the Earth surface. Such an approximation is acceptable in cases where the carrier trajectory and the terrain may be only slightly curved. However, a priori information about the curvature of the trajectory and the topography of the surface is required for obtaining high-precision images, and in some cases such information is available.

Let's define a Cartesian coordinates system OXYZ on the examined surface (fig. 2). We will assume that the radar carrier moves along a straight line parallel to the OX axis at the height H. Let us introduce the coordinate ξ along the carrier's trajectory, so that $\xi{=}x$. For convenience of further presentation, let's translate from the coordinate y to the coordinate ρ, where $\rho = \sqrt{y^2 + H^2}$. The coordinates $(x,\ \rho)$ define a point on the surface and are called the azimuthal distance and the slant-range distance.

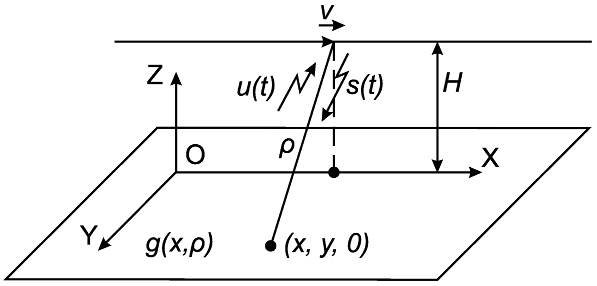


Figure 2. The scheme of synthetic aperture radar

We characterize the reflective properties of the Earth's surface by the coefficient $g(x,\rho)$. The function $g(x,\rho)$ is the sought-for value in the SAR imaging problems. Let's denote the signal emitted by the radar as $s(t)$. Due to the principle of superposition, the full signal $\hat{u}\left(\xi\left(t\right)\right)$ received by the radar at the coordinate $\xi\left(t\right)$ at the moment t, can be expressed as an integral over the plane of the examined surface (x,ρ) :

$$\hat{u}\left(\xi\left(t\right)\right) = \int\!\!\int s\left(t - \frac{2R\left(\xi\left(t\right), x, \rho\right)}{c}\right) P\left(\xi\left(t\right), x, \rho\right) g\left(x, \rho\right)\, \mathrm{d}x\, \mathrm{d}\rho, \tag{7}$$

where $R(\xi(t), x, \rho)$ is the distance between the radar carrier and the point (x, ρ) on the surface; the function P takes into account the beam shape of the antenna; the delay $\frac{2}{c}R(\xi(t), x, \rho)$ takes into account the duration of signal travel from the carrier to the point (x, ρ) and back; c is the speed of radio wave propagation.

Let's specify the probing signals. We will assume that pulse signals are emitted periodically, with the period of $2T_z$, in the form of a finite-duration train of high frequency oscillations with the frequency ω_0. Let's use the start-stop approximation, assuming that during a period of $2T_z$ the radar receives reflected signals being at the point $\xi_j = \xi(t_j) = vt_j$, which is the coordinate of the carrier along the trajectory, where v is the speed of the radar carrier movement. In the theory of synthetic aperture, t_j are called "slow" time. The parameter τ is called the "fast" time, and characterizes the slant-range distance from the radar carrier to the point on the studied surface. It can be shown that $\tau = const$ corresponds to a segment of the $y = const$ line on the ground.

For typical parameters of real radar systems [15], the equation (7) can be approximated as (8), using the second-order approximation [17]. For simplicity, here and further we will omit index j.

$$u(\xi, \tau) = \int_{\xi-L}^{\xi+L} \exp\left(\frac{i2\omega_0}{c^2\tau}(x-\xi)^2\right) g(x, \tau)\,\mathrm{d}x. \tag{8}$$

In this approximation, the mathematical model of the signal registration in the case of short probing pulses has a form of convolution with the Fresnel kernel in the segment $(-L, L)$ by the coordinate x for $-\infty < \xi < \infty$. By solving the inverse problem, we obtain the reflectivity coefficient $g(x, \tau)$ using approximately specified radar data $u(\xi, \tau)$.

A characteristic feature of the integral equation (8) is that it breaks down into independent one-dimensional integral equations. For each value of τ it is necessary to solve an integral equation of the first kind with the Fresnel kernel, the support set of which is limited to the interval $(-L, L)$. For typical radar parameters, the interval $(-L, L)$ contains hundreds of the Fresnel zones corresponding to the kernel $\exp\left(i2\omega_0 (x-\xi)^2/(c\rho)\right)$, which allows obtaining high-resolution image along the x coordinate using real radar data as input.

For the approximate computation of $g(x, \tau)$ one can use the following integral operator — "the inverse Fresnel transformation" with finite limits of integration:

$$\tilde{g}(x, \tau) = \int_{-L}^{L} \exp\left(-\frac{i2\omega_0}{c^2\tau}\xi^2\right) u(x-\xi, \tau)\,\mathrm{d}\xi. \tag{9}$$

In literature, the algorithm (9) is known as the focused synthesis algorithm [15]. As one can see, the focused synthesis algorithm employs a rather simple kind of one-dimensional convolution by the coordinate ξ . The fast Fourier transform (FFT) procedure can be used to implement the algorithm on a computer.

The data obtained by SAR can cover the size of the Earth's surface up to several tens of kilometers along the azimuthal and slant-range distances. The grid step of the recorded experimental data is about 1×1 m. This leads to the need in a huge amount of computations, sometimes performed in real time. However, the specificity of the algorithm is such that the calculations may be performed almost independently for each value of τ (the slant-range distance). This makes it possible to efficiently parallelize the calculations by allocating a dedicated range of τ values for each parallel process.

Fig. 3 shows the results of digital reconstruction of the image from the SAR mounted on the "Almaz" spacecraft [17]. Fig. 3 shows the original image (on the left) — the $u(\xi, \tau)$ function obtained by probing the Earth's surface. On the right, fig. 3 shows a fragment of the reconstructed image obtained as a result of solving the inverse problem using the focused synthesis method. As one can see from this example, the use of developed methods allows the resolving power of 11 – 15 m.

Further improvement of the SAR resolving power sets the task of developing mathematical methods and models for SAR image reconstruction. Formulations of the problems in this case are more complicated, because the model includes factors such as curvilinear trajectory of the radar carrier, the effects of the probed terrain surface, the curvature of the operator kernel support set, etc.

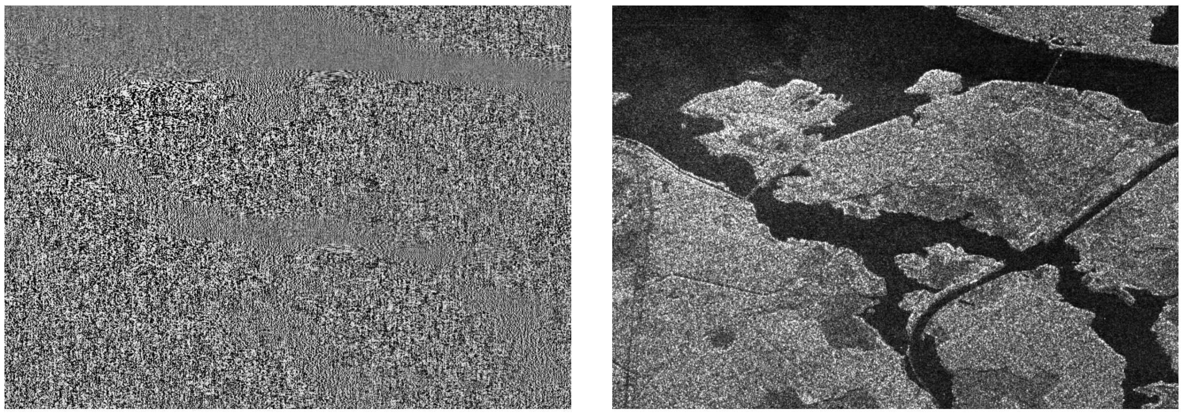


Figure 3. Reconstruction of the Earth's surface using the data obtained from the SAR on the "Almaz" spacecraft. Raw data is on the left, reconstructed image is on the right

2. The inverse problems of wave tomography in the framework of the integral approach

It has been shown above that efficient algorithms for solving inverse problems of X-ray tomography have been developed, which makes it possible to obtain detailed information about the internal structure of the examined objects. The results in the ultrasonic, acoustic, seismic and electromagnetic tomographic studies are significantly more modest. This is due to the fact that even if we use the simplest scalar wave models, interpreting the results of experiments requires solving complicated nonlinear three-dimensional inverse problems. Due to the significance of wave phenomena, generally speaking, it is incorrect to represent these three-dimensional problems as a set of two-dimensional problems and use the ray-based geometric optics models. Section 2 discusses the integral approach to solving the inverse problem of wave tomography based on the Green function method. Using the integral approach, the inverse problems can be reduced to a system of nonlinear Fredholm integral equations of the first kind [20–22]. The resulting problem is ill-posed. A rich arsenal of iterative methods for solving such problems has been developed [20]. To illustrate solving inverse problems in the integral approach using the iterative methods, we present the model problem of reconstructing the structure of the Earth's near-surface layers using wave sources of radiation. The model problems were solved using the "Lomonosov" supercomputer. The results show that the integral approach is efficient for solving the problems of wave diagnostics on coarse grids only.

2.1. Integral formulation of the inverse problem of wave tomography

Propagation of acoustic or electromagnetic radiation in an inhomogeneous medium in the simplest scalar wave approximation is described by the following well-known hyperbolic differential equation:

$$\Delta u(\boldsymbol{r}, \boldsymbol{q}, t) - \frac{1}{c^2(\boldsymbol{r})} u_{tt}(\boldsymbol{r}, \boldsymbol{q}, t) = \delta(\boldsymbol{r} - \boldsymbol{q}) f(t), \tag{10}$$

where $u(\boldsymbol{r},\boldsymbol{q},t)$ is a scalar wave field (for example, acoustic pressure in a liquid medium), which depends on the spatial variable $\boldsymbol{r} \in \mathbf{R}^3$ and the time $t \geq 0$, Δ is the Laplace operator, $c(\boldsymbol{r})$ is the velocity of wave propagation, $\delta(\boldsymbol{r}$ **-** $\boldsymbol{q})f(t)$ describes perturbation of the medium by a point source located at the point $\boldsymbol{q} \in \mathbf{R}^3$, $\delta(\cdot)$ is the Dirac's delta function. The initial conditions at t=0 are $u(\boldsymbol{r},\boldsymbol{q},0)=0$; $u_t(\boldsymbol{r},\boldsymbol{q},0)=0$.

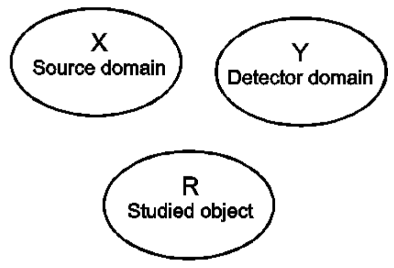


Figure 4. Scheme of tomographic examinations

Let us consider the following well-known general formulation of the inverse problem (fig. 4). Let the inhomogeneity of the medium be localized in the domain R. Let us assume that the sounding wave sources (positions of which are characterized by the vector $\boldsymbol{q}$) are located in the domain X ($\boldsymbol{q} \in$ X), and the detectors that measure the wave field $u(\boldsymbol{r},\boldsymbol{q},t)$ are present only in the domain Y ($\boldsymbol{r} \in$ Y). The domain R does not intersect with the domains X and Y, while X and Y may overlap or coincide. We will assume that $c(\boldsymbol{r})$ is a smooth function, which differs from the known constant c_0 only within the domain R ($0 < c_1 < c(\boldsymbol{r}) < c_2$). The function $f(t)$ that describes the initial pulse is known a priori. In the inverse problem, we are to determine the unknown function $c(\boldsymbol{r})$, $\boldsymbol{r} \in$ R, using experimental data $u(\boldsymbol{r},\boldsymbol{q},t)$ obtained for $\boldsymbol{r} \in$ Y. Such an inverse problem is nonlinear, since the function $u(\boldsymbol{r},\boldsymbol{q},t)$ is unknown, where $\boldsymbol{r} \notin$ Y.

Above, the problem has been formulated in the time domain. Let us present the formulation in the frequency domain. Applying the Fourier transform by the time variable t to the left and right part of the equation (10), the problem may be reduced to the Helmholtz equation, when the source is a point harmonic oscillator

$$\Delta u(\boldsymbol{r}, \boldsymbol{q}, \omega) \;+\; k^2(\boldsymbol{r}, \omega)\; u(\boldsymbol{r}, \boldsymbol{q}, \omega) \;=\; f(\boldsymbol{r}, \boldsymbol{q}, \omega), \tag{11}$$

where $k(\boldsymbol{r}, \omega) = \omega/c(\boldsymbol{r})$. In the scalar approximation, this equation describes the acoustic or electromagnetic field $u(\boldsymbol{r},\boldsymbol{q},\omega)$ generated by the source described by the function $f(\boldsymbol{r},\boldsymbol{q},\omega)$. If the source is located at the point $\boldsymbol{q} \in$ X, this function assumes the form $f(\boldsymbol{r},\boldsymbol{q},\omega) = -\delta(\boldsymbol{r}\text{-}\boldsymbol{q})$, where ω is the circular frequency of the sounding wave source. Inhomogeneity of the medium is caused only by changes in the phase velocity $c(\boldsymbol{r})$. Outside the inhomogeneous region $k(\boldsymbol{r},\omega) = k_0 = \omega/c_0$, where c_0=*const* is known.

In this inverse problem, same as above, we are to determine the unknown function $c(\boldsymbol{r})$, $\boldsymbol{r} \in \mathrm{R}$, using experimental data $u(\boldsymbol{r},\boldsymbol{q},\omega)$ obtained at points $\boldsymbol{r} \in \mathrm{Y}$ for some set of frequencies ω and source positions $\boldsymbol{q} \in \mathrm{X}$.

Let's consider the well-known approach to solving coefficient inverse problems in the framework of frequency-domain wave models, which is based on integral representation using the Green function. It is well known that the Green function for the equation (11) in a homogeneous medium satisfies the equation

$$\Delta G(\boldsymbol{r},\boldsymbol{q},\omega) + \frac{\omega^2}{c_0^2} G(\boldsymbol{r},\boldsymbol{q},\omega) = -\delta(\boldsymbol{r}-\boldsymbol{q})$$

and has the form

$$G(\boldsymbol{r},\boldsymbol{q},\omega) = \frac{1}{4\pi \left\|\boldsymbol{r}-\boldsymbol{q}\right\|}\ \exp(i\frac{\omega}{c_0}\left\|\boldsymbol{r}-\boldsymbol{q}\right\|).$$

Same as before, we assume that the wave field $u(\boldsymbol{r},\boldsymbol{q},\omega)$ is measured in the domain Y ($\boldsymbol{r} \in \mathrm{Y}$), i.e., the detectors run through the domain Y. The point sources run through the domain X ($\boldsymbol{q} \in \mathrm{X}$), the studied inhomogeneity is located in the compact domain R. By writing the equations separately for the domains R and Y, we will obtain the following nonlinear system of equations

$$F(Z) = 0, \tag{12}$$

where

$$F(Z) = \left\{ \begin{array}{c} u(\boldsymbol{r},\boldsymbol{q},\omega) - u_0(\boldsymbol{r},\boldsymbol{q},\omega) - \omega^2 \int_R G(\boldsymbol{r}',\boldsymbol{r},\omega)\xi(\boldsymbol{r}')u(\boldsymbol{r}',\boldsymbol{q},\omega)\,\mathrm{d}\boldsymbol{r}', \quad \boldsymbol{r} \in \mathrm{R} \\ \omega^2 \int_R G(\boldsymbol{r}',\boldsymbol{p},\omega)\xi(\boldsymbol{r}')u(\boldsymbol{r}',\boldsymbol{q},\omega)\,\mathrm{d}\boldsymbol{r}' - U(\boldsymbol{p},\boldsymbol{q},\omega), \quad \boldsymbol{p} \in \mathrm{Y} \end{array} \right. .$$

Here $U(\boldsymbol{p},\boldsymbol{q},\omega) = u(\boldsymbol{p},\boldsymbol{q},\omega) - u_0(\boldsymbol{p},\boldsymbol{q},\omega)$, $u_0(\boldsymbol{r},\boldsymbol{q},\omega) = -\int_{\mathrm{X}} G(\boldsymbol{r}',\boldsymbol{r},\omega)f(\boldsymbol{r}',\boldsymbol{q},\omega)d\boldsymbol{r}'$, $\xi(\boldsymbol{r}) = c_0^{-2} - c^{-2}(\boldsymbol{r})$, $Z = (\xi(\boldsymbol{r}), u(\boldsymbol{r},\boldsymbol{q},\omega))$.

The equation (12) is an integral equation of the first kind in certain Hilbert spaces with respect to $\xi(\boldsymbol{r})$ and $u(\boldsymbol{r},\boldsymbol{q},\omega)$. This equation is nonlinear, since the equation contains unknown functions as a product.

In this inverse problem, the properties of the medium $\xi(\boldsymbol{r})$ and the wave field $u(\boldsymbol{r},\boldsymbol{q},\omega)$ are considered unknown. The function $U(\boldsymbol{p},\boldsymbol{q},\omega)$ is obtained by measurement and is known in the domain $\boldsymbol{p} \in \mathrm{Y}$, $\boldsymbol{q} \in \mathrm{X}$ for some set of frequencies ω. The problem is to reconstruct the inhomogeneity of the medium $\xi(\boldsymbol{r})$ using these measurements.

2.2. Iterative algorithms for solving inverse problems of wave tomography in the integral approach

The resulting problem (12) is ill-posed and can be solved, for example, using the iterative-regularized Gauss — Newton method described in [20]

$$Z_{s+1} = Z_s - (F_s'^{*}F_s' + \alpha_s E)^{-1}(F_s'^{*}F(Z_s) + \alpha_s(Z_s - \zeta)), \tag{13}$$

where s is the iteration number, $F_s' = F'(Z_s)$ is the Frechet derivative for (12), $F_s'^{*}$ is the adjoint operator to F_s'. The suitable choice of the sequence α_s and the element ζ ensures regularizing properties of the algorithm.

It turns out that one can explicitly write down the formulas for $F'_s = F'(Z_s)$ and $F'^*_s F'_s$ [23]. However, even an ability to write explicit expressions for the matrices does not make it possible to effectively solve the inverse problem in question. The fact is that this task is extremely computationally expensive. The number of unknown values in the problem grows approximately as $O(tn^3)$, where n is the number of points in one direction of a three-dimensional grid, $t = N_\omega \cdot N_q$, where N_ω is the number of probing frequencies, N_q is the number of source positions.

In this Gauss – Newton procedure (13), the most computationally expensive operations are computing the matrix $F'^*_s F'_s$ and inverting the operator $(F'^*_s F'_s + \alpha_s E)$ at the each iteration of the Gauss–Newton procedure. It takes about $O(tn^9)$ addition and multiplication operations to compute the matrix. To invert the operator using an iterative methods it takes about $O(tn^6)$ operations at each step, and the number of steps is typically several hundred [24].

Such tasks with $n>20$ cannot be solved on a personal computer. A simple assessment shows that even if the computational power of the system is increased by 1000 times, we would only be able to increase the grid size by 2 times. Therefore, the presented integral algorithms are effective only on coarse grids.

2.3. Model calculations of inverse problems of wave tomography in integral formulation

To assess efficiency of the algorithms proposed in section 2.2, the model problems of reconstructing the structure of the Earth's subsurface layer have been solved [22, 23]. Numerical experiments have been performed with the use of the "Chebyshev" supercomputer. Iterative regularized Gauss–Newton method (13) was used to solve 3D inverse problems of wave tomography; the matrix was inverted using iterative methods that consisted of repeated vector-matrix multiplication operations. The advantage of such iterative methods, compared to direct methods of matrix inversion, is the possibility of parallelization — each parallel process has to store only several rows of the matrix and multiplies only these rows by the vector. Bearing in mind that the number of rows in the matrix can be up to several tens of thousands, such parallelization is very effective and, importantly, scalable to many processes.

