for wave surface behavior modeling. This model considers wave surface as a spatio-temporal field. Every point value of such field is calculated as an infinite weighted sum of previous point values in a given sub region plus white noise value. Therefore the state of a wave surface at a given time has an autoregressive dependency on a set of previous states and some random variable with normal distribution [8]. This dependency is defined by the following equation.

$$z_{x,y,t} = \sum_{i=0}^{p_1}\sum_{j=0}^{p_2}\sum_{k=0}^{p_3}\varphi_{i,j,k}\cdot z_{x-i,y-j,t-k} + \varepsilon_{x,y,t}\,, \quad (1)$$

where φ are autoregressive coefficients. They can be estimated from the auto covariate function (ACF) using Yule-Walker equations. The set of such equations for three-dimensional problem is given by the following formulas, where $\gamma_{i,j,k}$ are discrete values of ACF.

$$A\varphi = b,\ a_{i,j} = \gamma_{|x(i)-x(j)|,|y(i)-y(j)|,|t(i)-t(j)|},\ b_i = \gamma_{x(i),y(i),t(i)} \quad (2)$$

$x(i) = \mathrm{mod}((i+1)/(p_1p_2), p_1)$,

$y(i) = \mathrm{mod}((i+1)/p_3, p_2)$,

$t(i) = \mathrm{mod}(i+1, p_3)$.

The experimental investigations of sea waves [12, 16] shown, that the value of wave skewness is limited by 0.1-0.52, and values of kurtosis is limited by 0.1-0.7. Therefore the generated wave surface should be transformed to match this distribution. The transformation function z=f(y) for an arbitrary probability distribution F(z) defined as the solution of nonlinear static equation

$$F(z) = \Phi(y), \quad (3)$$

where Φ(y) is the one-dimensional uniform Gaussian probability distribution.

However, if nonlinear transformation of process is taken place, the autocovariance function of the process has to be transformed explicitly. The transformation is done by using Gram-Charlier series [5, 9] and the following formula, where z(y) is the solution of equation (3) and $H_m(y)$ is Hermite polynomial.

$$K_z(x,y,t) = \sum_{m=0}^{\infty} C_m^2 \frac{K_y(x,y,t)}{m!}, \quad (4)$$

$$C_m = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} f(y)H_m(y)\exp(-\frac{y^2}{2})dy \,.$$

The coefficients C_m are estimated analytically.

2.1. Implementation

Implementation has been done on the base of OpenCL, MPI and OpenMP. However algorithm is the same for all of them. The first step of algorithm is estimation of auto covariate function (ACF). Usually such estimation is carried out with the help of natural data. This approach results in overdetermined Yule-Walker system and involves least squares method using. However research [8, 16] showed that in this case using of approximation of preliminary smoothed ACF is more robust approach than direct using of natural data. Theoretically the number of AR coefficients tends to infinity but it is not practically possible. Therefore the approximation of ACF should be chosen tending to zero so that φ_i values gradually become smaller and could be considered as zeroes at some point of time. Sample of such approximation is given below.

$$\gamma(x, y, t) = e^{-\alpha(|x|+|y|+|t|)}\cdot\cos(\beta x)\cdot\cos(\beta y)\cdot\cos(\beta t)\,. \quad (5)$$

The next step is ACF transformation. It is done by choosing of the distribution function. One of the possible choices is asymmetric normal distribution denoted by the following formula, where T(z, α) is Owen's T function and F(z) is normal distribution function (a sample of this distribution function is shown in fig. 1).

$$\Phi(z) = F(z) - 2T(z, \alpha)\,. \quad (6)$$

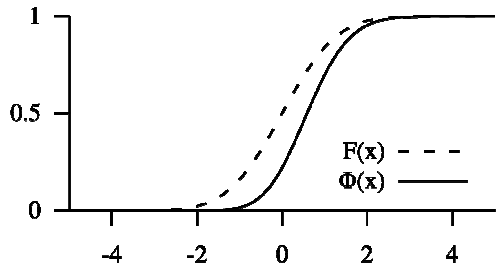

Fig. 1. Sample of asymmetric normal distribution function Φ(x), where α=1.2, mean=0.6130, variance=0.6243, skewness=0.2004, kurtosis=0.1026.

Solving equation (3) with chosen distribution function at each surface point is more precise but inefficient approach, therefore it is rational to solve equation at a certain grid points and to interpolate the results. The solution is approximated by the following polynomial.

$$z = f(y) \cong \sum_{i=0}^{N} d_i y^i \,. \quad (7)$$

The approximate parameters d_i are obtained by means of the least squares approach. In current implementation distribution function was approximated by the 12 order polynomial at 500 grid points in range [-5·Var(z), 5·Var(z)]. Choosing of higher order of polynomial resulted in floating point number overflows and additional coefficients tending to zero. Choosing of different grid size also had minor effect on the approximation. In most cases three coefficients were sufficient for ACF transformation; interpolation increased relative error from 10^{-5} to $0.43\cdot10^{-3}$.

