Virtual testbed: Ship motion simulation for personal workstations

Alexander Degtyarev[0000−0003−0967−2949],
Vasily Khramushin[0000−0002−3357−169X],
Ivan Gankevich*[0000−0001−7067−6928],
Ivan Petriakov,
Anton Gavrikov[0000−0003−2128−8368], and
Artemii Grigorev

Saint Petersburg State University, Russia
{a.degtyarev,v.khramushin,i.gankevich}@spbu.ru,
{st049350,st047437,st016177}@student.spbu.ru
https://spbu.ru/

Abstract. Virtual testbed is a computer programme that simulates ocean waves, ship motions and compartment flooding. One feature of this programme is that it visualises physical phenomena frame by frame as the simulation progresses. The aim of the studies reported here was to assess how much performance can be gained using graphical accelerators compared to ordinary processors when repeating the same computations in a loop. We rewrote programme's hot spots in OpenCL to able to execute them on a graphical accelerator and benchmarked their performance with a number of real-world ship models. The analysis of the results showed that data copying in and out of accelerator's main memory has major impact on performance when done in a loop, and the best performance is achieved when copying in and out is done outside the loop (when data copying inside the loop involves accelerator's main memory only). This result comes in line with how distributed computations are performed on a set of cluster nodes, and suggests using similar approaches for single heterogeneous node with a graphical accelerator.

Keywords: wavy surface · pressure field · pressure force · ship · wetted surface · OpenCL · GPGPU.

1 Introduction

Simulation of ship motion in ocean waves is done in several computer programmes [9, 11, 12] that differ in what physical phenomena they simulate (manoeuvring in waves, compartment flooding, regular and irregular waves, wind, real-time simulation or batch processing etc.) and application area (scientific studies, education or entertainment). These programmes are virtual analogues of ship model basins that are used to simulate characteristics and behaviour of ship in a particular sea conditions. The advantage of using virtual ship model basin over a physical one is that experiments are performed in real scale (with real-sized ships and ocean waves) and on a computer without the need to access high-technological facility.

Although, all numerical experiments are performed on a computer, one computer is not powerful enough to perform them fast. Often, this problem is solved by using a cluster of computer nodes or a supercomputer; however, a supercomputer or a cluster is another high-technological facility that a researcher have to gain access to. In that case virtual ship model basin has little advantage over a physical one: the research is slowed down by official documents' approvals and time-sharing of computing resources.

One way of removing this barrier is to use graphical accelerator to speed up computations. In that case simulation can be performed on a regular workstation that has a dedicated graphics card. Most of the researchers use GPU to make visualisation in real-time, but it is rarely used for speeding up simulation parts, let alone the whole programme. In [2] the authors use GPU to speed up computation of free surface motion inside a tank. In [14] the authors rewrite their simulation code using Fast Fourier transforms and propose to use GPU to gain more performance. In [6] the authors use GPU to simulate ocean waves. Nevertheless, the most efficient way of using GPU is to use it for both computation and visualisation: it allows to minimise data copying between CPU and GPU memory and use mathematical models, data structures and numerical methods that are tailored to graphical accelerators.

The present research proposes a numerical method for computing velocity potentials and wave pressures on a graphical accelerator, briefly explains other methods in the programme, and presents benchmarks for asynchronous visualisation and simulation.

2 Methods

Virtual testbed is a computer programme that simulates ocean waves, ship motions and compartment flooding. One feature that distinguishes it with respect to existing proposals is the use of graphical accelerators to speed up computations and real-time visualisation that was made possible by these accelerators.

The programme consists of the following modules: `vessel` reads ship hull model from an input file, `gui` draws current state of the virtual world and `core` computes each step of the simulation. The `core` module consists of components that are linked together in a pipeline, in which output of one component is the input of another one. The computation is carried out in parallel to visualisation, and synchronisation occurs after each simulation step. It makes graphical user interface responsive even when workstation is not powerful enough to compute in real-time.

Inside `core` module the following components are present: wavy surface generator, velocity potential solver, pressure force solver. Each component in the `core` module is interchangeable, which means that different wavy surface generators can be used with the same velocity potential solver. Once initialised, these components are executed in a loop in which each iteration computes the next time step of the simulation. Although, iterations of the loop are sequential, each component is internally parallel, i.e. each component uses OpenMP or OpenCL to perform computations on each processor or graphical core. In other words, Virtual testbed follows BSP model [13] for organising parallel computations, in which a programme consists of sequential steps each of which is internally parallel (fig. 1).

