

Testbed for the Extremal Waves Perturbations

Alexander V. Bogdanov(✉) and Vladimir V. Mareev(✉)

Faculty of Applied Mathematics and Control Processes, Saint-Petersburg State University, Petergof, 198504 Saint-Petersburg, Russia
{bogdanov,map}@csa.ru

Abstract. The problem of forming mesoscale ocean waves is important for shipping, especially in connection with the increase in the size of supertankers and ferries. Although much research has been done on the formation and propagation of such waves, the question of the effect of background waves on wave behavior is unclear. We consider a model problem of soliton perturbation in the regime of extreme growth by various background wave fields. Due to the high complexity of this task, to solve it, it was necessary to use a specialized software and hardware complex (testbed) with software adapted for a hybrid architecture and special measures to suppress artifacts in complex nonlinear systems. Due to the essentially nonlinear nature of the propagation of extreme waves, a strong influence of background wave fields on the main wave was found.

Keywords: Mesoscale waves · Digital models · Numerical methods · Testbed · Extreme conditions

1 Introduction

The problem of the propagation of two-dimensional nonlinear waves has long been in the focus of researchers' attention due to its numerous applications in shipbuilding. Recently, special attention has been paid to mesoscopic-scale waves due to the specific extreme behavior and the special significance that it has for extra-large ships (supertankers and ferries). The wavelength of such a wave is comparable to the size of a supertanker, and its extreme growth can lead to severe maritime accidents. Although recently it has been possible to understand the features of the formation and growth of such waves, their stability and the effect of external wave disturbances on them has not yet been studied.

In this communication, we consider a model problem of perturbing an extreme wave by a wave field of various types. Several attempts [1–7] to deal with these problems have shown that it is not possible to achieve the result by simple approaches to calculations. Instabilities, artifacts, and distortion of the wave picture appear. Therefore, it was decided to create a specialized software and hardware complex (testbed) for this task, on which the approaches developed by the authors were tested.

© Springer Nature Switzerland AG 2021
O. Gervasi et al. (Eds.): ICCSA 2021, LNCS 12956, pp. 486–495, 2021.
https://doi.org/10.1007/978-3-030-87010-2_36

2 The Problem Statement

Consider the Kadomtsev – Petviashvili I equation:

$$\left[u_t + 0.5\left(u^2\right)_x + \beta u_{xxx} - G\right]_x = \eta u_{yy} \tag{1}$$

Equation (1) for the function $u(x, y, t)$ is specified in the domain $x, y \in (-\infty, \infty), t \geq 0$.

Numerical modeling was carried out for the transformed form of the Eq. (1):

$$u_t + \frac{1}{2}\left(u^2\right)_x + \beta u_{xxx} = S(x, y, t) + G(x, y, t), \beta \geq 0 \tag{2}$$

$$S(x, y, t) = \eta \frac{\partial^2}{\partial y^2} \int_{-\infty}^{x} u\left(x', y, t\right) dx', \eta \geq 0$$

where the source $G(x, y, t)$ in (2) is the specified perturbation.

The initial distribution is taken as a Gaussian distribution:

$$u(x, y, 0) = \sigma \exp\left[-d\left(x^2/a^2 + y^2/b^2\right)\right], \sigma > 0 \tag{3}$$

With the volume of the perturbation $V = \sigma \pi ab/d$ and a, b being the semiaxes.

Basic information on the construction of a numerical method for Eq. (2) can be found in our works [8–15].

Numerical modeling of Eq. (1) is carried out in a finite computational domain $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}] \times [0, T]$. In the computational domain, a uniform space-time mesh is specified in coordinates x, y and time t:

$$x_j = j\Delta x, j \in [1, M], x_{\min} = x_1, x_{\max} = x_M$$

$$y_k = k\Delta y, k \in [1, L], y_{\min} = y_1, y_{\max} = y_L$$

$t^n = n\Delta t, n = 0, 1, 2, \ldots, T/\Delta t - 1$, with $\Delta x, \Delta y$ being the steps over the space coordinates, Δt being the time step.