Autocovariate function can be transformed using formula (4). The criterion of stopping of calculation process is defined as the follow [5, 9].

$$\sigma_z^2 = \sum\frac{C_m^2}{m!}\,. \quad (8)$$

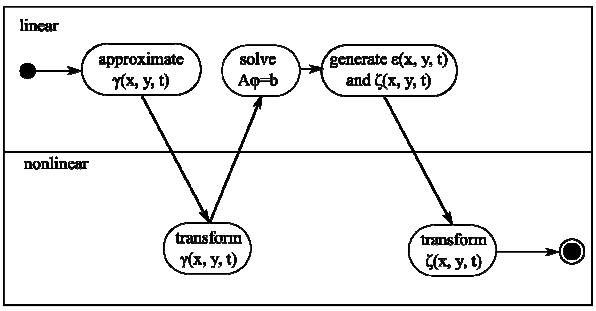

Fig. 2. Algorithm of wave surface generation

The next step is to determine AR coefficients by solving Yule-Walker equations. The optimum Yule-Walker system size should be chosen by hand (or by any heuristic approach) without invoking least squares method often used in solving one-dimensional problem. The matrix of Yule-Walker system is positive definite and symmetric, therefore it is efficient to solve it using Cholesky decomposition. The following step in the algorithm is generation of white noise with normal distribution. It is done by parallel implementation of Mersenne Twister algorithm. Separate unique generator initialized by a standalone dcmt program [15] is used in each

thread. This approach reduces random number correlation between generators [14].

Knowing of AR coefficients and white noise realization are sufficient for wave surface generation. This process starts with generation of separate parts of wave surface in each thread. These parts have intentionally larger sizes. Each part begins with so-called acceleration interval. On this interval points are generated using partial set of AR coefficients resulting in inaccurate wave surface. Therefore this interval is cut and remaining points are merged using bilateral autoregressive dependency (with interval between parts initially filled with zeroes) [7]. This technique is sometimes called "sewing". It is done by the following formula.

$$z_{x,y,t} = \sum_i \sum_j \sum_k \varphi_{i,j,k} z_{x-i,y-j,z-k} + \sum_i \sum_j \sum_k \varphi_{i,j,k} z_{x+i,y+j,z+k} + \varepsilon_{x,y,t} \quad (9)$$

The generation of wave surface is done when mathematical methods were heavy used. So its validity should be statistically proven. It is done by comparing certain wave surface characteristics (ACF, variance, spectrum) and by comparing distributions (z, slopes, heights and periods). Partial results of such tests are given in the fig. 3. General algorithm for wave surface generation is shown in the fig. 2.

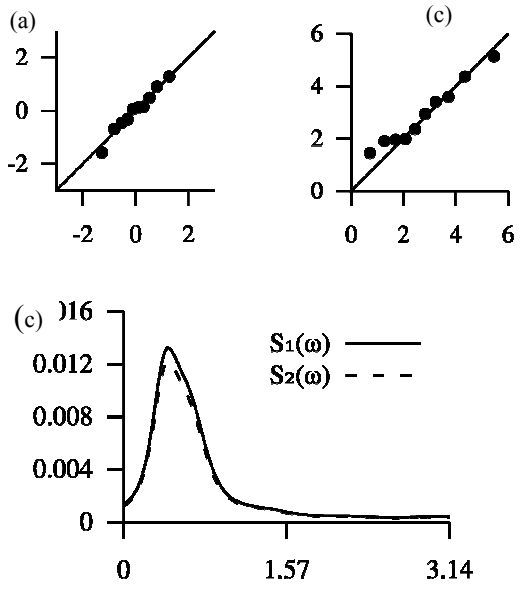

Fig. 3. (a) wave z distribution; (b) real S_1 and model S_2 spectrum comparison; (c) wave heights distribution;

3. COMPARISON OF OPENCL, OPENMP AND MPI IMPLEMENTATIONS

Implementation has been made using OpenCL, OpenMP and MPI technologies. OpenCL (Open Computing Language) provides user with tools to run programs on GPUs and standalone programming language based on C99. OpenMP (Open Multi-Processing) provides user with tools to parallelize programs on CPUs supporting programming languages C/C++ and Fortran. Both technologies support data and task parallelism, offering shared memory for processes and thread synchronization tools. The significant difference between technologies is their hardware base (CPU and GPU architecture).

The GPU design was initially aimed for maximizing computational throughput; therefore a general graphics card contains several computation units that work in parallel. These units are optimized for data processing, and they are able to run multiple threads at a time. GPU thread is a lightweight hardware process, it is not a subject for a process context switches and has inexpensive spawning time. Described design resulted in GPU having more threads running at a time in comparison with CPU.