2.1 Wavy surface generation

There are three models that are used for wavy surface generation in Virtual testbed: autoregressive moving average model (ARMA), Stokes wave, and plane sine/cosine wave. It is not beneficial in terms of performance to execute ARMA model on a graphical accelerator [5]: its algorithm does not use transcendental mathematical functions, has nonlinear memory access pattern and complex information dependencies. It is much more efficient (even without serious optimisations) to execute it on a processor. In contrast, the other two wave models are embarrassingly parallel and easy to rewrite in OpenCL.

Each wave model outputs three-dimensional (one temporal and two spatial dimensions) field of wavy surface elevation, and ARMA model post-processes this field using the following algorithm. First, autocovariance function (ACF) is estimated from the input field using Wiener—Khinchin theorem. Then ACF is used to build autocovariance matrix and determine autoregressive model coefficients. Finally, the coefficients are used to generate new wavy surface elevation field.

The resulting field is stochastic, but has the same integral characteristics as the original one. In particular, probability distribution function of wavy surface elevation, wave height, length and period are preserved. Using ARMA model for post-processing has several advantages.

- It makes wavy surface aperiodic (its period equals period of pseudo-random number generator, which can be considered infinite for all practical applications) which allows to perform statistical studies using Virtual testbed.
- It is much faster to generate wavy surface with this model than with the original model, because ARMA model involves only multiplications and additions rather than transcendental mathematical functions.
- This model allows to use any wavy surface as the input (not only plane and Stokes waves). Frequency-directional spectrum of a particular ocean region can be used instead.

This paper gives only a short description of the model, please refer to [4, 5] for in-depth study.

To summarise, wavy surface generator produces wavy surface elevation field using one of the models described above. For ARMA model it is impractical to generate it using graphical accelerator, and for other models it is to trivial to discuss. This field is an input for velocity potential solver.

2.2 Velocity potential computation

Since wavy surface generator produces discretely given elevation field we may not use formula from linear wave theory to compute velocity potential; instead, we derived a formula for arbitrary surface for inviscid incompressible fluid:

$$\phi(x,y,z,t) = \mathcal{F}^{-1}_{x,y}\left\{ \frac{\cosh\left(2\pi|\boldsymbol{k}|(z+h)\right)}{2\pi|\boldsymbol{k}|} \frac{\mathcal{F}_{u,v}\{f(x,y,t)\}}{\mathcal{F}_{u,v}\{\mathcal{D}_3\left(x,y,\zeta\left(x,y\right)\right)\}} \right\}, \tag{1}$$

where

$$\begin{aligned}
f(x,y,t) &= \zeta_t(x,y,t) / \left(i f_1(x,y) + i f_2(x,y) - f_3(x,y)\right), \\
f_1(x,y) &= \zeta_x / \sqrt{1+\zeta_x^2+\zeta_y^2} - \zeta_x, & \mathcal{F}_{u,v}\{\mathcal{D}_3\left(x,y,z\right)\} = \cosh\left(2\pi|\boldsymbol{k}|z\right), \\
f_2(x,y) &= \zeta_y / \sqrt{1+\zeta_x^2+\zeta_y^2} - \zeta_y, & |\boldsymbol{k}| = \sqrt{u^2+v^2}, \\
f_3(x,y) &= 1 / \sqrt{1+\zeta_x^2+\zeta_y^2}.
\end{aligned}$$

Here $\boldsymbol{k}$ is wave number, ζ — wavy surface elevation, h — water depth, $\mathcal{F}$ — Fourier transform, ϕ — velocity potential. The formula is derived as a solution for continuity equation with kinematic boundary condition

$$\begin{aligned}
& \nabla^2\phi = 0, \\
& \phi_t + \frac{1}{2}|\boldsymbol{v}|^2 + g\zeta = -\frac{p}{\rho}, & \text{at } z = \zeta(x,y,t), \\
& D\zeta = \nabla\phi\cdot\boldsymbol{n}, & \text{at } z = \zeta(x,y,t),
\end{aligned} \tag{2}$$

without assumptions of linear wave theory (wave length is much larger than wave height). Hence it can be used for arbitrary-amplitude ocean waves. Here the first equation is continuity equation, the second is dynamic boundary condition, and the last one is kinematic boundary condition; p — pressure, ρ — fluid density, $\boldsymbol{v} = (\phi_x, \phi_y, \phi_z)$ — velocity vector, g — acceleration of gravity, and D — substantial (Lagrange) derivative. Since we solve for ϕ, dynamic boundary condition becomes explicit formula for pressure and is used to compute pressure force acting on a ship hull (see sec. 2.3).