In the problems of seismic surveys, sources and detectors are generally located on the Earth's surface. This, of course, imposes limitations on the resolving power of seismic methods. The situation in civil engineering, where small areas with dimensions of tens of meters are studied, is somewhat better. In this case, one can position the sources and detectors not only on the surface, but in the boreholes, surrounding the area being examined. Fig. 5 shows a layout of a model experiment (sources are marked with cones, detectors are marked with points). Sources and detectors were located on the Earth's surface (in the z=0 plane) in a 20×20 m area, and in vertical boreholes up to 10 m deep. The inhomogeneity (numerical phantom) was chosen in the form of two cubes. The dimensions of the cubes were 3 m and 6 m, the grid size was 1.5 m. The phase velocity was equal to 2000 m·s^{-1} in the cubes and 1000 m·s^{-1} in the surrounding volume. The problem was solved at the single frequency ω = 100 Hz. Fig. 6 shows a cross section of the reconstructed image in the plane OYZ on a 18×18×18-point grid.

As one can see from the numerical experiments, the nonlinear inverse problem in the integral representation is too computationally expensive to solve even on modern supercomputers. Even if 512 processor cores are used, computations can be performed in a reasonable time only on a 20^3–30^3-point grid, which is insufficient for complex tasks involving real measured data. As

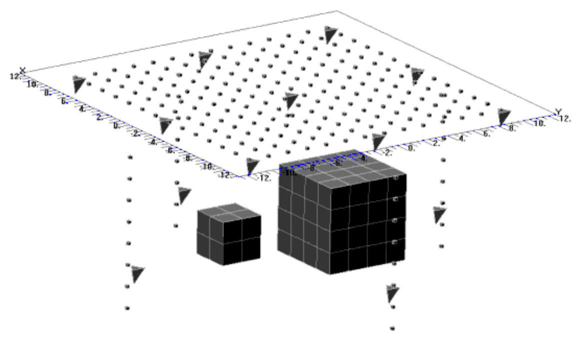


Figure 5. Layout of the numerical experiment with boreholes

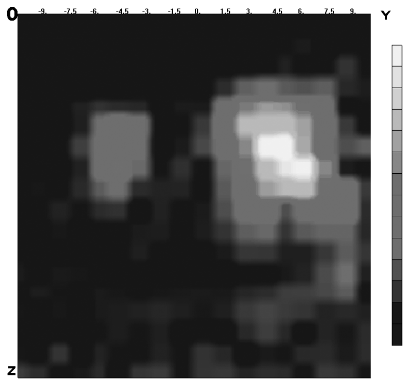


Figure 6. Cross-section in the OYZ plane of the reconstructed 3D image on a 18 × 18 × 18-point grid

the number of grid points increases, the number of operations required to solve the 3D inverse problem using the integral approach increases as n^9, where n is the number of grid points along either one coordinate. Thus, for obtaining more detailed information that is characteristic for tomography, it does not seem possible to implement the integral approach.

The natural solution in this case is linearization of integral equations (Born, Rytov approximations, etc.) [20]. Using of linearized approximations in the integral approach reduces significantly the amount of computations. However, the possibility of application of these approximations to real problems for complex heterogeneous media is rather limited due to the fact that the model is simplified. However, linearized approximations may be used in the case of a simple structure of the studied object (e.g., localized small inhomogeneities) or for obtaining the initial approximations for iterative procedures to solve nonlinear inverse problems.

3. Differential approach to inverse problems of wave tomography

In the previous section, the integral approach has been discussed. From the formal point of view, there are algorithms that are quite suitable for solving the obtained system of integral equations. However, implementation of these algorithms even on modern supercomputers is very difficult. Therefore, simplified models seem to be attractive in application to solving 3D problems of wave tomography.

The simplest model is the linear ray-based model similar to X-ray tomography, in which the ultrasonic pulses' travel time between sources and detectors is measured. Such models have been first used in the 70-ies [25] and are called time-of-flight (TOF) tomography. The integrals of the sound speed along the rays are obtained by measurements, and the sound speed distribution inside the object may be reconstructed using the inverse Radon transform (2). However, unlike X-rays, which practically do not deviate from straight lines, ultrasonic waves are subject to refraction, diffraction, multiple scattering, etc. Therefore, the ray model does not work well in heterogeneous media, and the reconstructed images are of poor quality. The advantage of the linear model is its simplicity and a possibility to implement the method even on a personal computer.

The ray model may be improved by introducing a correction for refraction. Having obtained a rough approximation of the sound speed using the linear model, one can plot the trajectories of the rays through the object with regard to refraction and measure the time of pulse arrival along the new ray path, thus defining the velocity structure with higher accuracy. This problem is a nonlinear one, and iterative methods are used to solve it. The required amount of computations is significantly greater than for the linear model, and graphics accelerators are used for image reconstruction [26].

Knowing the approximate velocity distribution, one can use the reflected ultrasonic waves for obtaining the reflectivity image. This method is called refraction-corrected reflection [8]. Without knowing the sound velocity, it is impossible to link the reflection to a specific point of the object. This leads to distortion of the reflectivity image obtained with regular, non-tomographic devices. The refraction-corrected reflection method allows obtaining a more accurate image; however, it has several disadvantages. First of all, the reflectivity image carries no information about parameters of the tissue, but only about the shape of the inhomogeneity's boundaries. Secondly, the ray model does not take into account diffraction effects, which become significant when the size of inhomogeneities is about 1–2 wavelengths. As a result, speed of sound image, which could be used for tissue characterization, though is better than in the linear model, still has too low quality for a number of tasks, such as early cancer diagnosis.

In the three dimensional variant, one can attempt to use the synthetic aperture method for tomographic image reconstruction. Presenting an unknown object as a set of points that scatter ultrasonic radiation in all directions, it is possible to determine accurately the coordinates of these points. However, the model of the synthesized aperture works well only in a homogeneous media. Diffraction can be partially taken into account, but accounting for refraction and multiple scattering in such a model is difficult. The resulting reflectivity image still has no information about the type of the tissue, but only about the shape of inhomogeneities. Implementation of the 3D method requires fairly large amount of computation, and GPU clusters are to be used [16].

Wave models describe ultrasound wave propagation in the tissue most accurately, and therefore make it possible to use the results of the measurements to the most extent. However, the inverse problems of image reconstruction for wave models are nonlinear and very complicated. The described above integral approach requires n^9 operations for solving a 3D problem on a n^3 grid. In a simplified formulation of parabolic approximation, the wave methods are developed in the work [8].

In recent years, breakthrough results have been achieved in the field of inverse problems of wave tomography [27–29]. In these works, the inverse problem of wave tomography is formulated as a coefficient inverse problem for a wave equation. The developed gradient-descent methods for solving inverse problems using the differential (FDTD) approach make it possible to solve the 3D problem of tomographic image reconstruction using just n^4 operations. With the help of the gradient methods, one can obtain a sound speed image, the accuracy of which is limited only by the accuracy of the measurements performed, and by the adequacy of the mathematical model to the real wave propagation processes. In practice, the computations may be performed in reasonable time using powerful graphics clusters containing about 30–40 GPU devices. Such clusters may be used as computing devices in medical and other tomographic systems. With the constant improvement of the technology, performance of the devices increases and power consumption decreases; this will make possible the mass use of such computing devices in ultrasonic tomographic systems for diagnosing breast cancer.

3.1. Formulation of the inverse problem of wave tomography as a coefficient inverse problem for wave equation

Let us consider the problem of reconstructing an unknown sound speed inside an object using the data obtained by a tomographic examination. First, let us consider a two-dimensional problem. Despite the fact that the process of wave propagation in an inhomogeneous medium is always three-dimensional, 2D (layer-by-layer) measurements make it possible to obtain acceptable image quality in some applications. Implementation of a layer-by-layer model requires much simpler equipment and less amount of computation.

The scheme of 2D (layer-by-layer) tomography is shown in fig. 7. The studied object G is placed in the homogeneous medium L with known properties. At the border of the examined area there are ultrasound sources 1 and detectors 2.

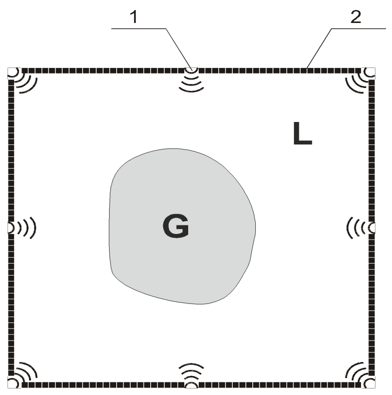


Figure 7. Scheme of 2D tomography

We will use a simple scalar wave model that describes pressure waves in liquid media. It is applicable for soft tissues, too. The acoustic pressure $u(\boldsymbol{r},t)$ satisfies the equation

$$c(\boldsymbol{r})u_{tt}(\boldsymbol{r},t) - \Delta u(\boldsymbol{r},t) = \delta(\boldsymbol{r}-\boldsymbol{q})\ f(t) \tag{14}$$

$$u(\boldsymbol{r},t=0) = u_t(\boldsymbol{r},t=0) = 0, \quad \partial_n u|_{ST} = p(\boldsymbol{r},t). \tag{15}$$

Here $c^{-0.5}(\boldsymbol{r}) = v(\boldsymbol{r})$ is the speed of ultrasound propagation, $\partial_n u|_{ST}$ is the derivative along the normal to the surface S in the domain $S \times (0,T)$, $\delta(\boldsymbol{r}-\boldsymbol{q})f(t)$ describes the initial pulse emitted by a point source at the point $\boldsymbol{q} \in \mathbf{R}^2$, Δ is the Laplace operator. Let us assume that heterogeneity of the studied object is caused only by the changes in the sound speed $c(\boldsymbol{r})$, while the sound speed in the medium L is constant and known, $c(\boldsymbol{r})$=c_0=const. The function $p(\boldsymbol{r},t)$ is also known. The inverse problem consists in finding unknown sound speed $c(\boldsymbol{r})$, which is a coefficient in the differential equation (14), using the measured acoustic pressure $U(\boldsymbol{s},t)$ at the detectors $\boldsymbol{s} \in$ S during the time $(0,T)$ with different positions of the ultrasound source $\boldsymbol{r}_0$. The inverse problem is formulated as a problem of minimization of the quadratic residual functional $\Phi(u(c))$ by c.

$$\Phi(u(c)) = \frac{1}{2}\int_0^T \int_S (u(\boldsymbol{s},t) - U(\boldsymbol{s},t))^2 \,\mathrm{d}\boldsymbol{s}\,\mathrm{d}t.$$

3.2. Representation of the gradient of the residual functional

We use gradient iterative methods to minimize the residual functional. Breakthrough results in solving coefficient inverse problems are associated with the possibility to formulate explicitly and compute the gradient $\Phi'(u(c))$ of the residual functional. In various formulations, the expression for the gradient has been derived in the works [27, 28, 24, 30]. According to [24], the gradient of the residual functional for the inverse problem (14–15) has the following form:

$$\Phi_c'(u(c)) = \int_0^T w_t(\boldsymbol{r},t)u_t(\boldsymbol{r},t)\,\mathrm{d}t. \tag{16}$$

Here $u(\boldsymbol{r},t)$ is the solution of the primary problem (14–15), and $w(\boldsymbol{r},t)$ ***s*** the solution of the "conjugate" problem:

$$c(\boldsymbol{r})w_{tt}(\boldsymbol{r},t) - \Delta w(\boldsymbol{r},t) = 0, \tag{17}$$

$$w(\boldsymbol{r},t=T) = w_t(\boldsymbol{r},t=T) = 0, \;\; \partial_n w|_{ST} = u|_{ST} - U. \tag{18}$$

Thus, for computing the gradient of the residual functional, it is necessary to solve both the main and the "conjugate" problems. Knowing Φ_c' from (16), we can build various iterative schemes for gradient minimization of the residual functional.

3.3. Numerical methods for solving coefficient inverse problems of wave tomography

We will use the finite-difference time-domain method (FDTD) to solve the inverse problem. In this formulation, solving the differential wave equations is reduced to solving algebraic finite-difference equations. For the sake of simplicity, we describe the two-dimensional version of the method. We define a uniform discrete grid in the computational domain:

$$v_{ijk} = \{(x_i, y_j, t_k) : x_i = ih, 0 \le i < n;\; y_j = jh, 0 \le j < n;\; t_k = k\tau, 0 \le k < m\},$$

where h is the grid step for spatial variables, τ is the time step. Let's use the following second-order approximation for second derivatives in the equation (14):

$$u_{tt}(\boldsymbol{r},t) = \frac{u_{ij}^{k+1} - 2u_{ij}^k + u_{ij}^{k-1}}{\tau^2}, \;\; u_{xx}(\boldsymbol{r},t) = \frac{u_{i+1j}^k - 2u_{ij}^k + u_{i-1j}^k}{h^2}.$$

A similar finite-difference scheme is used for $u_{yy}(\boldsymbol{r},t)$. In the area excluding the ultrasound sources, we obtain an explicit scheme for differential equation (14) for calculating the wave field u at the next time step *(k+1)*:

$$u_{ij}^{k+1} = \frac{1}{c_{ij}}\tau^2 \Delta u_{ij}^k + 2u_{ij}^k - u_{ij}^{k-1}, \tag{19}$$

where $\Delta u_{ij}^k = \frac{1}{h^2}\left\{(u_{i+1j}^k - 2u_{ij}^k + u_{i-1j}^k) + (u_{ij+1}^k - 2u_{ij}^k + u_{ij-1}^k)\right\}$ is a two-dimensional discrete Laplacian at the point $(i,\, j)$ at the time step k; u_{ij}^k is the value of $u(\boldsymbol{r},t)$ at the same point, c_{ij} is the value of c($\boldsymbol{r}$) at the point $(i,\, j)$. Parameters h and τ are connected by the Courant–Friedrichs–Lewy stability condition: $c^{-0.5}\tau < \frac{h}{\sqrt{2}}$. The time T is chosen long enough so that all ultrasonic waves have time to reach the detectors, and the signal level $U(\boldsymbol{s},t)$ for $t > T$ is negligible.

The gradient Φ'_c (4) is computed using the following formula

$$(grad\ \Phi)_{ij} = \sum_{k=0}^{m} \frac{u_{ij}^{k+1} - u_{ij}^{k-1}}{2\tau} \frac{w_{ij}^{k+1} - w_{ij}^{k-1}}{2\tau} \tau. \tag{20}$$

For the three-dimensional variant, the FDTD formulas are similar. Three-dimensional Laplacian can be approximated as described in [31].

We used the following iterative process to minimize the residual functional (4). For the initial approximation, we assume $c^{(0)} = c_0 = const$. At each iteration (m), the following steps are performed:

1. Calculation of the initial pulse from each source.

2. Solving the direct problem (14–15) for the current iterative approximation $c^{(m)}$. Propagation of the ultrasonic wave field $u^{(m)}(\boldsymbol{r},t)$ is calculated by formula (19). Values of $u(\boldsymbol{s},t)$ are calculated at the detectors $\boldsymbol{s}$.

3. Calculation of the residual functional $\Phi^{(m)} = \Phi(u^{(m)}(\boldsymbol{r}))$.

4. Solving the "conjugate" problem (17–18) for $w^{(m)}(\boldsymbol{r},t)$.

5. Calculation of the gradient by formula (20) for all the sources. The gradient $grad\Phi^{(m)}$ is the sum of gradients obtained for each source.

6. Correction of the current iterative approximation: $c^{(m+1)} = c^{(m)} + \lambda^{(m)}\ grad\ \Phi^{(m)}$. The process returns to stage 2.

The increment $\lambda^{(m)}$ is determined from a priori considerations. During subsequent iterations, $\lambda^{(m)}$ is increased if $\Phi^{(m)} < \Phi^{(m-1)}$ and decreased if $\Phi^{(m)} \geq \Phi^{(m-1)}$. It takes $50 - 200$ gradient descent iterations to find the solution, depending on the tomographic scheme and accuracy of measurements.

3.4. Numerical simulations of layer-by-layer ultrasonic tomography problems

The tomographic scheme and the parameter values for the numerical experiments were chosen primarily for the purpose of modeling the problem of breast ultrasound tomography. We used the mathematical model described by the equation (14–15). Modeling was performed according to the scheme in fig. 7. First, we computed acoustic pressure $U(\boldsymbol{s},t)$ at the detectors for the given sound speed distribution $c(\boldsymbol{r})$, and then we solved the inverse problem: the sound speed $c(\boldsymbol{r})$ was reconstructed using the data $U(\boldsymbol{s},t)$ at the detectors.

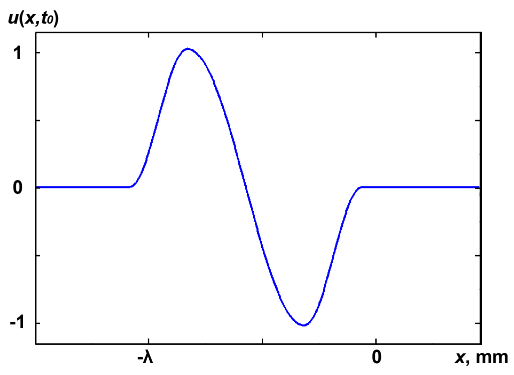


Figure 8. Waveform of the initial pulse

Waveform of the initial pulse is shown in fig. 8. For a 2D model, the following values of parameters were used: pulse width λ=5 mm, sound speed range within the object — 1.43–1.6 km $\cdot$ s^{-1} , in the medium — 1.5 km$\cdot$ s^{-1} , the size of the computational domain h=204 mm, FDTD grid size 1024×1024 pixels. Eight sources were used in the numerical experiment, by two on each side of the square. The detectors were located around the perimeter of the square at the distance of 0.8 mm from each other.

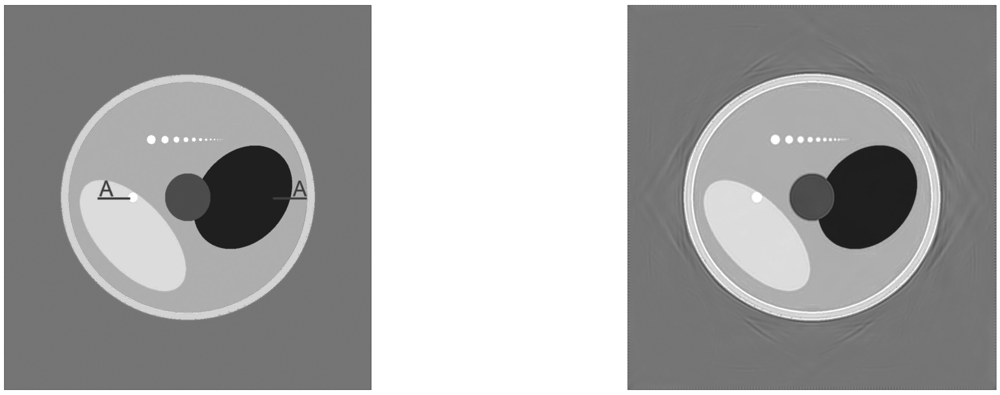


Figure 9. The two-dimensional numerical experiment: exact image (numerical phantom) – (left); reconstructed image – (right)

Fig. 9 (left) shows the numerical sound speed phantom, fig. 8 (right) shows the reconstructed image. As one can see in the figures, in such a configuration of sources and detectors, the image may be reconstructed quite accurately even when using a small number of sources, assuming that measurement errors are sufficiently low. The computing time for solving a 2D problem using 8 NVIDIA Tesla X2070 graphics processors (one for each source) was about 30 minutes, with 200 iterations performed.

3.5. Applicability of the layer-by-layer models to solving three-dimensional problems of wave tomography

The use of layer-by-layer tomographic schemes is not strictly justified for ultrasonic applications. This idea has been borrowed from X-ray tomography, where the use of layer-by-layer examination is possible due to the fact that X-rays almost do not deviate from a straight line. In ultrasonic tomography, the hope for allocating thin layers similar to X-ray tomography is not always justified, because the rays can bend due to refraction both within the studied two-dimensional layer and in the third dimension, and leave the layer. Such wave phenomena as diffraction and multiple scattering aren't taken into account by the ray-based model too.

We performed numerical simulations to study the possibility of using two-dimensional layer-by-layer schemes for ultrasonic tomographic reconstruction of 3D objects. Such studies are of interest because of great popularity of the layered approach in ultrasonic tomography [1, 29, 32, 33]. In this paper we focus on the task of differential diagnostics of breast cancer, which requires resolving small-sized and low-contrast inhomogeneities in the medium.

