Saint Petersburg State University

a manuscript

thesis for candidate of sciences degree

Simulation modelling of irregular waves for marine object dynamics programmes

Ivan Gennadevich Gankevich

Speciality 05.13.18
Mathematical modelling, numerical methods and programme complexes

Supervisor
Alexander Borisovich Degtyarev, ScD

St. Petersburg, 2017

Contents

1 Introduction

Topic relevance. Software programmes, which simulate ship behaviour in sea waves, are widely used to model ship motion, estimate impact of external forces on floating platform or other marine object, and estimate capsize probability under given weather conditions; however, to model sea waves most of them use linear wave theory [1–4], in the framework of which it is difficult to reproduce certain peculiarities of wind wave climate. Among them are transition between normal and storm weather, and sea composed of multiple wave systems — both wind waves and swell — heading from multiple directions. Another shortcoming of linear wave theory is an assumption, that wave amplitude is small compared to wave length. This makes calculations inaccurate when modelling ship motion in irregular waves, for which the assumption does not hold. So, studying new and more advanced models and methods for sea simulation software would increase number of its application scenarios and foster studying ship motion in extreme conditions in particular.

State of the art. Autoregressive moving average (ARMA) model is a response to difficulties encountered by practitioners who used wave simulation models developed in the framework of linear wave theory. The problems they have encountered with Longuet—Higgins model (a model which is entirely based on linear wave theory) are the following.

1. *Periodicity.* Linear wave theory approximates waves by a sum of harmonics, hence period of the whole wavy surface realisation depends on the number of harmonics in the model. The more realisation size is, the more coefficients are required to eliminate periodicity, therefore, generation time grows non-linearly with realisation size. This in turn results in overall low efficiency

of any model based on this theory, no matter how optimised the software implementation is.

2. *Linearity.* Linear wave theory gives mathematical definition for sea waves which have small amplitudes compared to their lengths. Waves of this type occur mostly in open ocean, so near-shore waves as well as storm waves, for which this assumption does not hold, are inaccurately modelled in the framework of linear theory.

3. *Probabilistic convergence.* Phase of a wave, which is generated by pseudo random number generator (PRNG), has uniform distribution, which leads to low convergence rate of wavy surface integral characteristics (average wave height, wave period, wave length etc.).

These difficulties became a starting point in search for a new model which is not based on linear wave theory, and ARMA process studies were found to have all the required mathematical apparatus.

1. ARMA process takes auto-covariate function (ACF) as an input parameter, and this function can be directly obtained from wave energy or frequency-directional spectrum (which is the input parameter for Longuet—Higgins model). So, inputs for one model can easily be converted to each other.

2. There is no small-amplitude waves assumption. Wave may have any amplitude, and can be generated as steep as it is possible with real sea wave ACF.

3. Period of the realisation equals the period of PRNG, so generation time grows linearly with the realisation size.

4. White noise — the only probabilistic term in ARMA process — has Gaussian distribution, hence convergence rate is not probabilistic.

Goals and objectives. ARMA process is the basis for ARMA sea simulation model, but due to its non-physical nature the model needs to be adapted to be used for wavy surface generation.

1. Investigate how different ACF shapes affect the choice of ARMA parameters (the number of moving average and autoregressive processes coefficients).
2. Investigate a possibility to generate waves of arbitrary profiles, not only cosines (which means taking into account asymmetric distribution of wavy surface elevation).
3. Develop a method to determine pressure field under discretely given wavy surface. Usually, such formulae are derived for a particular model by substituting wave profile formula into the system of equations for pressure determination (1), however, ARMA process does not provide explicit wave profile formula, so this problem has to be solved for general wavy surface (which is not defined by a mathematical formula), without linearisation of boundaries and assumption of small-amplitude waves.
4. Verify that wavy surface integral characteristics match the ones of real sea waves.
5. Develop software programme that implements ARMA model and pressure calculation method, and allows to run simulations on both shared and distributed memory computer systems.

Scientific novelty. ARMA model, as opposed to other sea simulation models, does not use linear wave theory. This makes it capable of

- generating waves with arbitrary amplitudes by adjusting wave steepness via ACF;
- generating waves with arbitrary profiles by adjusting asymmetry of wave elevation distribution via non-linear inertialess transform (NIT).

This makes it possible to use ARMA process to model transition between normal and storm weather taking into account climate spectra and assimilation data of a particular ocean region, which is not possible with models based on linear wave theory.

The distinct feature of this work is

- the use of *three-dimensional* AR and MA models in all experiments,
- the use of velocity potential field calculation method suitable for discretely given wavy surface, and
- the development of software programme that implements sea wavy surface generation and pressure field computation on both shared memory (SMP) and distributed memory (MPP) computer systems, but also hybrid systems (GPGPU) which use graphical coprocessors to accelerate computations.

Theoretical and practical significance. Implementing ARMA model, that does not use assumptions of linear wave theory, will increase quality of ship motion and marine object behaviour simulation software.

1. Since pressure field method is developed for discrete wavy surface and without assumptions about wave amplitudes, it is applicable to any wavy surface of incompressible inviscid fluid (in particular, it is applicable to wavy surface generated by LH model). This allows to use this method without being tied to ARMA model.
2. From computational point of view this method is more efficient than the corresponding method for LH model, because integrals in its formula are reduced to Fourier transforms, for which there is fast Fourier transform (FFT) family of algorithms, optimised for different processor architectures.

3. Since the formula which is used in the method is explicit, there is no need in data exchange between parallel processes, which allows to scale performance to a large number of processor cores.

4. Finally, ARMA model is itself more efficient than LH model due to absence of trigonometric functions in its formula: In fact, wavy surface is computed as a sum of large number of polynomials, for which there is low-level FMA (Fused Multiply-Add) assembly instruction, and memory access pattern allows for efficient use of CPU cache.

Methodology and research methods. Software implementation of ARMA model and pressure field calculation method was created incrementally: a prototype written in high-level engineering languages (Mathematica [5] and Octave [6]) was rewritten in lower level language (C++). Due to usage of different abstractions and language primitives, implementation of the same algorithms and methods in languages of varying levels allows to correct errors, which would left unnoticed, if only one language was used. Wavy surface, generated by ARMA model, as well as all input parameters (ACF, distribution of wave elevation etc.) were inspected via graphical means built into the programming language.

Theses for the defence.

- Sea wave simulation model which allows to generate wavy surface realisations with large period and consisting of waves of arbitrary amplitudes;

- Pressure field calculation method derived for this model without assumptions of linear wave theory;

- Software implementation of the simulation model and the method for shared and distributed memory computer systems.

Results verification and approbation. ARMA model is verified by comparing generated wavy surface integral characteristics (distribution of wave elevation, wave heights and lengths etc.) to the ones of real sea waves. Pressure field calculation method was developed using Mathematica language, where resulting formulae are verified by built-in graphical means.

ARMA model and pressure field calculation method were incorporated into Large Amplitude Motion Programme (LAMP) — an ship motion simulation software programme — where they were compared to previously used LH model. Numerical experiments showed higher computational efficiency of ARMA model.

2 Problem statement

The aim of the study reported here is to apply ARMA process mathematical apparatus to sea wave modelling and to develop a method to compute pressure field under discretely given wavy surface for inviscid incompressible fluid without assumptions of linear wave theory.

- In case of small-amplitude waves the resulting pressure field must correspond to the one obtained via formula from linear wave theory; in all other cases field values must not tend to infinity.
- Integral characteristics of generated wavy surface must match the ones of real sea waves.
- Software implementation of ARMA model and pressure field computation method must work on shared and distributed memory systems.

Pressure field formula. The problem of finding pressure field under wavy sea surface represents inverse problem of hydrodynamics for incompressible inviscid fluid. System of equations for it in general case is written as [7]

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

where ϕ is velocity potential, ζ — elevation (z coordinate) of wavy surface, p — wave pressure, ρ — fluid density, $\vec{v} = (\phi_x, \phi_y, \phi_z)$ — velocity vector, g — acceleration of gravity, and D — substantial (Lagrange) derivative. The first equation is called continuity (Laplace) equation, the second one is the conservation of momentum law (the so called dynamic boundary condition); the third one is kinematic boundary condition for free wavy surface, which states that a rate of change of wavy surface elevation ($D\zeta$) equals to the change of velocity potential derivative in the direction of wavy surface normal ($\nabla\phi \cdot \vec{n}$, see section 10.2).

Inverse problem of hydrodynamics consists in solving this system of equations for ϕ. In this formulation dynamic boundary condition becomes explicit formula to determine pressure field using velocity potential derivatives obtained from the remaining equations. From mathematical point of view inverse problem of hydrodynamics reduces to Laplace equation with mixed boundary condition — Robin problem.

Inverse problem is feasible because ARMA model generate hydrodynamically adequate sea wavy surface: distributions of integral characteristics and dispersion relation match the ones of real waves [8, 9].

3 ARMA model for sea wave simulation

3.1 Sea wave models analysis

Pressure computation is only possible when the shape of wavy surface is known. It is defined either at discrete grid points, or continuously via some analytic formula. As will be shown in section 4.1, such formula may simplify pressure computation by effectively reducing the task to pressure field generation, instead of wavy surface generation.

Longuet—Higgins model. The simplest model, formula of which is derived in the framework of linear wave theory (see sec. 10.1), is Longuet—Higgins (LH) model [10]. In-depth comparative analysis of this model and ARMA model is done in [8, 9].

LH model represents sea wavy surface as a superposition of sine waves with random amplitudes c_n and phases ϵ_n, uniformly distributed on interval $[0, 2\pi]$. Wavy surface elevation (z coordinate) is defined by

$$\zeta(x,y,t) = \sum_n c_n \cos(u_n x + v_n y - \omega_n t + \epsilon_n). \tag{2}$$

Here wave numbers (u_n, v_n) are continuously distributed on plane (u, v), i.e. area $du \times dv$ contains infinite quantity of wave numbers. Frequency is related to wave numbers via dispersion relation $\omega_n = \omega(u_n, v_n)$. Function $\zeta(x, y, t)$ is a three-dimensional ergodic stationary homogeneous Gaussian process defined by

$$2E_\zeta(u,v)\,du\,dv = \sum_n c_n^2,$$

where $E_{\zeta}(u,v)$ — two-dimensional wave energy spectral density. Coefficients c_n are derived from wave energy spectrum $S(\omega)$ via

$$c_n = \sqrt{\int\limits_{\omega_n}^{\omega_{n+1}} S(\omega) d\omega}.$$

Disadvantages of Longuet-Higgins model. Despite simplicity of LH model numerical algorithm, in practice it has several disadvantages.

1. The model simulates only stationary Gaussian field. This is a consequence of central limit theorem (CLT): sum of large number of sines with random amplitudes and phases has normal distribution, no matter what spectrum is used as the model input. Using lower number of coefficients may solve the problem, but also make realisation period smaller. Using LH model to simulate waves with non-Gaussian distribution of elevation — a distribution which real sea waves have [11,12] — is impractical.

2. From computational point of view, the deficiency of the model is non-linear increase of wavy surface generation time with the increase of realisation size. The larger the size of the realisation, the higher number of coefficients (discrete points of frequency-directional spectrum) is needed to eliminate periodicity. This makes LH model inefficient for long-time simulations.

3. Finally, there are peculiarities which make LH model unsuitable as a base for building more advanced simulation models.

   - In software implementation convergence rate of (2) is low due to uniform distribution of phases ϵ_n.

   - It is difficult to generalise LH model for non-Gaussian processes as it involves incorporating non-linear terms in (2) for which there is no known formula to determine coefficients [13].

To summarise, LH model is applicable to generating sea wavy surface only in the framework of linear wave theory, inefficient for long-time simulations, and has a number of deficiencies which do not allow to use it as a base for more advanced models.

ARMA model. In [14] ARMA model is used to generate time series spectrum of which is compatible with Pierson—Moskowitz (PM) spectrum. The authors carry out experiments for one-dimensional AR, MA and ARMA models. They mention excellent agreement between target and initial spectra and higher performance of ARMA model compared to models based on summing large number of harmonic components with random phases. The also mention that in order to reach agreement between target and initial spectrum MA model require lesser number of coefficients than AR model. In [15] the authors generalise ARMA model coefficients determination formulae for multi-variate (vector) case.

One thing that distinguishes present work with respect to afore-mentioned ones is the study of three-dimensional (2D in space and 1D in time) ARMA model, which is mostly a different problem.

1. Yule—Walker system of equations, which are used to determine AR coefficients, has complex block-block structure.

2. Optimal model order (in a sense that target spectrum agrees with initial) is determined manually.

3. Instead of PM spectrum, analytic formulae for standing and propagating waves ACF are used as the model input.

4. Three-dimensional wavy surface should be compatible with real sea surface not only in terms of spectral characteristics, but also in the shape of wave profiles. So, model verification includes distributions of various parameters of generated waves (lengths, heights, periods etc.).

Multi-dimensionality of investigated model not only complexifies the task, but also allows to carry out visual validation of generated wavy surface. It is the opportunity to visualise output of the programme that allows to ensure that generated surface is compatible with real sea surface, and is not abstract multi-dimensional stochastic process that corresponds to the real one only statistically.

In [16] AR model is used to predict swell waves to control wave-energy converters (WEC) in real-time. In order to make WEC more efficient its internal oscillator frequency should match the one of sea waves. The authors treat wave elevation as time series and compare performance of AR model, neural networks and cyclical models in forecasting time series future values. AR model gives the most accurate prediction of low-frequency swell waves for up to two typical wave periods. It is an example of successful application of AR process to sea wave modelling.

3.2 Governing equations for 3-dimensional ARMA process

ARMA sea simulation model defines sea wavy surface as three-dimensional (two dimensions in space and one in time) autoregressive moving average process: every surface point is represented as a weighted sum of previous in time and space points plus weighted sum of previous in time and space normally distributed random impulses. The governing equation for 3-D ARMA process is

$$\zeta_{\vec{i}} = \sum_{\vec{j}=\vec{0}}^{\vec{N}} \Phi_{\vec{j}} \zeta_{\vec{i}-\vec{j}} + \sum_{\vec{j}=\vec{0}}^{\vec{M}} \Theta_{\vec{j}} \epsilon_{\vec{i}-\vec{j}}, \tag{3}$$

where ζ — wave elevation, Φ — AR process coefficients, Θ — MA process coefficients, ϵ — white noise with Gaussian distribution, $\vec{N}$ — AR process order, $\vec{M}$ — MA process order, and $\Phi_{\vec{0}} \equiv 0$, $\Theta_{\vec{0}} \equiv 0$. Here arrows denote multi-component indices with a component for each dimension. In general, any scalar quantity can be a component (temperature, salinity, concentration of some substance in water etc.). Equation parameters are AR and MA process coefficients and order.

Sea simulation model is used to simulate realistic realisations of wind induced wave field and is suitable to use in ship dynamics calculations. Stationarity and invertibility properties are the main criteria for choosing one or another process to simulate waves with different profiles, which are discussed in section 3.2.

Autoregressive (AR) process. AR process is ARMA process with only one random impulse instead of theirs weighted sum:

$$\zeta_{\vec{i}} = \sum_{\vec{j}=\vec{0}}^{\vec{N}} \Phi_{\vec{j}} \zeta_{\vec{i}-\vec{j}} + \epsilon_{i,j,k}. \tag{4}$$

The coefficients Φ are calculated from ACF via three-dimensional Yule—Walker (YW) equations, which are obtained after multiplying both parts of the previous equation by $\zeta_{\vec{i}-\vec{k}}$ and computing the expected value. Generic form of YW equations is

$$\gamma_{\vec{k}} = \sum_{\vec{j}=\vec{0}}^{\vec{N}} \Phi_{\vec{j}}\, \gamma_{\vec{k}-\vec{j}} + \sigma_\epsilon^2 \delta_{\vec{k}}, \qquad \delta_{\vec{k}} = \begin{cases} 1, & \text{if } \vec{k} = 0 \\ 0, & \text{if } \vec{k} \neq 0, \end{cases} \tag{5}$$

where γ — ACF of process ζ, σ_ϵ^2 — white noise variance. Matrix form of three-dimensional YW equations, which is used in the present work, is

$$\Gamma \begin{bmatrix} \Phi_{\vec{0}} \\ \Phi_{0,0,1} \\ \vdots \\ \Phi_{\vec{N}} \end{bmatrix} = \begin{bmatrix} \gamma_{0,0,0} - \sigma_\epsilon^2 \\ \gamma_{0,0,1} \\ \vdots \\ \gamma_{\vec{N}} \end{bmatrix}, \qquad \Gamma = \begin{bmatrix} \Gamma_0 & \Gamma_1 & \cdots & \Gamma_{N_1} \\ \Gamma_1 & \Gamma_0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \Gamma_1 \\ \Gamma_{N_1} & \cdots & \Gamma_1 & \Gamma_0 \end{bmatrix},$$

where $\vec{N} = (p_1, p_2, p_3)$ and

$$\Gamma_i = \begin{bmatrix} \Gamma_i^0 & \Gamma_i^1 & \cdots & \Gamma_i^{N_2} \\ \Gamma_i^1 & \Gamma_i^0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \Gamma_i^1 \\ \Gamma_i^{N_2} & \cdots & \Gamma_i^1 & \Gamma_i^0 \end{bmatrix} \qquad \Gamma_i^j = \begin{bmatrix} \gamma_{i,j,0} & \gamma_{i,j,1} & \cdots & \gamma_{i,j,N_3} \\ \gamma_{i,j,1} & \gamma_{i,j,0} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \gamma_{i,j,1} \\ \gamma_{i,j,N_3} & \cdots & \gamma_{i,j,1} & \gamma_{i,j,0} \end{bmatrix},$$

Since $\Phi_{\vec{0}} \equiv 0$, the first row and column of Γ can be eliminated. Matrix Γ is block-block-toeplitz, positive definite and symmetric, hence the system is efficiently solved by Cholesky decomposition, which is particularly suitable for these types of matrices.

After solving this system of equations white noise variance is estimated from (5) by plugging $\vec{k} = \vec{0}$:

$$\sigma_\epsilon^2 = \sigma_\zeta^2 - \sum_{\vec{j}=\vec{0}}^{\vec{N}} \Phi_{\vec{j}} \, \gamma_{\vec{j}}.$$

Moving average (MA) process. MA process is ARMA process with $\Phi \equiv 0$:

$$\zeta_{\vec{i}} = \sum_{\vec{j}=\vec{0}}^{\vec{M}} \Theta_{\vec{j}} \epsilon_{\vec{i}-\vec{j}}. \tag{6}$$

MA coefficients Θ are defined implicitly via the following non-linear system of equations:

$$\gamma_{\vec{i}} = \left[\sum_{\vec{j}=\vec{i}}^{\vec{M}} \Theta_{\vec{j}} \Theta_{\vec{j}-\vec{i}} \right] \sigma_\epsilon^2.$$

The system is solved numerically by fixed-point iteration method via the following formulae

$$\Theta_{\vec{i}} = -\frac{\gamma_{\vec{0}}}{\sigma_\epsilon^2} + \sum_{\vec{j}=\vec{i}}^{\vec{M}} \Theta_{\vec{j}} \Theta_{\vec{j}-\vec{i}}.$$

Here coefficients Θ are calculated from back to front: from $\vec{i} = \vec{M}$ to $\vec{i} = \vec{0}$. White noise variance is estimated by

$$\sigma_\epsilon^2 = \frac{\gamma_{\vec{0}}}{1 + \sum\limits_{\vec{j}=\vec{0}}^{\vec{M}} \Theta_{\vec{j}}^2}.$$

Authors of [17] suggest using Newton—Raphson method to solve this equation with higher precision, however, in this case it is difficult to generalise the method for three dimensions. Using slower method does not have dramatic effect on the overall programme performance, because the number of coefficients is small and most of the time is spent generating wavy surface.

Stationarity and invertibility of AR and MA processes. In order for modelled wavy surface to represent physical phenomena, the corresponding process must be stationary and invertible. If the process is *invertible*, then there is a reasonable connection of current events with the events in the past, and if the process is *stationary*, the modelled physical signal amplitude does not increase infinitely in time and space.

AR process is always invertible, and for stationarity it is necessary for roots of characteristic equation

$$1 - \Phi_{0,0,1} z - \Phi_{0,0,2} z^2 - \cdots - \Phi_{\vec{N}} z^{N_0 N_1 N_2} = 0,$$

to lie *outside* the unit circle. Here $\vec{N}$ is AR process order and Φ are coefficients.

MA process is always stationary, and for invertibility it is necessary for roots of characteristic equation

$$1 - \Theta_{0,0,1} z - \Theta_{0,0,2} z^2 - \cdots - \Theta_{\vec{M}} z^{M_0 M_1 M_2} = 0,$$

to lie *outside* the unit circle. Here $\vec{M}$ is three-dimensional MA process order and Θ are coefficients.

Stationarity and invertibility properties are the main criteria in selection of the process to model different wave profiles, which are discussed in section 3.2.

Mixed autoregressive moving average (ARMA) process. Generally speaking, ARMA process is obtained by plugging MA generated wavy surface as random impulse to AR process, however, in order to get the process with desired ACF one should re-compute AR coefficients before plugging. There are several approaches to “mix” AR and MA processes.

- The approach proposed in [17] which involves dividing ACF into MA and AR part along each dimension is not applicable here, because in three dimensions such division is not possible: there always be parts of the ACF that are not taken into account by AR and MA process.

- The alternative approach is to use the same (undivided) ACF for both AR and MA processes but use different process order, however, then realisation characteristics (mean, variance etc.) become skewed: these are characteristics of the two overlapped processes.

For the first approach the authors of [17] propose a formula to correct AR process coefficients, but there is no such formula for the second approach. So, the only proven solution for now is to simply use AR and MA process exclusively.