Integral in (1) converges when summation goes over a range of wave numbers that are actually present in discretely given wavy surface. This range is determined numerically by finding crests and troughs for each spatial dimension of the wavy surface with polynomial interpolation and using these values to determine wave length. For small-amplitude waves this approach gives the same values of velocity potential field as direct application of the formula from linear wave theory.

Formula (1) is particularly suitable for computation on a graphical accelerator: it contains transcendental mathematical functions (complex exponents) that help offset slow global memory loads and stores, it is explicit which makes it easy to compute in parallel, and it is written using Fourier transforms that are efficient to compute on a graphical accelerator [15].

This paper gives only a short description of the method, please refer to [4, 5] for in-depth study.

2.3 Pressure force computation

There are three stages of pressure force computation: determining wetted surface of a ship hull, computing wave pressure field under wavy surface, and computing pressure force acting on a ship hull.

In order to determine wetted surface, ship hull is decomposed into triangular panels (faces) that approximate its geometry. Then for each panel the algorithm determines its position relative to wavy surface. If it is fully submerged, it is considered wetted; if it is partially submerged, the algorithm computes intersection points using bisection method and wavy surface interpolation, and slices the part of the panel which is above the wavy surface (for simplicity the slice is assumed to be straight line, as is the case for sufficiently small panels). Wave pressure at any point under wavy surface is computed using dynamic boundary condition from (2) as an explicit formula. Then the pressure is interpolated in the centre of each panel to compute pressure force acting on a ship hull.

It is straightforward to rewrite pressure computation for a graphical accelerator as its algorithm reduces to looping over a large collection of panels and performing the same calculations for each of them; however, dynamic boundary condition contains temporal and spatial derivatives that have to be computed. Although, computing derivatives on a processor is fast, copying the results to accelerator's main memory proved to be inefficient as there are four arrays (one for each dimension) that need to be allocated and transferred. Simply rewriting code for OpenCL proved to be even more inefficient due to irregular memory access pattern for different array dimensions. So, we resorted to implementing the algorithm described in [10], that stores intermediate results in the local memory of the accelerator. Using this algorithm allowed us to store arrays of derivatives entirely in graphical accelerator's main memory and eliminate data transfer altogether.

2.4 Translational and angular ship motion computation

In order to compute ship position, translational velocity, angular displacement and angular velocity for each time step we solve equations of translational and angular motion (adapted from [9]) using pressure force computed for each panel:

$$\dot{\boldsymbol{v}} = \frac{\boldsymbol{F}}{m} + g\boldsymbol{\tau} - \boldsymbol{\Omega} \times \boldsymbol{v} - \lambda\boldsymbol{v}, \qquad \dot{\boldsymbol{h}} = \boldsymbol{G} - \boldsymbol{\Omega} \times \boldsymbol{h}, \qquad \boldsymbol{h} = \mathcal{I} \cdot \boldsymbol{\Omega}.$$

Here $\boldsymbol{\tau} = (-\sin\theta, \cos\theta\sin\phi, -\cos\theta\cos\phi)$ is a vector that transforms g into body-fixed coordinate system, $\boldsymbol{v}$ — translational velocity vector, g — gravitational acceleration, $\boldsymbol{\Omega}$ — angular velocity vector, $\boldsymbol{F}$ — vector of external forces, m — ship mass, λ — damping coefficient, h — angular momentum, $\boldsymbol{G}$ — the moment of external forces, and $\mathcal{I}$ — inertia matrix.

We compute total force $\boldsymbol{F}$ and momentum $\boldsymbol{G}$ acting on a ship hull by adding forces acting on each panel. Then we solve the system of equations using numerical Runge—Kutta—Fehlberg method [8]. The vector of values determined by the method consists of all components of the following vectors: ship position, translational velocity, angular displacement and angular velocity (twelve variables in total). Since we are not interested in angular momentum, we use inertia matrix to obtain it from angular velocity and inverse inertia matrix to convert it back. Twelve vector components are too few to efficiently execute this method on a graphical accelerator and there is no other way of making iterative method parallel, so we execute it on a processor.