The scheme of the numerical experiment is shown in fig. 10. It shows a cubic computational domain, inside which the simulation of ultrasonic wave interaction with the object Ω (designated as 1) was performed. We chose the simplest three-dimensional object — a homogeneous ball placed in a homogeneous medium. The choice of a ball as a test object was determined by the fact that for a spherically symmetric object it is possible to obtain explicitly the analytical solution

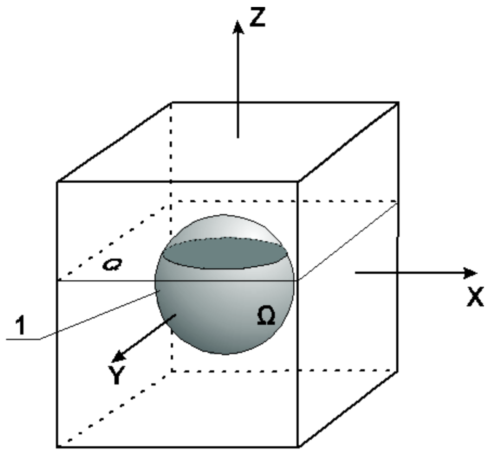


Figure 10. Scheme of the 3D numerical experiment

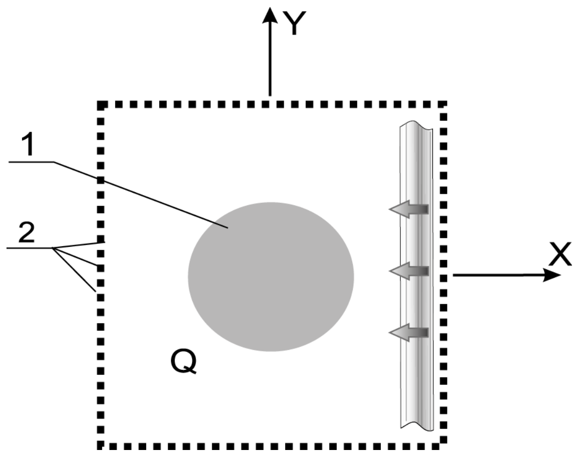


Figure 11. Scheme of the 2D numerical experiment

for the direct problem of pressure-wave propagation using decomposition by special functions (Legendre polynomials and spherical Hankel functions of the second kind). This approach has been well studied in literature [34]. Using the method of decomposing the solution into series of special functions, it is possible to calculate the wave field at any point in the selected plane Q of three-dimensional space that crosses the ball. This way, we solve the three-dimensional forward problem of wave propagation. Next, we solve the two-dimensional inverse problem (fig. 11) based on the data at the detectors (designated as 2) in the plane Q using numerical methods, and then we can compare the obtained result with the exact cross-section of the ball (designated by the digit 1).

The center of the cube is located at the point of reference, the Z axis is directed upwards. The size of the cube is 20×20×20 cm. The homogeneous medium has the density of 1 g/cm^3 and the sound speed of c_0=1500 m·s^{-1} (water). The inhomogeneity has a spherical shape with radius of 6 cm and the sound speed of c_Ω=1600 m·s^{-1}. The density of the ball is assumed to be equal to the density of the environment, 1 g/cm^3.

A plane wave falling on the ball from four sides along the X and Y axes (indicated by arrows on fig. 11) was used as the sounding wave for analytical solution of the direct problem in 3D. The pulse width was λ=7 mm. The distance between the detectors located on the sides of the examined cross-section Q did not exceed $\lambda/3$.

For the purpose of studying the possibilities of using layer-by-layer schemes for 3D wave-tomography imaging, two-dimensional inverse problems were solved numerically in various cross-sections Q perpendicular to the Z-axis. For numerical solution of 2D inverse problems, the size of cross-section Q was 20×20 cm, FDTD grid size was 1000×1000 pixels.

Fig. 12 b shows the reconstructed velocity structure in the planes $z = 0$ cm, and $z = 4$ cm, where the plane z=0 passes through the center of the ball. Fig. 12 a shows the exact cross-sections of the ball for the same z. It can be seen that for the central cross-section $z = 0$, the reconstructed solution coincides with the exact solution quite well. With increasing z, the effect of refraction increases too, this results in distortion of the shape and appearance of artifacts.

Thus, we can assume that in principle the layer-by-layer scheme is applicable to ultrasound tomography, it is possible to obtain both the shape of the heterogeneous object (albeit with some inaccuracy) and the absolute sound speed value. However, the use of layer-by-layer schemes can lead to the distortion of geometrical shape, appearance of artifacts; the images are blurred along

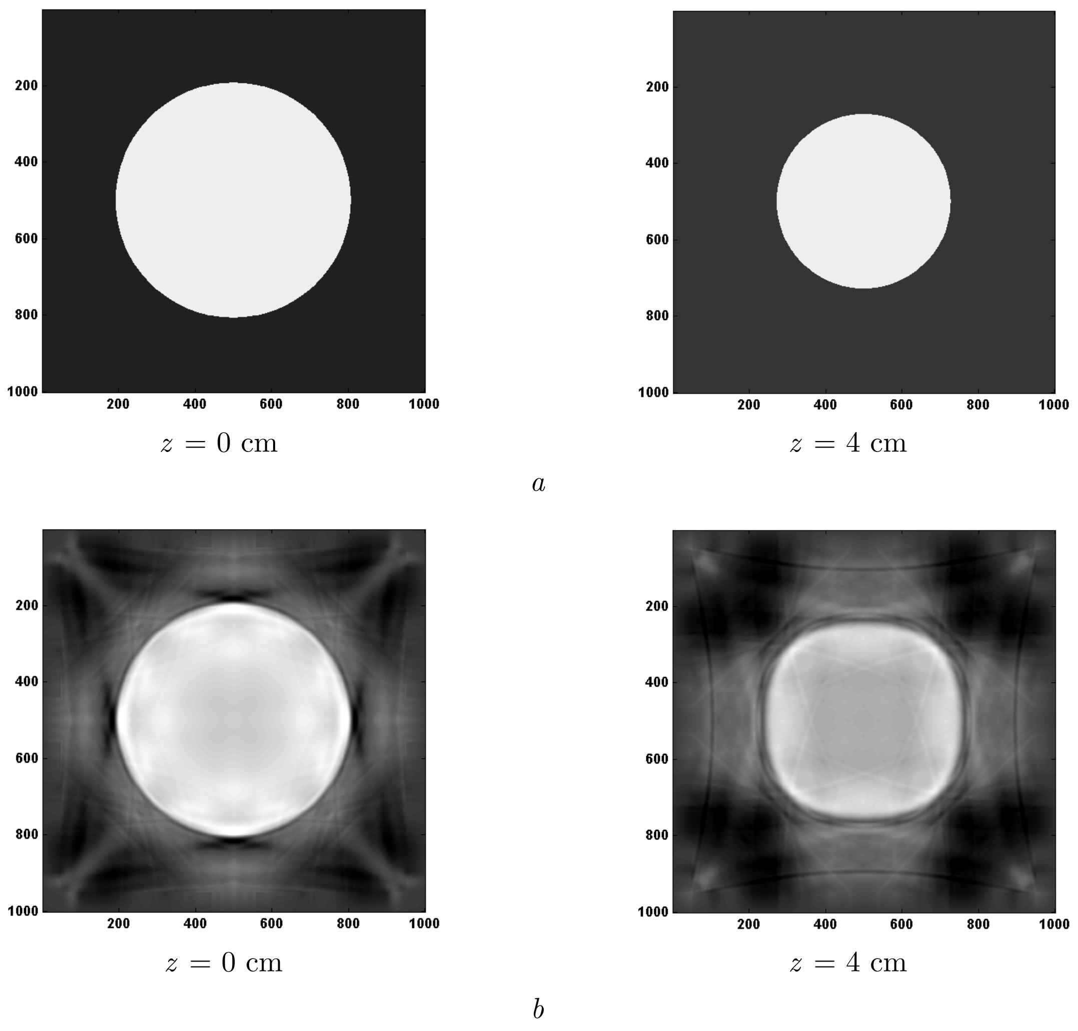


Figure 12. The model problem of reconstruction of a three-dimensional ball: *a* — ball cross-sections in planes z=const, *b* — reconstructed images

the Z-axis, which is consistent with the results of [35]. The problems that occur in layer-by-layer tomography can be most easily described at the physical level as restrictions of the ray-based approximation. Unlike X-rays, ultrasonic beams are not straight lines. Moreover, they can leave the studied cross-section. These issues restrict the use of layer-by-layer imaging schemes.

3.6. Numerical simulations. The three-dimensional formulation of the inverse problem

In the three-dimensional numerical simulation of wave-tomography imaging, the studied object was placed inside a cubic computational domain, at the borders of which the sources and detectors were located. The scheme of the numerical experiment is shown in fig. 13. The ultrasound sources S were placed at 24 positions in two planes $z{=}h_0$ and $z{=}h_1$. The pulse width (wavelength) λ was chosen as 5 mm, the sound speed range inside the object was 1.43 – 1.6 km·s^{-1}, the sound speed in the medium was 1.5 km·s^{-1}.

The detectors were located on the faces of the 176×176×176 mm cubic computational domain Ω, including the bottom face $z{=}0$ and top face $z{=}H$. The distance between the adjacent detectors was 2 mm. Thus, each face of the cube is an array of 54×54 detectors. The cross-sections of the reconstructed 3D sound speed image in the $z{=}const$ planes are shown in fig. 14,

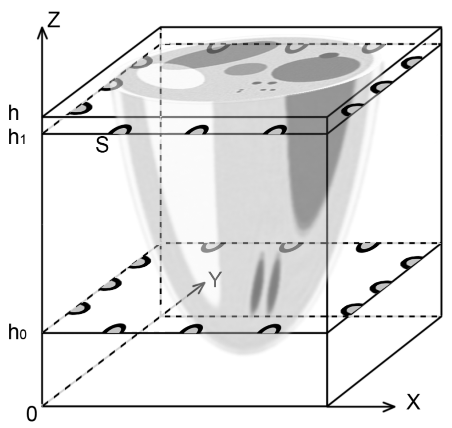


Figure 13. Scheme of the 3D numerical simulation

and in the *y=const* planes in fig. 15. It can be seen that in this scheme even the smallest inclusions (diameter of which in this example is about 1 mm) are well reconstructed.

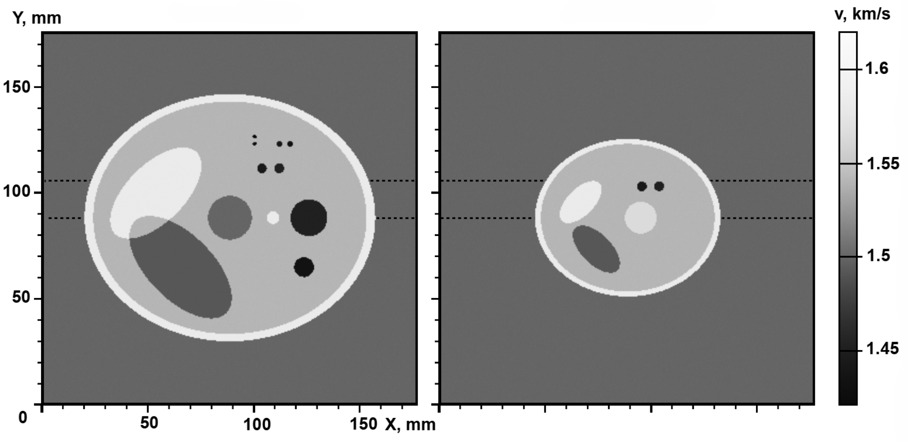


Figure 14. Cross-sections of the reconstructed 3D image in $z = const$ planes

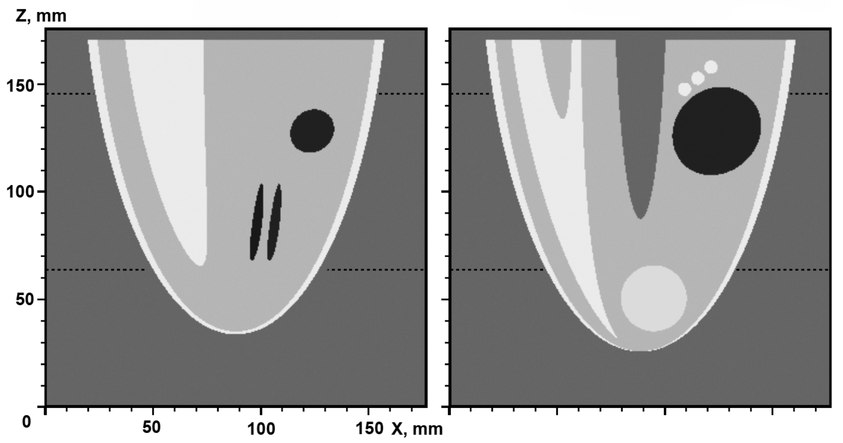


Figure 15. Cross-sections of the reconstructed 3D image in $y = const$ planes

The 3D problems were solved on a 352×352×352 - point FDTD grid. The time of computing a 3D task on a cluster of 24 NVidia Tesla X2070 devices was about 2 hours, 150 gradient descent iterations have been performed. As one can see from the figures, the sound speed image is reconstructed fairly well. Note that solving an inverse problem with over 30 million unknowns required less than 20 thousand detectors. Reconstructing a similar 3D image using a ray model would have required several million rays. Modern X-ray tomographs use up to 10^9 rays. It is almost impossible to implement such a number of ultrasonic detectors. One of the advantages of the wave model is the possibility of reconstructing the tomographic image using a much smaller number of sources and detectors, since the wave model uses all of the information about the wave field $u(\boldsymbol{r},t)$ obtained from the detectors. Using large amounts of information indicates the increase of complexity of image reconstruction algorithms, and the necessity of using supercomputers.

3.7. The problems of designing ultrasonic tomography devices for medical examinations

The developed algorithms and software make it possible to perform numerical simulations for various parameters of the experiment, such as the waveform of the sounding pulse, the number and location of sources and detectors, accuracy of measurement, etc. Supercomputer modeling makes it possible to determine the optimal parameters that could be used for developing tomographic systems [36, 37].

Wavelength (or pulse width and frequency bandwidth) of the ultrasonic radiation is an important parameter. Ultrasonic tomography equipment for early breast cancer diagnosis should detect inhomogeneities of about 3 mm in size and distinguish them from the surrounding tissues. Sound speed inside an inhomogeneity should be determined within several percents, since in soft tissues sound speed in the tumor and that in the healthy tissue differ only slightly — by less than 10%. There are two main factors that influence the choice of the ultrasonic radiation wavelength. First, it is ultrasound absorption in tissues, which increases as the frequency increases. Despite the fact that high frequencies in the 1 – 10 MHz range are widely used in ultrasound equipment, the use of such high frequencies in the tomographic scheme is problematic, since tomographic reconstruction requires high accuracy of measurements, and too weak signal does not allow this. Secondly, increasing the frequency results in increasing the complexity of equipment, such as the number of detectors, quality of the signal processing hardware, etc.

Let's show that it is possible to achieve the resolution of about 3 mm using the wavelength of about 5 mm. We will illustrate this fact by the following numerical simulations. In the 3D numerical experiment, 24 source were used, located in planes $z=h_0$ and $z=h_1$ as shown on fig. 13. The detectors were located on all the faces of the cubic computational domain; the inter-detector distance is 2 mm. The waveform of the sounding pulse is shown in fig. 8. The calculations were made for pulses with width λ=5 mm and λ=10 mm.

Fig. 16 shows cross-sections of the reconstructed 3D sound speed image for initial pulses with wavelength of λ=5 mm and 10 mm. Fig. 17 shows enlarged fragments of the images. It can be seen that at λ=10 mm, the quality of restored 2 mm details is not satisfactory. Therefore, the optimal wavelength would be $\lambda \approx 5$ mm. Note that such a high spatial resolution is obtained even having a very low contrast of the test object, sound speed variation in which does not exceed 10%.

The quality of reconstructed tomographic images depends on the accuracy of the input data. The error level of the input data depends on both random factors, such as crosstalk, equipment noise, etc., and on the setup, such as sampling frequency and the number of digitization levels.

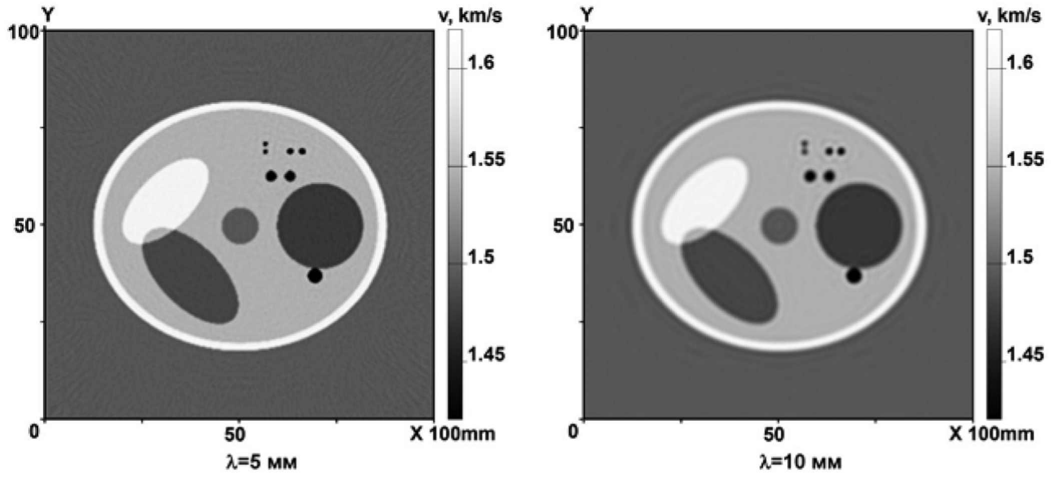


Figure 16. The images reconstructed using 5mm and 10mm initial pulses

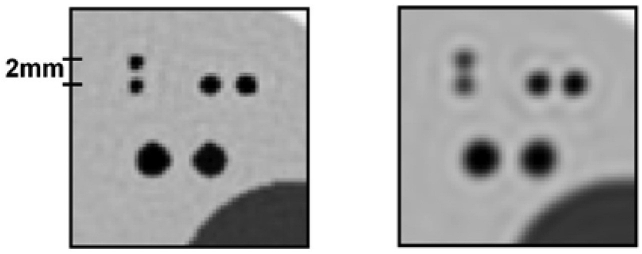


Figure 17. Magnified images reconstructed using 5mm (left) and 10mm (right) initial pulses

Let's define, with the help of mathematical modeling, the allowable error levels of the input data for ultrasound tomography. The level of input data error δ is measured relative to the maximum amplitude of the signal at the detectors: A = max($U(\boldsymbol{s},t)$) – min($U(\boldsymbol{s},t)$), which is assumed as 100%.

Fig. 18 shows cross-sections of the 3D images reconstructed using perturbed data. The error levels of 0.015·A (1.5%) and 0.05·A (5%) were simulated by adding pseudo-random noise to the input data $U(\boldsymbol{s},t)$ at the sampling rate of 10 MHz. The sources were located according to fig. 13, in two planes z=h_0 and z=h_1, the detectors were located on all the faces of the cube with 2.5 mm spacing between the adjacent detectors.

From the fig. 18 it can be seen that with the error level of 1.5%, it is still possible to obtain high-resolution images. With the error of 5%, the restored image still contains some information about the small inclusions present in the numerical phantom, but the image quality deteriorates significantly.

Comparison of the images (fig. 19) shows that precision of the input data is important, the quality of image reconstruction improves as the error decreases, and the tomographic instruments are intended to make this error as small as possible. Thus, the design should take into account the overall level of input data error not exceeding 3 to 5%.