Differential Eq. (2) is approximated on a given mesh and reduced to an implicit linearized finite-difference scheme. The resulting system of equations is solved using a five-point sweep.

The integral $S(x, y, t)$ is replaced by an approximate expression

$$\tilde{S}(x, y, t) = P(y) \cdot S(x, y, t),$$

with

$$P(y) = \begin{cases} 1, y \in [y_{\min} + \delta, y_{\max} - \delta] \\ h^-(y), y \in [y_{\min}, y_{\min} + \delta] \\ h^+(y), y \in [y_{\max} - \delta, y_{\max}] \end{cases} \tag{4}$$

The need to introduce such a procedure for smoothing the values of functions at the boundaries of the computational domain is discussed in detail in [13].

Finite smoothing on the segment δ is given in the form of a parabola

$$P(y) = ay^2 + by + c.$$

On the right side $h^-(y) = y^2$ with $y \in [y_{\min}, y_{\min} + \delta]$, and on the left side $h^+(y) = y^2 - 2y + 1$ with $y \in [y_{\max} - \delta, y_{\max}]$.

For the perturbation $G(x, y, t)$ at the edges of the region $x_{\min}$ and $x_{\max}$ similarly to expression (3) for the integral $\tilde{S}(x, y, t)$, the smoothing function is taken:

$$\tilde{G}(x, y, t) = P'(x) \cdot G(x, y, t),$$

где

$$P\prime(\mathrm{x}) = \begin{cases} 1, & x \in [x_{\min} + \delta\prime, x_{\min} - \delta\prime] \\ h\prime^-(\mathrm{x}), & x \in [x_{\min}, x_{\min} + \delta\prime] \\ h\prime^+(\mathrm{x}), & x \in [x_{\max} - \delta\prime, x_{\max}] \end{cases}, \tag{5}$$

Finite functions $P'(x)$, taking into account the direction of x, are similar to the case for the function $P(y)$ (4).

3 The Testbed

The performance of computation for such complex problem becomes crucial and due to matrix nature of internal loops it was decided to build a special testbed based on hybrid infrastructure.

The execution of a hybrid program consists of two parts: the host part of the program and the kernels that are executed on the GPU.

CPU AMD FX 8550 4 GHz
GPU AMD R9 380 970 Mhz

The host portion of the program defines the context in which the kernels are executed and controls their execution.

The main part of the execution model describes the execution of the kernels. When a kernel is queued for execution, an index space is defined. A copy of the kernel will be executed for each index from this space. A copy of the kernel running for a particular index is called a Work-Item or Thread and is identified by a point in the index space, that is, each "unit" is given a global identifier. Each unit executes the same code, but the specific path of execution (branches, etc.) and the data with which it operates may be different.

Work units are organized into groups (Work-Groups). Groups provide a larger partition in the index space. Each group is assigned a group ID with the same dimension that was used to address individual elements. Each element is associated with a unique, within the group, local identifier. In this way, work units can be addressed both by a global identifier and by a combination of a group and local identifier. There are slight differences between OpenCL and CUDA in how work units are organized and indexed, but these have no impact on the final functionality.

Units in a group are executed concurrently (in parallel) on the PE of one computing unit. The unified device model is clearly visible here: several PE ⇒ CU, several CU ⇒ device, several devices ⇒ heterogeneous system.

The working unit executing the kernel can use four different types of memory:

- *Global memory*. This memory provides read and write access to members of all groups. Each unit can write to and read from any part of the memory object. Writing and reading of global memory can be cached depending on the capabilities of the device.
- *Constant memory*. A region of global memory that remains constant during kernel execution. The host allocates and initializes memory objects located in constant memory.
- *Local memory*. A memory area local to the group. This memory area can be used to create variables that are shared by the entire group. It can be implemented as a separate memory on the device. Alternatively, this memory can be mapped as an area in global memory.
- *Private memory*. The memory area owned by the unit. Variables defined in the private memory of one unit are not visible to others

The specification defines 4 types of memory, but does not impose any requirements on the implementation of memory in the device itself. All 4 types of memory can be located in global memory, and the separation of types can be carried out at the driver level, and on the contrary, there can be a strict separation of memory types dictated by the architecture of the device. The existence of these types of memory is quite logical: the processor core has its own cache, the processor has a shared cache, and the entire device has a certain amount of memory.