3 Results

3.1 Test setup

Virtual testbed performance was benchmarked in a number of tests. Since we use both OpenMP and OpenCL technologies for parallel computing, we wanted to know how performance scales with the number of processor cores and with and without graphical accelerator.

Graphical accelerators are divided into two broad categories: for general purpose computations and for visualisation. Accelerators from the first category typically have more double precision arithmetic units and accelerators from the second category are typically optimised for single precision. The ratio of single to double precision performance can be as high as 32. Virtual testbed produces correct results for both single and double precision, but OpenCL version supports only single precision, and graphical accelerators that we used have higher single precision performance (tab. 1). So we choose single precision in all benchmarks.

Table 1. Hardware configurations for benchmarks. For all benchmarks we used GCC version 8.1.1 compiler and optimisation flags `-O3 -march=native`.

			GPU GFLOPS	
Node	CPU	GPU	Single	Double
Storm	Intel Q9550	Radeon R7 360	1613	101
GPUlab	AMD FX-8370	NVIDIA GTX1060	4375	137
Capybara	Intel E5-2630 v4	NVIDIA P5000	8873	277

Double precision was used only for computing autoregressive model coefficients, because round-off and truncation numerical errors make covariance matrices (from which coefficients are computed) non-positive definite. These matrices typically have very large condition numbers, and linear system which they represent cannot be solved by Gaussian elimination or $LDLT$ Cholesky decomposition, as these methods are numerically unstable (at least in our programme).

Since Virtual testbed does both visualisation and computation in real-time, we measured performance of each stage of the main loop (fig. 1) synchronously with the parameters that affect it. To assess computational performance we measured execution time of each stage in microseconds (wall clock time) together with the number of wetted panels, and wavy surface size. To assess visualisation performance we measured the execution time of each visualisation frame (one iteration of the visualisation main loop) and execution time of computational frame (one iteration of the computational loop), from which it is easy to compute the usual frames-per-second metric. The tests were run for one minute and were forcibly stopped after the time ran out. Wall clock time was measured as a median across all simulation steps (or visualisation frames).

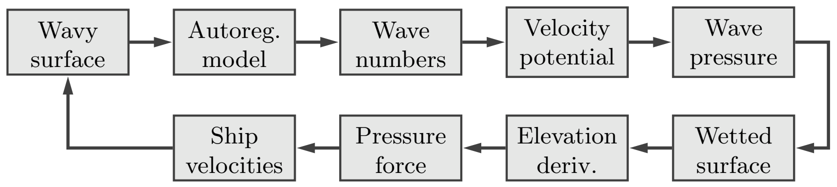

Fig. 1. Virtual testbed main loop.

We ran all tests on each node for increasing number of processor cores and with and without graphical accelerator. The code was compiled with maximum optimisation level including processor-specific optimisations which enabled auto-vectorisation for further performance improvements.

We ran all tests for each of the three ship hull models: Aurora cruiser, MICW (a hull with reduced moments of inertia for the current waterline) and a sphere. The first two models represent real-world ships with known characteristics and we took them from Vessel database [1] registered by our university which is managed by Hull programme [7]. Parameters of these ship models are listed in tab. 2, three-dimensional models are shown in fig. 2. Sphere was used as a geometric shape wetted surface area of which is close to constant under impact of ocean waves.

We ran all tests for each workstation from tab. 1 to investigate if there is a difference in performance between ordinary workstation and a computer for visualisation. Storm is a regular workstation with mediocre processor and graphical accelerator, GPUlab is a slightly more powerful workstation, and Capybara has the most powerful processor and professional graphical accelerator optimised for visualisation.

Table 2. Parameters of ship models that were used in the benchmarks.

	Aurora	MICW	Sphere
Length, m	126.5	260	100
Beam, m	16.8	32	100
Depth, m	14.5	31	100
No. of panels	29306	10912	5120

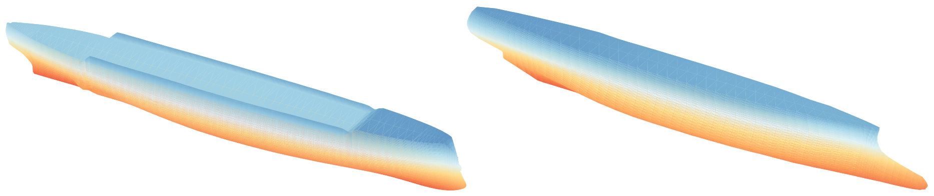

Fig. 2. Aurora and MICW three-dimensional ship hull models.