4. Discussion and conclusion

1. The article discusses various diagnostic tasks where the tomographic approach is used. The tomographic approach involves studying the object from various points of view. Tomographic methods can significantly increase the resolution of both medical and industrial di-

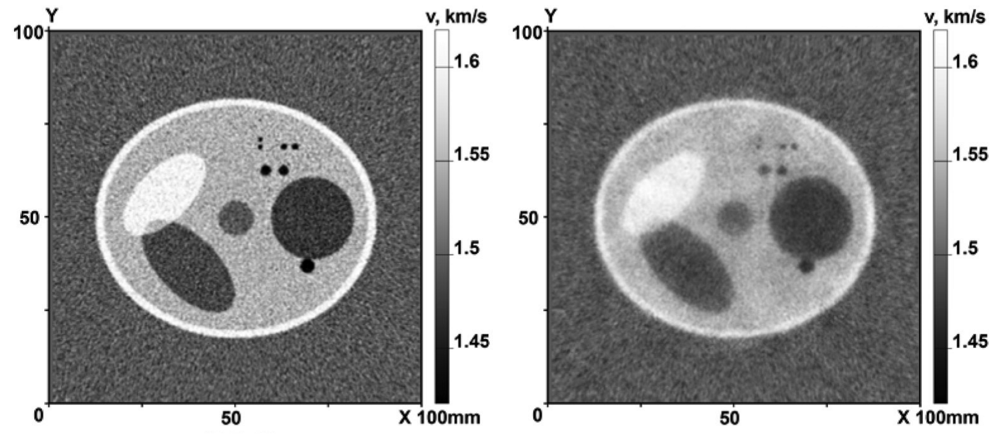


Figure 18. Cross-section in the Z=66 mm plane obtained with the error levels of 1.5% (left) and 5% (right)

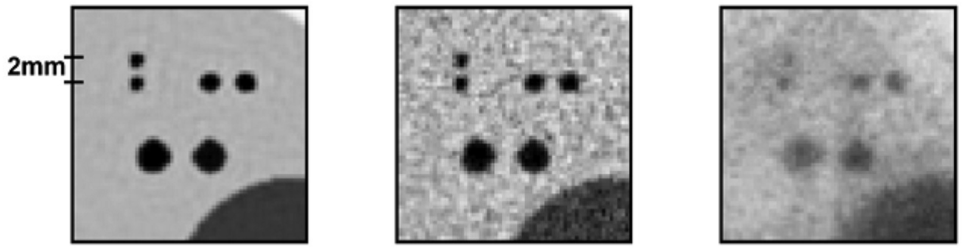


Figure 19. Magnified images obtained without error (on the left) with the error level of 1.5% (center) and 5% (right)

agnostic systems. One of the variants of tomographic imaging is the method of synthesized aperture, where the object is examined only from a narrow angular range. However, for some tasks, like radar surveys, this approach makes it possible to increase the resolution of the obtained images by hundreds of times. From the mathematical point of view, most of the tasks of synthetic aperture data interpretation can be reduced to solving linear problems. Using SAR in on-line mode involves processing huge amounts of data in real time, which requires the use of graphics accelerators.

Unfortunately, from the applications point of view, linear models of synthesized aperture are effective only for a limited class of applications. An example of such tasks may be studying the Earth's surface from a spacecraft in the centimeter wavelength range. An important limitation of the model is the fact that the medium between the radiation source and the studied object should be homogeneous. For reconstructing the internal structure of objects, the use of the synthetic aperture method is limited.

2. Tomographic examination, where the object can be studied using numerous source and detector positions, ideally — viewed under all angles (full-range tomography), allows precise reconstruction of the internal structure of examined objects. X-ray tomographs are used most widely; they are used both in medicine and in non-destructive testing. A modern trend in the medicine is limiting the use of harmful ionizing radiation. From this point of view, development of methods of ultrasonic tomography is promising. The most relevant medical application is differential diagnosis of breast cancer.

3. Unlike X-ray tomography, inverse problems of wave tomography are nonlinear and three-dimensional. This article describes modern methods of solving inverse problems of wave tomography as coefficient inverse problems for hyperbolic differential equations. Such problems cannot be solved without using supercomputers. This article describes numerical simulations that show efficiency of algorithms for solving 2D and 3D coefficient inverse problems arising in ultrasound tomography. Both CPU and GPU processors were used to solve the test problems.

4. The three-dimensional model is the most adequate to the reality. The developed algorithms are focused on using GPU clusters, which are most suitable for solving inverse problems in 3D formulation. The performance-comparable systems based on general-purpose processors have 5–10 times higher cost than GPU-based systems. The FDTD method employed can be efficiently parallelized on SIMD/SPMD architectures; however, this method is memory-intensive and requires high memory bandwidth. Numerical simulations have been performed using NVidia Tesla X2070, NVidia GeForce GTX 460 and NVidia GeForce GTX Titan graphics cards. The time spent for calculating the gradient for a single ultrasound source in a three-dimensional inverse problem for NVidia Tesla X2070 and GTX Titan was $\sim$45 seconds, therefore the complete image reconstruction ($\sim$120 gradient descent iterations) takes about 2 hours. The computation time was twice as long for the entry-level NVidia GeForce GTX 460 graphics card. Calculations for each source can be performed independently; therefore, the task can be easily parallelized to a number of devices corresponding to the number of sources (possibly 25–40 in an actual tomographic setup), so the inverse problem can be solved for all the sources in about 2 hours using such GPU cluster. Almost all the data involved in the process were placed in the onboard GPU memory, the required volume of which is 3 GB for the image size of 400^3. If larger image dimensions are required, computations for each source may be parallelized to 2–4 GPU devices to maintain acceptable computation time.

5. The technology of tomographic image reconstruction by solving coefficient inverse problems may be implemented at present time in tomographic facilities, although it is at the limit of the capabilities of modern technology. At present, supercomputers are just starting to be used for medical research. However, the rate of computer technology development has enormous potential for development of new generation diagnostic systems based on wave tomography. In particular, in 2016, launch of NVidia Pascal or AMD Polaris devices based on HBM (High Bandwidth Memory) architecture has been announced, which allows achieving the performance of up to 1 TB/sec, which would speed up the calculations by 3–4 times compared to the currently available graphics cards, simultaneously reducing power consumption. The use of such computing devices will make it possible to build computing systems that could be widely used in medical ultrasonic tomography facilities.

6. From the mathematical point of view, an important parameter for solving an inverse problem is the error of the obtained approximate solution to the inverse problem. It is a strictly mathematical concept. Let $\bar{c}(\boldsymbol{r})$ be the exact solution for the given input data $u{=}U$. Let the input information be specified with error δ, that is, u_δ is set so that $||u_\delta - u|| \leq \delta$. Following the basics of the regularizing algorithms theory, the concept of an algorithm R for the approximate solution R$=$R(u_δ, δ) is introduced [43]. The algorithm allows finding the approximate solution of the problem $c_\delta{=}\mathrm{R}(u_\delta, \delta)$, according to the given values of δ and u_δ. The algorithm is called regularizing if $c_\delta \to \bar{c}$ if $\delta \to 0$ [38, 39]. The value $\varepsilon(\delta) = ||c_\delta - \bar{c}\,||$ is the error of the approximate solution. Let us note right away that in fact the error of the approximate solution is not only the function of δ, but it also depends on the exact solution $\bar{c}$, that is,

$||c_\delta - \bar{c}|| = \varepsilon(\delta, \bar{c})$. Thus, the error of the approximate solution can be defined only at each fixed point $c = \bar{c}$. Naturally, the error of the approximate solution is different for different points $\bar{c}$. When solving the model problem, $\bar{c}$ is fixed, and the model problems demonstrate how the approximate solution is close to exact solution $\bar{c}$. With $\delta \to 0$, the approximate solution tends to the exact solution of the problem, which corresponds to the infinitely high resolution that can be obtained with the error tending to zero. The developed algorithms have all the characteristics listed above. The possibility to approximate the exact solution of the problem arbitrarily precise, if $\delta \to 0$ means that it is possible to achieve arbitrarily high resolving power, provided that the error level δ is sufficiently small.

The study of the error $\varepsilon(\delta,\ c)$ for a fixed level δ is most interesting. For solving the model problems, it is necessary to choose the test object (numerical phantom) that is as close to reality as possible. If δ and $\bar{c}$ are fixed, the magnitude of error $\varepsilon(\delta, \bar{c})$ depends on the scheme of the experiment, the number of the sources and their positions, the number of detectors and their size, the number of grid points, and on other parameters. The task of mathematical modeling is choosing the optimum parameters of ultrasonic equipment, which requires performing many numerical simulations with various parameters of the tomographic setup. When designing ultrasonic tomography equipment, mathematical modeling makes it possible to save not only money, but development time as well. The arising mathematical problems cannot be solved without using supercomputers.

This work was supported by the Russian Foundation for Basic Research (Project No. 14-07-00078-a). The reported study was supported by the Supercomputing Center of Lomonosov Moscow State University [40].

1. Glide-Hurst C.K., Duric N., Littrup P. Volumetric breast density evaluation from ultrasound tomography images // Medical Physics. 2008. 35. 3988-3997. DOI: 10.1118/1.2964092.
2. Matej S., Fessler J.A., Kazantsev I.G. Iterative tomographic image reconstruction using fourier-based forward and back-projectors IEEE Transactions on Medical Imaging. 2004. Vol. 23. p. 401. DOI: 10.1109/NSSMIC.2002.1239651.
3. Bazulin A.E., Bazulin E.G., Vopilkin A.K., Kokolev S.A., Romashkin S.V., Tikhonov D.S. Application of 3D coherent processing in ultrasonic testing // Russian journal of nondestructive testing, Vol. 50, No. 2, 2014. pp. 92-108. DOI: 10.1134/S1061830914020028
4. Levin G.G., Vishnyakov G.N., Minaev V.L., Latushko M.I., Pickalov V.V., Belyakov V.K., Sukhenko E.P., Demyanenko A.V. Shearing interference microscopy for tomography of living cells // Proceedings of SPIE, Vol. 9536, 2015. pp. 95360G. doi: 10.1117/12.2183717
5. Glinskii B.M., Sobisevich A.L., Khairetdinov M.S. Experience of vibroseismic sounding of complex geological structures (with the Shugo mud volcano as an example) Doklady earth sciences, 2007, Vol. 413, No. 3. pp.397-401. DOI: 10.1134/S1028334X07030178
6. Digisens 3D tomography software solutions. http://www.digisens3d.com/
7. Jia X., Yan H., Cervino L., Folkerts M., Jiang S.B. A GPU tool for efficient, accurate, and realistic simulation of cone beam CT projections // Med Phys. 2012 Dec;39 (12):7368-78. doi: 10.1118/1.4766436. DOI: 10.1118/1.4766436.
8. Wiskin J., Borup D., Johnson S., Berggren M., Robinson D., Smith J., Chen J., Parisky Y., Klock J. Inverse scattering and refraction corrected reflection for breast cancer imaging // Proc. of SPIE Vol. 7629 76290K-1. doi: 10.1117/12.844910.
9. Gemmeke H., Menshikov A., Tchernikovski D., Berger L., Gobel G., Birk M., Zapf M. and Ruiter N. V. 2010 Hardware setup for the next generation of 3D ultrasound computer

tomography Nuclear Science Symposium Conference Record (NSS/MIC) IEEE pp 2449-54. DOI: 10.1109/NSSMIC.2010.5874228

10. Burov V.A., Zotov D.I., Rumyantseva O.D. Reconstruction of spatial distributions of sound velocity and absorption in soft biological tissues using model ultrasonic tomographic data // Acoustical Physics, 2014, V 60, No. 4, pp. 479-491. DOI: 10.1134/S1063771014040022
11. Kak A, Slaney M. Principles of computerized tomographic imaging. SIAM, 2001.
12. Radon, J.; Parks, P.C. (translator) (1986), On the determination of functions from their integral values along certain manifolds, IEEE Transactions on Medical Imaging 5 (4): 170–176, doi:10.1109/TMI.1986.4307775, PMID 18244009.
13. Schmidt S., Gade-Nielsen N.F., Hstergaard M., Dammann B., Kazantsev I.G., High Resolution Orientation Distribution Function // Materials Science Forum, 2012, Vols. 702-703, pp. 536-539.
14. Barber B. C. Theory of digital imaging from orbital synthetic aperture radar // IJRS, **11**, 1983.
15. Elachi C., Bicknell T., Jordan R. L., Chialin Wu. Spaceborne synthetic-aperture imaging radars: Applications, techniques, and technology // Proceedings of the IEEE 1982, Vol 70, Issue: 10 pp.1174 - 1209.
16. Kretzek E., Ruiter N.V. GPU based 3D SAFT reconstruction including phase aberration // Proc. SPIE 9040, Medical Imaging 2014: Ultrasonic Imaging and Tomography, 90400W (March 20, 2014); doi:10.1117/12.2042669
17. Tikhonov A.N., Goncharskii A.V., Matvienko A.N., Romanov S.Y., Shchetinin V.G., Chubarov I.N., Markachev S.I., Grishko M.I., Ksenofontov E.A. Problems in the digital reconstruction of synthetic-aperture radar images // Soviet Physics Doklady, 1992, Vol. 37, pp. 79.
18. Rubin G., Berger D., Sager E. GPU Acceleration of SAR/ISAR Imaging Algorithms // Antenna Measurement Techniques Association Symposium 2010 (AMTA 2010), Proceedings of a meeting held 10-15 October 2010, Atlanta, Georgia, USA. A10-0059. pp. 430-435.
19. Maddikonda S.S., Shanmugha Sundaram G.A. SAR image processing using GPU // Communications and Signal Processing (ICCSP), 2014 International Conference on. pp. 448 – 452. DOI: 10.1109/ICCSP.2014.6949881
20. Bakushinsky A.B., Goncharsky A.V. Ill-posed problems. Theory and applications. Dordrect: Kluwer, 1994.
21. Goncharskii A.V., Romanov S.Y. On a three-dimensional diagnostics problem in the wave Approximation // Computational Mathematics and Mathematical Physics 2000;40:1308–1311. DOI: 10.3103/S0278641910010012
22. Goncharskii A.V., Ovchinnikov S.L., Romanov S.Y.. On the one problem of wave diagnostics // Moscow University Computational Mathematics and Cybernetics 2010;34:1–7. DOI: 10.3103/S0278641910010012
23. Ovchinnikov S.L., Romanov S.Yu. Organization of parallel computations when solving the inverse problem of wave diagnostics // Vychisl. Metody Programm. 2008. Vol.9. N 2. pp.338-345. (in Russian).
24. Goncharskii A.V., Romanov S.Yu. Two Approaches to the Solution of Coefficient Inverse Problems for Wave Equations // Computational Mathematics and Mathematical Physics. 2012, Vol. 52, No. 2, pp. 245–251.

25. Glover G.H. Computerized time-of-flight ultrasonic tomography for breast examination //Ultrasound in Medicine & Biology Vol, 3, No. 2–3, 1977, pp. 117-127. doi:10.1016/0301-5629(77)90064-3
26. Toward real-time bent-ray breast ultrasound tomography using GPUs// Proceedings of SPIE - The international society for optical engineering 9040 february 2014. DOI: 10.1117/12.2043127
27. Natterer F. Possibilities and limitations of time domain wave equation imaging. Contemporary Mathematics 2011;559:151–162.
28. Beilina L. and Klibanov M.V. 2012 Approximate Global Convergence and Adaptivity for Coefficient Inverse Problems (New York: Springer).
29. Goncharsky A.V., Romanov S.Y. Supercomputer technologies in inverse problems of ultrasound tomography // Inverse Problems. 2013, Vol. 29, No. 7. 075004. DOI: 10.1088/0266-5611/29/7/075004.
30. Goncharsky A.V., Romanov S.Y. Inverse problems of ultrasound tomography in models with attenuation. Phys Med Biol. 2014;59:1979–2004. DOI: 10.1088/0031-9155/59/8/1979
31. Goncharsky A.V., Romanov S.Y., Seryozhnikov S.Y. Inverse problems of 3D ultrasonic tomography with complete and incomplete range data. Wave motion 2014;51:389–404. DOI: 10.1016/j.wavemoti.2013.10.001
32. Huang L., Quan Y. Sound-speed tomography using first-arrival transmission ultrasound for a ring array // Proc. SPIE Medical Imaging. 2007. 6513. doi: 10.1117/12.709647.
33. Roy O., Jovanovic I., Hormati A., Parhizkar R., Vetterli M. Sound Speed Estimation Using Wave-based Ultrasound Tomography:Theory and GPU Implementation // Proc. SPIE 7629, Medical Imaging 2010: Ultrasonic Imaging, Tomography, and Therapy. 2010. 76290J. doi:10.1117/12.844691.
34. Varadan V.V., Ma Y., Varadan V.K., Lakhtakia A. Scattering of waves by spheres and cylinders // in: Field representations and Introduction to Scattering. (North-Holland), Amsterdam. 1991. 211-324.
35. Lavarello R.J., Oelze M.L. Tomographic Reconstruction of Three-Dimensional Volumes Using the Distorted Born Iterative Method // IEEE Trans. Med. Imaging. 2009. 28. 1643-1653. DOI: 10.1109/TMI.2009.2026274.
36. Goncharsky A.V., Romanov S.Yu., and Seryozhnikov S.Yu. Problems of limited-data wave tomography // Vychisl. Metody Programm. 2014 (15), 274–285 (in Russian).
37. Goncharsky A.V., Romanov S.Y., Seryozhnikov S.Y. A computer simulation study of soft tissue characterization using low-frequency ultrasonic tomography // Ultrasonics, 2016. V. 67, pp. 136–150. DOI: 10.1016/j.ultras.2016.01.008.
38. Tikhonov A.N. The solution of ill-posed problems and the regularization method, DAN SSSR 151: 3 (1963), pp.501-504 (in Russian)
39. Tikhonov A.N., Goncharsky A.V., Stepanov V.V., Yagola A.G. Numerical Methods for the Solution of Ill-Posed Problems. Kluwer acad.publ. Dordrecht/Boston/London 1995. ISBN 0-7923-3583-X
40. Sadovnichy V., Tikhonravov A., Voevodin Vl., and Opanasenko V. ”Lomonosov”: Supercomputing at Moscow State University. In Contemporary High Performance Computing: From Petascale toward Exascale (Chapman & Hall/CRC Computational Science), pp.283-307, Boca Raton, USA, CRC Press, 2013.

DOI: 10.14529/jsfi160104

Server Level Liquid Cooling: Do Higher System Temperatures Improve Energy Efficiency?

Egor A. Druzhinin[1], *Alexey B. Shmelev*[1], *Alexander A. Moskovsky*[1], *Vladimir V. Mironov*[2], *Andrey Semin*[3]

Liquid cooling is now a mainstream approach to boost energy efficiency for high performance computing systems. Higher coolant temperature is usually considered to be an advantage, since it allows heat reuse/recuperation and simplifies datacenter infrastructure by eliminating the need of chiller machine. However, the use of hot coolant imposes high requirements for cooling equipment. A promising approach is to utilize coldplates with channel structure and liquid circulation for heat removal from semiconductor components. We have designed a coldplate with low heat-resistance that ensures effective cooling with only 20-30° temperature difference between the coolant and electronic parts of a server. Under the stress-test conditions the coolant temperature rose up to 65 °C while server operation remained. We also studied power efficiency (expressed in floating point operations per watt) dependence on the coolant temperature (19-65 °C) on the individual server level (based on Intel Grantley platform with dual Intel Xeon E5-2697 v3 processors). The power performance ratio shows moderate ($\approx$10%) efficiency drop from 19 to 65 °C due to increase of leakage current in chipset components and reduction of processor frequency which resulted into proportional reduction of DGEMM benchmark performance. It must be taken into account by datacenter designers, that the amount of recuperated energy from 65 °C should be at least $\approx$10% to justify the choice of high temperature coolant solution.

Keywords: hot liquid cooling, cold plates, energy efficiency.

Introduction

Current power usage levels of top supercomputers place a heavy burden on system owners and maintainers. For example, the power usage of the number one supercomputer (positon №1 in the rating, Tianhe-2, $R_{max} = 33.86 \cdot 10^{15}$ floating point operations per second or FLOPS) in November 2015 Top500 list [1] is 17.8 MW under HPL [2] benchmark. More than 40 machines on the Top 500 report power dissipation of more than 1 MW. Even a 78 kW, which is an estimate for power consumption for the machine with the same HPL result as 500 system on the Top500 rating, is a substantial amount that requires sophisticated power supply and cooling equipment in place. With these figures, the energy efficiency that can be measured in FLOPS per watt is now reffered to as a key characteristic of modern high performance computing (HPC) systems. While Koomey's law [3] gives >50% energy efficiency improvement every year with the new generation of hardware platform, its also important to address the infrastructure components, like cooling subsystem, that can dramatically affect overall system efficiency. Addressing infrastructure efficiency would be also critical for attaining exascale level of performance for the future generation supercomputers, especially given power limits available in existing computing centers (e.g. notorious 20 MW power limit for an exaFLOPS-level machine [4, 5]).