Process selection criteria. One problem of ARMA model application to sea wave generation is that for different types of wave profiles different processes *must* be used: standing waves are modelled by AR process, and propagating waves by MA process. This statement comes from practice: if one tries to use the processes the other way round, the resulting realisation either diverges or does not correspond to real sea waves. (The latter happens for non-invertible MA process, as it is always stationary.) So, the best way to apply ARMA model to sea wave generation is to use AR process for standing waves and MA process for progressive waves.

The other problem is difficulty of determination of optimal number of coefficients for three-dimensional AR and MA processes. For one-dimensional processes there are easy to implement iterative methods [17], but for three-dimensional case there are analogous methods [18, 19] which are difficult to implement, hence they were not used in this work. Manual choice of model's order is trivial: choosing knowingly higher order than needed affects performance, but not realisation quality, because processes' periodicity does not depend on the number of coefficients. Software implementation of iterative methods of finding the optimal model order would improve automation level of wavy surface generator, but not the quality of the experiments being conducted.

In practice some statements made for AR and MA processes in [17] should be flipped for three-dimensional case. For example, the authors say that ACF of MA process cuts at q and ACF of AR process decays to nought infinitely, but in practice making ACF of 3-dimensional MA process not decay results in it being non-invertible and producing realisation that does not look like real sea waves, whereas doing the same for ACF of AR process results in stationary process and adequate realisation. Also, the authors say that one should allocate the first q points of ACF to MA process (as it often needed to describe the peaks in ACF) and leave the rest points to AR process, but in practice in case of ACF of a propagating wave AR process is stationary only for the first time slice of the ACF, and the rest is left to MA process.

To summarise, the only established scenario of applying ARMA model to sea wave generation is to use AR process for standing waves and MA process for propagating waves. Using mixed ARMA process for both types of waves might increase model precision, given that coefficient correction formulae for three dimensions become available, which is the objective of the future research.

Verification of wavy surface integral characteristics. In [8, 9, 20] AR model the following items are verified experimentally:

- probability distributions of different wave characteristics (heights, lengths, crests, periods, slopes, three-dimensionality),
- dispersion relation,
- retention of integral characteristics for mixed wave sea state.

In this work both AR and MA model are verified by comparing probability distributions of different wave characteristics.

In [13] the authors show that several sea wave characteristics (listed in table 1) have Weibull distribution, and wavy surface elevation has Gaussian distribution. In order to verify that distributions corresponding to generated realisation are correct, quantile-quantile plots are used (plots where analytic quantile values are used for OX axis and estimated quantile values for OY axis). If the estimated distribution matches analytic then the graph has the form of the straight line. Tails of the graph may diverge from the straight line, because they can not be reliably estimated from the finite-size realisation.

Characteristic	Weibull shape (k)
Wave height	2
Wave length	2.3
Crest length	2.3
Wave period	3
Wave slope	2.5
Three-dimensionality	2.5

Table 1: Values of Weibull shape parameter for different wave characteristics.

Verification was performed for standing and propagating waves. The corresponding ACFs and quantile-quantile plots of wave characteristics distributions are shown in fig. 1, 2, 3.

Graph tails in fig. 1 deviate from original distribution for individual wave characteristics, because every wave have to be extracted from the resulting wavy surface to measure its length, period and height. There is no algorithm that guar-

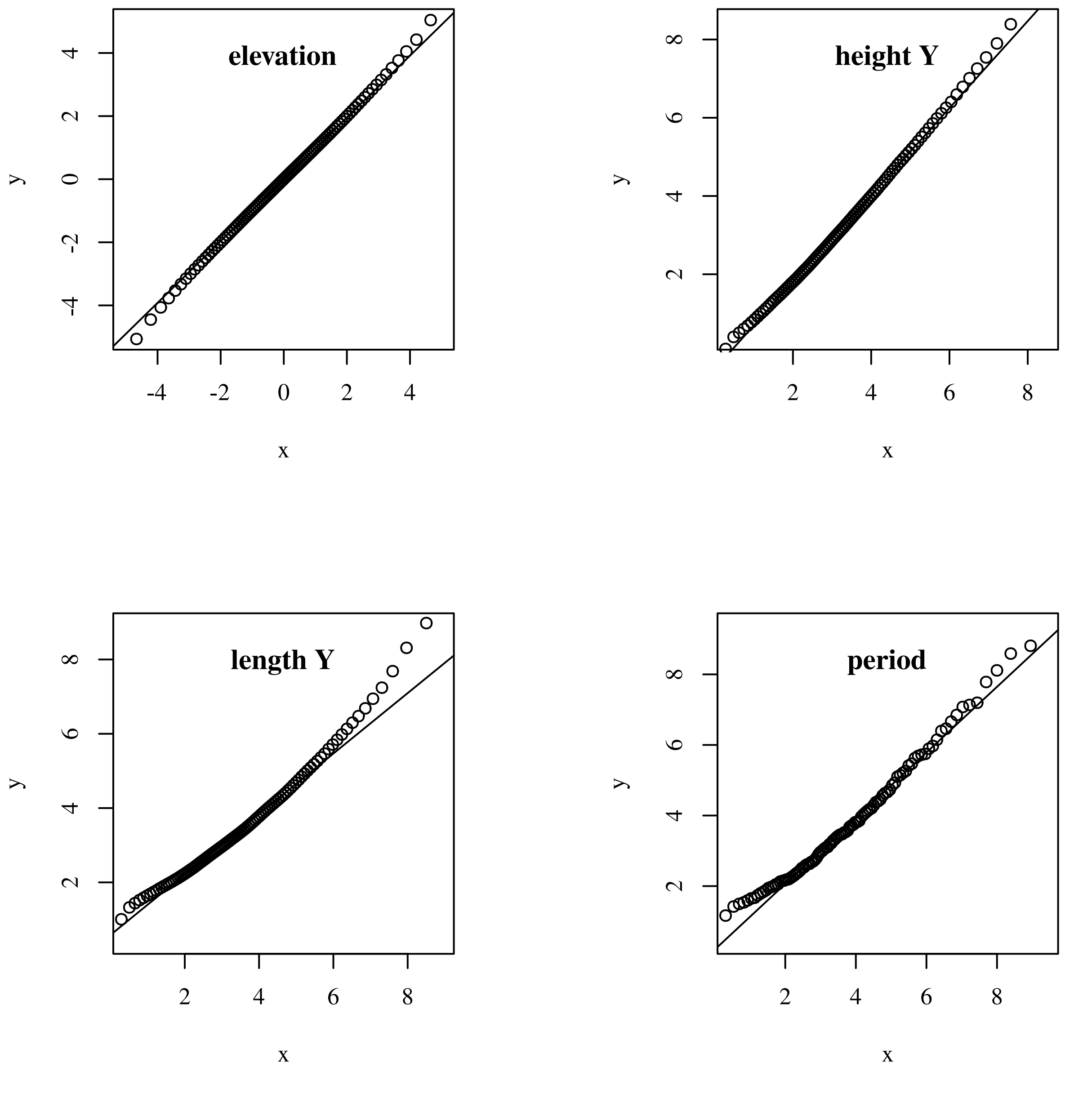


Figure 1: Quantile-quantile plots for propagating waves.

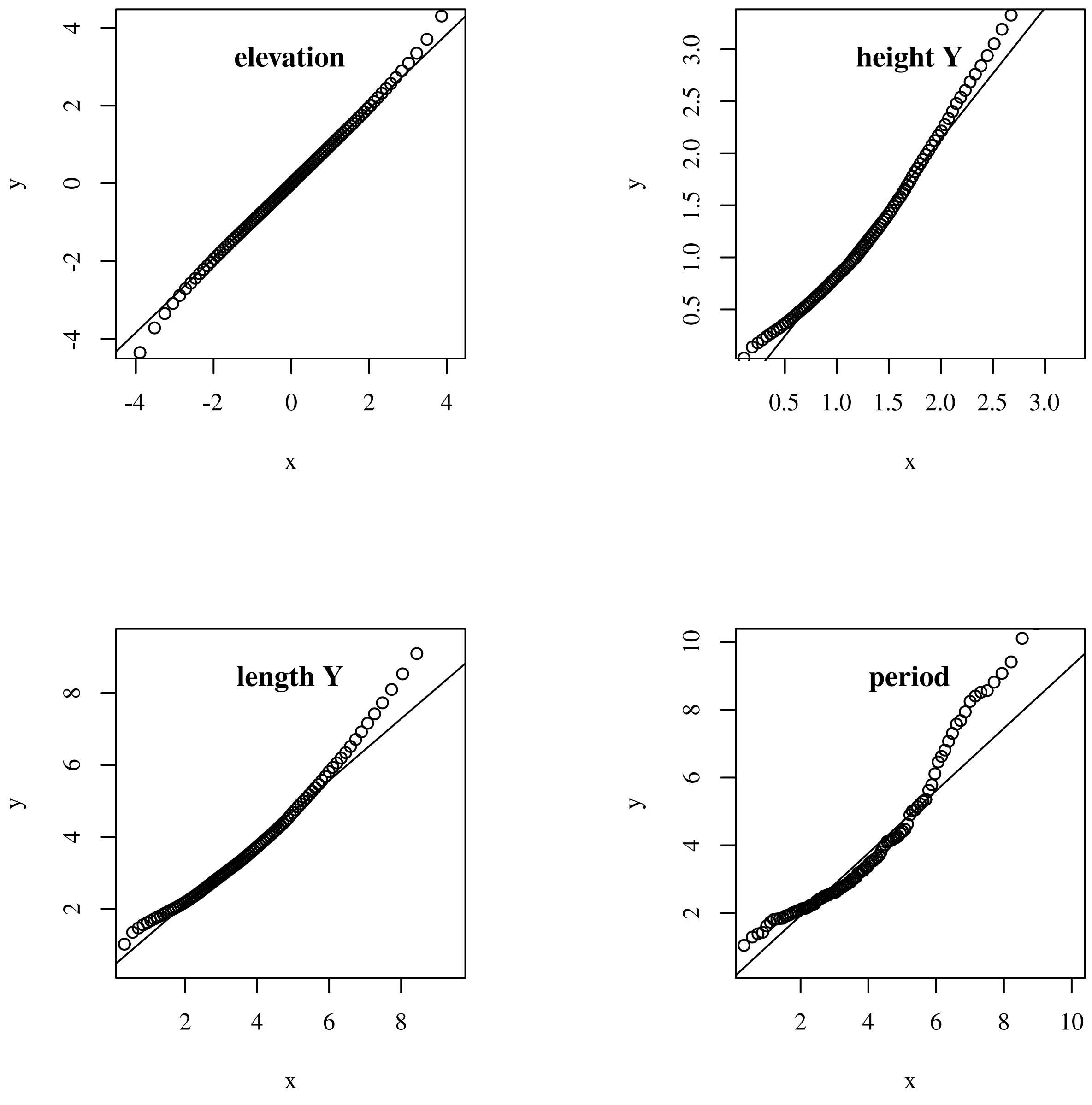


Figure 2: Quantile-quantile plots for standing waves.

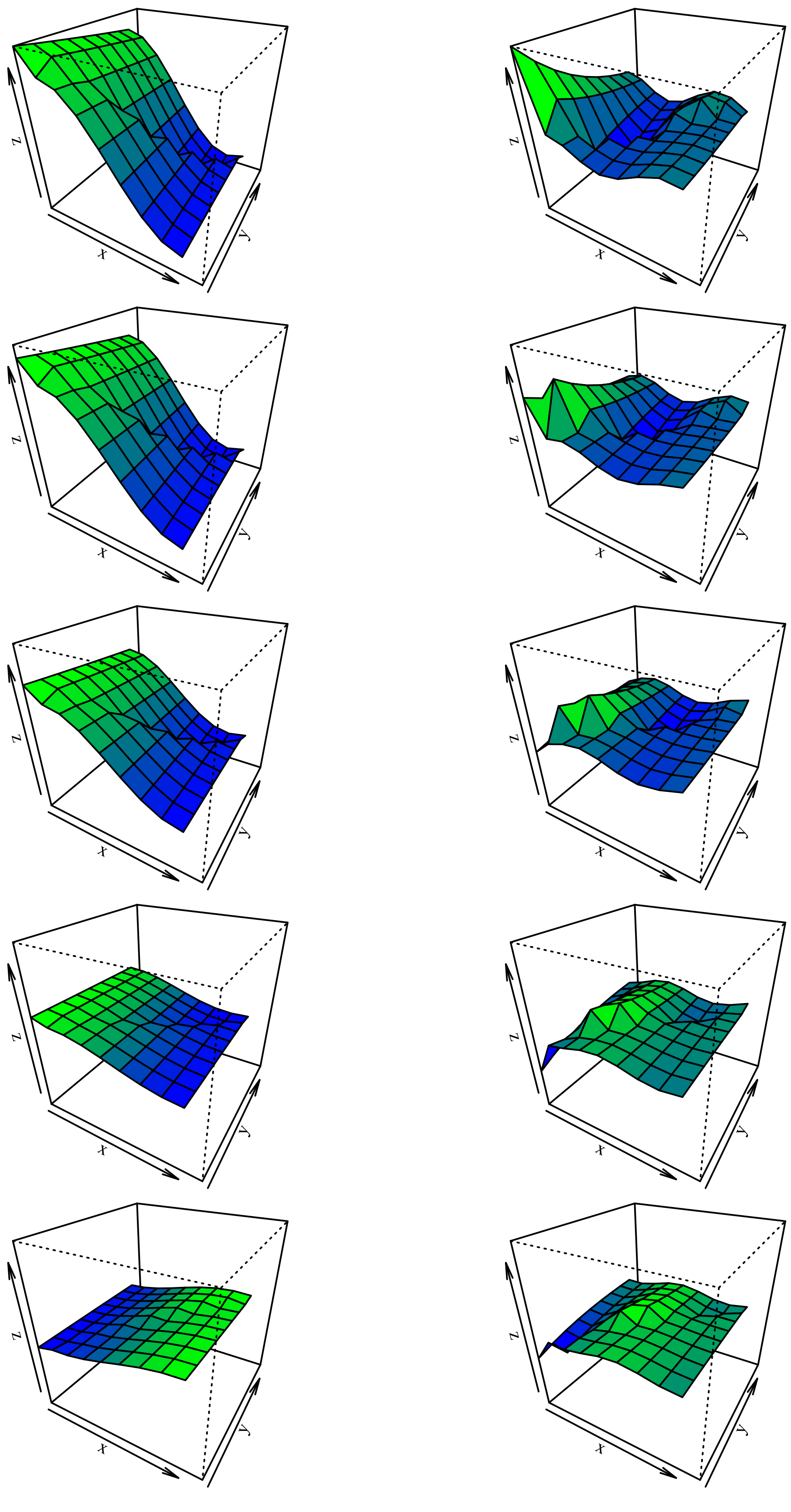


Figure 3: ACF time slices for standing (left column) and propagating waves (right column).

antees correct extraction of all waves, because they may and often overlap each other. Weibull distribution right tail represents infrequently occurring waves, so it deviates more than left tail.

Correspondence rate for standing waves (fig. 2) is lower for height and length, roughly the same for surface elevation and higher for wave period distribution tails. Lower correspondence degree for length and height may be attributed to the fact that Weibull distributions were obtained empirically for sea waves which are typically propagating, and distributions may be different for standings waves. Higher correspondence degree for wave periods is attributed to the fact that wave periods of standing waves are extracted more precisely as the waves do not move outside simulated wavy surface region. The same correspondence degree for wave elevation is obtained, because this is the characteristic of the wavy surface (and corresponding AR or MA process) and is not affected by the type of waves.

To summarise, software implementation of AR and MA models (which is described in detail in sec. 5) generates wavy surface, distributions of individual waves' characteristics of which correspond to *in situ* data.

3.3 Modelling non-linearity of sea waves

ARMA model allows to model asymmetry of wave elevation distribution, i.e. generate sea waves, distribution of z-coordinate of which has non-nought kurtosis and asymmetry. Such distribution is inherent to real sea waves [21] and given by either polynomial approximation of *in situ* data or analytic formula. Wave asymmetry is modelled by non-linear inertialess transform (NIT) of stochastic process, however, transforming resulting wavy surface means transforming initial

ACF. In order to alleviate this, ACF must be preliminary transformed as shown in [8].

Wavy surface transformation. Explicit formula $z = f(y)$ that transforms wavy surface to desired one-dimensional distribution $F(z)$ is the solution of non-linear transcendental equation $F(z) = \Phi(y)$, where $\Phi(y)$ — one-dimensional Gaussian distribution. Since distribution of wave elevation is often given by some approximation based on *in situ* data, this equation is solved numerically with respect to z_k in each grid point $y_k|_{k=0}^{N}$ of generated wavy surface. In this case equation is rewritten as

$$F(z_k) = \frac{1}{\sqrt{2\pi}} \int\limits_{0}^{y_k} \exp\left[-\frac{t^2}{2}\right] dt. \qquad (7)$$

Since, distribution functions are monotonic, the simplest interval halving (bisection) numerical method is used to solve this equation.

Preliminary ACF transformation. In order to transform ACF γ_z of the process, it is expanded as a series of Hermite polynomials (Gram—Charlier series)

$$\gamma_z(\vec{u}) = \sum_{m=0}^{\infty} C_m^2 \frac{\gamma_y^m(\vec{u})}{m!},$$

where

$$C_m = \frac{1}{\sqrt{2\pi}} \int\limits_{0}^{\infty} f(y) H_m(y) \exp\left[-\frac{y^2}{2}\right],$$

H_m — Hermite polynomial, and $f(y)$ — solution to equation (7). Plugging polynomial approximation $f(y) \approx \sum\limits_i d_i y^i$ and analytic formulae for Hermite polynomial yields

$$\frac{1}{\sqrt{2\pi}} \int\limits_{\infty}^{\infty} y^k \exp\left[-\frac{y^2}{2}\right] = \begin{cases} (k-1)!! & \text{if } k \text{ is even,} \\ 0 & \text{if } k \text{ is odd,} \end{cases}$$

which simplifies the former equation. Optimal number of coefficients C_m is determined by computing them sequentially and stopping when variances of both fields become equal with desired accuracy ϵ:

$$\left|\sigma_z^2 - \sum_{k=0}^{m} \frac{C_k^2}{k!}\right| \leq \epsilon. \tag{8}$$

In [8] the author suggests using polynomial approximation $f(y)$ also for wavy surface transformation, however, in practice sea surface realisation often contains points, in which z-coordinate is beyond the limits of the approximation, leading to sharp precision decrease. In these points it is more efficient to solve equation (7) by bisection method. Using the same approximation in Gram—Charlier series does not lead to such errors.

Gram—Charlier series expansion. In [11] the authors experimentally show, that PDF of sea surface elevation is distinguished from normal distribution by non-nought kurtosis and skewness. In [12] the authors show, that this type of PDF expands as a Gram—Charlier series (GCS):

$$\begin{gathered}
F(z;\mu=0,\sigma=1,\gamma_1,\gamma_2) \approx \Phi(z;\mu,\sigma)\frac{2+\gamma_2}{2} - \frac{2}{3}\phi(z;\mu,\sigma)\left(\gamma_2 z^3 + \gamma_1\left(2z^2+1\right)\right) \\
f(z;\mu,\sigma,\gamma_1,\gamma_2) \approx \phi(z;\mu,\sigma)\left[1 + \frac{1}{3!}\gamma_1 H_3\left(\frac{z-\mu}{\sigma}\right) + \frac{1}{4!}\gamma_2 H_4\left(\frac{z-\mu}{\sigma}\right)\right],
\end{gathered} \tag{9}$$

where $\Phi(z)$ — cumulative density function (CDF) of normal distribution, ϕ — PDF of normal distribution, γ_1 — skewness, γ_2 — kurtosis, f — PDF, F — cumulative distribution function (CDF). According to [13] for sea waves skewness is selected from interval $0.1 \leq \gamma_1 \leq 0.52]$ and kurtosis from interval $0.1 \leq \gamma_2 \leq 0.7$. Family of probability density functions for different parameters is shown in fig. 4.