3.1 Programming Model

The OpenCL and CUDA execution model supports two programming models: Data Parallel and Task Parallel, and hybrid models are also supported. The main model that drives the design of both platforms is data parallelism.

3.2 Programming Model with Data Parallelism

This model defines computation as a sequence of instructions applied to multiple elements of a memory object. The index space associated with the execution model defines work units and how data is distributed among them. In a strict data parallelism model, there is a strict one-to-one correspondence between a unit and an element in a memory object with which the kernel can run in parallel. OpenCL and CUDA implement a softer data parallelism model where strict one-to-one correspondence is not required.

Both platforms provide a hierarchical data parallelism model. There are two ways to define hierarchical division. In an explicit model, the programmer defines the total number of items to be executed in parallel and how these items will be grouped. In the implicit model, the programmer only determines the total number of elements that should be executed in parallel, and the division into work groups is done automatically.

3.3 Programming Model with Job Parallelism

In this model, each copy of the kernel runs independently of any index space. Logically, this is equivalent to the execution of the kernel on a computational unit with a group consisting of one element. In such a model, users express concurrency in the following ways:

- use vector data types implemented in the device;
- queue up a lot of jobs;
- queue kernels that use a programming model orthogonal to the platform.

The existence of two programming models is also a tribute to versatility. The first model is well suited for modern GPUs and Cells. But not all algorithms can be effectively implemented within the framework of such a model, and there is also the possibility of a device whose architecture will be inconvenient for using the first model. In this case, the second model allows you to write applications specific to a different architecture.

3.4 Algorithm Migration Process

Before proceeding to the direct implementation of the hybrid version of the considered algorithm, it is necessary to investigate its acceleration potential and identify areas for optimization specific to hybrid platforms.

Optimization of Memory Transfer. The biggest obstacle to using GPUs in computing is the need to transfer data to and from the device. This gives us an obvious vector for optimization. Leaving these tasks on the device for several iterations can greatly increase overall performance.

Optimization of Device Memory Usage. 7The memory on a GPU device is not homogeneous and consists of several regions. They vary in size and speed by many orders of magnitude, and their correct use can bring tangible performance gains. We are interested in Global Memory, Shared Memory and Registers.

Load Optimization. Before starting the kernel, you need to set the "run configuration", which determines the size and number of thread blocks. Previous work has shown that choosing the optimal configuration can bring a fourfold increase in performance without changing the source code of the program.

4 The Test Problem

When choosing the sources $G(x, y, t)$, we focused on cases associated with perturbations of the form of a traveling wave moving with the speed $c > 0$. For all variants, the initial distribution was set at the following values: $V = 120, \sigma = 10, a = 2, b = 3, d = 1.570796$, the values of the parameters of the difference scheme being: number of nodes $M \times L = 800 \times 600$, $\Delta t = 5 \cdot 10^{-5}$, $\Delta x = \Delta y = 0.1$, $\varepsilon = 0.01$. The parameter ε is the amplitude for artificial convection in the smoothing procedure [13].

For the testing purposes, the model size was limited to 960 by 640 cells, with 200 iterations, which equates to 0.01 s of simulation time. The main attention was paid on the inter-column parallelization, which was found surprisingly straightforward, with little code changes required. Five-point tridiagonal matrix algorithm implementation required buffer arrays for each row, which were allocated using private memory which for datasets of this size is automatically projected onto the global device memory, and as such does not require additional synchronization. This resulted in an expected increase of performance.

The last step was to improve the five-point tridiagonal matrix algorithm parallelization. In this case local memory would not help as there is simply not enough of it to contain all the required data. The solution was to create a single buffer matrix and utilize the comparatively large 4 Gb global memory space. Big memory usage is unpleasant, but acceptable if it solves a much larger problem of slow global memory access.