A traditional quantitate measure of how efficiently energy is utilized in datacenter is power usage effectiveness (PUE) developed by The Green Grid consortium [6]. PUE is the ratio of total amount of energy used by a computer data center facility to the energy delivered to computing equipment. The ideal PUE is 1.0 when all energy is consumed by computing equipment and the datacenter infrastructure (such as cooling) takes no energy. In a real datacenter, a lot of energy is used by infrastructure equipment and PUE value is higher than 1.0. One of the most important energy consumer in datacenter is the cooling

[1]ZAO "RSC Technologies", Moscow, Russia
[2]Lomonosov Moscow State University, Moscow, Russia
[3]Intel Deutschland GmbH, Munich, Germany

system: even in the optimized liquid cooled datacenters it could utilize as much as 30% of total energy supply [7]. PUE is significantly improved in modern datacenters with the advent of so-called free cooling [8]. They are specially designed for effective use of outdoor environment to remove the heat from the servers. However, free cooling is most effective if the datacenter is able to operate at high temperatures. Compactness is uniquely important for HPC design. To be efficient, the parallel processor components require as fast communication as possible. So, at the end, these components are to be placed as close to each other as possible to minimize the communication length. These rules out the use of free airflow cooling methods, sometimes applied in the datacenter design (e.g. in [9]). Nevertheless, supercomputers must be energy efficient due to the huge amount of electricity consumed by them. Additional savings are possible by reusing some of the heat, but they may require coolant temperatures to be even higher than a human could tolerate. Higher temperature coolant is usually considered to be a more energy efficient option due to the absence of chiller equipment, which reduces capital expenditure for system construction. The coldplate-based design enables the compact system setup, which is important for HPC. However, the semiconductors operating on higher temperatures may have higher leakage current resulting in degradation of energy efficiency. Tht interference of these two factors remains nontrivial, especially for the most recent hardware generation and multitude of semiconductors used in server design.

In the current work we have studied a traditional homogeneous server operating with hot liquid cooling setup. The liquid coolants have much better thermal properties than gases: the typical PUE values for water cooled datacenters utilizing free cooling could be less than 1.1. We have designed a coldplate with low thermal resistance: the temperature difference between the CPU top cover and liquid coolant is minimized. That design enabled us to explore server energy efficiency in a wide coolant temperature range.

The goal of this work is to measure the actual performance values of the practically important benchmark such as DGEMM for the real-world HPC solution that utilizes hot liquid cooling technology. There are many studies of hot liquid cooling systems; however, they cover mostly the heat reuse issue but not the efficiency of the computation. We have conducted a series of tests, which are described in the sections below. In the tests we studied not only the cooling efficiency (which can be considered among the best ones for high temperature coolant system), but also the impact of the coolant temperature on the server performance and the power consumption ratio.

1. Details of the experiment

The coolant temperature impact on performance and energy consumption has been studied on an individual server level. The server was thermally insulated from the environment to eliminate its impact on the experiment and to mimic a very high density packaging that is traditionally found within the blade server systems racks. The thoroughness of the isolation was examined with Fluke Ti32 thermal imager. The resulting experimental setup is presented on Figure 1.

1.1. Hardware specifications

A single RSC Tornado server was used to study the performance and energy efficiency of our liquid cooling solution. The server was based on the dual socket Intel ServerBoard S2600KPF with two Intel Xeon E5-2697 v3 (14 cores, 2.6 GHz, 145 W TDP) installed. A latest production BIOS was used. The server was also equipped with 64 GiB (8×8 GiB) of DDR4-2133 registered memory with ECC support and a single 120 GB solid state drive.

The coolant temperature was measured by thermocouples, installed on inlet and outlet of the server. The flow rate sensor was used to measure liquid flow rate through the system. The liquid-to-air heat exchanger was activated when the coolant temperature reached an experiment threshold; its efficiency was optimized to avoid the liquid temperature decline on its fan activation. The average rate of water flow in cooling system was 30 ml/s; the average difference between inlet and outlet water temperature was 3.5 °C.

1.2. Benchmark details

The DGEMM matrix multiplication benchmark was used to simulate stress-test conditions. Intel MKL (version 12.2.3) DGEMM implementation was used in this test. The dimensions of the matrices were selected to be 87936×192×87936 that fit in 58 GiB of memory and provides the stress level similar to the hot phase of the HPL benchmark.

The DGEMM kernel was running continuously for at least 1 hour for each of the different inlet temperatures of the coolant ranging from 19 to 70 °C. The performance information was collected for every DGEMM iteration during the benchmark. The power usage and temperatures of the CPU, memory as well as the whole system was monitored using out-of-band telemetry provided by Intel NodeManager. The state of liquid cooling system (temperature and inlet and outlet coolant flow rate) was also controlled during all benchmarks.

The data from the first 20-30 minutes of all benchmarks are omitted from to ensure stationary coolant temperature is reached and the temperature levels of all node components are stabilized.

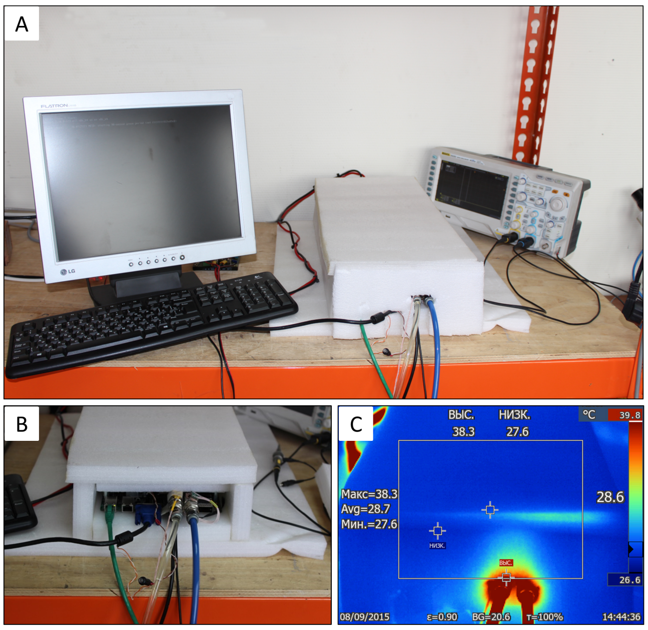


Figure 1. a) Overview of the experimental setup; b) liquid cooling and system connections; c) Fluke Ti32 view (T_{max} = 38.3 °C, T_{min} = 27.6 °C)

Table 1: Performance and power characteristics at different coolant temperatures

T_{LC} (°C)	Performance (GFLOPS)	System power (W)	Efficiency (GFLOPS/W)	Efficiency decrease (%)	Possible energy reuse[1](%)
19	974	359	2.72	–	-1.8
45	985	388	2.54	7.0	7.8
50	985	390	2.53	7.5	9.7
55	981	395	2.49	9.4	10.8
60	976	396	2.46	10.3	12.3
65	969	398	2.44	11.5	14.9

[1] assuming environment temperature T_0 = 25 °C

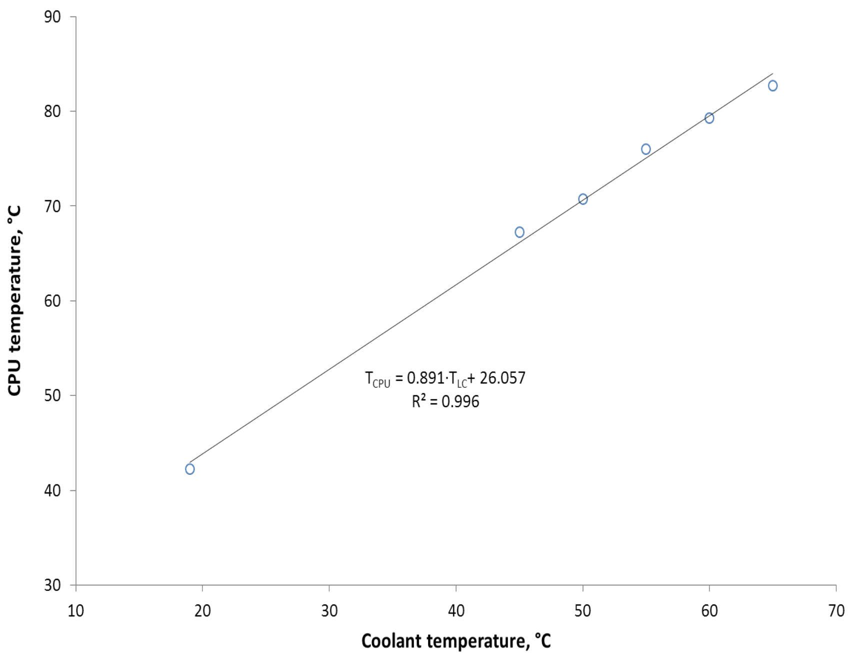


Figure 2. Dependence of the average measured CPU sensor temperature on coolant inlet temperature. T_{CPU} - CPU temperature, T_{LC} - coolant temperature

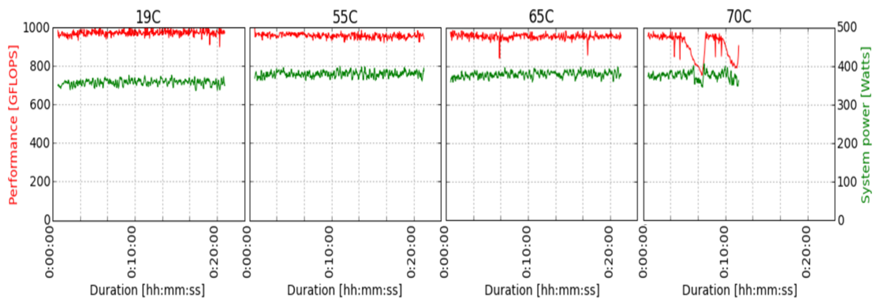


Figure 3. Illustration of system performance and power profiles at T_{LC} = 19, 55, 65 and 70 °C

2. Results

2.1. Liquid cooling performance

As one can see on the Figure 2, the liquid cooling solution that was used in this study has relatively low thermal resistance. The average difference between CPU (T_{CPU}) and liquid coolant (T_{LC}) temperatures is about 26 °C even at T_{LC} = 60-65 °C. The actual thermal resistance value was estimated to be 0.74 °C·in^2·W^{-1} that is on-par with commercially available coldplates (eq. Lytron plates [10]). We also monitored the data from the distributed temperature sensing (DTS) devices integrated in CPUs. These sensors show the difference between current and critical CPU temperature, at which it is considered to be overheated [11]. In this case DTS sensors return to zero. In current work we never get zero DTS value up to T_{LC} = 65 °C. However, at this coolant temperature the average DTS value corresponds to the temperature, which is only 5° below critical.

At the low temperature the performance is very stable for a long period of time and only at T_{LC} = 65 °C a very few and spiky performance drops are observed (Figure 3). Further growth of T_{LC} would result in scaling down of the CPU clock rate [12]. The performance of DGEMM is known to be linearly dependent on the CPU clock [13] and performance degradation would be observed on higher temperatures. Actually, our tests at T_{LC} = 70 °C show significant drop in performance. Almost half of the benchmark time the performance at 70 °C is 100-200 GFLOPS lower than the one at 65 °C. At the same time the power consumption changes only a little and a decrease of efficiency is also observed. For these reasons we dont refer to 70 °C results further.

2.2. Computational performance and energy efficiency

As it was discussed in previous section, the studied liquid cooling solution is very effective up to the 65 °C coolant temperature (Table 1). Indeed, upon this temperature we didnt observe any significant decrease of the average performance of the system (actual variations are ≈1-2%). At the same time, the average system power consumption is almost linearly dependent on temperature. These trends lead to efficiency decrease of ≈2.2% at every 10 °C growth of coolant temperature. Actually, the observed total change in efficiency is 11.5% upon growth of coolant temperature from 19 to 65 °C.

We also provided basic exergy analysis of possible hot water energy reuse. Exergy is the thermodynamic quantity that corresponds to the amount of energy which could be utilized for useful work. The following equation was used for exergy production rate (E_Q) calculation [14]:

$$\dot{E_Q} = \dot{m} C_p \left((T_{out} - T_{in}) - T_0 \ln \frac{T_{out}}{T_{in}} \right) \tag{1}$$

where $\dot{m}$ stands for water flow; C_p is heat capacity at constant pressure; T_0, T_{in}, T_{out} are environment, inlet and outlet coolant temperatures respectively. The estimates on energy reuse at T_0 = 25 °C are presented in Table 1. One could see that theoretical amount of energy, which could be reused would overcompensate the decrease of efficiency. However, the amount of reused energy would always be lower in real applications.

2.3. Power consumption of system components analysis

Modern Intel hardware allows fine-grain power profiling for many system components. In this study, we collected data on memory and CPU power consumption. As expected, the power consumption of CPU (Figure 4) and memory (21 W over all tests) modules depend only slightly (≈1%) on temperature. The tests were conducted with Turbo Boost feature enabled that adapts CPU performance to fit in the power limit. Moreover Haswell uses an improved version of Intel's 22-nm process where leakage current is relatively small [15].

The contribution to power consumption from other system components is much more sensitive to the temperature (Figure 5). The largest contribution to residual power consumption is made by the chipset components and voltage regulators. They don't have such power control and their power con-

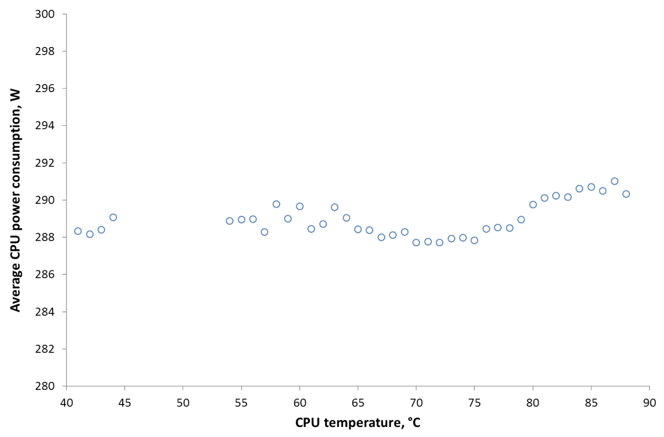


Figure 4. Dependence of CPU power consumption on its temperature. Gap at 45-55 °C is due to the lack of the data for this temperature interval

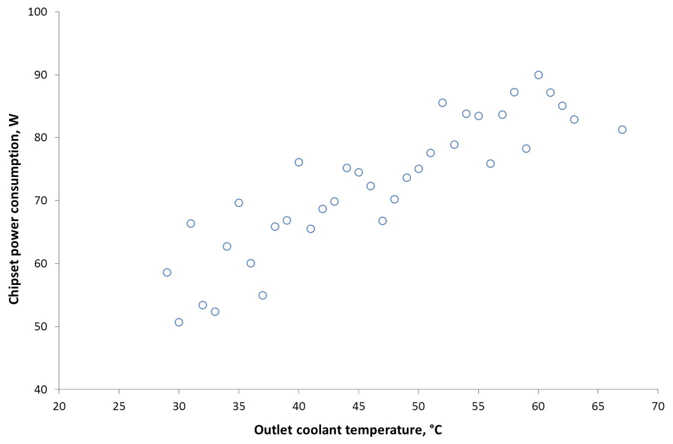


Figure 5. Dependence of the chipset and voltage regulators power consumption on temperature. The system temperature was estimated as the outlet coolant temperature. Its real value could be 10-20 °C higher

sumption grows with temperature due to increase of leakage current [16]. In our setup the residual power consumption almost doubles when the temperature rises from 30 to 65 °C.

3. Related work

An important problem of hot liquid cooling systems is the possibility of energy reuse. It should be stressed, that the gain from free cooling and energy recuperation/reuse should compensate computational efficiency decrease. Switching to the free cooling already provides notable energy efficiency growth. The problem of the heat reuse is much more complex. The most efficient way is to use hot water for heating [17, 18], however, it is very climate and country specific and not always possible. In many cases HPC datacenter designers have to think of possible energy conversion in useful work, however its mount is limited by thermodynamics second law. Energy recovery is most effective at very high coolant temperatures, but there are restrictions imposed by the hardware.

We observe a modest decrease (about 10%) of power efficiency when coolant temperature increases from 19 to 65 °C. The similar results have been obtained for other contemporary hot water cooling solutions, namely Aquasar (7% power consumption increase from 30 to 60 °C), CooLMUC (5% power consumption increase from 30 to 50 °C) and iDataCool (7% of efficiency decrease from 49 to 70 °C) [18–20]. In the study of Aquasar hot liquid cooling system [18] the exergy analysis showed about 10% of possible energy reuse at 50-60 °C water temperature. Our estimates also showed that the heat reuse could overcompensate the performance loss. Some additional gain is also possible due to free cooling. Thus the use of hot coolant is reasonable in datacenters and would result into reduction of their PUE value. One of such success stories of hot liquid cooling usage was presented by Eurotech [21]. They reported PUE value of 1.05 at 50 °C coolant temperature, however they didn't perform a detailed performance analysis that includes FLOPS/W metrics.

4. Conclusion

In this paper we explored the performance and power profiles of liquid cooling equipment with hot water, designed for RSC Tornado supercomputer architecture. This hardware exemplifies modern server platform with high temperature coolant option available due to appropriate design. Since the commodity semiconductor components are used (CPU, memory modules, server board), our results provide the insight on the modern hardware in general.

The power efficiency of our benchmark server is about 2.5 Gflops/W that is comparable to top 40 devices of current (November 2015) Green500 list [22]. Taking into account the possible energy recuperation will further increase efficiency of HPC system. We also demonstrated that the power efficiency decrease upon the coolant temperature growth should be compensated by possible heat reuse. A number of options for such reuse exist, starting from facility and building heating to adsorption by chiller machines. Climate-wise, 40-60 °C coolant temperatures enable free cooling on most of the Earth 24×7, except desert areas.

We are looking for the future to study behavior of the larger cluster with high temperature coolant, which is expected to be available in 2016, with respect to efficiency, reliability and ease of maintenance.