3.2 Benchmark results

The main result of the benchmarks is that Virtual testbed is capable of running on a regular workstation with or without a graphical accelerator in real-time with high frame rate and small simulation time steps.

– We achieved more than 60 simulation steps per second (SSPS) on each of the workstations. SSPS is the same metric as frames per second in visuliastion, but for simulation. For Storm and GPUlab the most performant programme version was the one for graphical accelerator and for Capybara the most performant version was the one for the processor (tab. 3).
– The most performant node is GPUlab with 104 simulation steps per second. Performance of Capybara is higher than of Storm, but it uses powerful server-grade processor to achieve it.
– Computational speedup for increasing number of parallel OpenMP threads is far from linear: we achieved only fourfold speedup for ten threads (fig. 3).
– Although, GPUlab's processor has higher frequency, even one core of Capybara's processor achieves slightly higher performance.
– The least powerful workstation (Storm) has the largest positive difference between graphical accelerator and processor performance (fig. 4). The most powerful workstation (Capybara) has comparable but negative difference.
– Usage of graphical accelerator increases time needed to synchronise simulation step with the visualisation frame (*exchange* stage in fig. 4).

4 Discussion

Although, graphical accelerator gives noticeable performance increase only for the least powerful workstation, we considered only the simplest simulation scenario (ship motions induced by a plane Stokes wave) in the benchmarks: The problem that we solve is too small to saturate graphical accelerator cores. We tried to eliminate expensive data copying operations between host and graphical accelerator memory, where possible, but we need to simulate more physical phenomena and at a larger scale (ships with large number of panels, large number of compartments, wind simulation etc.) to verify that performance gap increases

Table 3. Best median performance for each workstation and each ship hull. Here t is simulation step computation time, m — no. of simulation steps per second (SSPS), and n — the number of OpenMP threads, CL — OpenCL, MP — OpenMP.

	Sphere				Aurora				MICW			
Node	t, ms	m	n	ver.	t, ms	m	n	ver.	t, ms	m	n	ver.
Storm	16	64	1	CL	14	72	1	CL	29	34	1	CL
GPUlab	10	104	1	CL	9	112	1	CL	18	55	1	CL
Capybara	12	85	10	MP	15	66	10	MP	19	51	10	MP

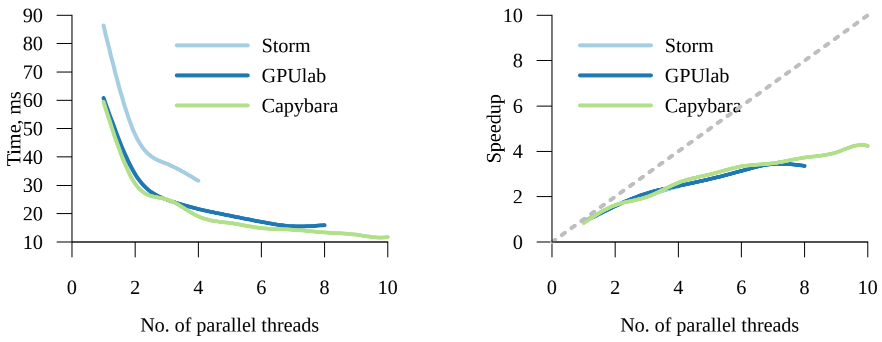

Fig. 3. Median simulation step computation time for different number of parallel threads (sphere).

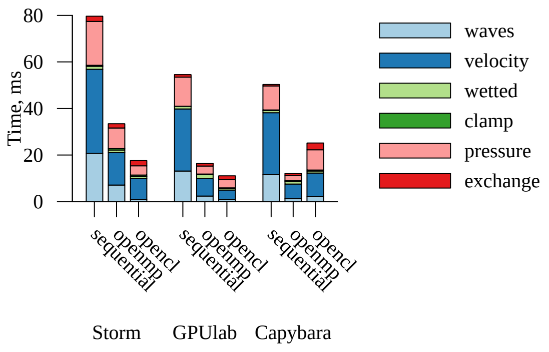

Fig. 4. Median computation time for each main loop stage, each node and sequential, OpenMP and OpenCL versions (sphere).

for powerful workstations. On the bright side, even if a computer does not have powerful graphical accelerator (e.g. a laptop with integrated graphics), it still can run Virtual testbed with acceptable performance.