Each matrix element is only dependent on the two previous ones, which allows us to store these two values in faster memory. This way, at the first stage of calculations, each iteration writes the data into global memory but does not read it, which allows the optimizer to perform a much faster asynchronous write. For subsequent iterations, the same data is stored in private memory, which is fast for both write and read. At the second stage of calculations, the intermediary data is only read from global memory, which avoid the need for additional synchronization. There was a risk that the intermediary data would not fit into the private memory, since it is very limited, but in practice, there were no problems with it, and it provided a significant boost to performance [10].

Next step was to implement the inter-row parallelism, at least in the trivial cases, such as calculation of U_{yy} and U_y matrices, which do not feature accumulation. This had significantly improved calculation times. This is most likely due to the memory block read optimization.

Some test results are given in the Table 1.

Table 1. Benchmark results

Version	Time (s)
CPU	28.683
GPU	1.355
GPU async	1.338
GPU 2d	0.304
GPU 2d async	0.302

Where CPU is the original implementation, GPU is simple column parallelization, GPU async is similar to GPU, but with asynchronous calling for cores, GPU 2d is parallelization in both rows and columns.

You can see that even simple parallelization by columns gave almost a 20-fold increase in performance, while parallelization by rows and columns increased it up to 90-fold.

4.1 Exponential Wave

Perturbation

$$G(x, y, t) = A \exp\left[-\theta(x - x_w + vt)^2\right].$$

is a wave. Two cases are considered when $\theta = 0.1$. For these cas, the maximum for the exponent is obviously equal to the amplitude A. Case a): wave moves fm left to right at $v = -4$. The initial position of the wave at the moment $t = 0$ is reached at $x_w = -5.6$. Case b): wave moves from right to left at $v = +3$. The initial position of the wave at the moment $t = 0$ is reached at $x_w = 15$.

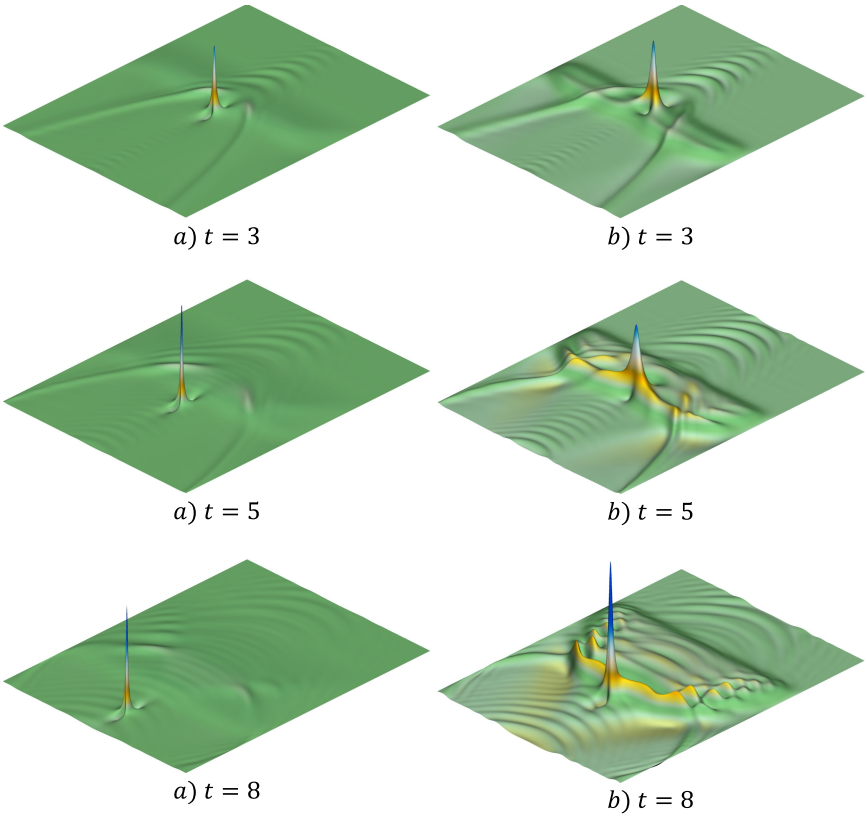


Fig. 1. Effect of an exponential wave formation of a soliton

To some extent, such a representation is like a single wave incident on a forming soliton.