References

1. TOP500 supercomputer sites. [Online]. Available: http://www.top500.org
2. A. Petitet, R. C. Whaley, J. Dongarra, and A. Cleary. (2008) HPL - a portable implementation of the high-performance Linpack benchmark for distributed-memory computers. [Online]. Available: http://www.netlib.org/benchmark/hpl/
3. J. Koomey, S. Berard, M. Sanchez, and H. Wong, "Implications of historical trends in the electrical efficiency of computing," *IEEE Annals of the History of Computing*, vol. 33, no. 3, pp. 46–54, 2011.
4. P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller, M. Richards, and A. Snavely, "ExaScale computing study: technology challenges in achieving exascale systems," *Government PROcurement*, vol. TR-2008-13, p. 278, 2008.
5. D. Wade, "ASC business plan (NA-ASC-104R-15-Vol.1-Rev.0)," Tech. Rep., 2015.
6. J. Haas, J. Froedge, J. Pflueger, and D. Azevedo, *Usage and public reporting guidelines for The Green Grid's infrastructure metrics (PUE/DCiE)*, The Green Grid, 2009. [Online]. Available: http://

www.thegreengrid.org/~/media/WhitePapers/WhitePaper22PUEDCiEUsageGuidelinesfinalv21.pdf

7. M. P. David, M. Iyengar, P. Parida, R. Simons, M. Schultz, M. Gaynes, R. Schmidt, and T. Chainer, "Experimental characterization of an energy efficient chiller-less data center test facility with warm water cooled servers," in *2012 28th Annual IEEE Semiconductor Thermal Measurement and Management Symposium (SEMI-THERM)*. IEEE, 2012, pp. 232–237.
8. C. Gough, I. Steiner, and W. A. Saunders, *Energy efficient servers: blueprints for data center optimization*, 1st ed. Apress, 2015.
9. Yahoo launches second 'computing coop' data center in New York state. [Online]. Available: http://www.datacenterknowledge.com/archives/2015/04/27/second-yahoo-data-center-comes-online-in-new-york-state/
10. The performance of standard cold plate technologies is compared in a graph showing local thermal resistance. Lytron Inc. [Online]. Available: http://www.lytron.com/Cold-Plates/Standard/Performance-Comparison
11. M. Berktold and T. Tian, *CPU monitoring with DTS/PECI*, 2010. [Online]. Available: http://www.intel.com/content/www/us/en/embedded/testing-and-validation/cpu-monitoring-dts-peci-paper.html
12. *Intel Turbo Boost technology 2.0*, Intel. [Online]. Available: http://www.intel.com/technology/turboboost/
13. J. Demmel and A. Gearhart, "Instrumenting linear algebra energy consumption via on-chip energy counters," UC at Berkeley, Tech. Rep., 2012. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-168.html
14. G. Evola and L. Marletta, "Exergy and thermoeconomic optimization of a water-cooled glazed hybrid photovoltaic/thermal (PVT) collector," *Solar Energy*, vol. 107, pp. 12–25, 2014.
15. P. Hammarlund, A. J. Martinez, A. A. Bajwa, D. L. Hill, E. Hallnor, H. Jiang, M. Dixon, M. Derr, M. Hunsaker, R. Kumar, R. B. Osborne, R. Rajwar, R. Singhal, R. D'Sa, R. Chappell, S. Kaushik, S. Chennupaty, S. Jourdan, S. Gunther, T. Piazza, and T. Burton, "Haswell: the fourth-generation Intel Core processor," *IEEE Micro*, vol. 34, no. 2, pp. 6–20, 2014.
16. Nam Sung Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, Jie S. Hu, M. Irwin, M. Kandemir, and V. Narayanan, "Leakage current: Moore's law meets static power," *Computer*, vol. 36, no. 12, pp. 68–75, 2003.
17. S. Zimmermann, M. K. Tiwari, I. Meijer, S. Paredes, B. Michel, and D. Poulikakos, "Hot water cooled electronics: exergy analysis and waste heat reuse feasibility," *International Journal of Heat and Mass Transfer*, vol. 55, no. 23-24, pp. 6391–6399, 2012.
18. S. Zimmermann, I. Meijer, M. K. Tiwari, S. Paredes, B. Michel, and D. Poulikakos, "Aquasar: a hot water cooled data center with direct energy reuse," *Energy*, vol. 43, no. 1, pp. 237–245, 2012.
19. N. Meyer, M. Ries, S. Solbrig, and T. Wettig, "iDataCool: HPC with hot-water cooling and energy reuse," in *Supercomputing: 28th International Supercomputing Conference*, 2013, pp. 383–394.
20. A. Auweter and H. Huber, "Direct warm water cooled Linux cluster Munich: CooLMUC," *Inside*, vol. 10, no. 1, pp. 81–82, 2012.
21. Hot water cooled supercomputer. Eurotech. [Online]. Available: http://www.eurotech.com/en/hpc/hpc+solutions/liquid+cooling

22. The Green500 list. [Online]. Available: http://www.green500.org

DOI: 10.14529/jsfi160105

Data Compression for Climate Data

Michael Kuhn[1], *Julian Kunkel*[2], *Thomas Ludwig*[2]

The different rates of increase for computational power and storage capabilities of supercomputers turn data storage into a technical and economical problem. As storage capabilities are lagging behind, investments and operational costs for storage systems have increased to keep up with the supercomputers' I/O requirements. One promising approach is to reduce the amount of data that is stored. In this paper, we take a look at the impact of compression on performance and costs of high performance systems. To this end, we analyze the applicability of compression on all layers of the I/O stack, that is, main memory, network and storage. Based on the Mistral system of the German Climate Computing Center (Deutsches Klimarechenzentrum, DKRZ), we illustrate potential performance improvements and cost savings. Making use of compression on a large scale can decrease investments and operational costs by 50 % without negative impact on the performance. Additionally, we present ongoing work for supporting enhanced adaptive compression in the parallel distributed file system Lustre and application-specific compression.

Keywords: data compression, storage system, climate data, cost efficiency.

Introduction

Throughout the history of supercomputers as recorded by the TOP500 list, the computational power has been increasing exponentially, doubling roughly every 14.5 months [36]. While this increase in computational power has allowed more detailed numerical simulations to be performed, this has also caused the simulation results to grow in size exponentially. Computational speed and storage capacity have roughly increased by factors of 300 and 100 every 10 years, respectively. The storage speed, however, has only grown by a factor of 20 every 10 years, even when taking newer technologies such as SSDs into account. The different rates of improvement can be seen in Figure 1.[3] Thus, the importance of performing I/O efficiently and storing the resulting data cost-efficiently increases [23].

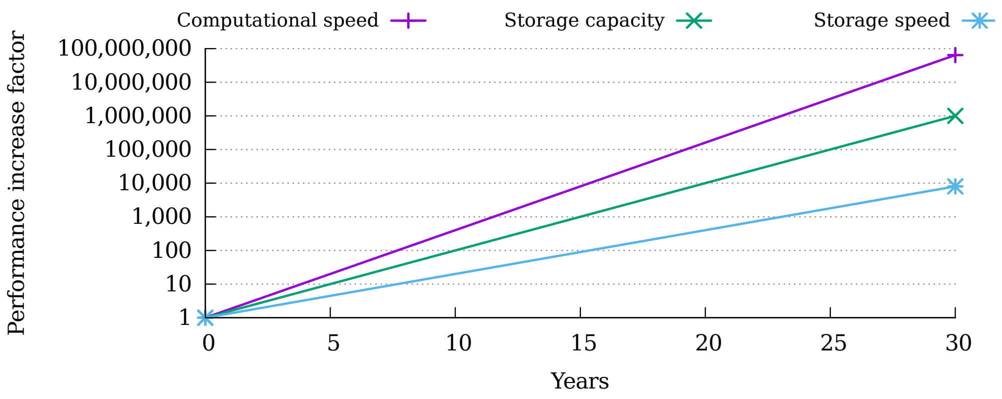


Figure 1. Development of computational speed, storage capacity and storage speed

Although it is theoretically possible to compensate for this fact in the short term by simply buying more storage hardware, new approaches are required to use the storage infrastructure as efficiently as possible due to the ever increasing gap between the faster-growing processing

[1]Universität Hamburg, Hamburg, Germany
[2]Deutsches Klimarechenzentrum GmbH, Hamburg, Germany
[3]It has to be noted that the computational speed is based on the TOP500 list while the storage capacity and speed are based on single devices.

power on the one hand and the lagging storage capacity and throughput on the other hand. Data reduction techniques present one such approach and play an important role in today's high performance computing (HPC) systems. Data reduction can be used to reduce the costs and size of storage systems, and to increase performance.

Overall, the storage subsystems can be responsible for a significant portion of a system's total cost of ownership. In the case of Mistral, it accounts for roughly 20 % of the overall costs. This amount of storage is necessary due to the huge amounts of data. In phase 5 of the Coupled Model Intercomparison Project (CMIP), a volume of 1.8 PB of data and 4.3 million files were created [1]. An estimate of the Earth System Grid Federation (ESGF)[4] for the upcoming phase 6 project is between 36 PB and 90 PB [9].

In an effort to keep storage costs from increasing, data reduction techniques are increasingly being deployed. Previous studies have shown that certain data reduction techniques can be beneficial for large-scale storage systems [25]. However, due to their inherent costs and complexities, techniques such as deduplication and re-computation are not suitable without restrictions. In particular, deduplication typically yields lower data reduction rates and may degrade performance, thus making it unsuitable for HPC [29, 30]. Compression can be deployed with relatively little overhead and the integration into existing systems is much easier.

Being such a versatile concept, compression can be introduced with different goals on various hardware and software levels. In this paper, we will investigate the possibilities of applying compression to various levels of the HPC hardware/software stack. In this regard, we will analyze typical storage behavior from a datacenter perspective and not focus on particular use cases or data formats. Specifically, the following goals will be addressed:

1. Memory capacity. By compressing data in memory, additional memory becomes usable, and out-of-core computing can be avoided. Usually, this can be done by providing two pools of memory: one compressed pool for slower access and a regular pool, in which the working set fits. Each time data from the compressed memory pool is required, it must be decompressed and either migrated into the regular pool or directly into the CPU caches.
2. Network throughput. By compressing data before transmitting it via the network, communication performance can be increased. This typically requires fast compression algorithms, but not necessarily compression speeds higher than the throughput of the network. In case of storage systems, data does not have to be decompressed on the receiving side, leading to further benefits.
3. I/O throughput. Compressing data before writing it to the storage hardware can increase overall I/O performance because less data has to be written. Conversely, less data has to be read from the storage hardware, also increasing performance for the read case.
4. Storage capacity. In addition to the already mentioned performance benefits, compressing data reduces its storage footprint, allowing more data to be stored with the same storage hardware.
5. Cost and energy efficiency. Compression can be used to improve the cost efficiency of data storage. Additionally, if data has to be archived for long periods of time, it can be worthwhile compressing it using slower algorithms yielding higher compression ratios.[5]

[4]`http://esgf.llnl.gov/`

[5]We explicitly define compression ratio as the fraction of uncompressed size over compressed size, that is, compression ratio $= \frac{\text{uncompressed size}}{\text{compressed size}}$. Due to its convenience, the inverse compression ratio, that is, 1 divided by the compression ratio, will also be used at some points in the paper; it indicates the fraction to which data can be compressed.

A proper integration of the mentioned approaches also allows avoiding decompression at different levels of the stack, further increasing its effectiveness. Compression offers the chance of improving the cost and energy efficiency of existing hardware, and also allows improving performance. With respect to the above list, the main contributions of this paper are:

1. A thorough analysis of existing compression technologies regarding their applicability in high performance computing and the I/O stack.
2. Models to estimate the impact of compression on performance and cost of supercomputers and their storage systems.

This paper is structured as follows: Section 1 contains state-of-the-art and related work regarding the compression and its applicability in HPC contexts. In Section 2, we model the impact of different compression techniques on performance and cost. We then describe ongoing work for applying different types of compression in a parallel distributed file system and applications in Section 3. Section 4 concludes the paper and discusses future work.

1. State-of-the-art and related work

A multitude of different compression algorithms and possibilities for applying compression exist. In this section, we will describe approaches for addressing the goals in more details and quantify their respective advantages and disadvantages.

1.1. Compression algorithms

There is a wide range of compression algorithms. General *lossless* compression algorithms, such as DEFLATE [10] and LZO [20], consider data to be an array of characters. Therewith, the algorithm has to ensure that any sequence of characters can be compressed and reconstructed. Usually, applications store more complex data types and compound structures. Knowing the data structures would allow to reduce the required information in the compressed representation as impossible configurations of data can be ignored. For instance, several lossless compression algorithms specifically tailored to floating-point data are available [27, 33].

To improve performance, several approaches have been investigated. LZRW [39] is a variant of the LZ77 [42] compression algorithm tuned for speed. The lz4fast algorithm is a new variant of the lz4 compression algorithm that allows users to specify an *acceleration factor* to improve compression speed by sacrificing compressibility [41]. Preliminary tests have shown that lz4fast can reach up to several GB/s per core both for compression and decompression, making it especially interesting for HPC. Because throughput requirements are increasing, algorithms have also been ported to accelerators [32] and have been implemented in hardware [2, 4, 8] to accelerate compression speed. With SLDC [12], the tape technology LTO offers hardware-accelerated compression performed by the tape drive. One drawback of this approach is that it is not possible to know in advance how much capacity a tape drive actually has.

If the application knows the tolerable errors in the representation, *lossy* compression schemes can reduce data further to the level that is needed to reconstruct the data with the required precision. However, such an application-specific schema requires insight from the domain. ISABELA [26] and ZFP [28] are compression schemes for floating-point data. ISABELA is a preconditioner that reorders data to improve the efficiency of the later compression stages. ZFP supports lossy compression and allows to define either the relative error tolerance or absolute error tolerance. Multidimensional variables usually exhibit some locality and this smoothness of data even increases with the resolution of the simulation domain. Such regularities on grids can be exploited by preconditioning the input data, for example, as done by ZFP and our own approach MAFISC [18]. A compression scheme may even be tailored for a particular grid structure. Wang et al. exploit the topology of icosahedral grids that are now common in atmospheric models by mapping each rombus to a 2D matrix that can be compressed with wavelet schemes [37].

1.2. Memory capacity

Since the early days of personal computers, in-memory compression has been used to virtually enlarge memory capacity. For example, in the early days of Intel's 386 and MS-DOS, the Quarterdeck Expanded Memory Manager (QEMM) has been widely used. For modern systems, Linux's *zram* framework provides a simple way to create compressed block devices [31]. To increase the main memory capacity, these block devices can then be used as swap devices. For instance, given a machine with 8 GiB of physical main memory, an additional 8 GiB zram block device can be created. Assuming a compression ratio of 2.0, the zram block device would require 4 GiB of physical main memory, providing a total of 12 GiB of main memory. Obviously, the final amount of available main memory heavily depends on the data written to the compressed block device.

The impact on performance of this solution can be hard to predict because the data is not compressed and decompressed on every access, but only if Linux decides that a page should be swapped in or out. That is, live data will typically stay uncompressed and only be compressed when the page containing it is swapped out. Whether a page should be swapped out is determined by the kernel's `swappiness` parameter.

Listing 1 shows the steps of setting up a compressed zram block device and using it as a swap device. First, the `zram` kernel module has to be loaded to be able to use the zram framework. Afterwards, the `zramctl` command line tool can be used to set up compressed block devices. In this case, a block device with 8 GiB of space is set up using the first free identifier. It is also possible to change the used compression algorithm and the number of compression streams to increase performance. By default, the `lzo` algorithm and only one compression stream are used. Given sufficient parallelism from the application side, increasing the number of compression streams can be used to effectively scale the compression throughput.

```
$ modprobe zram
$ zramctl --find --size 8G
/dev/zram0
$ mkswap /dev/zram0
$ swapon /dev/zram0
```

Listing 1. Setting up a compressed block device with a size of 8 GiB and using it as swap space

1.3. Network throughput

There are several approaches for message compression in the Message Passing Interface (MPI), all of them virtually increasing network throughput. CoMPI allows compressing MPI messages by adding a compression layer to the ADI of MPICH [16]. While beneficial in many cases, the evaluation of HPC applications is conducted using Fast Ethernet and, thus, the applicability to recent HPC systems is limited. Adaptive-CoMPI improves this approach by selecting the compression algorithm at runtime and allowing developers to specify guiding information [17].

Since the compression ratio depends on data properties such as the inherent redundancy, the evaluation of this approach shows that it is able to improve throughput for some of the tested scientific applications and may only degrade performance slightly. The PRAcTICaL-MPI wrapper transfers the strategy to a library that utilizes the MPI standard profiling interface (PMPI) [14]. Due to the portability of PMPI, it can be deployed without changing code or MPI implementation.

In [32], it has been discussed whether it is worthwhile to compress CPU/GPU data transfers over the PCI-Express bus. However, due to the speed of PCIe 3.0 (x16) in the order of 16 GB/s, this idea has been discarded.

1.4. Storage capacity and throughput

Welton et al. compress network traffic between clients and the I/O forwarding layer to improve performance [38]. Their evaluation is conducted on an Ethernet and a QDR InfiniBand connected cluster. While on Ethernet, LZO increases performance significantly and even does not degrade performance for random workloads, the performance of IB networks is degraded when applying compression. The Autonomous and Parallel Compressed File System [22] is a FUSE file system stacked on top of a local file system that transparently compresses data passed through it. It uses LZ77 and adaptive Huffman encoding. The authors aim to select adaptively the algorithm based on the data type (for example, text and movies).

In [15], Filgueira et al. apply their adaptive compression scheme of CoMPI to I/O. They implement the I/O compression into the Papio parallel storage system that offers quality of service (QoS). Writes are performed in atoms of `stream_width`, that is, the number of strips times the `stripe_size`. Their approach provides heuristics to estimate compression rate and network performance, and, thus, the achieved speedup. Based on the estimate and selected QoS, the appropriate algorithm is selected. Compression of the data can then be performed by multiple threads. The evaluation is conducted on a 10 Gbit Ethernet system but QoS limits throughput to 300 MB/s. It is demonstrated that the heuristics approximate the speedup of the compression schemes well. The adaptive compression improves speedup in all cases but degrades with increasing throughput. For reads, the achieved speedup is about $^3/_4$ of the compression speed.

1.4.1. File systems

Currently, only a hand full of file systems support compression natively. All of them are local file systems, the most important ones being btrfs [21], NTFS and ZFS [5]. Due to NTFS only being available on Windows, however, it typically does not play a role in HPC.

btrfs supports compression on a per-extent level with a maximum size of 128 KiB. It supports the compression algorithms LZO and zlib, which can be selected at mount time by specifying the mount options `compress` or `compress-force`; they will either skip compression of incompressible files or force compression of all files, respectively. Compression can also be enabled and disabled for individual files and directories using the `c` attribute of `chattr`. ZFS supports compression on a per-record level with a typical maximum size of 128 KiB; newer versions of ZFS allow recording sizes of up to 1 MiB, which can increase the efficiency of compression algorithms. ZFS has support for zero-length encoding (zle), gzip (with levels 1–9), lzjb and lz4. Compression can be set individually for each file system using the `zfs set compression` command.

1.5. Cost and energy efficiency

Cost and energy efficiency are important topics in high performance computing, as demonstrated by initiatives, such as the Green500 list [35]. However, the impact of compression on these two metrics has so far mainly been analyzed in the context of embedded and mobile devices, where data transmission is especially expensive [3, 11, 34, 40]. Studies have shown that compression can increase energy consumption instead of decreasing it. This is due to the fact that algorithms might spend a disproportional amount of energy for compressing the data. That is, it is also important to look at how much energy is consumed to compress a given amount of data (MB/J) instead of only compression throughput (MB/s). While there have been studies on the energy and performance impact of compression for data-intensive workloads such as MapReduce [7] and for file systems in general [24], the cost aspect of the overall system has often been neglected. Our own studies have shown that efficient compression algorithms can be beneficial both in terms of performance and energy consumption even for HPC applications [6].

2. Modeling the impact of compression

An important factor when considering compression is its impact on performance. Depending on the used compression algorithm, its settings and where compression is applied, it can be either beneficial or detrimental for performance. The overall impact of introducing compression into scientific workflows depends on the strategy, that is, which data to compress and when to compress it, the characteristics of the data, the characteristics of the compression algorithm and the hardware characteristics. In detail, the strategy determines the number of times data is compressed/decompressed in the workflow. Time needed to compress/decompress is mainly determined by the compression algorithm and the CPU but also influenced by the structure of the data. The achieved compression ratio is mainly determined by the algorithm but is also influenced by the compressibility of the data. The benefit of any strategy comes from virtually increasing throughput and storage capacity in hardware components. Since characteristics of compression algorithms vary, the best fitting algorithm for a given scenario can be chosen or adaptively determined.

2.1. Performance considerations

For the following discussion, we assume an application consists of the three phases: compute, communication and I/O, as shown in Figure 2. The overall runtime of the application is $t = t_{cpu} + t_{net} + t_{io}$. Note that if multiple phases of the same type are observable, we just accumulate the time for each phase.

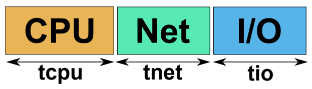


Figure 2. Phases of an HPC application

It could be an iterative algorithm – such as shown in Figure 3 – that uses asynchronous communication and I/O to hide the time needed for communication and I/O. The exact composition of the application does not matter, the total time spent in computation, communication and I/O are labeled as t_{cpu}, t_{net} and t_{io}, respectively, since they utilize these hardware components.

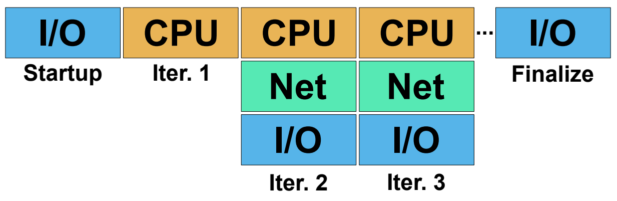


Figure 3. Phases of an HPC application when using asynchronous communication and I/O

When introducing a compression strategy to communication and I/O on the compute nodes, this requires additional time to compress (t_c) when sending/writing data and to decompress (t_d) when receiving/reading data. Similarly, when compressing data structures in memory, the computation pattern changes to include phases to compress/decompress from the compressed memory pool to the CPU cache (not shown in the figures).