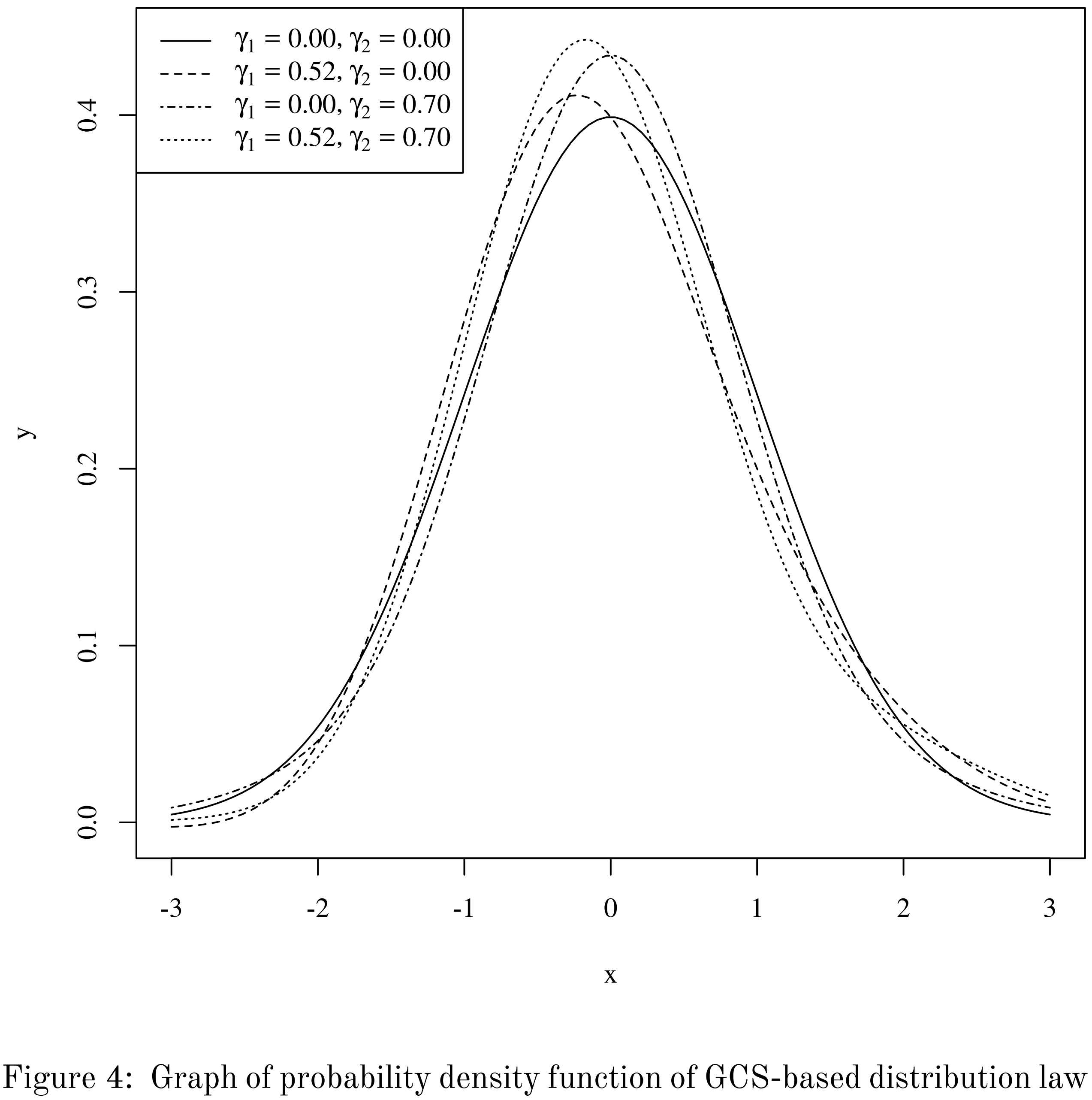


Figure 4: Graph of probability density function of GCS-based distribution law for different values of skewness γ_1 and kurtosis γ_2.

Skew-normal distribution. Alternative approach is to approximate distribution of sea wavy surface elevation by skew-normal (SN) distribution:

$$F(z;\alpha) = \frac{1}{2}\text{erfc}\left[-\frac{z}{\sqrt{2}}\right] - 2T(z,\alpha),$$
$$f(z;\alpha) = \frac{e^{-\frac{z^2}{2}}}{\sqrt{2\pi}}\text{erfc}\left[-\frac{\alpha z}{\sqrt{2}}\right], \quad (10)$$

where T — Owen T-function [22]. Using this formula it is impossible to specify skewness and kurtosis separately — both values are adjusted via α parameter. The only advantage of the formula is its relative computational simplicity: this function is available in some programmes and mathematical libraries. Its graph for different values of α is shown in fig. 5.

Evaluation. In order to measure the effect of NIT on the shape of the resulting wavy surface, three realisations were generated:

- realisation with Gaussian distribution (without NIT),
- realisation with Gram—Charlier series (GCS) based distribution, and
- realisation with skew normal distribution.

The seed of PRNG was set to be the same for all programme executions to make ARMA model produce the same values for each realisation. There we two experiments: for standing and propagating waves with ACFs given by formulae from section 3.4.

While the experiment showed that applying NIT with GCS-based distribution increases wave steepness, the same is not true for skew normal distribution (fig. 6). Using this distribution results in wavy surface each z-coordinate of which is always greater or equal to nought. So, skew normal distribution is unsuitable for NIT. NIT increases the wave height and steepness of both standing and propagating waves. Increasing either skewness or kurtosis of GCS-based distribution increases both wave height and steepness. The error of ACF approximation (eq. (8))

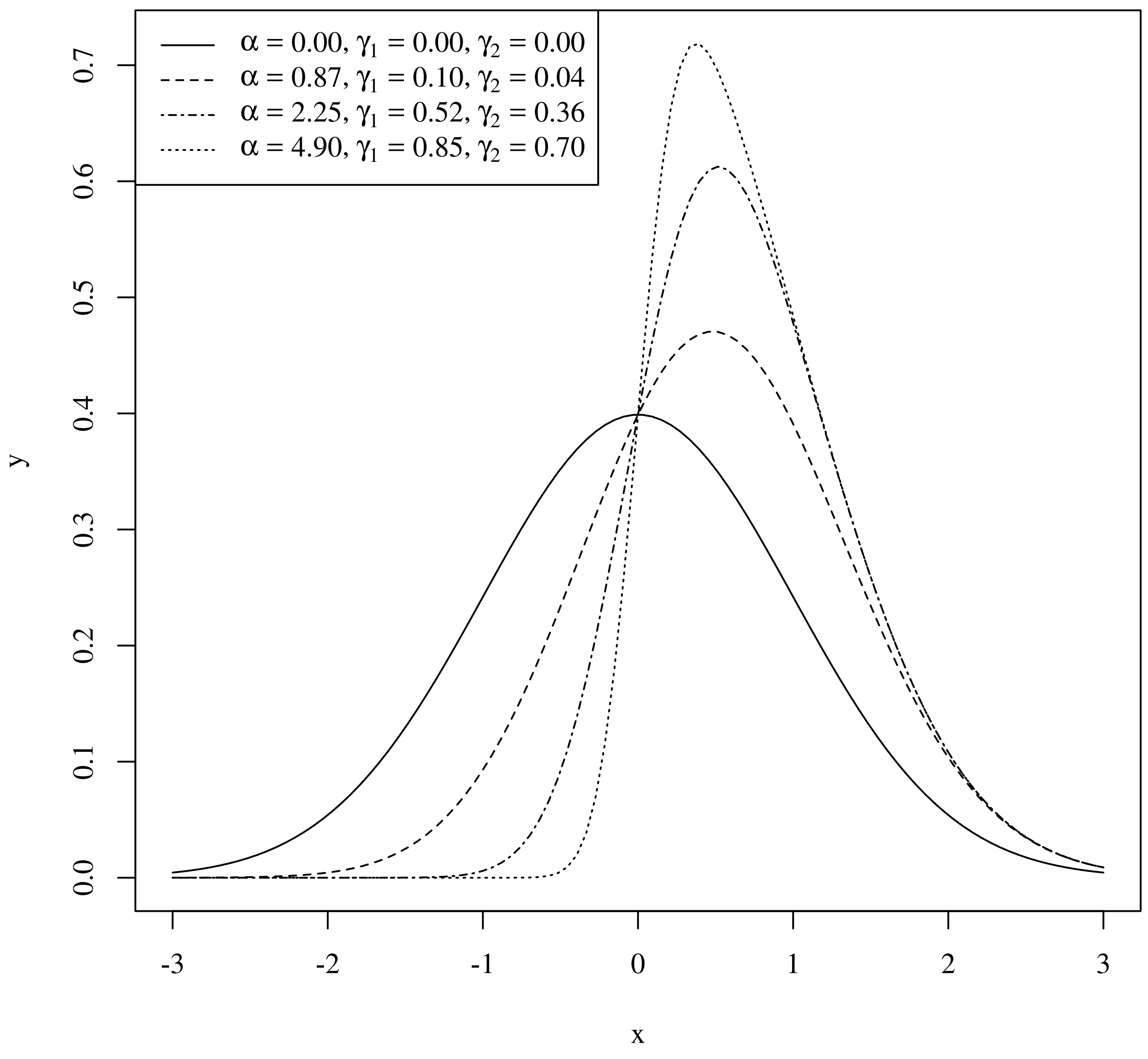


Figure 5: Graph of PDF of skew-normal distribution for different values of skewness coefficient α.

ranges from 0.20 for GCS-based distribution to 0.70 for skew normal distribution (table 2).

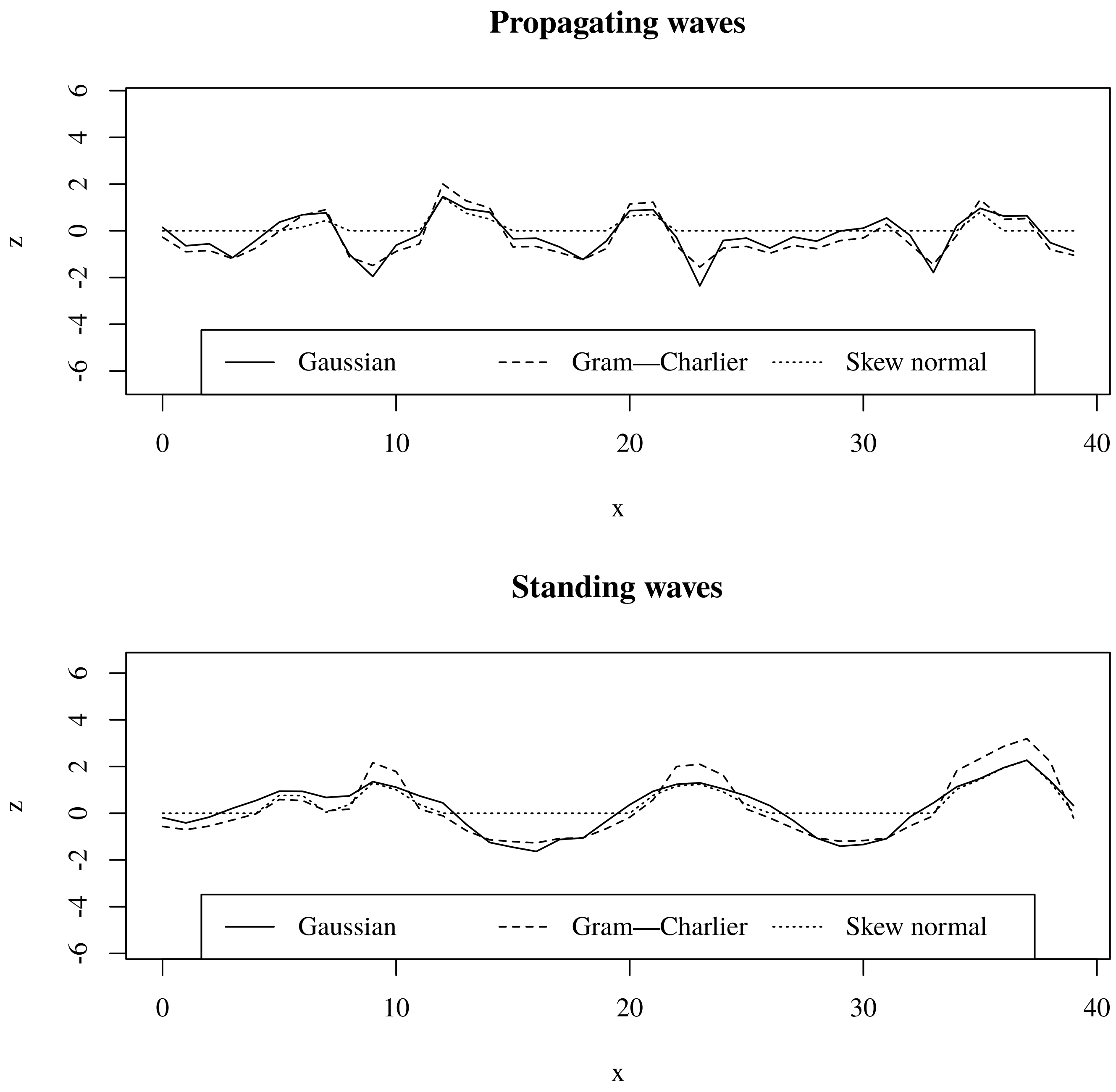


Figure 6: Wavy surface slices with different distributions of wave elevation (Gaussian, GCS-based and SN).

To summarise, the only test case that showed acceptable results is realisation with GCS-based distribution for both standing and propagating waves. Skew normal distribution warps wavy surface for both types of waves. GCS-based realisations have large error of ACF approximation, which results in increase of

Wave type	Distribution	γ_1	γ_2	α	Error	N	Wave height
propagating	Gaussian						2.41
propagating	GCS-based	2.25	0.4		0.20	2	2.75
propagating	skew normal			1	0.70	3	1.37
standing	Gaussian						1.73
standing	GCS-based	2.25	0.4		0.26	2	1.96
standing	skew normal			1	0.70	3	0.94

Table 2: Errors of ACF approximations (the difference of variances) for different wave elevation distributions. N is the number of coefficients of ACF approximation.

wave height. The reason for the large error is that GCS approximations are not accurate as they do not converge for all possible functions [23]. Despite the large error, the change in wave height is small (table 2).

3.4 The shape of ACF for different types of waves

Analytic method of finding the ACF. The straightforward way to find ACF for a given sea wave profile is to apply Wiener—Khinchin theorem. According to this theorem the autocorrelation K of a function ζ is given by the Fourier transform of the absolute square of the function:

$$K(t) = \mathcal{F}\left\{\left|\zeta(t)\right|^2\right\}. \tag{11}$$

When ζ is replaced with actual wave profile, this formula gives you analytic formula for the corresponding ACF.

For three-dimensional wave profile (2D in space and 1D in time) analytic formula is a polynomial of high order and is best obtained via symbolic computation programme. Then for practical usage it can be approximated by superposition of exponentially decaying cosines (which is how ACF of a stationary ARMA process looks like [17]).

Empirical method of finding the ACF. However, for three-dimensional case there exists simpler empirical method which does not require sophisticated software to determine shape of the ACF. It is known that ACF represented by exponentially decaying cosines satisfies first order Stokes' equations for gravity waves [24]. So, if the shape of the wave profile is the only concern in the simulation, then one can simply multiply it by a decaying exponent to get appropriate ACF. This ACF does not reflect other wave profile parameters, such as wave height and period, but opens possibility to simulate waves of a particular shape by defining their profile with discretely given function, then multiplying it by an exponent and using the resulting function as ACF. So, this empirical method is imprecise but offers simpler alternative to Wiener—Khinchin theorem approach; it is mainly useful to test ARMA model.

Standing wave ACF. For three-dimensional plain standing wave the profile is given by

$$\zeta(t,x,y) = A \sin(k_x x + k_y y) \sin(\sigma t). \tag{12}$$

Find ACF via analytic method. Multiplying the formula by a decaying exponent (because Fourier transform is defined for a function f that $f \underset{x\to\pm\infty}{\longrightarrow} 0$) yields

$$\zeta(t,x,y) = A \exp\left[-\alpha(|t|+|x|+|y|)\right] \sin(k_x x + k_y y) \sin(\sigma t). \tag{13}$$

Then, apply three-dimensional Fourier transform to both sides of the equation via symbolic computation programme, fit the resulting polynomial to the following approximation:

$$K(t,x,y) = \gamma \exp\left[-\alpha(|t|+|x|+|y|)\right] \cos\beta t \cos\left[\beta x + \beta y\right]. \tag{14}$$

So, after applying Wiener—Khinchin theorem we get initial formula but with cosines instead of sines. This difference is important because the value of ACF at $(0, 0, 0)$ equals to the ARMA process variance, and if one used sines the value would be wrong.

If one tries to replicate the same formula via empirical method, the usual way is to adapt (13) to match (14). This can be done either by changing the phase of the sine, or by substituting sine with cosine to move the maximum of the function to the origin of coordinates.

Propagating wave ACF. Three-dimensional profile of plain propagating wave is given by

$$\zeta(t, x, y) = A \cos(\sigma t + k_x x + k_y y). \tag{15}$$

For the analytic method repeating steps from the previous two paragraphs yields

$$K(t, x, y) = \gamma \exp\left[-\alpha(|t| + |x| + |y|)\right] \cos\left[\beta(t + x + y)\right]. \tag{16}$$

For the empirical method the wave profile is simply multiplied by a decaying exponent without need to adapt the maximum value of ACF (as it is required for standing wave).

Comparison of studied methods. To summarise, the analytic method of finding sea wave's ACF reduces to the following steps.

- Make wave profile decay when approaching $\pm\infty$ by multiplying it by a decaying exponent.
- Apply Fourier transform to the absolute square of the resulting equation using symbolic computation programme.
- Fit the resulting polynomial to the appropriate ACF approximation.

Two examples in this section showed that in case of standing and propagating waves their decaying profiles resemble the corresponding ACFs with the exception that the ACF’s maximum should be moved to the origin to preserve simulated process variance. Empirical method of finding ACF reduces to the following steps.

- Make wave profile decay when approaching $\pm\infty$ by multiplying it by a decaying exponent.
- Move maximum value of the resulting function to the origin by using trigonometric identities to shift the phase.

3.5 Summary

ARMA model, owing to its non-physical nature, does not have the notion of sea wave; it simulates wavy surface as a whole instead. Motions of individual waves and their shape are often rough, and the total number of waves can not be determined precisely. However, integral characteristics of wavy surface match the ones of real sea waves.

Theoretically, sea waves themselves can be chosen as ACFs, the only preprocessing step is to make them decay exponentially. This may allow to generate waves of arbitrary profiles without using NIT, and is one of the directions of future work.

4 Pressure field under discretely given wavy surface

4.1 Known pressure field determination formulae

Small amplitude waves theory. In [25–27] the authors propose a solution for inverse problem of hydrodynamics of potential flow in the framework of small-amplitude wave theory (under assumption that wave length is much larger than height: $\lambda \gg h$). In that case inverse problem is linear and reduces to Laplace equation with mixed boundary conditions, and equation of motion is solely used to determine pressures for calculated velocity potential derivatives. The assumption of small amplitudes means the slow decay of wind wave coherence function, i.e. small change of local wave number in time and space compared to the wavy surface elevation (z coordinate). This assumption allows to calculate elevation z derivative as $\zeta_z = k\zeta$, where k is wave number. In two-dimensional case the solution is written explicitly as

$$\left.\frac{\partial\phi}{\partial x}\right|_{x,t} = -\frac{1}{\sqrt{1+\alpha^2}} e^{-I(x)} \int\limits_0^x \frac{\partial\dot{\zeta}/\partial z + \alpha\dot{\alpha}}{\sqrt{1+\alpha^2}} e^{I(x)} dx, \tag{17}$$

$$I(x) = \int\limits_0^x \frac{\partial\alpha/\partial z}{1+\alpha^2} dx,$$

where α is wave slope. In three-dimensional case the solution is written in the form of elliptic partial differential equation (PDE):

$$\begin{aligned}
&\frac{\partial^2\phi}{\partial x^2}\left(1+\alpha_x^2\right)+\frac{\partial^2\phi}{\partial y^2}\left(1+\alpha_y^2\right)+2\alpha_x\alpha_y\frac{\partial^2\phi}{\partial x\partial y}+\\
&\left(\frac{\partial\alpha_x}{\partial z}+\alpha_x\frac{\partial\alpha_x}{\partial x}+\alpha_y\frac{\partial\alpha_x}{\partial y}\right)\frac{\partial\phi}{\partial x}+\\
&\left(\frac{\partial\alpha_y}{\partial z}+\alpha_x\frac{\partial\alpha_y}{\partial x}+\alpha_y\frac{\partial\alpha_y}{\partial y}\right)\frac{\partial\phi}{\partial y}+\\
&\frac{\partial\dot{\zeta}}{\partial z}+\alpha_x\dot{\alpha_x}+\alpha_y\dot{\alpha_y}=0.
\end{aligned}$$

The authors suggest transforming this equation to finite differences and solve it numerically.

As will be shown in section 4.3, formula (17) diverges when attempted to calculate velocity field for large-amplitude waves, and this is the reason that it can not be used in conjunction with a model, that generates arbitrary-amplitude waves.

Linearisation of boundary condition. LH model allows to derive an explicit formula for velocity field by linearising kinematic boundary condition. Velocity potential formula is written as

$$\phi(x,y,z,t)=\sum_n\frac{c_n g}{\omega_n}e^{\sqrt{u_n^2+v_n^2}z}\sin(u_n x+v_n y-\omega_n t+\epsilon_n).$$

This formula is differentiated to obtain velocity potential derivatives, which are plugged to dynamic boundary condition to determine pressure field.

4.2 Determining wave pressures for discretely given wavy surface

Analytic solutions to boundary problems in classical equations are often used to study different properties of the solution, and for that purpose general solution formula is too difficult to study, as it contains integrals of unknown functions. Fourier method is one of the methods to find analytic solutions to PDE. It is based on Fourier transform, applying of which to some PDEs reduces them to algebraic equations, and the solution is written as inverse Fourier transform of some function (which may contain Fourier transforms of other functions). Since it is not possible to write analytic forms of these Fourier transforms in some cases, unique solutions are found and their behaviour is studied in different domains instead. At the same time, computing discrete Fourier transforms numerically is possible for any discretely defined function using FFT algorithms. These algorithms use symmetry of complex exponentials to decrease asymptotic complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \log_2 n)$. So, even if general solution contains Fourier transforms of unknown functions, they still can be computed numerically, and FFT family of algorithms makes this approach efficient.

Alternative approach to solve PDEs is to reduce them to difference equations, which are solved by constructing various numerical schemes. This approach leads to approximate solution, and asymptotic complexity of corresponding algorithms is comparable to that of FFT. For example, for stationary elliptic PDE an implicit numerical scheme is constructed; the scheme is computed by iterative method on each step of which a tridiagonal or five-diagonal system of algebraic equations is solved by Thomas algorithm. Asymptotic complexity of this approach is $\mathcal{O}(nm)$, where n is the number of wavy surface grid points and m is the number of iterations.