Large SSPS is needed neither for smooth visualisation, nor for accurate simulation; however, it gives performance reserve for further increase in detail and scale of simulated physical phenomena. We manually limit simulation time step to a minimum of 1/30 of the second to prevent floating-point numerical errors due to small time steps. Also, we limit maximum time step to have wave frequency greater or equal to Nyquist frequency for precise partial time derivatives computation.

Real-time simulation is essential not only for educational purposes, but also for on-board intelligent systems. These systems analyse data coming from a multitude of sensors the ship equips, calculate probability of occurrence of a particular dangerous situation (e.g. large roll angle) and try to prevent it by notifying ship's crew and an operator on the coast. This is one of the directions of future work.

Overall performance depends on the size of the ship rather than the number of panels. MICW hull has less number of panels than Aurora, but two times larger size and two times worse performance (tab. 3). The size of the hull affects the size of the grid in each point of which velocity potential and then pressure is computed. These routines are much more compute intensive in comparison to wetted surface determination and pressure force computation, performance of which depends on the number of panels.

Despite the fact that Capybara has the highest floating-point performance across all workstations in the benchmarks, Virtual testbed runs faster on its processor, not the graphical accelerator. Routine-by-routine investigation showed that this graphics card is simply slower at computing even fully parallel Stokes wave generator OpenCL kernel. This kernel fills three-dimensional array using explicit formula for the wave profile, it has linear memory access pattern and no information dependencies between array elements. It seems, that P5000 is not optimised for general purpose computations. We did not conduct visualisation benchmarks, so we do not know if it is more efficient in that case.

Although, Capybara's processor has 20 hardware threads (2 threads per core), OpenMP performance does not scale beyond 10 threads. Parallel threads in our code do mostly the same operations but with different data, so switching between different hardware threads running on the same core in the hope that the second thread performs useful work while the first one stalls on input/output or load/store operation is not efficient. This problem is usually solved by creating a pipeline from the main loop in which each stage is executed in parallel and data constantly flows between subsequent stages. This approach is easy to implement when computational grid can be divided into distinct parts, which is not the case for Virtual testbed: there are too many dependencies between parts and the position and the size of each part can be different in each stage. Graphical accelerators have more efficient hardware threads switching which, and pipeline would probably not improve their performance, so we did not take this approach.

Our approach for performing computations on a heterogeneous node (a node with both a processor and a graphical accelerator) is similar to the approach followed by the authors of Spark distributed data processing framework [16]. In this framework data is first loaded into the main memory of each cluster node and then processed in a loop. Each iteration of this loop runs by all nodes in parallel and synchronisation occurs at the end of each iteration. This is in contrast to MapReduce framework [3] where after each iteration the data is written to stable storage and then read back into the main memory to continue processing. Not interacting with slow stable storage on every iteration allows Spark to achieve an order of magnitude higher performance than Hadoop (open-source version of MapReduce) on iterative algorithms.

For a heterogeneous node an analogue of stable storage, read/writes to which is much slower than accesses to the main memory, is graphical accelerator memory. To minimise interaction with this memory, we do not read intermediate results of our computations from it, but reuse arrays that already reside there. (As a concrete example, we do not copy pressure field from a graphical accelerator, only the forces for each panel.) This allows us to eliminate expensive data transfer between CPU and GPU memory. In early versions of our programme this copying slowed down simulation significantly.

Although, heterogeneous node is not a cluster, efficient programme architecture for such a node is similar to distributed data processing systems: we process data only on those device main memory of which contains the data and we never transfer intermediate computation results between devices. To implement this principle the whole iteration of the programme's main loop have to be executed either on a processor or a graphical accelerator. Given the time constraints, future maintenance burden and programme's code size, it was difficult to fully follow this approach, but we came to a reasonable approximation of it. We still have functions (*clamp* stage in fig. 4 that reduces the size of the computational grid to the points nearby the ship) in Virtual testbed that work with intermediate results on a processor, but the amount of data that is copied to and from a graphical accelerator is relatively small.