Let us first discuss the benefit of hardware-accelerated compression performed by the network interface, this would allow to compress communication but also data on the I/O path without overhead for the computation while still allowing RDMA. Moreover, if the I/O server can disable decompression while receiving data blocks to be stored, that data can be stored in its compressed form without additional costs (similarly for the read path). The application runtime using this strategy becomes $\hat{t} = t_{cpu} + \hat{t}_{net} + \hat{t}_{io}$, where $\hat{t}$ includes the time for the compression and decompression. For larger amounts of data, the network interface can overlap compression with sending/receiving of data. Thus, it can hide the time needed for compression as long as the performance of the compression algorithm is, at least, as fast as the network throughput. $\hat{t}_{net} = \frac{t_{net}}{cr_{net}}$ and $\hat{t}_{io} = \frac{t_{io}}{cr_{io}}$, where cr_p is the compression ratio of phase p. In this case, this leads to a total time $\hat{t}$, as shown in Equation (1).

$$\hat{t} = t_{cpu} + \frac{t_{net}}{cr_{net}} + \frac{t_{io}}{cr_{io}} \tag{1}$$

As hardware-accelerated compression/decompression is typically not available using current hardware, it is important to also model the unaccelerated case. If the network interface does not support compression and data is uncompressed in memory, then we have to explicitly compress/decompress data, increasing the required CPU time t_{cpu} by the time required for compression (t_c) and decompression (t_d). This leads to an overall time $\hat{t}^+$, as shown in Equation (2), where s_p denotes the data size of phase p and p_p denotes the performance of phase p.

$$\begin{aligned} \hat{t}^+ &= t_{cpu} + \frac{t_{net}}{cr_{net}} + \frac{t_{io}}{cr_{io}} + t_c + t_d \\ &= t_{cpu} + \frac{s_{net}}{p_{net} \cdot cr_{net}} + \frac{s_{io}}{p_{io} \cdot cr_{io}} + \frac{s_c}{p_c} + \frac{s_d}{p_d} \end{aligned} \tag{2}$$

In a perfectly balanced system, $p_{io} = p_{net}$. Performing this process only pays off, if the time saved through compression ($t - t_{compressed}$) is larger than the time it takes to perform it ($t_c + t_d$). This leads us to the estimation in Equation (3), where cr^{-1} is the inverse compression ratio.

$$\begin{aligned}
t_c + t_d &< t - t_{compressed} & \Leftrightarrow \\
\frac{s}{p_c} + \frac{s}{p_d} &< \frac{s}{p} - \frac{s \cdot cr^{-1}}{p} & \Leftrightarrow \\
\frac{p_c + p_d}{p_c \cdot p_d} &< \frac{1}{p} \cdot (1 - cr^{-1}) & \Leftrightarrow \\
p &< \frac{p_c \cdot p_d}{p_c + p_d} \cdot (1 - cr^{-1}) & (3)
\end{aligned}$$

If we can pipeline compression and decompression on sender and receiver side, this equation becomes $p < \min(p_c, p_d) \cdot (1 - cr^{-1})$. Using this equation, we can generate thresholds for when compression pays off; the thresholds for three algorithms as measured on Mistral are shown in Table 1. Actually, the network interface bandwidth is shared among all cores, thus, the value determined can be multiplied with the number of cores.

Table 1. Network throughput thresholds for compression to pay off without hardware acceleration (network throughput has to be below given value)

Algorithm	**CPU**	**Pipelined**	**CPU (24 cores)**	**Pipelined (24 cores)**
pithy	791 MB/s	1,170 MB/s	19,000 MB/s	28,000 MB/s
blosc	389 MB/s	442 MB/s	9,300 MB/s	10,600 MB/s
lz4fast	914 MB/s	1,330 MB/s	21,900 MB/s	31,900 MB/s

The performance benefit or drawback, when compressing network communication, can be visualized in a 2D graph with the number of cores and the compression ratio as axes. Figure 4 illustrates the speedup for Mistral's FDR InfiniBand and using the lz4fast compression algorithm without hardware support. On our system, lz4fast achieves a throughput of roughly 2,900 MB/s for compression and 6,500 MB/s for decompression.[6] It can be observed that using 17 cores and an inverse compression ratio of 0.5, a speedup of 1.5 is possible. This almost matches the speedup when upgrading from FDR to EDR InfiniBand (which is faster by a factor of 1.77) and can thus be a viable approach to improve throughput without increasing costs for network equipment.

2.1.1. Parallel distributed file systems

Considering parallel distributed file systems, compression can either be applied on the clients or on the servers. Both approaches have benefits and limitations. The maximum throughput (p) in a parallel distributed file system is limited by several factors. The most important ones are client (p_c), network (p_n) and server (p_s) throughput, as shown in Equation (4).

$$p = \min(p_c, p_n, p_s) \tag{4}$$

p_n is static and independent on any potential data reduction taking place. p_c and p_s, however, depend on memory throughput and are therefore heavily dependent on which compression algo-

[6] A random selection of files on the storage system has been chosen for analysis (7,335 files with a volume of 300 GiB). The files contain a representative sample of scientific data formats and text files. They are compressed running independent compression/decompression jobs on 24 cores of a compute node concurrently. Throughout the paper, all specified values are the arithmetic mean across these files.

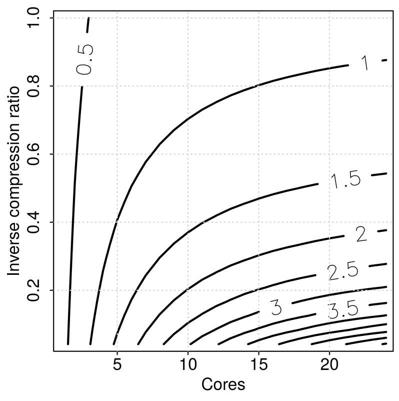
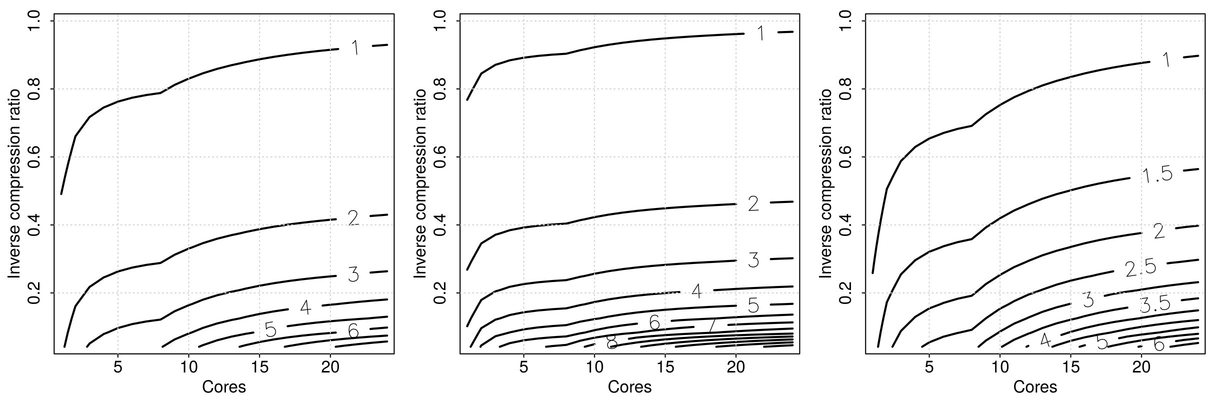


Figure 4. Performance multiplier when compressing communication on Mistral's FDR InfiniBand

rithm and settings are used. In real systems, slow compression algorithms can be counterbalanced to a certain extent by using write-behind caching, which allows applications to progress immediately and compression to take place in the background. For the sake of simplicity, we will assume that all data has to be written immediately and performance is thus limited by the compression algorithm's throughput. Slow compression algorithms can therefore slow down overall I/O speed significantly; their main benefits lie in further reducing the amount of data, saving costs. When applying compression in a file system, there are several possible approaches:

1. When writing data, it is compressed on the client, sent to the server and stored there in its compressed form. When reading the data again, the server returns the compressed data, which is then decompressed on the client. This approach might have disadvantages when performing small accesses because complete blocks of data have to be compressed and decompressed, causing additional overhead. Further analyses regarding the actual impact are necessary as HPC applications typically perform large accesses.
2. When writing data, it is compressed on the client and sent to the server. The server then decompresses the data and stores it in its decompressed form. When reading the data again, the server compresses the data, which is then decompressed on the client. This approach causes additional overhead on the server but does not suffer from the small access problem mentioned above because data can be accessed at finer granularity.



a) Write (compress) b) Read (decompress) c) Write with server decompress

Figure 5. Performance multiplier when (de)compressing data in the Lustre client

Figures 5a to 5c show the visualization of possible performance that increases for Mistral's Lustre file system. While Figures 5a and 5b contain the performance multipliers for writing and reading the data using the first approach, respectively, Figure 5c shows the performance when writing the data using the second approach. As can be seen, it is easier to improve performance when decompressing data on read than when compressing it on write due to the much higher decompression speeds of lz4fast. Additionally, compressing the data on the client and decompressing it on the server before actually writing it incurs additional overhead that requires more cores to be worth it. Overall, while compression requires a significant amount of cores to improve performance, it is also unlikely to degrade performance. Decompression, however, can achieve higher performance with even a small amount of cores, showing the usefulness of this approach.

2.2. Cost considerations

Besides its impact on performance, compression can also be an important factor in reducing costs. While this mainly applies to the reduction of the data's footprint, it can also have secondary benefits such as being able to spend less money on main memory or network infrastructure. Additionally, increase in throughput typically decrease costs indirectly because execution times are reduced. There are two approaches when utilizing compression:

1. Using compression to reach the desired metric, such as storage capacity, main memory capacity or network throughput. This allows spending less money for the respective component and using the remaining money for different purposes.
2. Using compression to improve the desired metric. This involves spending the same amount of money for the respective component while gaining more performance and/or capacity.

The following considerations will take both approaches into account. Depending on the examined hardware component, one or the other makes more sense. All cost considerations will be inspired by DKRZ's current Mistral supercomputer.

Table 2. Compression throughput and ratio for selected compression algorithms as measured using real data

Algorithm	Compression	Decompression	Ratio
lz4fast	2,945 MB/s	6,460 MB/s	1.825
lz4	1,796 MB/s	5,178 MB/s	1.923
lz4hc	258 MB/s	4,333 MB/s	2.0
lzo	380 MB/s	1,938 MB/s	1.887
xz	26 MB/s	97 MB/s	2.632
zlib	95 MB/s	610 MB/s	2.326
zstd	658 MB/s	2,019 MB/s	2.326

Table 2 shows the compression throughput and ratio for selected compression algorithms as measured on a set of real data on Mistral's storage system (for details, see Footnote 6). As can be seen, the algorithms have vastly different compression throughputs and ratios. Some algorithms such as zstd outperform other algorithms such as lzo for both throughput and ratio, but usually algorithms with lower throughputs also achieve higher compression ratios.

2.2.1. Main memory

Typically, the amount of main memory in a supercomputer depends on the applications that will be executed on it. Therefore, we will consider the goal of reaching a given amount of the main memory per node and use compression to reduce the amount of necessary hardware. Mistral has a total main memory capacity of roughly 320 TB. Assuming costs of €200 per 32 GB of the main memory, the total costs for main memory only are €2,000,000. Additionally, we will assume that each node is equipped with 128 GB of the main memory, resulting in 2,500 nodes.[7]

Since we are interested in reaching a per-node main memory capacity of 128 GB, we can calculate the amount of necessary physical main memory $mem = \frac{mem_c}{ratio}$, where mem is the amount of physical main memory, mem_c is the amount of the main memory after compression and $ratio$ is the average compression ratio. As mentioned previously, zram supports the lz4 and lzo compression algorithms. For lz4 this would be $\frac{128}{1.923} = 66.56\,\text{GB}$ and for lzo $\frac{128}{1.887} = 67.83\,\text{GB}$. The main memory capacity of nodes can not be chosen freely and some main memory should be reserved for keeping uncompressed data. Therefore, it would make sense to either use a configuration with 96 GB of the main memory or to use a 64 GB configuration that relaxes the 128 GB requirement slightly. This configuration could reserve 4 GB for uncompressed data and provide $60\,\text{GB} \cdot 1.923 = 115.38\,\text{GB}$ or $60\,\text{GB} \cdot 1.887 = 113.2\,\text{GB}$ of compressed main memory for lz4 or lzo, respectively. One important factor that has to be kept in mind is the main memory throughput. While lz4 is typically fast enough to saturate the memory bus if enough parallel compression streams are used, lzo is much slower. Even with 24 parallel streams, lzo can only achieve $24 \cdot 380\,\text{MB/s} = 9.12\,\text{GB/s}$.

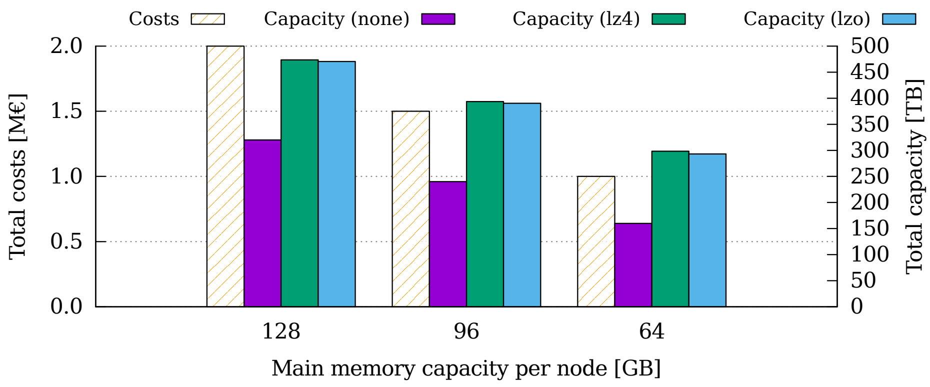


Figure 6. Total costs and capacity in relation to the amount of main memory per node

Figure 6 shows three node configurations with 128, 96 and 64 GB of the main memory. It contains the total costs required to procure the main memory and total capacities with no compression, lz4 and lzo. For the configurations with 128 and 96 GB, the main memory is split into an uncompressed and a compressed pool dynamically; only the necessary amount of the main memory to reach a per-node capacity of at least 128 GB is compressed, the remaining main memory is left uncompressed. For example, $\frac{128\,\text{GB}}{1.923} = 66.56\,\text{GB}$ are compressed for lz4, the remaining part is left uncompressed. For the 64 GB configuration, 60 GB are compressed while 4 GB are left uncompressed, resulting in less than 128 GB capacity per node. As it can be seen, the main memory's cost efficiency – that is, the amount of memory per € – increases as more main memory is compressed. Reducing the amount of the main memory is worthwhile because it is responsible for roughly 10 % of the overall power consumption according to [23, page

[7]Mistral has roughly 3,000 nodes and features different main memory configurations.

165]. However, as mentioned previously, zram's impact on performance can be hard to predict. Therefore, keeping a pool of uncompressed main memory for fast access makes sense. We will investigate this approach and its impact on performance in more detail in the future.

2.2.2. Network

Compression can also be used to virtually increase network throughput. In the following, we will investigate whether this can be translated into cost savings. As with the main memory, the network can not be scaled arbitrarily but is limited to certain stages given by the used network technology. For high performance computing, InfiniBand is one of the most common network technologies and is available in several different speeds. The most widely used version is FDR, which offers a latency of $0.7\,\mu$s and a throughput of 54.54 Gbit/s when using four links. The newest version, EDR, offers a latency of $0.5\,\mu$s and a throughput of 96.97 Gbit/s using four links. However, the older version QDR and Ethernet networks are also interesting due to their potentially lower costs.

Mistral is using FDR InfiniBand. When using lz4fast and 24 cores, we can reach a maximum compression throughput of $24 \cdot 2,945$ MB/s $= 70,680$ MB/s, which is enough to saturate even the fastest network. We can saturate the FDR InfiniBand network using only three cores and achieve a throughput of 99.54 Gbit/s due to the compression ratio of 1.825. This is even faster than what EDR InfiniBand can offer. When looking at saving costs by choosing a slower interconnect, however, even QDR InfiniBand could be used with its 32 Gbit/s throughput to achieve the same performance as FDR InfiniBand. The throughput increases to 32 Gbit/s $\cdot$ 1.825 = 58.4 Gbit/s when applying compression with lz4fast in this case.

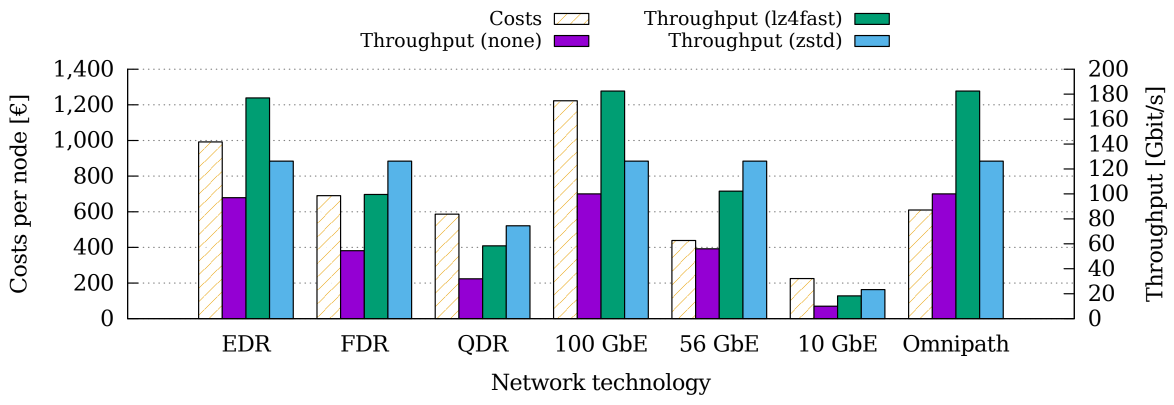


Figure 7. Costs and throughput for network technology when compressing communication

Figure 7 shows the costs and throughput per node when equipping nodes (and switches) with EDR, FDR, or QDR InfiniBand, 100 Gbit/s, 56 Gbit/s, or 10 Gbit/s Ethernet or Omnipath. The maximum throughput is shown for lz4fast and zstd. When using zstd with networks faster than 54 Gbit/s, the maximum throughput is limited due to zstd's compression throughput. Therefore, lz4fast is usually the better choice for high performance networks. Mistral's FDR InfiniBand network could be replaced with QDR InfiniBand when applying lz4fast compression for all network communication, decreasing costs by 15 %. Alternatively, the per-node throughput could be improved to roughly 100 Gbit/s or 125 Gbit/s when using lz4fast or zstd, respectively.

2.2.3. Storage

As mentioned previously, Mistral's storage system cost approximately € 6,000,000. The storage is distributed across roughly 60 Scalable Storage Units (SSUs), which contain two complete storage servers, and 60 Expansion Storage Units (ESUs) that are just JBODs, each connected to one SSU. Therefore, we can assume a cost of € 100,000 per SSU/ESU pair with a base cost of € 10,000 per SSU/ESU pair and up to € 90,000 for the HDDs.[8] Additionally, each SSU/ESU pair is capable of providing 833 TB of capacity and delivering 10.8 GB/s of throughput. For the following considerations, we will only take server-side compression into account because the performance aspect of client-side compression has been implicitly included in the network throughput analysis.

Since the storage capacity and throughput can be scaled linearly by simply adding or removing more SSU/ESU pairs, the resulting storage capacity only depends on the compression ratio. The storage throughput depends on the compression throughput and the number of SSU/ESU pairs. Therefore, we will consider different compression algorithms in this case. Additionally, we will assume that the SSU's CPUs are completely busy with the normal file system load, that is, we will have to buy additional CPUs for the compression overhead. Each SSU/ESU pair is already equipped with two 8-core CPUs and we will include additional costs of € 1,500 per SSU/ESU pair for compression. To take different key aspects into account, we will consider two scenarios:

- S1: We determine the number of SSU/ESU pairs necessary to achieve a capacity of 50 PB and only purchase this amount. This scenario will typically result in lower costs and decreased throughput.
- S2: We determine the number of HDDs necessary to achieve a capacity of 50 PB and spread these disks evenly across 60 SSU/ESU pairs. This scenario will typically result in both slightly higher costs and higher throughput than S1.