Despite their wide spread, iterative algorithms are inefficient on parallel computer architectures due to inevitable process synchronisation after each iteration; in particular, their mapping to co-processors may involve copying data in and out of the co-processor on each iteration, which negatively affects their performance. At the same time, high number of Fourier transforms in the solution is an advantage, rather than a disadvantage. First, solutions obtained by Fourier method are explicit, hence their implementations scale with the large number of parallel computer cores. Second, there are implementations of FFT optimised for different processor architectures as well as co-processors (GPU, MIC) which makes it easy to get high performance on any computing platform. These advantages substantiate the choice of Fourier method to obtain explicit solution to the problem of determining pressures under wavy sea surface.

4.2.1 Two-dimensional velocity potential field

Formula for infinite depth fluid. Two-dimensional Laplace equation with Robin boundary condition is written as

$$\begin{aligned}
&\phi_{xx}+\phi_{zz}=0, \\
&\zeta_t+\zeta_x\phi_x=\frac{\zeta_x}{\sqrt{1+\zeta_x^2}}\phi_x-\frac{1}{\sqrt{1+\zeta_x^2}}\phi_z, \qquad \text{at } z=\zeta(x,t).
\end{aligned} \tag{18}$$

Use Fourier method to solve this problem. Applying Fourier transform to both sides of the equation yields

$$-4\pi^2\left(u^2+v^2\right)\mathcal{F}_{u,v}\{\phi(x,z)\}=0,$$

hence $v = \pm iu$. Hereinafter we use the following symmetric form of Fourier transform:

$$\mathcal{F}_{u,v}\{f(x,y)\} = \iint\limits_{-\infty}^{\infty} f(x,y) e^{-2\pi i(xu+yv)} dx dy.$$

We seek solution in the form of inverse Fourier transform $\phi(x,z) = \mathcal{F}^{-1}_{x,z}\{E(u,v)\}$. Plugging[1] $v = iu$ into the formula yields

$$\phi(x,z) = \mathcal{F}^{-1}_{x}\left\{e^{2\pi uz} E(u)\right\}. \tag{19}$$

In order to make substitution $z = \zeta(x,t)$ not interfere with Fourier transforms, we rewrite (19) as a convolution:

$$\phi(x,z) = \mathcal{D}_1(x,z) * \mathcal{F}^{-1}_{x}\{E(u)\},$$

where $\mathcal{D}_1(x,z)$ — a function, form of which is defined in section 4.2.3 and which satisfies equation $\mathcal{F}_u\{\mathcal{D}_1(x,z)\} = e^{2\pi uz}$. Plugging formula ϕ into the boundary condition yields

$$\zeta_t = \left(if(x) - 1/\sqrt{1+\zeta_x^2}\right)\left[\mathcal{D}_1(x,z) * \mathcal{F}^{-1}_{x}\{2\pi u E(u)\}\right],$$

where $f(x) = \zeta_x/\sqrt{1+\zeta_x^2} - \zeta_x$. Applying Fourier transform to both sides of this equation yields formula for coefficients E:

$$E(u) = \frac{1}{2\pi u} \frac{\mathcal{F}_u\left\{\zeta_t / \left(if(x) - 1/\sqrt{1+\zeta_x^2}\right)\right\}}{\mathcal{F}_u\{\mathcal{D}_1(x,z)\}}$$

[1] $v = -iu$ is not applicable because velocity potential must go to nought when depth goes to infinity.

Finally, substituting z for $\zeta(x,t)$ and plugging resulting equation into (19) yields formula for $\phi(x,z)$:

$$\boxed{\phi(x,z) = \mathcal{F}_x^{-1}\left\{ \frac{e^{2\pi uz}}{2\pi u} \frac{\mathcal{F}_u\left\{\zeta_t / \left(if(x) - 1/\sqrt{1+\zeta_x^2}\right)\right\}}{\mathcal{F}_u\{\mathcal{D}_1\left(x, \zeta(x,t)\right)\}} \right\}.} \tag{20}$$

Multiplier $e^{2\pi uz}/(2\pi u)$ makes graph of a function to which Fourier transform is applied asymmetric with respect to OY axis. This makes it difficult to use FFT, because it expects periodic function which takes nought values on both ends of the interval. At the same time, using numerical integration instead of FFT is not faster than solving the initial system of equations with numerical schemes. This problem is alleviated by using formula (22) for finite depth fluid with knowingly large depth h. This formula is derived in the following section.

Formula for finite depth fluid. On the sea bottom vertical fluid velocity component equals nought, i.e. $\phi_z = 0$ on $z = -h$, where h — water depth. In this case equation $v = -iu$, which came from Laplace equation, can not be neglected, hence the solution is sought in the following form:

$$\phi(x,z) = \mathcal{F}_x^{-1}\left\{ \left(C_1 e^{2\pi uz} + C_2 e^{-2\pi uz}\right) E(u) \right\}. \tag{21}$$

Plugging ϕ into the boundary condition on the sea bottom yields

$$C_1 e^{-2\pi uh} - C_2 e^{2\pi uh} = 0,$$

hence $C_1 = \frac{1}{2} C e^{2\pi uh}$ and $C_2 = -\frac{1}{2} C e^{-2\pi uh}$. Constant C may take arbitrary values here, because after plugging it becomes part of unknown coefficients $E(u)$. Plugging formulae for C_1 and C_2 into (21) yields

$$\phi(x,z) = \mathcal{F}_x^{-1}\{\cosh\left(2\pi u(z+h)\right) E(u)\}.$$

Plugging ϕ into the boundary condition on the free surface yields

$$\begin{aligned}\zeta_t = f(x)\mathcal{F}_x^{-1}\{2\pi i u \cosh\left(2\pi u(z+h)\right) E(u)\} \\ - \frac{1}{\sqrt{1+\zeta_x^2}}\mathcal{F}_x^{-1}\{2\pi u \sinh\left(2\pi u(z+h)\right) E(u)\}.\end{aligned}$$

Here sinh and cosh give similar results near free surface, and since this is the main area of interest in practical applications, we assume that $\cosh\left(2\pi u(z+h)\right) \approx \sinh\left(2\pi u(z+h)\right)$. Performing analogous to the previous section transformations yields final formula for $\phi(x,z)$:

$$\boxed{\phi(x,z,t) = \mathcal{F}_x^{-1}\left\{ \frac{\cosh\left(2\pi u(z+h)\right)}{2\pi u} \frac{\mathcal{F}_u\left\{\zeta_t / \left(i f(x) - 1/\sqrt{1+\zeta_x^2}\right)\right\}}{\mathcal{F}_u\{\mathcal{D}_2\left(x,\zeta(x,t)\right)\}} \right\},} \qquad (22)$$

where $\mathcal{D}_2\left(x,z\right)$ is a function, a form of which is defined in section 4.2.3 and which satisfies equation $\mathcal{F}_u\{\mathcal{D}_2\left(x,z\right)\} = \cosh\left(2\pi u z\right)$.

Reduction to the formulae from linear wave theory. Check the validity of derived formulae by substituting $\zeta(x,t)$ with known analytic formula for plain waves. Symbolic computation of Fourier transforms in this section were performed in Mathematica [5]. In the framework of linear wave theory assume that waves have small amplitude compared to their lengths, which allows us to simplify initial system of equations (18) to

$$\begin{aligned}&\phi_{xx} + \phi_{zz} = 0, \\ &\zeta_t = -\phi_z && \text{at } z = \zeta(x,t),\end{aligned}$$

solution to which is written as

$$\phi(x,z,t) = -\mathcal{F}_x^{-1}\left\{ \frac{e^{2\pi u z}}{2\pi u} \mathcal{F}_u\{\zeta_t\} \right\}.$$

Propagating wave profile is defined as $\zeta(x,t) = A\cos(2\pi(kx-t))$. Plugging this formula into (20) yields $\phi(x,z,t) = -\frac{A}{k}\sin(2\pi(kx-t))\cosh(2\pi kz)$. In order to reduce it to the formula from linear wave theory, rewrite hyperbolic cosine in exponential form and discard the term containing $e^{-2\pi kz}$ as contradicting condition $\phi \underset{z\rightarrow-\infty}{\longrightarrow} 0$. Taking real part of the resulting formula yields $\phi(x,z,t) = \frac{A}{k}e^{2\pi kz}\sin(2\pi(kx-t))$, which corresponds to the known formula from linear wave theory. Similarly, under small-amplitude waves assumption the formula for finite depth fluid (22) is reduced to

$$\phi(x,z,t) = -\mathcal{F}_x^{-1}\left\{ \frac{\cosh(2\pi u(z+h))}{2\pi u\cosh(2\pi uh)} \mathcal{F}_u\{\zeta_t\} \right\}.$$

Substituting $\zeta(x,t)$ with propagating plain wave profile formula yields

$$\phi(x,z,t) = \frac{A}{k}\frac{\cosh(2\pi k(z+h))}{\cosh(2\pi kh)}\sin(2\pi(kx-t)), \tag{23}$$

which corresponds to the formula from linear wave theory for finite depth fluid.

Different forms of Laplace equation solutions, in which decaying exponent is written with either “$+$” or “-” signs, may cause incompatibilities between formulae from linear wave theory and formulae derived in this work, where sinh is used instead of cosh. Equality $\frac{\cosh(2\pi k(z+h))}{\cosh(2\pi kh)} \approx \frac{\sinh(2\pi k(z+h))}{\sinh(2\pi kh)}$ becomes strict on the free surface, and difference between left-hand and right-hand sides increases when approaching sea bottom (for sufficiently large depth difference near free surface is negligible). So, for sufficiently large depth any function (cosh or sinh) may be used for velocity potential computation near free surface.

Reducing (20) and (22) to the known formulae from linear wave theory shows, that formula for infinite depth (20) is not suitable to compute velocity potentials with Fourier method, because it does not have symmetry, which is required for Fourier transform. However, formula for finite depth can be used instead by setting h to some characteristic water depth. For standing wave reducing to linear wave theory formulae is made under the same assumptions.

4.2.2 Three-dimensional velocity potential field

Three-dimensional version of (1) is written as

$$\begin{aligned}
&\phi_{xx} + \phi_{yy} + \phi_{zz} = 0, && (24)\\
&\zeta_t + \zeta_x\phi_x + \zeta_y\phi_y = \frac{\zeta_x}{\sqrt{1+\zeta_x^2+\zeta_y^2}}\phi_x + \frac{\zeta_y}{\sqrt{1+\zeta_x^2+\zeta_y^2}}\phi_y - \frac{1}{\sqrt{1+\zeta_x^2+\zeta_y^2}}\phi_z, \\
&\text{at } z=\zeta(x,y,t).
\end{aligned}$$

Again, use Fourier method to solve it. Applying Fourier transform to both sides of Laplace equation yields

$$-4\pi^2\left(u^2+v^2+w^2\right)\mathcal{F}_{u,v,w}\{\phi(x,y,z)\} = 0,$$

hence $w = \pm i\sqrt{u^2+v^2}$. We seek solution in the form of inverse Fourier transform $\phi(x,y,z) = \mathcal{F}^{-1}_{x,y,z}\{E(u,v,w)\}$. Plugging $w = i\sqrt{u^2+v^2} = i|\vec{k}|$ into the formula yields

$$\phi(x,y,z) = \mathcal{F}^{-1}_{x,y}\Big\{\left(C_1 e^{2\pi|\vec{k}|z} - C_2 e^{-2\pi|\vec{k}|z}\right)E(u,v)\Big\}.$$

Plugging ϕ into the boundary condition on the sea bottom (analogous to two-dimensional case) yields

$$\phi(x,y,z) = \mathcal{F}^{-1}_{x,y}\Big\{\cosh\left(2\pi|\vec{k}|(z+h)\right)E(u,v)\Big\}. \qquad (25)$$

Plugging ϕ into the boundary condition on the free surface yields

$$\begin{aligned}\zeta_t =& i f_1(x,y)\mathcal{F}^{-1}_{x,y}\Big\{2\pi u \cosh\left(2\pi|\vec{k}|(z+h)\right) E(u,v)\Big\}\\ &+ i f_2(x,y)\mathcal{F}^{-1}_{x,y}\Big\{2\pi v \cosh\left(2\pi|\vec{k}|(z+h)\right) E(u,v)\Big\}\\ &- f_3(x,y)\mathcal{F}^{-1}_{x,y}\Big\{2\pi|\vec{k}| \sinh\left(2\pi|\vec{k}|(z+h)\right) E(u,v)\Big\}\end{aligned}$$

where $f_1(x,y) = \zeta_x/\sqrt{1+\zeta_x^2+\zeta_y^2} - \zeta_x$, $f_2(x,y) = \zeta_y/\sqrt{1+\zeta_x^2+\zeta_y^2} - \zeta_y$ and $f_3(x,y) = 1/\sqrt{1+\zeta_x^2+\zeta_y^2}$.

Like in section 4.2.1 we assume that $\cosh\left(2\pi u(z+h)\right) \approx \sinh\left(2\pi u(z+h)\right)$ near free surface, but in three-dimensional case this is not enough to solve the problem. In order to get explicit formula for coefficients E we need to assume, that all Fourier transforms in the equation have radially symmetric kernels, i.e. replace u and v with $|\vec{k}|$. There are two points supporting this assumption. First, in numerical implementation integration is done over positive wave numbers, so the sign of u and v does not affect the solution. Second, the growth rate of cosh term of the integral kernel is much higher than the one of u or $|\vec{k}|$, so the substitution has small effect on the magnitude of the solution. Despite these two points, a use of more mathematically rigorous approach would be preferable.

Making the replacement, applying Fourier transform to both sides of the equation and plugging the result into (25) yields formula for ϕ:

$$\phi(x,y,z,t) = \mathcal{F}^{-1}_{x,y}\left\{\frac{\cosh\left(2\pi|\vec{k}|(z+h)\right)}{2\pi|\vec{k}|}\frac{\mathcal{F}_{u,v}\{\zeta_t/\left(i f_1(x,y) + i f_2(x,y) - f_3(x,y)\right)\}}{\mathcal{F}_{u,v}\{\mathcal{D}_3\left(x,y,\zeta\left(x,y\right)\right)\}}\right\}, \tag{26}$$

where $\mathcal{F}_{u,v}\{\mathcal{D}_3\left(x,y,z\right)\} = \cosh\left(2\pi|\vec{k}|z\right)$.

4.2.3 Velocity potential normalisation formulae

In solutions (20) and (22) to two-dimensional pressure determination problem there are functions $\mathcal{D}_1(x,z) = \mathcal{F}_x^{-1}\left\{e^{2\pi u z}\right\}$ and $\mathcal{D}_2(x,z) = \mathcal{F}_x^{-1}\{\cosh(2\pi u z)\}$ which has multiple analytic representations and are difficult to compute. Each function is a Fourier transform of linear combination of exponents which reduces to poorly defined Dirac delta function of a complex argument (see table 3). The usual way of handling this type of functions is to write them as multiplication of Dirac delta functions of real and imaginary part, however, this approach does not work here, because applying inverse Fourier transform to this representation does not produce exponent, which severely warp resulting velocity field. In order to get unique analytic definition normalisation factor $1/\cosh(2\pi u h)$ (which is also included in formula for $E(u)$) may be used. Despite the fact that normalisation allows to obtain adequate velocity potential field, numerical experiments show that there is little difference between this field and the one produced by formulae from linear wave theory, in which terms with ζ are omitted. As a result, the formula for three-dimensional case was not derived.

Function	Without normalisation	Normalised
$\mathcal{D}_1(x,z)$	$\delta(x+iz)$	$\frac{1}{2h}\operatorname{sech}\left(\frac{\pi(x-i(h+z))}{2h}\right)$
$\mathcal{D}_2(x,z)$	$\frac{1}{2}\left[\delta(x-iz)+\delta(x+iz)\right]$	$\frac{1}{4h}\left[\operatorname{sech}\left(\frac{\pi(x-i(h+z))}{2h}\right)+\operatorname{sech}\left(\frac{\pi(x+i(h+z))}{2h}\right)\right]$

Table 3: Formulae for computing $\mathcal{D}_1(x,z)$ and $\mathcal{D}_2(x,z)$ from sec. 4.2.1, that use normalisation to eliminate uncertainty from definition of Dirac delta function of complex argument.

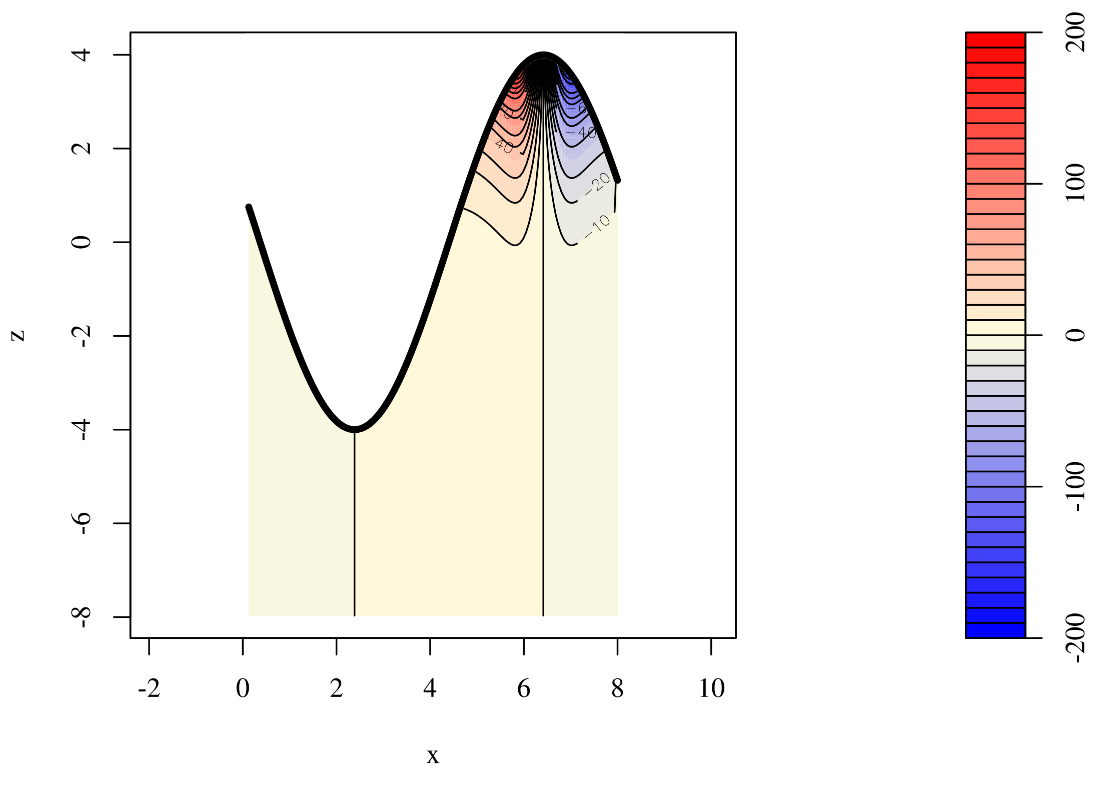
4.3 Verification of velocity potential fields

Comparing obtained generic formulae (20) and (22) to the known formulae from linear wave theory allows to see the difference between velocity fields for both large and small amplitude waves. In general case analytic formula for velocity potential in not known, even for plain waves, so comparison is done numerically. Taking into account conclusions of section 4.2.1, only finite depth formulae are compared.

The difference with linear wave theory formulae. In order to obtain velocity potential fields, plain waves of varying amplitude were generated. Wave numbers in Fourier transforms were chosen on the interval from 0 to the maximal wave number determined numerically from the resulting wavy surface. Experiments were conducted for waves of both small and large amplitudes.

The experiment showed that velocity potential fields for large-amplitude waves produced by formula (22) for finite depth fluid and formula (23) from linear wave theory are qualitatively different (fig. 7). First, velocity potential contours have sinusoidal shape, which is different from oval shape described by linear wave theory. Second, velocity potential decays more rapidly than in linear wave theory as getting closer to the bottom, and the region where the majority of wave energy is concentrated is closer to the wave crest. Similar numerical experiment, in which all terms of (22) that are neglected in the framework of linear wave theory are eliminated, shows no difference (as much as machine precision allows) in resulting velocity potential fields.