Smoothing segments length δ from the system (4) and δ' from (5) is 10% of the length of $(x_{\max} - x_{\min})$ and $(y_{\max} - y_{\min})$, that is over x axis $\delta^{'} = 8$, and over y axis $\delta = 6$, correspondingly. Let us present successive graphs of such an incident wave for $A = 1$ (Fig. 1).

4.2 Harmonic Wave

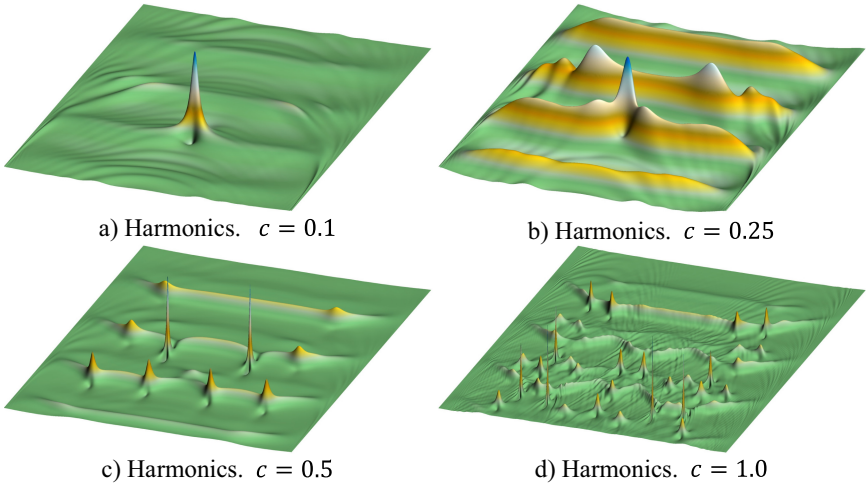
The perturbation form is represented as a harmonic:

$$G(x, y, t) = A \cos(\chi \cdot x - \omega \cdot t),$$

with A being the amplitude, $\varkappa$ being the wave number, $\omega = c \cdot \chi$, and c being the group speed.

Such a perturbation can be considered as a kind of ripple imposed on the forming soliton.

The length of the smoothing segments is 8% of the length of the sides of the computational domain, that is over x axis $\delta^{'} = 6.4$, and over y axis $\delta = 4.8$, correspondingly.



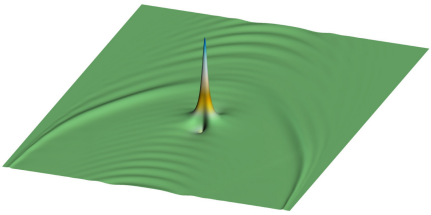
a) Harmonics. $c = 0.1$ b) Harmonics. $c = 0.25$

c) Harmonics. $c = 0.5$ d) Harmonics. $c = 1.0$

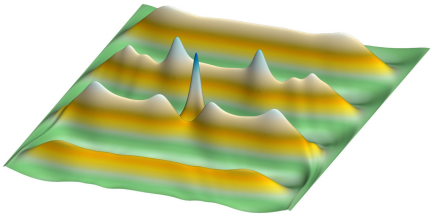
Fig. 2. Influence of harmonic perturbations on the destruction of a soliton

Let us present the results of calculations, fixing the amplitude $A = 1$ and changing the speed c. In this case, the wave number $\varkappa = \pi/10 = 0.3141592$. Figure 2 shows sequentially wave configurations for different velocities for the time moment $t = 8$.

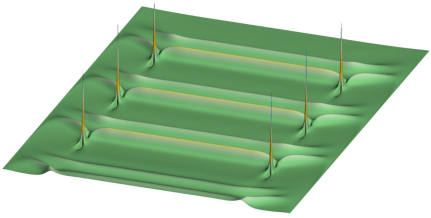
It is interesting to compare the influence of perturbations at $t = 6$ on the soliton (Fig. 3b) for two extreme cases: without perturbations (Fig. 3a), that is $G(x, y, t) \equiv 0$, and only with perturbations without initial distribution (Fig. 3c), that is $u(x, y, 0) = 0$ (3).



a) Without perturbation $G(x, y, t) \equiv 0$



b) Influence of perturbations on a soliton



c) Without soliton $u(x, y, 0) = 0$

Fig. 3. Comparison of different cases for $A = 1$, $c = 0.5$

5 Conclusion

The proposed hardware and software complex perfectly coped with the most difficult problems. The tests carried out have shown the stability of the system in different modes, reaching asymptotic and correct reproduction of wave effects. As for the physical results, research performed in this communication shows a strong influence of the background wave perturbation on the evolution of the extreme wave. Therefore, for real problems of the impact of extreme waves on ships, it is necessary to carefully consider the wave situation at sea.