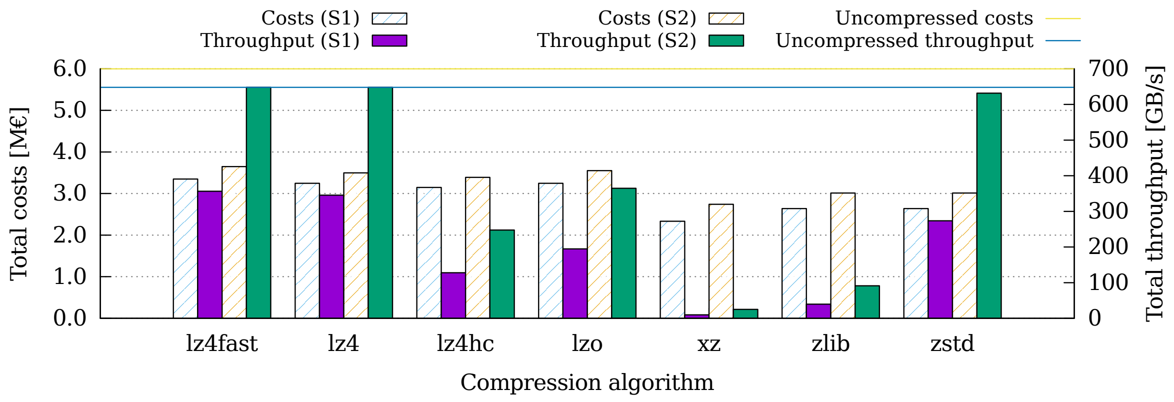


Figure 8. Storage costs and throughputs for different compression algorithms

Figure 8 shows the total costs and total throughput of Mistral's storage system when using the compression algorithms, shown in Table 2. First, we will take a look at scenario S1. As can be seen, employing compression allows decreasing the amount of necessary SSU/ESU pairs and thus costs. However, this also reduces the total throughput of the system because each SSU/ESU pair can only deliver at most 10.8 GB/s. The lz4fast and lz4 algorithms manage to achieve more than 10.8 GB/s per SSU/ESU pair and thus do not influence the throughput negatively on their own. The maximum throughput of zstd is only slightly less than that; therefore, the performance

[8]An SSU/ESU pair might be cheaper because auxiliary infrastructure is required in addition to the pair. However, for the purposes of modeling the possible savings, this estimation is good enough.

degradation can be mostly neglected. For these algorithms, the decrease in total throughput is caused exclusively by the reduction of the number of SSU/ESU pairs. However, lz4hc, xz and zlib have low throughputs and therefore do decrease total throughput.

When looking at scenario S2, costs are slightly increased overall due to the base cost of each SSU/ESU pair. However, the higher number of SSU/ESU pairs significantly increases throughput. For lz4 and lz4fast, performance is not degraded at all and costs are decreased to roughly €3,500,000. In zstd's case, throughput is decreased insignificantly by 20 GB/s while costs are reduced by 50 % to €3,000,000.

2.3. Summary

The results presented in the previous sections draw a clear picture: using compression on different levels of the I/O stack presents opportunities for both performance increases and cost reductions. In the best case, the different approaches should be combined for their synergetic effects. Table 3 shows an overview of the advantages and disadvantages on CPU utilization, memory capacity, network throughput, storage capacity and cost savings when deploying compression on different levels of the I/O stack.

Table 3. Design choices for compression on various levels

Strategy	CPU	Memory	Network	Storage	Cost savings
Memory	−	+	+	0	0
Communication	−	0	+	0	0
I/O (client)	−	0	+	+	+
I/O (server)	−	0	0	+	+
All levels	−	+	+	+	+

3. Ongoing work

Due to the very promising perspectives regarding performance improvements and cost savings, we have started to integrate compression into the HPC I/O stack on several levels. In the following, two approaches are described, one for lossless compression within the file system and one for lossy application-specific compression.

3.1. File system compression

Parallel distributed file systems typically do not have support for compression. However, several of them use underlying local file systems such as btrfs or ZFS and thus allow using those with compression support. While OrangeFS can use any POSIX file system, Lustre is limited to either ldiskfs or ZFS [19]. Consequently, using Lustre with its ZFS backend, it is possible to enable compression on a Lustre file system. However, Lustre currently has no support for compression, that is, it can not enable it by itself and has no knowledge of whether it is active or not.

When using this approach, data is only compressed within ZFS, as can be seen on the left side of Figure 9. This has several implications: since data is only compressed when arriving on the file system servers, there are no benefits regarding network throughput. If the network is a bottleneck for overall I/O performance, this can be a significant disadvantage. However, since compression is performed on the servers, it also can not have an influence on application performance. This influence might be problematic, if I/O is performed asynchronously and in parallel to the computation. Another benefit is that this setup can be deployed without any modifications to Lustre as long as Lustre's ZFS backend is used. Depending on the chosen compression algorithm, CPU overhead might be introduced on the file system servers, which might also have an influence on energy consumption. If particularly slow algorithms are used, they will also negatively impact I/O performance. However, this level of support already enables the cost savings presented in Section 2.2.3.

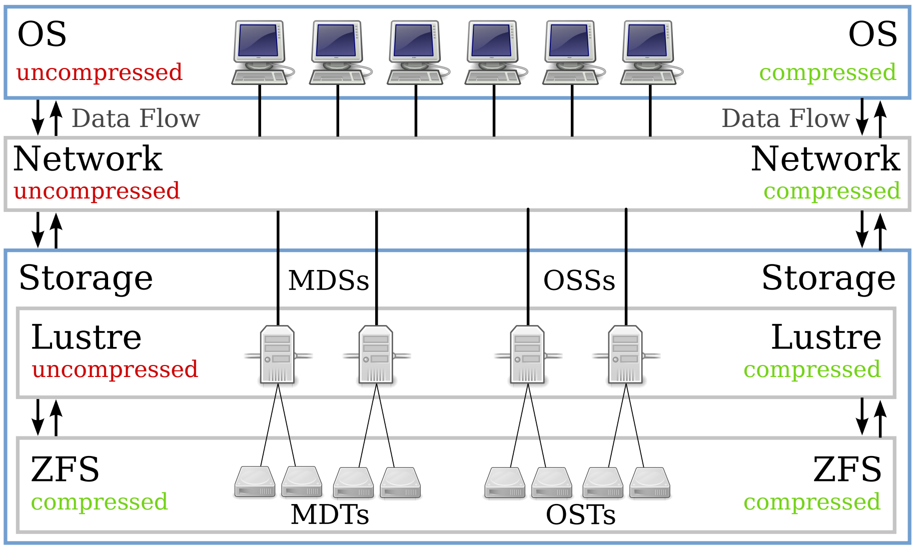


Figure 9. Comparison of different levels of support for compression in Lustre. Left: Compression is only performed on the servers. Right: Compression is performed on the clients

As part of our work within the Intel Parallel Computing Center "Enhanced Adaptive Compression in Lustre," we will add compression support to Lustre. This will include support for performing compression on both the clients and servers. Additionally, compression will be adaptive, that is, adjust itself according to the data to be compressed and several performance metrics. Applications will be able to influence the compression via Lustre's `ladvise` interface. This will allow proper integration with third-party applications and I/O libraries. On the one hand, applications already performing application-specific compression will be able to turn off compression as necessary through this interface. On the other hand, the compression support will be completely transparent to applications and libraries if they do not make use of this interface.

Once finished, this will bring compression support to the level depicted on the right side of Figure 9, that is, Lustre will have information about the compression status and act accordingly. For instance, data compressed on the client can be stored directly on the file system servers without decompression, reducing overhead. However, in case if a long-term archival is desired, the data might also be recompressed using different compression algorithms on the servers for higher space savings. While performing the compression on the clients might negatively influence applications, network throughput can be virtually increased by reducing the amount of data that has to be transferred via the network. However, as this compression will only be performed during the applications' I/O phases, interference should be minimal as long as sufficiently fast compression algorithms are used and/or I/O is not performed asynchronously.

Another major new feature will be adaptive compression. We have started working on support for adaptive compression within ZFS [13]. This support will be further modularized and its underlying functionality will be used in the Lustre client. Even though ZFS stores the used compression algorithm on a per-record level, it can only be set on a per-file-system level. Because modifying this file system parameter frequently does not make sense and is too coarse-grained, additional functionality is required. While lz4 is an appropriate compression algorithm for high performance I/O, gzip is more suitable for archival purposes due to its higher compression ratios. Lustre uses a single ZFS file system for each OST and MDT, therefore making it impossible to work around the problem using multiple file systems. Consequently, Lustre could either be tuned for high performance or archival, but not both. Adaptive compression allows selecting an appropriate compression algorithm based on different selection criteria during runtime. Based on client-provided information about the desired compression mode (high performance or archival), a cost function can be used to select the best compression algorithm for the current data. Additionally, performance metrics will be included in the decision process to make sure that clients and servers do not get overloaded.

3.2. Application-specific compression

To complement our file system approach, we have started to develop the Scientific Compression Library (SCIL) that allows fine-grained control over expected data accuracy metrics and performance behavior within the DFG-funded project AIMES.[9] Based on user-supplied hints, data properties and system performance characteristics, the library will adaptively choose the compression pipeline (algorithms) on behalf of the user. Therewith, users do not have to predefine the algorithm but can define the parameters that matter from the application point of view. Deploying an improved algorithm can then automatically benefit many existing applications. Amongst other goals, this library will be embedded as a filter into HDF5. We also push domain-specific solutions for icosahedral grids. Integrating approaches on both the application level and within the file system will allow us to maximize the benefits achieved by compression.

4. Conclusion and future work

Based on the results obtained by modeling the impact of compression on performance and costs, applying compression throughout the whole I/O stack is a viable approach to improve performance and decrease costs. Due to new and upcoming high performance compression algorithms, such as lz4fast, it is now possible to compress data at all levels of the storage hierarchy without sacrificing performance or increasing costs. In fact, our analyses have shown that it is possible to reduce costs of large-scale storage systems by up to 50 % without any negative impact on throughput or capacity. If costs are not a concern, compression can be used to improve performance by up to 133 % for network communication.

To leverage these benefits, we will integrate support for adaptive compression in Lustre. This will allow users to increase their file system's capacity without significant performance impacts. However, as the file system is limited in its insight into the applications' data, we will continue to explore new possibilities of application-specific compression with our Scientific Compression Library. Both approaches will interact to avoid unnecessary overhead and increase efficiency.

[9] `https://wr.informatik.uni-hamburg.de/research/projects/aimes/`

We thank Intel Corporation for funding our Intel Parallel Computing Center to integrate enhanced adaptive compression into Lustre. AIMES is supported by the German Research Foundation (DFG) through the Priority Programme 1648 "Software for Exascale Computing."

References

1. CMIP5 – Overview. `http://cmip-pcmdi.llnl.gov/cmip5/`. Last accessed: 2016-04.

2. Mohamed S. Abdelfattah, Andrei Hagiescu, and Deshanand Singh. Gzip on a chip: High performance lossless data compression on fpgas using opencl. In *Proceedings of the International Workshop on OpenCL 2013 &#38; 2014*, IWOCL '14, pages 4:1–4:9, New York, NY, USA, 2014. ACM.

3. Kenneth C. Barr and Krste Asanović. Energy-aware lossless data compression. *ACM Trans. Comput. Syst.*, 24(3):250–291, August 2006.

4. L. Benini, D. Bruni, A. Macii, and E. Macii. Hardware-assisted data compression for energy minimization in systems with embedded processors. In *Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings*, pages 449–453, 2002.

5. Jeff Bonwick, Matt Ahrens, Val Henson, Mark Maybee, and Mark Shellenbaum. The Zettabyte File System. 2003.

6. Konstantinos Chasapis, Manuel Dolz, Michael Kuhn, and Thomas Ludwig. Evaluating Power-Performace Benefits of Data Compression in HPC Storage Servers. In Steffen Fries and Petre Dini, editors, *IARIA Conference*, pages 29–34. IARIA XPS Press, 04 2014.

7. Yanpei Chen, Archana Ganapathi, and Randy H. Katz. To compress or not to compress - compute vs. io tradeoffs for mapreduce energy efficiency. In *Proceedings of the First ACM SIGCOMM Workshop on Green Networking*, Green Networking '10, pages 23–28, New York, NY, USA, 2010. ACM.

8. D. J. Craft. A fast hardware data compression algorithm and some algorithmic extensions. *IBM J. Res. Dev.*, 42(6):733–745, November 1998.

9. Sébastien Denvil. The ESGF's organization with a detailed discussion of the CMIP6 project and upcoming challenges. talk, `https://rd-alliance.org/sites/default/files/attachment/RDA-ESGF-2015.pdf`, 2015.

10. Peter Deutsch. DEFLATE Compressed Data Format Specification version 1.3. RFC 1951, 1996.

11. A. Dzhagaryan, A. Milenkovic, and M. Burtscher. Energy efficiency of lossless data compression on a mobile device: An experimental evaluation. In *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on*, pages 126–127, April 2013.

12. ECMA. Standard ECMA-321: Streaming Lossless Data Compression Algorithm – (SLDC). `http://www.ecma-international.org/publications/standards/Ecma-321.htm`, June 2011.

13. Florian Ehmke. Adaptive Compression for the Zettabyte File System. Master's thesis, Universität Hamburg, 02 2015.

14. Rosa Filgueira, Malcolm Atkinson, Alberto Nuñez, and Javier Fernández. *Euro-Par 2012 Parallel Processing: 18th International Conference, Euro-Par 2012, Rhodes Island, Greece, August 27-31, 2012. Proceedings*, chapter An Adaptive, Scalable, and Portable Technique for Speeding Up MPI-Based Applications, pages 729–740. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

15. Rosa Filgueira, Malcolm Atkinson, Yusuke Tanimura, and Isao Kojima. *Euro-Par 2014 Parallel Processing: 20th International Conference, Porto, Portugal, August 25-29, 2014. Proceedings*, chapter Applying Selectively Parallel I/O Compression to Parallel Storage Systems, pages 282–293. Springer International Publishing, Cham, 2014.

16. Rosa Filgueira, David E. Singh, Alejandro Calderón, and Jesús Carretero. CoMPI: Enhancing MPI Based Applications Performance and Scalability Using Run-Time Compression. In *Proceedings of the 16th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 207–218, Berlin, Heidelberg, 2009. Springer-Verlag.

17. Rosa Filgueira, David E. Singh, Jesús Carretero, Alejandro Calderón, and Félix García. Adaptive-Compi: Enhancing Mpi-Based Applications - Performance and Scalability by Using Adaptive Compression. *Int. J. High Perform. Comput. Appl.*, 25(1):93–114, February 2011.

18. Nathanel Hübbe and Julian Kunkel. Reducing the HPC-Datastorage Footprint with MAFISC – Multidimensional Adaptive Filtering Improved Scientific data Compression. *Computer Science - Research and Development*, pages 231–239, 05 2013.

19. Intel High Performance Data Division. Lustre – The High Performance File System, 2013.

20. J. Kane and Q. Yang. Compression speed enhancements to lzo for multi-core systems. In *Computer Architecture and High Performance Computing (SBAC-PAD), 2012 IEEE 24th International Symposium on*, pages 108–115, Oct 2012.

21. Meaza Taye Kebede. Performance Comparison of Btrfs and Ext4 Filesystems. Master's thesis, University of Oslo, 2012.

22. Kush K. Kella and Aasia Khanum. APCFS: Autonomous and Parallel Compressed File System. *International Journal of Parallel Programming*, 39(4):522–532, 2010.

23. Peter Kogge, Keren Bergman, Shekhar Borkar, Dan Campbell, William Carlson, William Dally, Monty Denneau, Paul Franzon, William Harrod, Kerry Hill, Jon Hiller, Sherman Karp, Stephen Keckler, Dean Klein, Robert Lucas, Mark Richards, Al Scarpelli, Steven Scott, Allan Snavely, Thomas Sterling, R. Stanley Williams, and Katherine Yelick. ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems, DARPA report. `http://www.cse.nd.edu/Reports/2008/TR-2008-13.pdf`, Sep 2008.

24. Rachita Kothiyal, Vasily Tarasov, Priya Sehgal, and Erez Zadok. Energy and Performance Evaluation of Lossless File Data Compression on Server Systems. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, SYSTOR '09, pages 4:1–4:12, New York, NY, USA, 2009. ACM.

25. Julian Kunkel, Michael Kuhn, and Thomas Ludwig. Exascale Storage Systems – An Analytical Study of Expenses. *Supercomputing Frontiers and Innovations*, pages 116–134, 06 2014.

26. Sriram Lakshminarasimhan, Neil Shah, Stephane Ethier, Seung-Hoe Ku, Choong-Seock Chang, Scott Klasky, Rob Latham, Rob Ross, and Nagiza F Samatova. ISABELA for effective in situ compression of scientific data. *Concurrency and Computation: Practice and Experience*, 25(4):524–540, 2013.

27. P. Lindstrom and M. Isenburg. Fast and efficient compression of floating-point data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1245–1250, Sept 2006.

28. Peter Lindstrom. Fixed-Rate Compressed Floating-Point Arrays. *Visualization and Computer Graphics, IEEE Transactions on*, 20(12):2674–2683, 2014.

29. Dirk Meister, Jürgen Kaiser, Andre Brinkmann, Michael Kuhn, Julian Kunkel, and Toni Cortes. A Study on Data Deduplication in HPC Storage Systems. In *Proceedings of the ACM/IEEE Conference on High Performance Computing (SC)*, 11 2012.

30. Dutch T. Meyer and William J. Bolosky. A study of practical deduplication. In *Proceedings of the 9th USENIX Conference on File and Stroage Technologies*, FAST'11, pages 1–1, Berkeley, CA, USA, 2011. USENIX Association.

31. Nitin Gupta. zram: Compressed RAM based block devices. `https://www.kernel.org/doc/Documentation/blockdev/zram.txt`, 11 2015. Last accessed: 2016-04.

32. Ritesh A Patel, Yao Zhang, Jason Mak, Andrew Davidson, and John D Owens. *Parallel lossless data compression on the GPU.* IEEE, 2012.

33. P. Ratanaworabhan, Jian Ke, and M. Burtscher. Fast lossless compression of scientific floating-point data. In *Data Compression Conference (DCC'06)*, pages 133–142, March 2006.

34. Christopher M. Sadler and Margaret Martonosi. Data compression algorithms for energy-constrained devices in delay tolerant networks. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, SenSys '06, pages 265–278, New York, NY, USA, 2006. ACM.

35. The Green500 Editors. Green500. `http://www.green500.org/`, 2016. Last accessed: 2016-04.

36. The TOP500 Editors. TOP500. `http://www.top500.org/`, 06 2014. Last accessed: 2016-04.

37. Ning Wang, Jian-Wen Bao, Jin-Luen Lee, Fanthune Moeng, and Cliff Matsumoto. Wavelet Compression Technique for High-Resolution Global Model Data on an Icosahedral Grid. *Journal of Atmospheric and Oceanic Technology*, 32(9):1650–1667, 2015.

38. Benjamin Welton, Dries Kimpe, Jason Cope, Christina M. Patrick, Kamil Iskra, and Robert Ross. Improving I/O Forwarding Throughput with Data Compression. In *Proceedings of the 2011 IEEE International Conference on Cluster Computing*, CLUSTER '11, pages 438–445, Washington, DC, USA, 2011. IEEE Computer Society.

39. R.N. Williams. An extremely fast Ziv-Lempel data compression algorithm. In *Data Compression Conference, 1991. DCC '91.*, pages 362–371, Apr 1991.

40. Rong Xu, Zhiyuan Li, Cheng Wang, and Peifeng Ni. Impact of data compression on energy consumption of wireless-networked handheld devices. In *Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on*, pages 302–311, May 2003.

41. Yann Collet. lz4. `http://www.lz4.org/`, 04 2016. Last accessed: 2016-04.

42. J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *Information Theory, IEEE Transactions on*, 23(3):337–343, May 1977.