The difference with small-amplitude wave theory. The experiment, in which velocity fields produced numerically by different formulae were compared, shows that velocity fields produced by formula (22) and (17) correspond to each other for small-amplitude waves. Two sea wavy surface realisations were made by AR model: one containing small-amplitude waves, other containing large-amplitude



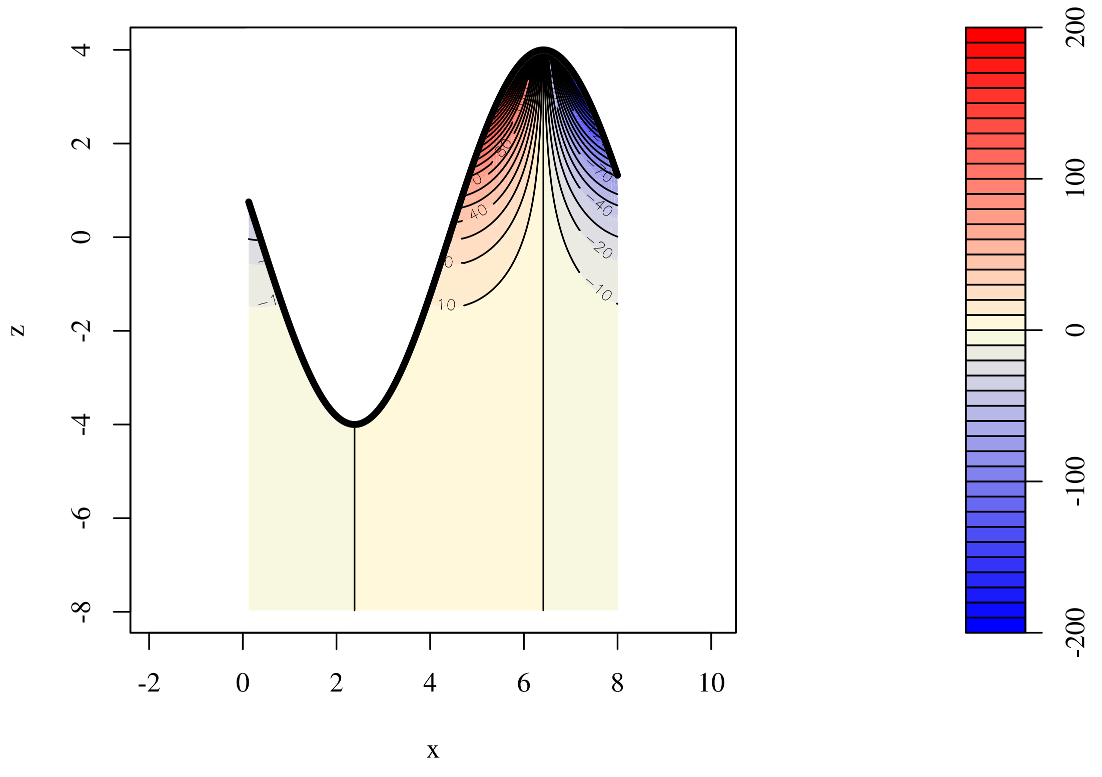
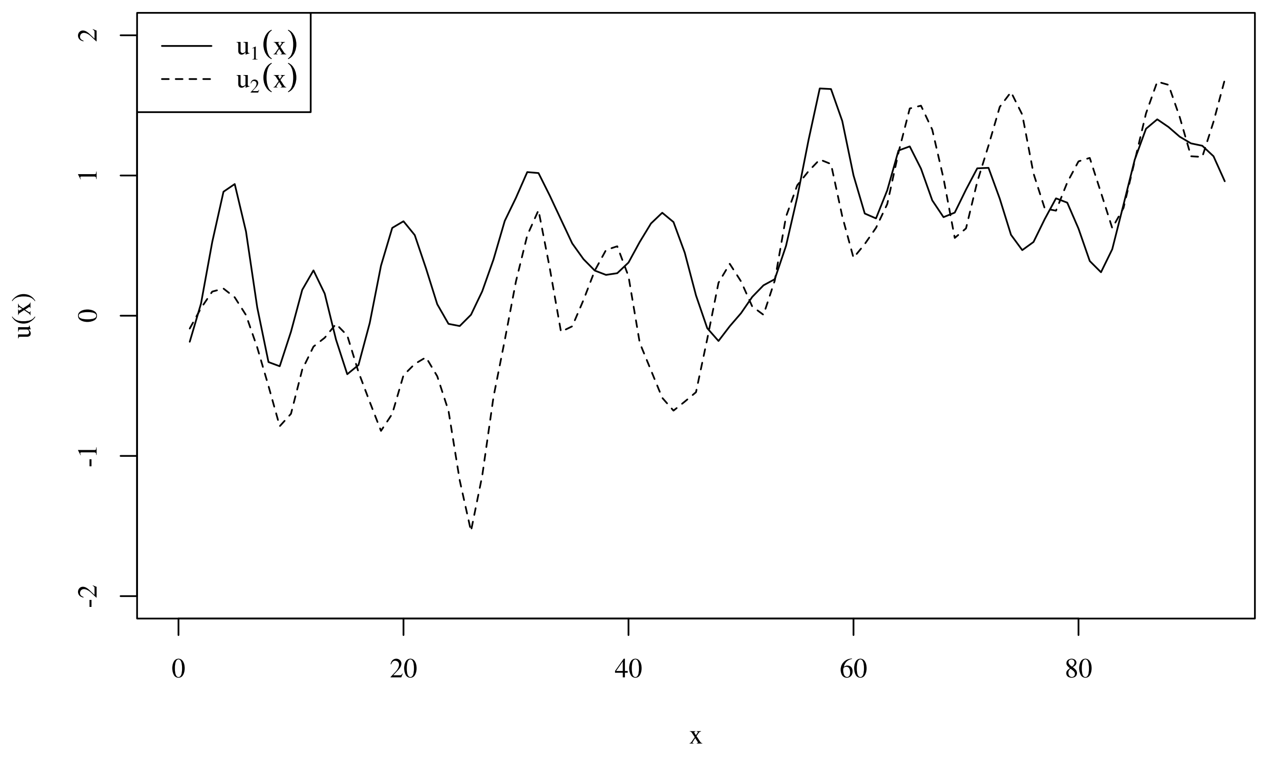


Figure 7: Comparison of velocity potential fields for propagating wave $\zeta(x, y, t) = \cos(2\pi x - t/2)$. Field produced by generic formula (top) and linear wave theory formula (bottom).

waves. Integration in formula (22) was done over wave numbers range extracted from the generated wavy surface. For small-amplitude waves both formulae showed comparable results (the difference in the velocity is attributed to the stochastic nature of AR model), whereas for large-amplitude waves stable velocity field was produced only by formula (22) (fig. 8). So, generic formula (22) gives satisfactory results without restriction on wave amplitudes.

4.4 Summary

Formulae derived in this section allow to numerically compute velocity potential field (and hence pressure field) near discretely or mathematically given wavy sea surface, bypassing assumptions of linear wavy theory and theory of small amplitude waves. For small amplitude waves new formulae produce the same velocity potential field, as formulae from linear wave theory. For large-amplitude waves the usage of new formulae results in a shift of a region where the majority of wave energy is concentrated closer to the wave crest.



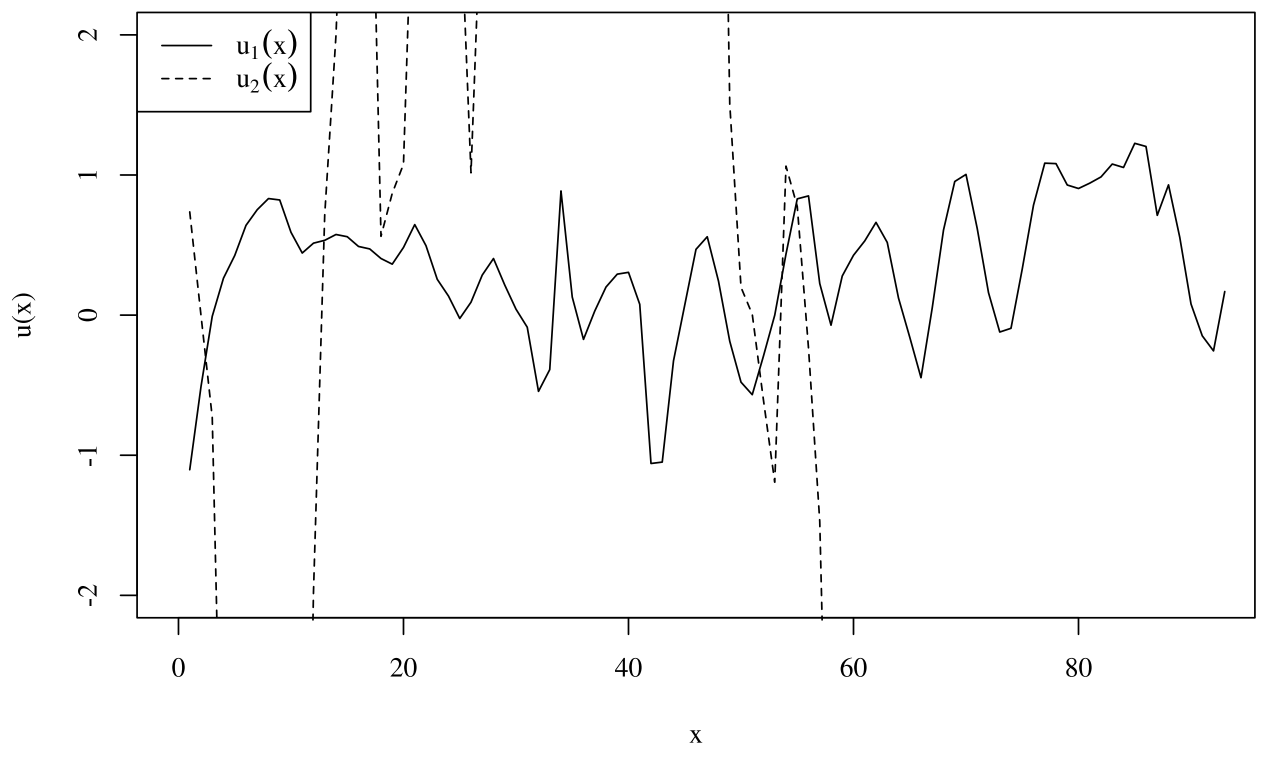


Figure 8: Comparison of velocity potential fields obtained by generic formula (u_1) and formula for small-amplitude waves (u_2): velocity field for realisations containing small-amplitude (top) and large-amplitude (bottom) waves.

5 High-performance software implementation of sea wave simulation

5.1 SMP implementation

5.1.1 Wavy surface generation

Parallel AR, MA and LH model algorithms. Although, AR and MA models are part of the single mixed model they have disparate parallel algorithms, which are different from trivial one of LH model. AR algorithm consists in partitioning wavy surface into equally-sized parts in each dimension and computing them in parallel taking into account causal constraints imposed by autoregressive dependencies between surface points. There are no such dependencies in MA model, and its formula represents convolution of white noise with model coefficients, which is reduced to computation of three Fourier transforms via convolution theorem. So, MA algorithm consists in parallel computation of the convolution which is based on FFT computation. Finally, LH algorithm is made parallel by simply computing each wavy surface point in parallel in several threads. So, parallel implementation of ARMA model consists of two parallel algorithms, one for each part of the model, which are more sophisticated than the one for LH model.

AR model's formula main feature is autoregressive dependencies between wavy surface points in each dimension which prevent computing each surface point in parallel. Instead the surface is partitioned along each dimension into equal three-dimensional parts, and for each part information dependencies, which define computation order, are established. Figure 9 shows these dependencies. An arrow denotes dependency of one part on availability of another, i.e. computation of a part may start only when all parts on which it depends were computed. Here part A does not have dependencies, parts B and D depend only on A, and part E depends on A, B and C. In general, each part depends on all parts that have previous index in at least one dimension (if such parts exist). The first part does not have any dependencies; and the size of each part along each dimension is made greater or equal to the corresponding number of coefficients along the dimension to consider only adjacent parts in dependency resolution.

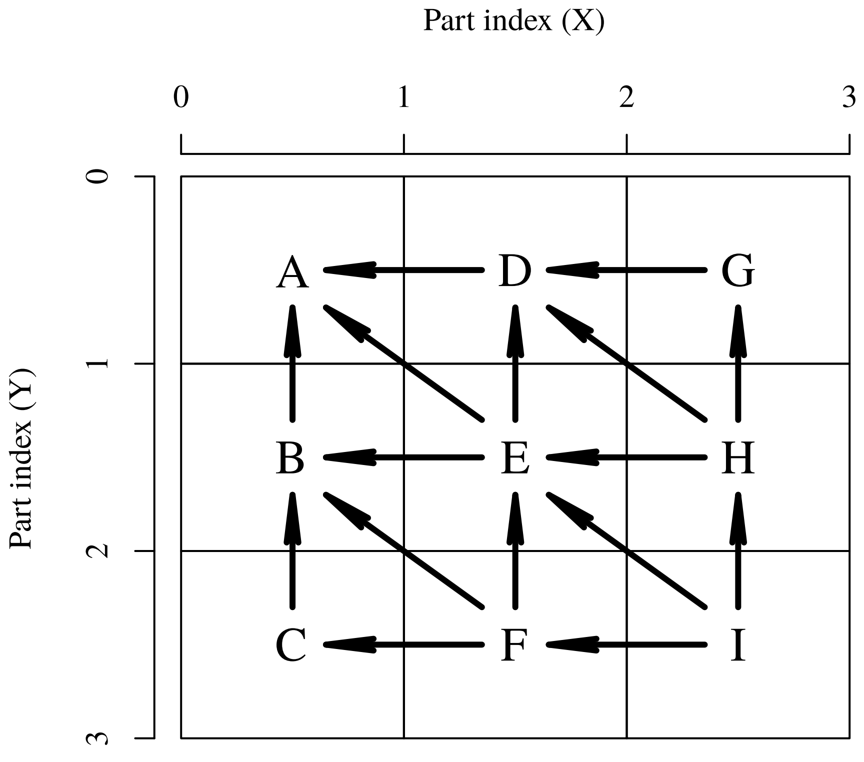


Figure 9: Autoregressive dependencies between wavy surface parts.

Each part has an three-dimensional index and a completion status. The algorithm starts by submitting objects containing this information into a queue. After that parallel threads start, each thread finds the first part for which all dependencies are satisfied (by checking the completion status of each part), removes the part from the queue, generates wavy surface for this part and sets completion status. The algorithm ends when the queue becomes empty. Access to the queue from different threads is synchronised by locks. The algorithm is suitable for SMP machines; for MPP all parts on which the current one depends, dependent parts needs to be copied to the node where computation is carried out.

So, parallel AR model algorithm reduces to implementing minimalistic job scheduler, in which

- each job corresponds to a wavy surface part,
- the order of execution of jobs is defined by autoregressive dependencies, and
- job queue is processed by a simple thread pool in which each thread in a loop removes the first job from the queue for which all dependent jobs have completed and executes it.

In contrast to AR model, MA model does not have autoregressive dependencies between points; instead, each surface point depends on previous in time and space white noise values. MA model's formula allows for rewriting it as a convolution of white noise with the coefficients as a kernel. Using convolution theorem, the convolution is rewritten as inverse Fourier transform of the product of Fourier transforms of white noise and coefficients. Since the number of MA coefficients is much smaller than the number of wavy surface points, parallel FFT implementation is not suitable here, as it requires padding the coefficients with noughts to match the size of the surface. Instead, the surface is divided into parts along each dimension which are padded with noughts to match the number of the coefficients along each dimension multiplied by two. Then Fourier transform of each part is computed in parallel, multiplied by previously computed Fourier transform of the coefficients, and inverse Fourier transform of the result is computed. After that, each part is written to the output array with overlapping (due to padding) points added to each other. This algorithm is commonly known in signal processing as "overlap-add" [28–30]. Padding with noughts is needed to prevent aliasing errors: without it the result would be circular convolution.

Despite the fact that MA model algorithm partitions the surface into the same parts (but possibly of different sizes) as AR model algorithm, the absence of autoregressive dependencies between them allows to compute them in parallel without the use of specialised job scheduler. However, this algorithm requires padding parts with noughts to make the result of calculations correspond to the result obtained with original MA model's formula. So, MA model's algorithm is more scalable to a large number of nodes as it has no information dependencies between parts, but the size of the parts is greater than in AR model's algorithm.

The distinct feature of LH model's algorithm is its simplicity: to make it parallel, wavy surface is partitioned into parts of equal sizes and each part is generated in parallel. There are no information dependencies between parts, which makes this algorithm particularly suitable for computation on GPU: each hardware thread simply computes its own point. In addition, sine and cosine functions in the model's formula are slow to compute on CPU, which makes GPU even more favourable choice.

To summarise, even though AR and MA models are part of the single mixed model, their parallel algorithms are fundamentally different and are more complicated than trivial parallel algorithm of LH model. Efficient implementation AR algorithm requires specialised job scheduler to manage autoregressive dependencies between wavy surface parts, whereas MA algorithm requires padding each part with noughts to be able to compute them in parallel. In contrast to these models, LH model has no information dependencies between parts, but requires more computational resources (floating point operations per seconds) for transcendental functions in its formula.

Parallel white noise generation algorithm. In order to eliminate periodicity from wavy surface generated by sea wave model, it is imperative to use

PRNG with sufficiently large period to generate white noise. Parallel Mersenne Twister [31] with a period of $2^{19937}-1$ is used as a generator in this work. It allows to produce aperiodic sea wavy surface realisations in any practical usage scenarios.

There is no guarantee that multiple PRNGs executed in parallel threads with distinct initial states produce uncorrelated pseudo-random number sequences, and algorithm of dynamic creation of Mersenne Twisters [32] is used to provide such guarantee. The essence of the algorithm is to find matrices of initial generator states, that give maximally uncorrelated pseudo-random number sequences when Mersenne Twisters are executed in parallel with these initial states. Since finding such initial states consumes considerable amount of processor time, vector of initial states is created beforehand with knowingly larger number of parallel threads and saved to a file, which is then read before starting white noise generation.

Ramp-up interval elimination. In AR model value of wavy surface elevation at a particular point depends on previous in space and time points, as a result the so called *ramp-up interval* (see fig. 10), in which realisation does not correspond to specified ACF, forms in the beginning of the realisation. There are several solutions to this problem which depend on the simulation context. If realisation is used in the context of ship stability simulation without manoeuvring, ramp-up interval will not affect results of the simulation, because it is located on the border (too far away from the studied marine object). Alternative approach is to generate sea wavy surface on ramp-up interval with LH model and generate the rest of the realisation with AR model. If ship stability with manoeuvring is studied, then the interval may be simply discarded from the realisation (the size of the interval approximately equals the number of AR coefficients). However, this may lead to a loss of a very large number of points, because discarding occurs for each of three dimensions.

Performance of OpenMP and OpenCL implementations. Differences in models' parallel algorithms make them efficient on different processor architec-

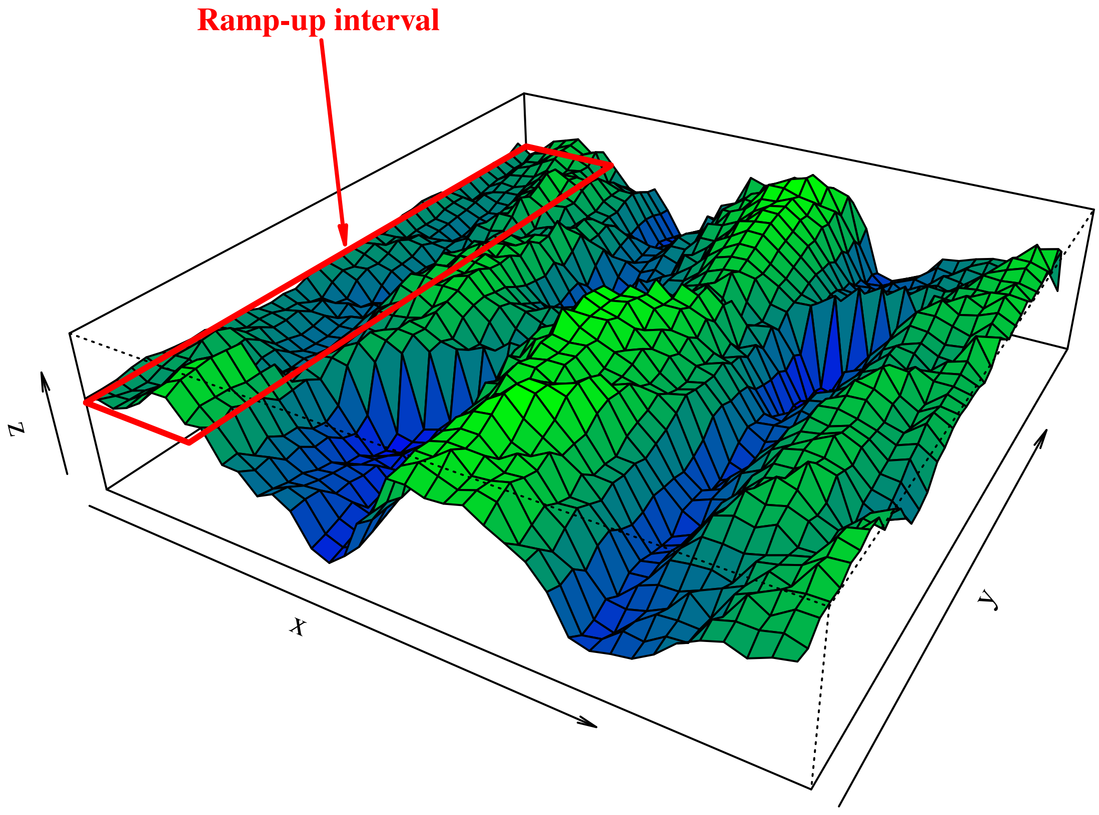


Figure 10: Ramp-up interval at the beginning of AR process realisation.

tures, and to find the most efficient one, all the models were benchmarked in both CPU and GPU.

AR and MA models do not require highly optimised codes to be efficient, their performance is high even without use of co-processors; there are two main causes of that. First, these models do not use transcendental functions (sines, cosines and exponents) as opposed to LH model. All calculations (except model coefficients) are done via polynomials, which can be efficiently computed on modern processors using FMA instructions. Second, pressure computation is done via explicit formula using a number of nested FFTs. Since two-dimensional FFT of the same size is repeatedly applied to every time slice, its coefficients (complex exponents) are pre-computed one time for all slices, and further computations involve only a few transcendental functions. In case of MA model, performance is also increased by doing convolution with FFT. So, high performance of AR and MA models is due to scarce use of transcendental functions and heavy use of FFT, not to mention that high convergence rate and non-existence of periodicity allows to use far fewer coefficients compared to LH model.

CPU	AMD FX-8370
RAM	16Gb
GPU	GeForce GTX 1060
GPU memory	6GB
HDD	WDC WD40EZRZ, 5400rpm
No. of CPU cores	4
No. of threads per core	2

Table 4: “Gpulab” system configuration.