5 Conclusion

We showed that ship motion simulation can be performed on a regular workstation with or without graphical accelerator. Our programme includes only minimal number of mathematical models that allow ship motions calculation, but has performance reserve for inclusion of additional models. We plan to implement rudder and propeller, compartment flooding and fire, wind and trochoidal waves simulation. Apart from that, the main direction of future research is creation of on-board intelligent system that would include Virtual testbed as an integral part for simulating and predicting physical phenomena.

Acknowledgements. Research work is supported by Saint Petersburg State University (grant no. 26520170 and 39417213).

References

1. Bogdanov, A., Khramushin, V.: Vessel: Blueprints for the analysis of hydrostatic characteristics, stability and propulsion of the ship (in Russian) (2015), http://www1.fips.ru/fips_servl/fips_servlet?DB=EVM&DocNumber=2015621368&TypeFile=html
2. Cercos-Pita, J.L., Bulian, G., Pérez-Rojas, L., Francescutto, A.: Coupled simulation of nonlinear ship motions and a free surface tank. Ocean Engineering **120**, 281–288 (2016). https://doi.org/10.1016/j.oceaneng.2016.03.015
3. Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. Communications of the ACM **51**(1), 107–113 (Jan 2008). https://doi.org/10.1145/1327452.1327492
4. Gankevich, I.: Simulation modelling of irregular waves for marine object dynamics programmes. Ph.D. thesis, Saint Petersburg State University, Saint Petersburg, Russia (June 2018)
5. Gankevich, I., Degtyarev, A.: Simulation of Standing and Propagating Sea Waves with Three-Dimensional ARMA Model, pp. 249–278. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-71934-4_18
6. Keeler, T., Bridson, R.: Ocean waves animation using boundary integral equations and explicit mesh tracking. In: Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation. pp. 11–19. SCA'14, Eurographics Association, Aire-la-Ville, Switzerland (2014), http://dl.acm.org/citation.cfm?id=2849517.2849520
7. Khramushin, V.: Analytic ship hull shape construction, wave resistance calculations, theoretical blueprint feature curve calculations, and ship stability diagrams (in Russian) (2010), http://www1.fips.ru/fips_servl/fips_servlet?DB=EVM&DocNumber=2010615849&TypeFile=html
8. Mathews, J.H., Fink, K.D.: Numerical methods using MATLAB. Pearson Prentice Hall, London, 4th edn. (2004)
9. Matusiak, J.: Dynamics of a Rigid Ship. No. 11/2013 in SCIENCE + TECHNOLOGY, Aalto University; Aalto-yliopisto (2013), http://urn.fi/URN:ISBN:978-952-60-5205-2
10. Micikevicius, P.: 3D finite difference computation on GPUs using CUDA. In: Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units. pp. 79–84. GPGPU-2, ACM, New York, NY, USA (2009). https://doi.org/10.1145/1513895.1513905
11. Shin, Y., Belenky, V., Lin, W., Weems, K., Engle, A., McTaggart, K., Falzarano, J.M., Hutchison, B.L., Gerigk, M., Grochowalski, S.: Nonlinear time domain simulation technology for seakeeping and wave-load analysis for modern ship design. authors' closure. Transactions-Society of Naval Architects and Marine Engineers **111**, 557–583 (2003)
12. Ueng, S.K., Lin, D., Liu, C.H.: A ship motion simulation system. Virtual Reality **12**(1), 65–76 (Mar 2008). https://doi.org/10.1007/s10055-008-0088-8
13. Valiant, L.G.: A bridging model for parallel computation. Communications of the ACM **33**(8), 103–111 (Aug 1990). https://doi.org/10.1145/79173.79181

14. Varela, J.M., Soares, C.G.: Interactive simulation of ship motions in random seas based on real wave spectra. In: Proceedings of the International Conference on Computer Graphics Theory and Applications. pp. 235–244 (2011)
15. Volkov, V., Kazian, B.: Fitting FFT onto the G80 architecture. Tech. Rep. 6, University of California, Berkeley (May 2008)
16. Zaharia, M., Xin, R.S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M.J., Ghodsi, A., Gonzalez, J., Shenker, S., Stoica, I.: Apache spark: A unified engine for big data processing. Commun. ACM **59**(11), 56–65 (October 2016). https://doi.org/10.1145/2934664