MPI (Message Passing Interface) provides user with tools for parallelizing of programs on multiple computers or processors. This technology also supports data and task parallelism but instead of shared memory it offers interprocess communication and synchronization routines.

Algorithms for wave surface generation using OpenCL and OpenMP are identical. OpenCL algorithm having multiple threads generates single wave surface part. OpenMP algorithm has one thread for each part. MPI implementation is quite different. It includes explicit synchronization before sewing. Each process receives small chunk of preceding wave surface part that sufficient to merge it with the next part. As soon as sewing is finished all parts are sent to main process.

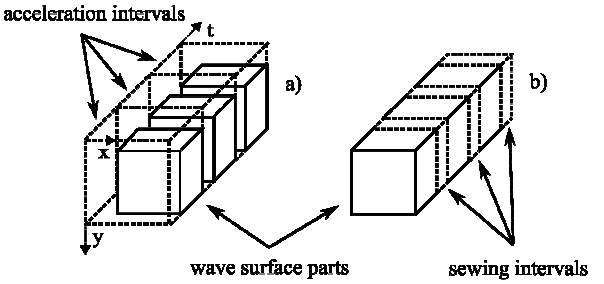

Fig. 4. Wave surface representation in device memory (3d view).

Programs were tested on the following hardware configuration: NVIDIA 8800GT graphics card with 112 stream processors at 1.5 GHz (10752 threads approx.), 256 Mb shared memory and 2 CPUs with 2 cores at 3 GHz (4 threads) connected directly via 1 Gbit Ethernet cable. Although graphics card is superior CPU in terms of performance, it has not been proved by real test results due to differences in technologies and algorithm implementations used. All tests were performed with single precision operation type due to OpenCL implementation constraints.

In accordance with benchmarks results of OpenCL and MPI programs execution times are linearly dependent on wave surface size, but increasing number of parts has small impact on performance. Such behavior has different causes. In MPI program the more we have wave surface parts the more interprocess communication is needed. As a result increased number of parts reduces wave generation time by half and doubles synchronization time, therefore total execution time remains roughly the same.

OpenCL program with large number of threads produces large amount of write operations. It causes high load on memory and therefore increases execution time. This problem can be solved by coalescing global memory accesses [17]; however this solution proved to be inappropriate considering current algorithm. Large number of parts results in a smaller interval between them and therefore to inferior part sewing. Small number of parts results in parallel threads not being fully utilized and algorithm being inefficient. OpenCL implementation is considered to be the most complicated, slow and having unstable scale of execution time among variable input parameters.

OpenCL and MPI implementations work faster with large amount of parts but it increases memory overhead for OpenCL and communication overhead for MPI. As opposed to this, the execution time of OpenMP program is linearly dependent on wave surface part count as well as on wave surface size. So we can consider OpenMP implementation as a good linear scaling way and the best performance in comparison with OpenCL and MPI implementations. Detailed test results are shown in table 1.

OpenMP technology has certain memory issues. The first problem is false sharing [18]. When one thread copies region of shared memory to local cache and the other thread writes to this memory region, it causes cache invalidation, increasing memory access time. The second problem is memory allocation issue [18]. Modern operating systems use first touch policy to allocate data objects. Under this policy the thread initializing data object gets the page associated with that data item in the memory local to processor it is currently executing on. Such strategy works well for single threaded applications; however first touch policy is not appropriate for multithreaded program. It can result in slow down memory accesses due to processes changing computing units after context switches and due to thread's irregular data access pattern including different memory pages.

Table 1. Performance test results for OpenCL, OpenMP and MPI implementations (single precision).

Wave surface parts			Size	Time, s		
OpenCL	OpenMP	MPI	All	OpenCL	OpenMP	MPI
256	1	2	13452	0.61	0.47	0.66
256	1	2	26905	1.23	0.94	1.38
256	1	2	53810	2.60	1.97	2.82
1024	1	2	107620	4.25	4.07	5.60
1024	1	2	215240	9.41	8.32	11.22
512	2	4	13452	0.51	0.24	0.64
512	2	4	26905	0.99	0.47	1.32
512	2	4	53810	2.36	1.07	2.67
2048	2	4	107620	3.90	2.20	5.35
2048	2	4	215240	8.08	4.42	10.94

Mentioned issues are important for large multiprocessor systems with hundreds of threads running simultaneously; however in the current configuration they have minor impact on program performance.

4. CONCLUSION

Different ship infrastructure can lead to different onboard intelligent system or virtual testbed configurations as well as different parallel computing technologies have their own advantages and disadvantages. Therefore the final choice is often dependent on a problem and available resources. Here all technologies and their benefits will be compared.