Software implementation uses several libraries of mathematical functions, numerical algorithms and visualisation primitives (listed in table 5), and was implemented using several parallel programming technologies (OpenMP, OpenCL) to have a possibility to choose the most efficient one. For each technology and each model an optimised wavy surface generation was implemented (except for MA model for which only OpenMP implementation was done). Velocity potential computation was done in OpenMP and was implemented in OpenCL only for real-time visualisation of wavy surface. For each technology the programme was recompiled and run multiple times and performance of each top-level subroutine was measured using system clock.

Results of benchmarks of the technologies are summarised in table 6. All benchmarks were run on a machine equipped with a GPU, characteristics of which are summarised in table 4. All benchmarks were run with the same input parameters for all the models: realisation length 10000s and output grid size 40×40m. The only parameter that was different is the order (the number of coefficients): order of AR and MA model was $7, 7, 7$ and order of LH model was $40, 40$. This is due to higher number of coefficient for LH model to eliminate periodicity.

In all benchmarks wavy surface generation and NIT take the most of the running time, whereas velocity potential calculation together with other subroutines only a small fraction of it.

Library	What it is used for
DCMT [32]	parallel PRNG
Blitz [33,34]	multidimensional arrays
GSL [35]	PDF, CDF, FFT computation
	checking process stationarity
clFFT [36]	FFT computation
LAPACK, GotoBLAS [37,38]	finding AR coefficients
GL, GLUT [39]	three-dimensional visualisation
CGAL [40]	wave numbers interpolation

Table 5: A list of libraries used in software implementation.

AR model exhibits the highest performance in OpenMP and the lowest performance in OpenCL implementations, which is also the best and the worst performance across all model and framework combinations. In the most optimal model and framework combination AR performance is 4.5 times higher than MA performance, and 20 times higher than LH performance; in the most suboptimal combination — 137 times slower than MA and two times slower than LH. The ratio between the best (OpenMP) and the worst (OpenCL) AR model performance is several hundreds. This is explained by the fact that the model formula (4) is efficiently mapped on the CPU architecture, which is distinguished from GPU architecture by having multiple caches, low-bandwidth memory and small number of floating point units compared to GPU.

- This formula does not contain transcendental mathematical functions (sines, cosines and exponents),
- all of the multiplications and additions in the formula can be implemented using FMA processor instructions, and
- efficient use (locality) of cache is achieved by using Blitz library which implements optimised traversals for multidimensional arrays based on Hilbert space-filling curve.

In contrast to CPU, GPU has less number of caches, high-bandwidth memory and large number of floating point units, which is the worst case scenario for AR model:

- there are no transcendental functions which could compensate high memory latency,
- there are FMA instructions in GPU but they do not improve performance due to high latency, and
- optimal traversal of multidimensional arrays was not used due to a lack of libraries implementing it for a GPU.

Finally, GPU architecture does not contain synchronisation primitives that allow to implement autoregressive dependencies between distinct wavy surface parts; instead of this a separate OpenCL kernel is launched for each part, and information dependency management between them is done on CPU side. So, in AR model case

CPU architecture is superior compared to GPU due to better handling of complex information dependencies, simple calculations (multiplications and additions) and complex memory access patterns.

	OpenMP			OpenCL	
Subroutine	AR	MA	LH	AR	LH
Determine coefficients	0.02	0.01	0.19	0.01	1.19
Validate model	0.08	0.10		0.08	
Generate wavy surface	1.26	5.57	350.98	769.38	0.02
NIT	7.11	7.43		0.02	
Copy data from GPU				5.22	25.06
Compute velocity potentials	0.05	0.05	0.06	0.03	0.03
Write output to files	0.27	0.27	0.27	0.28	0.27

Table 6: Running time (s.) for OpenMP and OpenCL implementations of AR, MA and LH models.

In contrast to AR model, LH model exhibits the best performance on GPU and the worst performance on CPU. The reasons for that are

- the large number of transcendental functions in its formula which offset high memory latency,
- linear memory access pattern which allows to vectorise calculations and coalesce memory accesses by different hardware threads, and
- no information dependencies between wavy surface points.

Despite the fact that GPU on the test system is more performant than CPU (in terms of floating point operations per second), the overall performance of LH model compared to AR model is lower. The reason for that is slow data transfer between GPU and CPU memory.

MA model is faster than LH model and slower than AR model. As the convolution in its formula is implemented using FFT, its performance depends on the performance of FFT implementation: GSL for CPU and clFFT for GPU. In this work performance of MA model on GPU was not tested due to unavailability of the three-dimensional FFT in clFFT library; if the transform was available, it could made the model even faster than AR.

NIT takes less time on GPU and more time on CPU, but taking data transfer between them into consideration makes their execution time comparable. This is explained by the large amount of transcendental functions that need to be computed for each wavy surface point to transform distribution of its z-coordinates. For each point a non-linear transcendental equation (7) is solved using bisection method. GPU performs this task several hundred times faster than CPU, but spends a lot of time to transfer the result back to the processor memory. So, the only possibility to optimise this routine is to use root finding method with quadratic convergence rate to reduce the number of transcendental functions that need to be computed.

I/O performance. Although, in the benchmarks from the previous section writing data to files does not consume much of the running time, the use of network-mounted file systems may slow down this stage of the programme. To optimise it wavy surface parts are written to a file as soon as generation of the whole time slice is completed (fig. 11): a separate thread starts writing to files as soon as the first time slice is available and finishes it after the main thread group finishes the computation. The total time spent to perform I/O is increased, but the total programme running time is decreased, because the I/O is done in parallel to computation (table 7). Using this approach with local file system has the same effect, but performance gain is lower.

	I			II		
Subroutine	XFS	NFS	GlusterFS	XFS	NFS	GlusterFS
Generate wavy surface	1.26	1.26	1.33	1.33	3.30	11.06
Write output to files	0.28	2.34	10.95	0.00	0.00	0.00

Table 7: Running time of subroutines (s.) with the use of XFS, NFS and GlusterFS along with sequential (I) and parallel (II) file output.

5.1.2 Velocity potential field computation

Parallel velocity potential field computation. The benchmarks for AR, MA and LH models showed that velocity potential field computation consumes only a fraction of total programme execution time, however, the absolute computation time over a dense XY grid may be greater. One application where dense grid is used is real-time simulation and visualisation of wavy surface. Real-time visualisation allows to

- adjust parameters of the model and ACF function, getting the result of the changes immediately, and
- compare the size and the shape of regions where the most wave energy is concentrated.

Since visualisation is done by GPU, doing velocity potential computation on CPU may cause data transfer between memory of these two devices to become a bottleneck. To circumvent this, the programme uses OpenCL/OpenGL interoperability API which allows creating buffers, that are shared between OpenCL and OpenGL contexts thus eliminating copying of data between devices. In addition to this, three-dimensional velocity potential field formula (26) is particularly suitable for computation by GPUs:

- it contains transcendental functions (hyperbolic cosines and complex exponents);
- it is computed over large four-dimensional (t,x,y,z) region;
- it is explicit with no information dependencies between individual points in t and z dimensions.

These considerations make velocity potential field computation on GPU advantageous in the application to real-time simulation and visualisation of wavy surface.

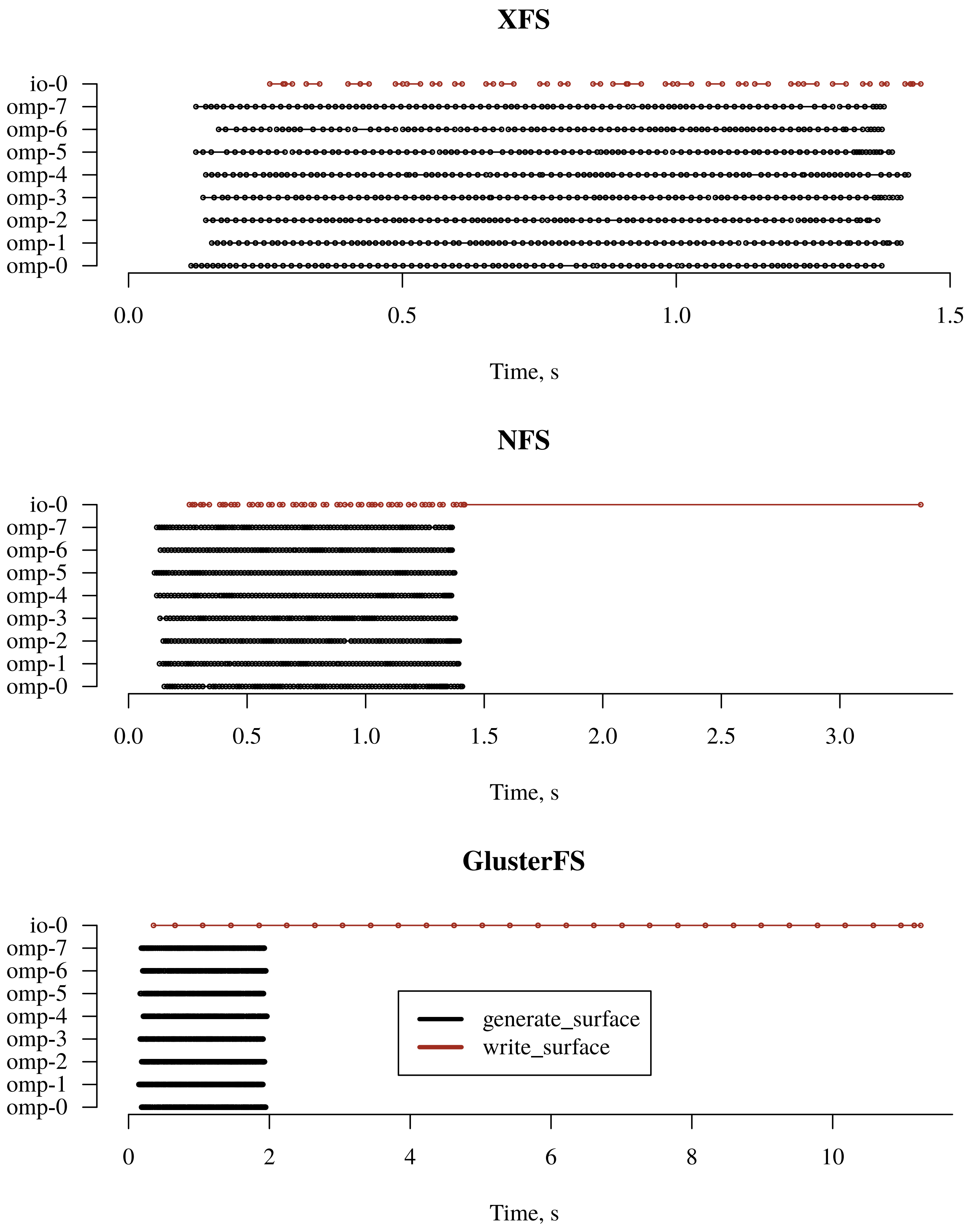


Figure 11: Event graph for XFS, NFS and GlusterFS that shows time intervals spent for I/O (red) and computation (black) by different threads.

In order to investigate, how much the use of GPU can speed-up velocity potential field computation, we benchmarked simplified version of (26):

$$\begin{aligned}\phi(x,y,z,t) &= \mathcal{F}^{-1}_{x,y}\left\{ \frac{\cosh\left(2\pi|\vec{k}|(z+h)\right)}{2\pi|\vec{k}|\cosh\left(2\pi|\vec{k}|h\right)} \mathcal{F}_{u,v}\{\zeta_t\} \right\} \\ &= \mathcal{F}^{-1}_{x,y}\{g_1(u,v)\mathcal{F}_{u,v}\{g_2(x,y)\}\}\,. \end{aligned} \tag{27}$$

Velocity potential computation code was rewritten in OpenCL and its performance was compared to an existing OpenMP implementation.

For each implementation running time of the corresponding subroutines and time spent for data transfer between devices was measured. Velocity potential field was computed for one t point, for 128 z points below wavy surface and for each x and y point of four-dimensional (t,x,y,z) grid. Between programme runs the size of the grid along x dimension was varied.

A different FFT library was used for each implementation: GNU Scientific Library (GSL) [41] for OpenMP and clFFT [36] for OpenCL. FFT routines from these libraries have the following features:

- The order of frequencies in FFT is different. In case of clFFT library elements of the resulting array are additionally shifted to make it correspond to the correct velocity potential field. In case of GSL no shift is needed.

- Discontinuity at $(x,y) = (0,0)$ is handled automatically by clFFT library, whereas GSL library produce skewed values at this point, thus in case of GSL these points are interpolated.

Other differences of FFT subroutines, that have impact on performance, were not discovered.

Performance of velocity potential solver. The experiments showed that OpenCL outperforms OpenMP implementation by a factor of 2–6 (fig. 12), however, distribution of running time between subroutines is different (table 9). For

CPU the most of the time is spent to compute g_1, whereas for GPU time spent to compute g_1 is comparable to g_2. Copying the resulting velocity potential field between CPU and GPU consumes $\approx 20\%$ of the total field computation time. Computing g_2 consumes the most of the execution time for OpenCL and the least of the time for OpenMP. In both implementations g_2 is computed on CPU, because subroutine for derivative computation in OpenCL was not found. For OpenCL the result is duplicated for each z grid point in order to perform multiplication of all XYZ planes along z dimension in single OpenCL kernel, and, subsequently copied to GPU memory which has negative impact on performance. All benchmarks were run on a machine equipped with a GPU, characteristics of which are summarised in table 8.

CPU	Intel Core 2 Quad Q9550
RAM	8Gb
GPU	AMD Radeon R7 360
GPU memory	2GB
HDD	Seagate Barracuda, 7200 rpm
No. of CPU cores	4

Table 8: “Storm” system configuration.

The reason for different distribution of time between OpenCL and OpenMP subroutines is the same as for different AR model performance on CPU and GPU: GPU has more floating point units and modules for transcendental mathematical functions, than CPU, which are needed for computation of g_1, but lacks caches which are needed to optimise irregular memory access pattern of g_2. In contrast to AR model, performance of multidimensional derivative computation on GPU is easier to improve, as there are no information dependencies between points: in this work optimisation was not done due to unavailability of existing implementation. Additionally, such library may allow to efficiently compute the non-simplified formula entirely on GPU, since omitted terms also contain derivatives.

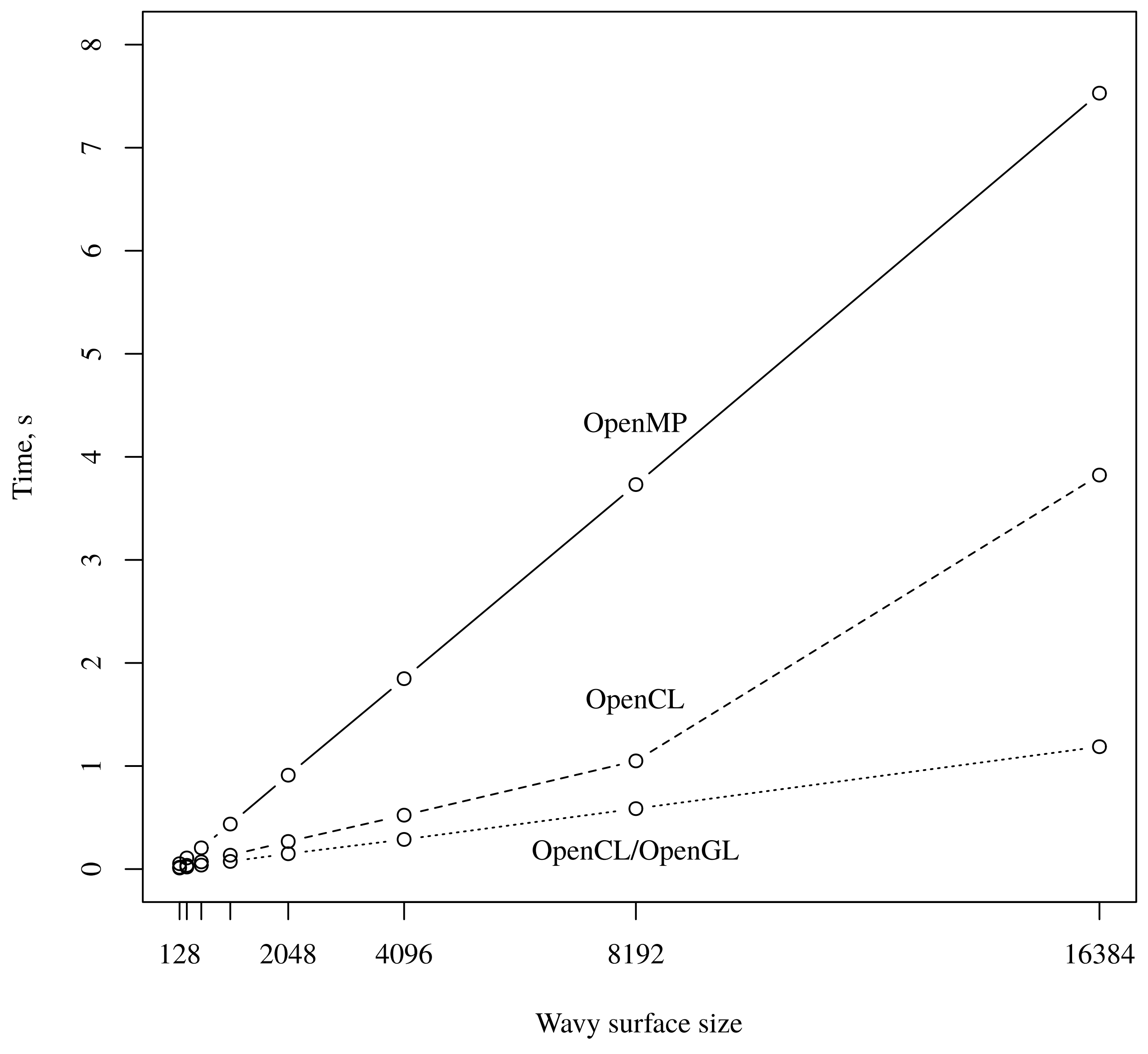


Figure 12: Performance comparison of CPU (OpenMP) and GPU (OpenCL) versions of velocity potential calculation code.

Subroutine	OpenMP	OpenCL
g_1 function	4.6730	0.0038
g_2 function	0.0002	0.8253
FFT	2.8560	0.3585
Copy data from GPU		2.6357

Table 9: Running time (s.) of real-time velocity potential calculation subroutines for wavy surface (OX size equals 16384).

As expected, sharing the same buffer between OpenCL and OpenGL contexts increases overall solver performance by eliminating data transfer between CPU and GPU memory, but also requires for the data to be in vertex buffer object format, that OpenGL can operate on. Conversion to this format is fast, but the resulting array occupies more memory, since each point now is a vector with three components. The other disadvantage of using OpenCL and OpenGL together is the requirement for manual locking of shared buffer: failure to do so results in appearance of screen image artefacts which can be removed only by rebooting the computer.

5.1.3 Summary

Benchmarks showed that GPU outperforms CPU in arithmetic intensive tasks, i.e. tasks requiring large number of floating point operations per second, however, its performance degrades when the volume of data that needs to be copied between CPU and GPU memory increases or when memory access pattern differs from linear. The first problem may be solved by using a co-processor where high-bandwidth memory is located on the same die together with the processor and the main memory. This eliminates data transfer bottleneck, but may also increase execution time due to smaller number of floating point units. The second problem may be solved programmatically, if OpenCL library that computes multi-dimensional derivatives were available.

AR and MA models outperform LH model in benchmarks and does not require GPU to do so. From computational point of view their strengths are

- absence of transcendental mathematical functions, and

- simple algorithm for both AR and MA model, performance of which depends on the performance of multi-dimensional array library and FFT library.

Providing main functionality via low-level libraries makes performance of the programme portable: support for new processor architectures can be added by substituting the libraries. Finally, using explicit formula makes velocity potential field computation consume only a small fraction of total programme execution time. If such formula did not exist or did not have all integrals as Fourier transforms, velocity potential field computation would consume much more time.

5.2 Fault-tolerant batch job scheduler

5.2.1 System architecture

Physical layer. Consists of nodes and direct/routed physical network links. On this layer full network connectivity, i.e. an ability to send network packets between any pair of cluster nodes.

Daemon process layer. Consists of daemon processes running on cluster nodes and hierarchical (master/slave) logical links between them. Only one daemon process is launched per node, so these terms are use interchangeably in this work. Master and slave roles are dynamically assigned to daemon processes, i.e. any physical cluster node may become a master or a slave, or both simultaneously. Dynamic role assignment uses leader election algorithm that does not require periodic broadcasting of messages to all cluster nodes, instead the role is determined by node’s IP address. Detailed explanation of the algorithm is provided in 5.2.2. Its strengths are scalability to a large number of nodes and low overhead, which makes it suitable for large-scale high-performance computations; its weakness is in artificial dependence of node's position in the hierarchy on its IP address, which makes it unsuitable for virtual environments, where nodes' IP addresses may change dynamically.

The only purpose of tree hierarchy is to balance the load between cluster nodes. The load is distributed from the current node to its neighbours in the hierarchy by simply iterating over them. When new links appear in the hierarchy or old links change (due to new node joining the cluster or due to node failure), daemon processes communicate each other the number of processes *behind* the corresponding link in the hierarchy. This information is used to distribute the load evenly, even if distributed programme is launched on slave node. In addition, this hierarchy reduces the number of simultaneous network connections between nodes: only one network connection is established and maintained for each hierarchical link — this decreases probability of network congestion when there is a large number of nodes in the cluster.