Acknowledgement. The paper has been prepared within the scope of the project of St. Petersburg State University (id 51129503, 51129820).

References

1. Infeld, E., Rowlands, G.: Nonlinear Waves, Solitons and Chaos, 2nd edn., p. 406. Cambridge University Press, Cambridge (2000)
2. Rushchitsky, J.J.: Nonlinear ElasticWaves in Materials, p. 400. Springer International Publishing, Cham (2016)https://doi.org/10.1007/978-3-319-00464-8
3. Zeytounian, R.K.: Theory and Applications of Nonviscous Fluid Flows, p. 295.Springer Science & Business Media, Heidelberg (2012)https://doi.org/10.1007/978-3-642-56215-0
4. Kundu, A. (ed.) Tsunami and Nonlinear Wave, p. 316. Springer Science & Business Media, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71256-5
5. Fujimoto, M.: Introduction to the Mathematical Physics of Nonlinear Waves, p. 156. Morgan & Claypool Publishers, San Rafael (2014)
6. Yang, J.: Nonlinear Waves in Integrable and Non-integrable Systems, p. 430. Society for Industrial and Applied Mathematics, Philadelphia (2010)

7. Todorov, M.D.: Nonlinear Waves: Theory, Computer Simulation, Experiment, p. 184. Morgan & Claypool Publishers, San Rafael (2018)
8. Bogdanov, A., Mareev, V.: Numerical simulation perturbed KdVB equation. In: EPJ Web of Conferences, vol. 108, p. 02014 (2016). MMCP 2015, Stará Lesná (2015)
9. Bogdanov, A.V., Mareev, V.V.: Numerical simulation KPI equation. In: Proceedings of the 15th International Ship Stability Workshop, 13–15 June 2016, Stockholm, Sweden, pp. 115–117.
10. Bogdanov, A.V., Mareev, V.V.: Hybrid approach for nonlinear wave equation. In: AIP Conference Proceedings, vol. 1863, p. 110005 (2017). Published by the American Institute of Physics (ICNAAM 2016), Rhodes, 9–25 September 2016
11. Bogdanov, A.V., Mareev, V.V., Kulabukhova, N.V., Degtyarev, A.B., Shchegoleva, N.L.: Influence of external source on KPI equation. In: Gervasi, O., et al. (eds.) ICCSA 2018. LNCS, vol. 10963, pp. 123–135. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-95171-3_11
12. Bogdanov, A.V., Mareev, V.V., Storublevtcev, N.: Algorithms for the calculation of nonlinear processes on hybrid architecture clusters. In: Proceedings of the VIII International Conference "Distributed Computing and Grid-technologies in Science and Education" (GRID 2018), Dubna, Moscow region, Russia, 10–14 September, 2018, pp. 333–336
13. Bogdanov, A.V., Mareev, V.V.: On some problems of the numerical implementation of nonlinear systems on example of KPI equation. In: EPJ Web of Conferences, vol. 226, p. 01003 (2020). Mathematical Modeling and Computational Physics MMCP2019, Stara Lesha, 1–5 July 2019. https://doi.org/10.1051/epjconf/202022601003
14. Oran, E.S., Boris, J.P.: Numerical Simulation of Reactive Flow. Elsevier, Amsterdam (1987)
15. Feng, B.-F., Mitsui, T.: A finite difference method for the Korteweg-de Vries and the Kadomtsev-Petviashvili equations. J. Comput. Appl. Math. **90**(1), 95–116 (1998). https://doi.org/10.1016/S0377-0427(98)00006-5