The use of OpenMP can ensure lower software development costs considering more elaborate and simple API among other technologies as well as good performance (according to test results); however multiprocessor systems tend to have higher market prices and low hardware scalability issues compared to computer clusters.

The use of OpenCL is justified only in solving certain problems that map well to GPU architecture and require real-time visualization. OpenGL and OpenCL interoperability reduces cost of transferring data between CPU and GPU memory and it is a great benefit from the visualization point of view (that is very important for virtual testbed development). However it is hard to fully utilize GPU power to solve wave surface generation problem; high discretion of wave surface often results in non stationary autoregressive process, therefore waves visualization can be performed in conjunction with any other technology.

OpenCL and OpenMP are evolving technologies whereas MPI is a standard for solving large scale problems. It has certain debugging issues as OpenCL does; however it provides uniform synchronization through the interprocess communication and simple architecture as OpenMP does.

All the technologies have their own integration scope and final decision should be made considering external factors and specified requirements.

5. ACKNOWLEDGEMENT

Computations were partly carried out on cluster HPC-0011654-001 of Saint-Petersburg State University, Faculty of Applied Mathematics and Control Processes.

REFERENCES

[1] Belenky V.L.&Sevastianov N.B.: Stability and Safety of Ships. Risk of Capsizing. SNAME, Jersey City (2007)

[2] Bogdanov A., Degtyarev A., Nechaev Yu.: Problems of development of virtual testbed for complex dynamic processes modeling. In: Intern. Conf. on Supercomputer Systems and its Applications (SSA'2004), Minsk, pp.31--37. (2004) (in Russian)

[3] Bogdanov A., Degtyarev A., Nechaev Yu.: Parallel algorithms for virtual testbed. In: 5th Intern. Conf. on Computer Science and Information Technologies, pp.393-398. NAS RA, Yerevan (2005)

[4] Bogdanov A., Degtyarev A., Soe Moe Lwin, Thurein Kyaw Lwin: Problems of Development of Complex Multi-layered Applications in Distributed Environment. In: 4th Intern. Conf. Distributed Computing and Grid-Technologies in Science and Education, pp.51-56. JINR, Dubna (2010)

[5] Boukhanovsky A., Degtyarev A.: Probabilistic Modelling of Stormy Sea Fields. In: Proc. of Intern. Conf. Navy and Shipbuilding Nowadays, A2-29, 10p., St.Petersburg (1996) (in Russian)

[6] Boukhanovsky A., Rozhkov V., Degtyarev A.: Peculiarities of Computer Simulation and Statistical Representation of Time-Spatial Metocean Fields. In: Alexandrov V.N. et al. (eds.) Computational Science – ICCS 2001. LNCS, vol.2073, part I, pp.463--472. Springer, Heidelberg (2001)

[7] Boukhanovky A. V., Ivanov S. V.: Parallel processing of data in information control systems. In: National Conference Control and information technology UIT-2003, v.2, pp. 64-68. Saint-Petersburg (2003) (in Russian)

[8] Box G., Jenkins G.: Time series analysis: Forecasting and control, Holden-Day, San Francisco (1970)

[9] Degtyarev A., Boukhanovsky A.: Peculiarities of Motion of Ship with Low Buoyancy on Asymmetrical Random Waves. In: Renilson M. (ed.) 7th Intern. Conf. on Stability of Ships and Ocean Vehicles. vol.B, pp.665—679. Launceston, Tasmania, Australia (2000)

[10] Degtyarev A., Mareev V.: Climatic Spectra and Long-Term Risk Assessment. In: 11th Intern. Ship Stability Workshop, pp.108-114. Wageningen, The Netherlands (2010)

[11] Gallopoulos E., Houstis E., and Rice J.R.: Computer as thinker doer: Problem-solving environments for computational science. *IEEE Computational Science and Engineering*, 2:13--23 (1994)

[12] Huang N.E., Long S.R., Chi-Chao Tung e.a.: A non-linear statistical model for surface elevation of nonlinear random wave fields. *J.Geophys. Res.*, 88, No.12 (1983).

[13] Korkhov V.V.: Hierarchical Resource Management in Grid Computing, PhD thesis, Universiteit van Amsterdam, 130 p. Publ: Ipskamp drukkers B.V., Enschede, the Netherlands (2009)

[14] Makoto Matsumoto, Takuji Nishimura: Dynamic Creation of Pseudorandom Generators

[15] Podlozhnyuk V. Parallel Mersenne Twister (2007)

[16] Rozhkov V.A., Trapeznikov U.A.: Probabilistic models of ocean processes. Gidrometeoizdat, Leningrad (1990) (in Russian)

[17] OpenCL Programming Guide for the CUDA Architecture (2010)

[18] Ruud van der Pas: OpenMP and performance. In: International Workshop on OpenMP, Dresden (2009)