Load balancing is implemented as follows. When node A tries to become a subordinate of node B, it sends a message to a corresponding IP address telling how many daemon processes are already linked to it in the hierarchy (including itself). After all hierarchical links are established, every node has enough information to tell, how many nodes exist behind each link. If the link is between a slave and a master, and the slave wants to know, how many nodes are behind the link to the master, then it subtracts the total number of nodes behind all of its slave nodes from the total number of nodes behind the master to get the correct amount. To distribute kernels (see sec. 5.2.1) across nodes simple round-robin algorithm is used, i.e. iterate over all links of the current node including the link to its master and taking into account the number of nodes behind each link. So, even if a programme is launched on a slave node in the bottom of the hierarchy, the kernels are distributed evenly between all cluster nodes.

The proposed approach can be extended to include sophisticated logic into load distribution algorithm. Along with the number of nodes behind the hierarchical link, any metric, that is required to implement this logic, can be sent. However, if an update of the metric happens more frequently, than a change in the hierarchy occurs, or it changes periodically, then status messages will be also sent more frequently. To ensure that this transfers do not affect programmes' performance, a separate physical network may be used to transmit the messages. The other disadvantage is that when reconfiguration of the hierarchy occurs, the kernels that are already sent to the nodes are not taken into account in the load distribution, so frequent hierarchy changes may cause uneven load on cluster nodes (which, however, balances over time).

Dynamic master/slave role assignment coupled with kernel distribution makes overall system architecture homogeneous within the framework of single cluster. On every node the same daemon is run, and no prior configuration is needed to construct a hierarchy of daemons processes running on different nodes.

Control flow objects layer. The key feature that is missing in the current parallel programming technologies is a possibility to specify hierarchical dependencies between tasks executed in parallel. When such dependency exists, the principal task becomes responsible for re-executing a failed subordinate task on the survived cluster nodes. To re-execute a task which does not have a principal, a copy of it is created and sent to a different node (see sec. 5.2.3). There exists a number of systems that are capable of executing directed acyclic graphs of tasks in parallel [42,43], but graphs are not suitable to determine principal-subordinate relationship between tasks, because a node in the graph may have multiple incoming edges (and hence multiple principal nodes).

The main purpose of the proposed approach is to simplify development of distributed batch processing applications and middleware. The idea is to provide resilience to failures on the lowest possible level. The implementation is divided into two layers: the lower layer consists of routines and classes for single node applications (with no network interactions), and the upper layer for applications that run on an arbitrary number of nodes. There are two kinds of tightly coupled entities — *control flow objects* (or *kernels* for short) and *pipelines* — which form a basis of the programme.

Kernels implement control flow logic in theirs `act` and `react` methods and store the state of the current control flow branch. Both logic and state are implemented by a programmer. In `act` method some function is either directly computed or decomposed into nested function calls (represented by a set of subordinate kernels) which are subsequently sent to a pipeline. In `react` method subordinate kernels that returned from the pipeline are processed by their parent. Calls to `act` and `react` methods are asynchronous and are made within threads attached to a pipeline. For each kernel `act` is called only once, and for multiple kernels the calls are done in parallel to each other, whereas `react` method is called once for each subordinate kernel, and all the calls are made in the same thread to prevent race conditions (for different parent kernels different threads may be used).

Pipelines implement asynchronous calls to `act` and `react`, and try to make as many parallel calls as possible considering concurrency of the platform (no. of cores per node and no. of nodes in a cluster). A pipeline consists of a kernel pool, which contains all the subordinate kernels sent by their parents, and a thread pool that processes kernels in accordance with rules outlined in the previous paragraph. A separate pipeline is used for each device: There are pipelines for parallel processing, schedule-based processing (periodic and delayed tasks), and a proxy pipeline for processing of kernels on other cluster nodes (see fig. 13).

In principle, kernels and pipelines machinery reflect the one of procedures and call stacks, with the advantage that kernel methods are called asynchronously and in parallel to each other (as much as programme logic allows). Kernel fields are stack local variables, `act` method is a sequence of processor instructions before nested procedure call, and `react` method is a sequence of processor instructions after the call. Constructing and sending subordinate kernels to the pipeline is nested procedure call. Two methods are necessary to make calls asynchronous, and replace active wait for completion of subordinate kernels with passive one. Pipelines, in turn, allow to implement passive wait, and call correct kernel methods by analysing their internal state.

Software implementation. For efficiency reasons object pipeline and fault tolerance techniques (which will be described later) are implemented in the C++ framework: From the author's perspective C language is deemed low-level for distributed programmes, and Java incurs too much overhead and is not popular in HPC. The framework is called Bscheduler, it is now in proof-of-concept development stage.

Application programming interface. Each kernel has four types of fields (listed in table 10):

- fields related to control flow,
- fields defining the source location of the kernel,
- fields defining the current location of the kernel, and
- fields defining the target location of the kernel.

Upon creation each kernel is assigned a parent and a pipeline. If there no other fields are set, then the kernel is an *upstream* kernel — a kernel that can be distributed on any node and any processor core to exploit parallelism. If principal field is set, then the kernel is a *downstream* kernel — a kernel that can only be sent to its principal, and a processor core to which the kernel is sent is defined by the principal memory address/identifier. If a downstream kernel is to be sent to another node, the destination IP-address must be set, otherwise the system will not find the target kernel.

When kernel execution completes (its `act` method finishes), the kernel is explicitly sent to some other kernel, this directive is explicitly called inside `act` method. Usually, after the execution completes a kernel is sent to its parent by

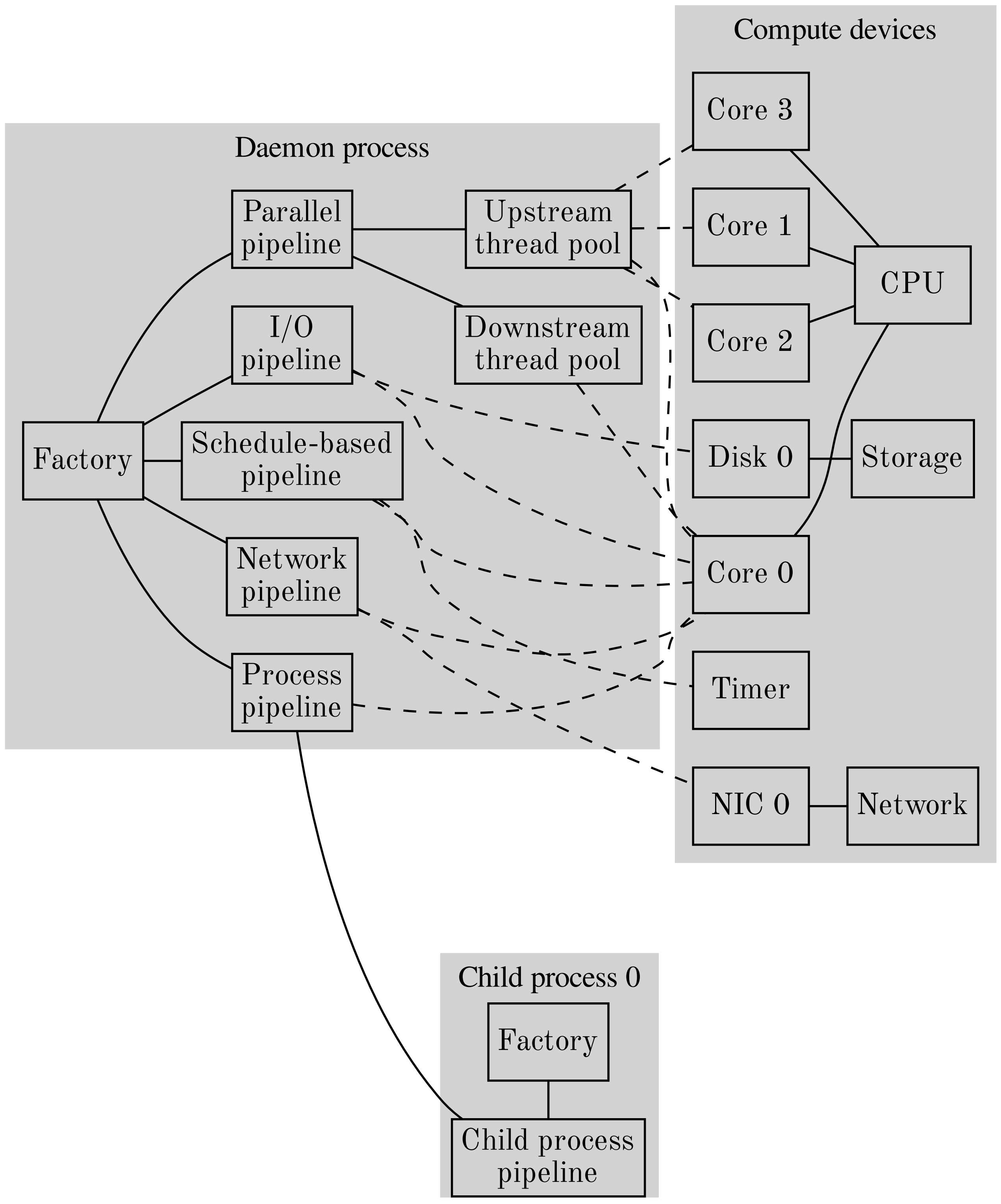


Figure 13: Mapping of parent and child process pipelines to compute devices. Solid lines denote aggregation, dashed lines — mapping between logical and physical entities.

Field	Description
`process_id`	Identifier of a cluster-wide process (application) a kernel belongs to.
`id`	Identifier of a kernel within a process.
`pipeline`	Identifier of a pipeline a kernel is processed by.
`exit_code`	A result of a kernel execution.
`flags`	Auxiliary bit flags used in scheduling.
`time_point`	A time point at which a kernel is scheduled to be executed.
`parent`	Address/identifier of a parent kernel.
`src_ip`	IP-address of a source cluster node.
`from_process_id`	Identifier of a cluster-wide process from which a kernel originated.
`principal`	Address/identifier of a target kernel (a kernel to which the current one is sent or returned).
`dst_ip`	IP-address of a destination cluster node.

Table 10: Description of kernel fields.

setting `principal` field to the address/identifier of the parent, destination IP-address field to the source IP-address, and process identifier to the source process identifier. After that kernel becomes a downstream kernel and is sent by the system to the node, where its current principal is located without invoking load balancing algorithm. For downstream kernel `react` method of its parent is called by a pipeline with the kernel as the method argument to make it possible for a parent to collect the result of the execution.

There is no way to provide fine-grained resilience to cluster node failures, if there are downstream kernels in the programme, except the ones returning to their parents. Instead, an exit code of the kernel is checked and a user-provided recovery subroutine is executed. If there is no error checking, the system restarts execution from the first parent kernel, which did not produce any downstream kernels. This means, that if a problem being solved by the programme has information dependencies between parts that are computed in parallel, and a node failure occurs during computation of these parts, then this computation is restarted from the very beginning, discarding any already computed parts. This does not occur for embarrassingly parallel programmes, where parallel parts do not have such information dependencies between each other: in this case only parts from failed nodes are recomputed and all previously computed parts are retained.

Unlike `main` function in programmes based on MPI library, the first (the main) kernel is initially run only on one node, and other cluster nodes are used on-demand. This allows to use more nodes for highly parallel parts of the code, and less nodes for other parts. Similar choice is made in the design of big data frameworks [44, 45] — a user submitting a job does not specify the number of hosts to run its job on, and actual hosts are the hosts where input files are located.

Mapping of simulation models on system architecture. Software implementation of AR and MA models works as a computational pipeline, in which each joint applies some function to the output coming from the pipe of the previous joint. Joints are distributed across computer cluster nodes to enable function parallelism, and then data flowing through the joints is distributed across processor cores to enable data parallelism. Figure 14 shows a diagram of such pipeline. Here rectangles with rounded corners denote joints, regular rectangles denote arrays of problem domain objects flowing from one joint to another, and arrows show flow direction. Some joints are divided into *sections* each of which process a separate part of the array. If joints are connected without a *barrier* (horizontal or vertical bar), then transfer of separate objects between them is done in parallel to computations, as they become available. Sections work in parallel on each processor core (or node of the cluster). There is surjective mapping between a set of processor cores, a set of pipeline joint sections and objects, i.e. each processor core may run several sections, each of which may sequentially process several objects, but a section can not work simultaneously on several processor cores, and an object can not be processed simultaneously by several sections. So, the programme is a pipeline through which control objects flow.

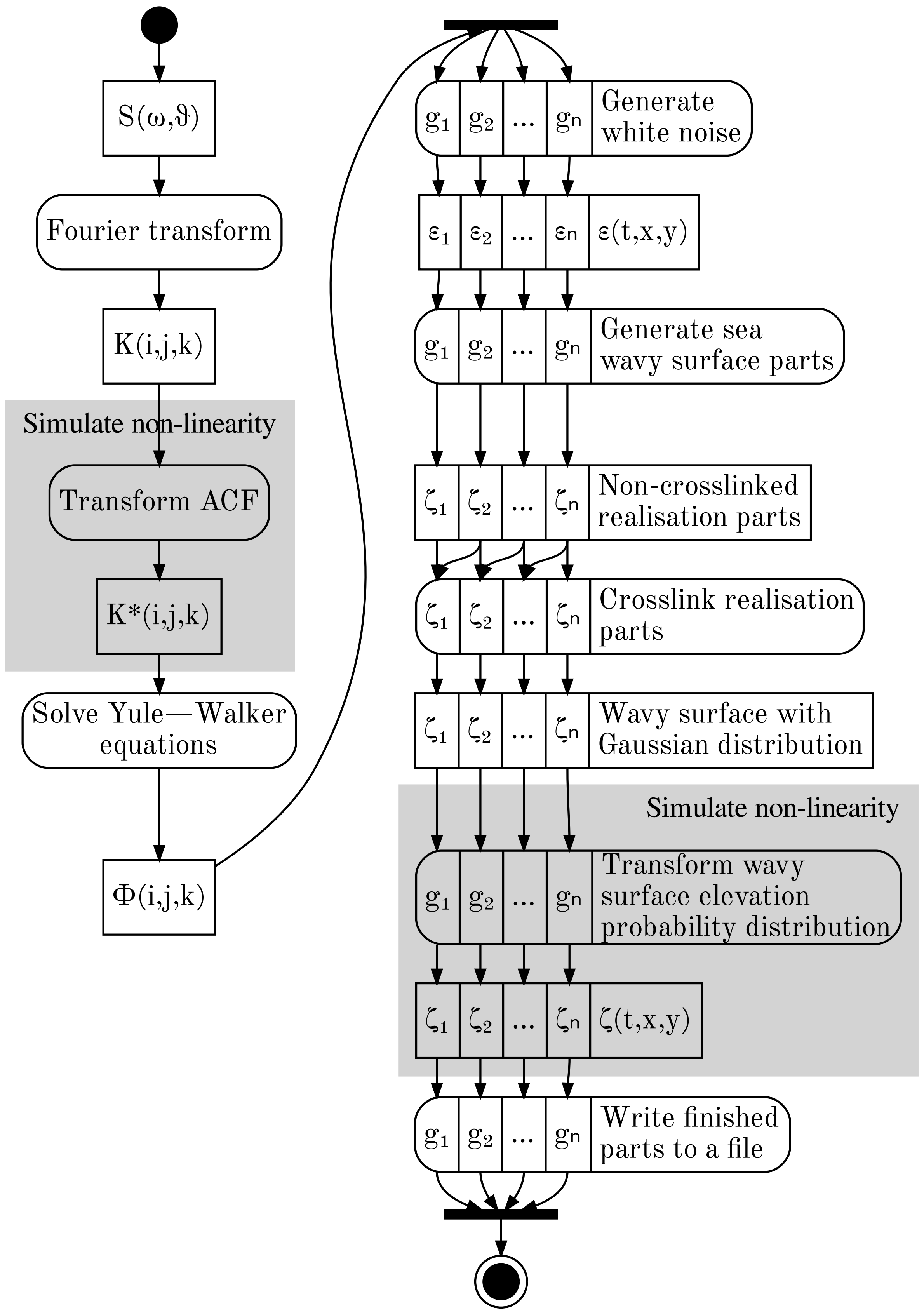

S(ω,ϑ)
Fourier transform
K(i,j,k)
Simulate non-linearity
Transform ACF
K*(i,j,k)
Solve Yule—Walker equations
Φ(i,j,k)
g1 g2 ... gn Generate white noise
ε1 ε2 ... εn ε(t,x,y)
g1 g2 ... gn Generate sea wavy surface parts
ζ1 ζ2 ... ζn Non-crosslinked realisation parts
ζ1 ζ2 ... ζn Crosslink realisation parts
ζ1 ζ2 ... ζn Wavy surface with Gaussian distribution
Simulate non-linearity
g1 g2 ... gn Transform wavy surface elevation probability distribution
ζ1 ζ2 ... ζn ζ(t,x,y)
g1 g2 ... gn Write finished parts to a file

Figure 14: Diagram of data processing pipeline, that implements sea wavy surface generation via AR model.

Object pipeline may be seen as an improvement of Bulk Synchronous Parallel (BSP) model [46], which is used in graph processing [47,48]. Pipeline eliminates global synchronisation (where it is possible) after each sequential computation step by doing data transfer between joints in parallel to computations, whereas in BSP model global synchronisation occurs after each step.

Object pipeline speeds up the programme by parallel execution of code blocks that work with different compute devices: while the current part of wavy surface is generated by a processor, the previous part is written to a disk. This approach allows to get speed-up (see sec. 5.1.1) because compute devices operate asynchronously, and their parallel usage increases the whole programme performance.

Since data transfer between pipeline joints is done in parallel to computations, the same pipeline may be used to run several copies of the application but with different parameters (generate several sea wavy surfaces having different characteristics). In practise, high-performance applications do not always consume 100% of processor time spending a portion of time on synchronisation of parallel processes and writing data to disk. Using pipeline in this case allows to run several computations on the same set of processes, and use all of the computer devices at maximal efficiency. For example, when one object writes data to a file, the other do computations on the processor in parallel. This minimises downtime of the processor and other computer devices and increases overall throughput of the computer cluster.

Pipelining of otherwise sequential steps is beneficial not only for code working with different devices, but for code different branches of which are suitable for execution by multiple hardware threads of the same processor core, i.e. branches accessing different memory blocks or performing mixed arithmetic (integer and floating point). Code branches which use different CPU modules are good candidates to run in parallel on a processor core with multiple hardware threads.

So, computational model with a pipeline can be seen as *bulk-asynchronous model*, because of the parallel nature of programme steps. This model is the basis of the fault-tolerance model which will be described later.

5.2.2 Cluster node discovery

Leader election algorithms. Many batch job scheduling systems are built on the principle of *subordination*: there is master node in each cluster which manages job queue, schedules job execution on subordinate nodes and monitors their state. Master role is assigned either *statically* by an administrator to a particular physical node, or *dynamically* by periodically electing one of the cluster nodes as master. In the former case fault tolerance is provided by reserving additional spare node which takes master role when current master fails. In the latter case fault tolerance is provided by electing new master node from survived nodes. Despite the fact that dynamic role assignment requires leader election algorithm, this approach becomes more and more popular as it does not require spare reserved nodes to recover from master node failure [49–51] and generally leads to a symmetric system architecture, in which the same software stack with the same configuration is installed on every node [52, 53].

Leader election algorithms (which sometimes referred to as *distributed consensus* algorithms are special cases of wave algorithms. In [54] the author defines them as algorithms in which termination event is preceded by at least one event occurring in *each* parallel process. Wave algorithms are not defined for anonymous networks, that is they apply only to processes that can uniquely define themselves. However, the number of processes affected by the “wave” can be determined in the course of an algorithm. For a distributed system this means that wave algorithms work for computer clusters with dynamically changing number of nodes, and the algorithm is unaffected by some nodes going on-line and off-line.

The approach for cluster node discovery does not use wave algorithms, and hence does not require communicating with each node of the cluster to determine a leader. Instead, each node enumerates all nodes in the network it is part of, and converts this list to a *tree hierarchy* with a user-defined *fan-out* value (maximal number of subordinate nodes a node may have). Then the node determines its hierarchy level and tries to communicate with nodes from higher levels to become their subordinate. First, it checks the closest ones and then goes all the way to the top. If there is no top-level nodes or the node cannot connect to them, then the node itself becomes the master of the whole hierarchy.

Tree hierarchy of all hosts in a network defines strict total order on a set of cluster nodes. Although, technically any function can be chosen to map a node to a number, in practise this function should be sufficiently smooth along the time axis and may have infrequent jumps: high-frequency oscillations (which are often caused by measurement errors) may result in constant passing of master role from one node to another, which makes the hierarchy unusable for load balancing. The simplest such function is the position of an IP address in network IP address range.

Tree hierarchy creation algorithm. Strict total order on the set $\mathcal{N}$ of cluster nodes connected to a network is defined as

$$\forall n_1 \forall n_2 \in \mathcal{N}, \forall f \colon \mathcal{N} \rightarrow \mathcal{R}^n \Rightarrow (f(n_1) < f(n_2) \Leftrightarrow \neg(f(n_1) \geq f(n_2))),$$

where f maps a node to its level and operator $<$ defines strict total order on $\mathcal{R}^n$. Function f defines node's sequential number, and $<$ makes this number unique.

The simplest function f maps each node to its IP address position in network IP address range. Without the use of tree hierarchy a node with the lowest position in this range becomes the master. If IP-address of a node occupies the first position in the range, then there is no master for it, and it continues to be at the top of the hierarchy until it fails. Although, IP address mapping is simple to implement, it introduces artificial dependence of the master role on the address of a node. Still, it is useful for initial configuration of a cluster when more complex mappings are not applicable.

To make node discovery scale to a large number of nodes, IP address range is mapped to a tree hierarchy. In this hierarchy each node is uniquely identified by its hierarchy level l, which it occupies, and offset o, which equals to the sequential number of node on its level. Values of level and offset are computed from the following optimisation problem.

$$n = \sum_{i=0}^{l(n)} p^i + o(n), \quad l \to \min, \quad o \to \min, \quad l \geq 0, \quad o \geq 0$$

where n is the position of node's IP address in network IP address range and p is fan-out value (the maximal number of subordinates, a node can have). The master of a node with level l and offset o has level $l - 1$ and offset $\lfloor o/p \rfloor$. The distance between any two nodes in the tree hierarchy with network positions i and j is computed as

$$\begin{aligned} &\langle \operatorname{lsub}(l(j), l(i)), \quad |o(j) - o(i)/p| \rangle, \\ &\operatorname{lsub}(l_1, l_2) = \begin{cases} \infty & \text{if } l_1 \geq l_2, \\ l_1 - l_2 & \text{if } l_1 < l_2. \end{cases} \end{aligned}$$

The distance is compound to account for level in the first place.

To determine its master, each node ranks all nodes in the network according to their position $\langle l(n), o(n) \rangle$, and using distance formula chooses the node which is closest to potential master position and has lower level. That way IP addresses of off-line nodes are skipped, however, for sparse networks (in which nodes occupy non-contiguous IP addresses) perfect tree is not guaranteed.

Formula for computing distance can be made arbitrary complex (e.g. to account for network latency and throughput or geographical location of the node), however, for its simplest form it is more efficient to use cluster node *traversal algorithm*. The algorithm requires less memory as there is no need to store ranked list of all nodes, but iterates over IP addresses of the network in the order defined by the fan-out value. The algorithm is as follows. First, the *base* node (a node which searches for its master) computes its potential master address and tries to connect to this node. If the connection fails, the base node sequentially tries to connect to each node from the higher hierarchy levels, until it reaches the top of the hierarchy (the root of the tree). If none of the connections succeed, the base node sequentially connects to all nodes on its own level having lower position in IP address range. If none of the nodes respond, the base node becomes the master node of the whole hierarchy, and the traversal repeats after a set period of time. An example of traversal order for a cluster of 11 nodes and a tree hierarchy with fan-out value of 2 is shown in fig. 15.

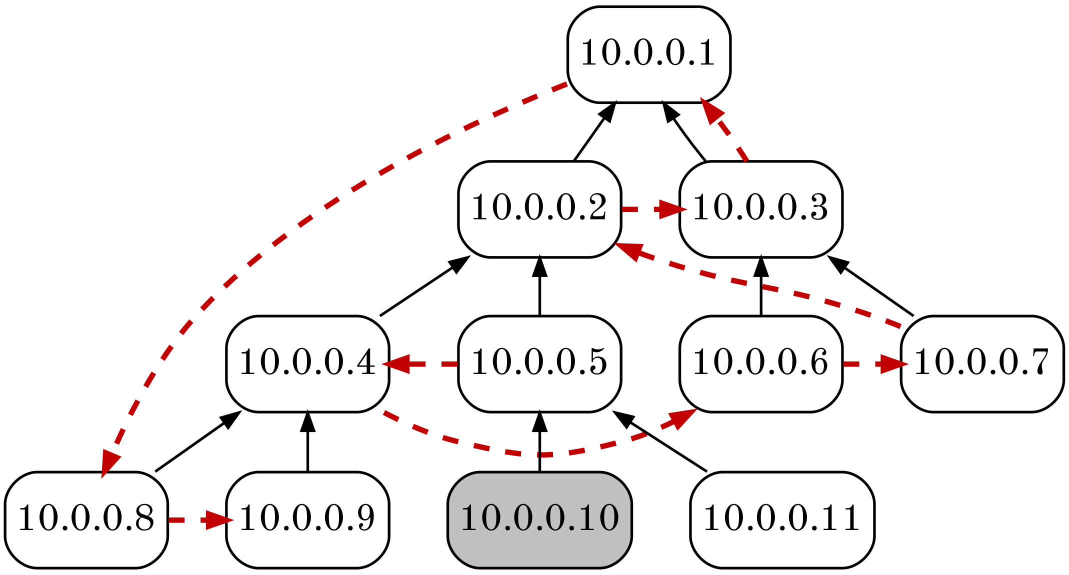


Figure 15: Tree hierarchy for 11 nodes with fan-out of 2. Red arrows denote hierarchy traversal order for a node with IP address 10.0.0.10.

Evaluation results. To benchmark performance of traversal algorithm on large number of nodes, several daemon processes were launched on each physical cluster node, each bound to its own IP address. The number of processes per physical core varied from 2 to 16. Each process was bound to a particular physical core to reduce overhead of process migration between cores. The algorithm has low requirements for processor time and network throughput, so running multiple processes per physical core is feasible, in contrast to HPC codes, where it often lowers performance. Test platform configuration is shown in table 11.

CPU	Intel Xeon E5440, 2.83GHz
RAM	4Gb
HDD	ST3250310NS, 7200rpm
No. of nodes	12
No. of CPU cores per node	8

Table 11: “Ant” system configuration.

Similar approach was used in in [55–57] where the authors reproduce various real-world experiments using virtual clusters, based on Linux namespaces, and compare the results to physical ones. The advantage of it is that the tests can be performed on a large virtual cluster using relatively small number of physical nodes. The advantage of the approach used in this work (which does not use Linux namespaces) is that it is more lightweight and larger number of daemon processes can be benchmarked on the same physical cluster.

Traversal algorithm performance was evaluated by measuring time needed for all nodes of the cluster to discover each other, i.e. the time needed for the tree hierarchy of nodes to reach stable state. Each change of the hierarchy, as seen by each node, was written to a log file and after a set amount of time all daemon processes (each of which models cluster node) were forcibly terminated. Daemon processes were launched sequentially with a 100ms delay to ensure that master nodes are always come online before subordinates and hierarchy does not change randomly as a result of different start time of each process. This artificial delay was subsequently subtracted from the results. So, benchmark results represent discovery time in an “ideal” cluster, in which every daemon process always finds its master on the first try.

The benchmark was run multiple times varying the number of daemon processes per cluster node. The experiment showed that discovery of 512 nodes (8 physical nodes with 64 processes per node) each other takes no more than two seconds (fig. 16). This value does not change significantly with the increase in the number of physical nodes. Using more than 8 nodes with 64 processes per node causes large variation in discovery time due to a large total number of processes connecting simultaneously to one master process (a fan-out value of 10000 was used for all tests), so these results were excluded from consideration.

Discussion. Traversal algorithm scales to a large number of nodes, because in order to determine its master, a node is required to communicate to a node address of which it knows beforehand. Communication with other nodes occurs only when the current master node fails. So, if cluster nodes occupy contiguous addresses in network IP address range, each node connects only to its master, and inefficient scan of the whole network by each node does not occur.

The following key features distinguish the proposed approach with respect to some existing approaches [58–60].

- **Multi-level hierarchy.** The number of master nodes in a network depends on the fan-out value. If it is lesser than the number of IP-addresses in the network, then there are multiple master nodes in the cluster. If it is greater or equal to the number of IP-addresses in the network, then there is only one master node. When some node fail, multi-level hierarchy changes locally, and only nodes that are adjacent to the failed one communicate. However, node weight changes propagate to every node in the cluster via hierarchical links.

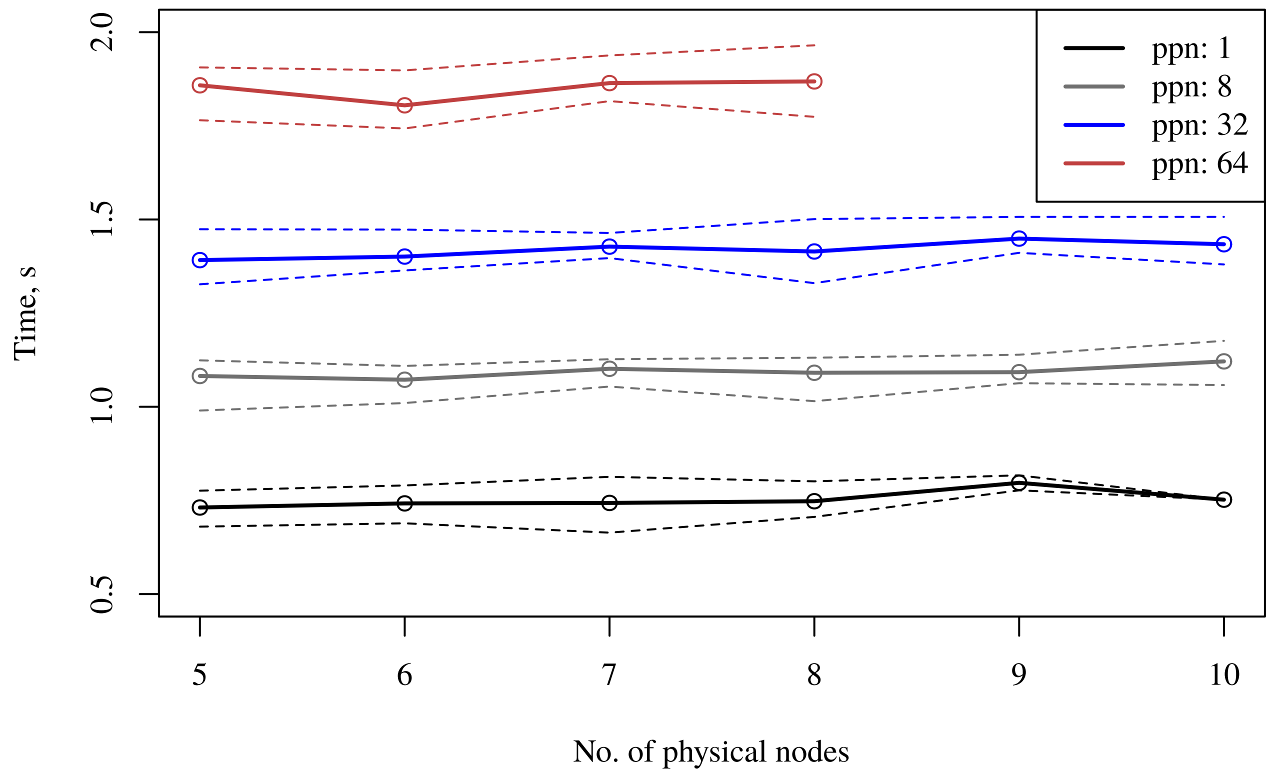


Figure 16: Time needed to discover all daemon processes running on the cluster depending on the number of daemon processes. Dashed lines denote minimum and maximum values, “ppn” is the number of daemon processes per node.

- **IP-address mapping.** Since hierarchy structure solely depends on the nodes' IP addresses, there is no election phase in the algorithm. To change the master each node sends a message to the old master and to the new one.

- **Completely event-based.** The messages are sent only when some node joins the cluster or fails, so there is no constant load on the network. Since the algorithm allows to tolerate failure of sending any message, there is no need in heartbeat packets indicating node serviceability in the network; instead, all messages play the role of heartbeats and packet send time-out is adjusted [61].

- **No manual configuration.** A node does not require any prior knowledge to find the master: it determines the network it is part of, calculates potential master IP-address and sends the message. If it fails, the process is repeated for the next potential master node. So, the algorithm is able to bootstrap a cluster (make all nodes aware of each other) without prior manual configuration, the only requirement is to assign an IP address to each node and start the daemon process on it.

To summarise, the advantage of the algorithm is that it

- scales to a large number of nodes by means of hierarchy with multiple master nodes,

- does not constantly load the network with node status messages and heartbeat packets,

- does not require manual configuration to bootstrap a cluster.

The disadvantage of the algorithm is that it requires IP-address to change infrequently, in order to be useful for load balancing. It is not suitable for cloud environments in which node DNS name is preserved, but IP-address may change over time. When IP-address changes, current connections may close, and node hierarchy is rebuilt. After each change in the hierarchy only newly created kernels are distributed using the new links, old kernels continue to execute on whatever nodes they were sent. So, environments where an IP address is not an unique identifier of the node are not suitable for the algorithm.

The other disadvantage is that the algorithm creates artificial dependence of node rank on an IP-address: it is difficult to substitute IP-address mapping with a more sophisticated one. If the mapping uses current node and network load to infer node ranks, measurement errors may result in unstable hierarchy, and the algorithm cease to be fully event-based as load levels need to be periodically collected on every node and propagated to each node in the cluster.

Node discovery algorithm is designed to balance the load on a cluster of compute nodes (see sec. 5.2.1), its use in other applications is not studied here. When distributed or parallel programme starts on any of cluster nodes, its kernels are distributed to all adjacent nodes in the hierarchy (including master node if applicable). To distribute the load evenly when the application is run on a subordinate node, each node maintains the weight of each adjacent node in the hierarchy. The weight equals to the number of nodes in the tree "behind" the adjacent node. For example, if the weight of the first adjacent node is 2, then round-robin load balancing algorithm distributes two kernels to the first node before moving to the next one.

To summarise, traversal algorithm is

- designed to ease load balancing on the large number of cluster nodes,
- fully fault-tolerant, because the state of every node can be recomputed at any time,
- fully event-based as it does not overload the network by periodically sending state update messages.

5.2.3 Fail over algorithm

Checkpoints. Node failures in a distributed system are divided into two types: failure of a slave node and failure of a master node. In order for a job running on the cluster to survive slave node failure, job scheduler periodically creates checkpoints and writes them to a stable (redundant) storage. In order to create the checkpoint, the scheduler temporarily suspends all parallel processes of the job, copies all memory pages and all internal operating system kernel structures allocated for these processes to disk, and resumes execution of the job. In order to survive master node failure, job scheduler daemon process continuously copies its internal state to a backup node, which becomes the master after the failure.

There are many works on improving performance of checkpoints [62], and alternative approaches do not receive much attention. Usually HPC applications use message passing for parallel processes communication and store their state in global memory space, hence there is no way one can restart a failed process from its current state without writing the whole memory image to disk. Usually the total number of processes is fixed by the job scheduler, and all parallel processes restart upon a failure. There is ongoing effort to make it possible to restart only the failed process [63] by restoring them from a checkpoint on the surviving nodes, but this may lead to overload if there are other processes on these nodes. Theoretically, process restart is not needed, if the job can proceed on the surviving nodes, however, message passing library does not allow to change the number of processes at runtime, and most programmes assume this number to be constant. So, there is no reliable way to provide fault tolerance in message passing library other than restarting all parallel processes from a checkpoint.

At the same time, there is a possibility to continue execution of a job on lesser number of nodes than it was initially requested by implementing fault tolerance on job scheduler level. In this case master and slave roles are dynamically distributed between scheduler's daemon processes running on each cluster node, forming a tree hierarchy of cluster nodes, and parallel programme consists of kernels which use node hierarchy to dynamically distribute the load and use their own hierarchy to restart kernels upon node failure.

Dynamic role distribution. Fault tolerance of a parallel programme is one of the problems which is solved by big data and HPC job schedulers, however, most schedulers provide fault tolerance for slave nodes only. These types of failures are routinely handled by restarting the affected job (from a checkpoint) or its part on the remaining nodes, and failure of a principal node is often considered either improbable, or too complicated to handle and configure on the target platform. System administrators often find alternatives to application level fault tolerance: they isolate principal process of the scheduler from the rest of the cluster nodes by placing it on a dedicated machine, or use virtualisation technologies instead. All these alternatives complexify configuration and maintenance, and by decreasing probability of a machine failure resulting in a whole system failure, they increase probability of a human error.

From such point of view it seems more practical to implement master node fault tolerance at application level, but there is no proven generic solution. Most implementations are too tied to a particular application to become universally applicable. Often a cluster is presented as a machine, which has a distinct control unit (master node) and a number of identical compute units (slave nodes). In practice, however, master and slave nodes are physically the same, and are distinguished only by batch job scheduler processes running on them. Realisation of the fact that cluster nodes are merely compute units allows to implement middleware that distributes master and slave roles automatically and handles node failures in a generic way. This software provides an API to distribute kernels between currently available nodes. Using this API one can write a programme that runs on a cluster without knowing the exact number of working nodes. The middleware works as a cluster operating system in user space, allowing to execute distributed applications transparently on any number of nodes.

Symmetric architecture. Many distributed key-value stores and parallel file systems have symmetric architecture, in which master and slave roles are dynamically distributed, so that any node can act as a master when the current master node fails [50–53, 64]. However, this architecture is still not used in big data and HPC job schedulers. For example, in YARN scheduler [45] master and slave roles are static. Failure of a slave node is tolerated by restarting a part of a job, that worked on it, on one of the surviving nodes, and failure of a master node is tolerated by setting up standby master node [65]. Both master nodes are coordinated by Zookeeper service which uses dynamic role assignment to ensure its own fault-tolerance [66]. So, the lack of dynamic role distribution in YARN scheduler complicates the whole cluster configuration: if dynamic roles were available, Zookeeper would be redundant in this configuration.

The same problem occurs in HPC job schedulers where master node (where the main job scheduler process is run) is the single point of failure. In [67, 68] the authors replicate job scheduler state to a backup node to make the master node highly available, but backup node role is assigned statically. This solution is close to symmetric architecture, because it does not involve external service to provide high availability, but far from ideal in which backup node is dynamically chosen.

Finally, the simplest master node high availability is implemented in VRRP protocol (Virtual Router Redundancy Protocol) [69–71]. Although VRRP protocol does provide dynamic role distribution, it is designed to be used by routers and reverse proxy servers behind them. Such servers lack the state (a job queue) that needs to be restored upon node failure, so it is easier for them to provide high availability. The protocol can be implemented even without routers using Keepalived daemon [72] instead.

Symmetric architecture is beneficial for job schedulers because it allows to

- make physical nodes interchangeable,
- implement dynamic distribution of master and slave node roles, and
- implement automatic recovery after a failure of any node.

The following sections describe the components that are required to write parallel programme and job scheduler, that can tolerate failure of cluster nodes.

Definitions of hierarchies. To disambiguate hierarchical links between daemon processes and kernels and to simplify the discussion, the following naming conventions are used throughout the text. If the link is between two daemon processes, the relationship is denoted as *master-slave*. If the link is between two kernels, then the relationship is denoted as either *principal-subordinate* or *parent-child*. Two hierarchies are orthogonal to each other in a sense that no kernel may have a link to a daemon, and vice versa. Since daemon process hierarchy is used to distribute the load on cluster nodes, kernel hierarchy is mapped onto it, and this mapping can be arbitrary: It is common to have principal kernel on a slave node with its subordinate kernels distributed evenly between all cluster nodes (including the node where the principal is located). Both hierarchies can be arbitrarily deep, but "shallow" ones are preferred for highly parallel programmes, as there are less number of hops when kernels are distributed between cluster nodes. Since there is one-to-one correspondence between daemons and cluster nodes, they are used interchangeably in the text.

Handling nodes failures. Basic method to overcome a failure of a subordinate node is to restart corresponding kernels on a healthy node (Erlang language uses the same method to restart failed subordinate processes [73]). In order to implement this method in the framework of kernel hierarchy, sender node saves every kernel that is sent to remote cluster nodes, and in an event of a failure of any number of nodes, where kernels were sent, their copies are redistributed between the remaining nodes without custom handling by a programmer. If there are no nodes to sent kernels to, they are executed locally. In contrast to "heavy-weight" checkpoint/restart machinery employed by HPC cluster job schedulers, tree hierarchy of nodes coupled with hierarchy of kernels allows to automatically and transparently handle any number of subordinate node failures without restarting any processes of a parallel programme, and processes act as resource allocation units.

A possible way of handling failure of the main node (a node where the main kernel is executed) is to replicate the main kernel to a backup node, and make all updates to its state propagate to the backup node by means of a distributed transaction, but this approach does not correlate with asynchronous nature of kernels and too complex to implement. In practice, however, the main kernel usually does not perform operations in parallel, it is rather sequentially executes programme steps one by one, hence it has only one subordinate at a time. (Each subordinate kernel represents sequential computational step which may or may not be internally parallel.) Keeping this in mind, one can simplify synchronisation of the main kernel state: send the main kernel along with its subordinate to the subordinate node. Then if the main node fails, the copy of the main kernel receives its subordinate (because both of them are on the same node) and no time is spent on recovery. When the subordinate node, to which subordinate kernel together with the copy of the main kernel was sent, fails, the subordinate kernel is sent to a survived node, and in the worst case the current computational step is executed again.

The approach described above is designed for kernels that do not have a parent and have only one subordinate at a time, which means that it functions as checkpoint mechanism. The advantage of this approach is that it

- saves programme state after each sequential step, when memory footprint of a programme is low,
- saves only relevant data, and
- uses memory of a subordinate node rather than disk storage.

This simple approach allows to tolerate at most one failure of *any* cluster node per computational step or arbitrary number of subordinate nodes at any time during programme execution.

For a four-node cluster and a hypothetical parallel programme failure recovery algorithm works as follows (fig. 17).

1. **Initial state.** Initially, computer cluster does not need to be configured except setting up local network. The algorithm assumes full connectivity of cluster nodes, and works best with tree network topologies in which several network switches connect all cluster nodes.

2. **Build node hierarchy.** When the cluster is bootstrapped, daemon processes start on all cluster nodes and collectively build their hierarchy superimposed on the topology of cluster network. Position of a daemon process in the hierarchy is defined by the position of its node IP address in the network IP address range. To establish hierarchical link each process connects to its assumed master process. The hierarchy is changed only when a new node joins the cluster or an existing node fails.

3. **Launch main kernel.** The first kernel launches on one of the subordinate nodes (node B). Main kernel may have only one subordinate at a time, and backup copy of the main kernel is sent along with the subordinate kernel T_1 to master node A. T_1 represents one sequential step of a programme. There can be any number of sequential steps in a programme, and when node A fails, the current step is restarted from the beginning.

4. **Launch subordinate kernels.** Kernels S_1, S_2, S_3 are launched on subordinate cluster nodes. When node B, C or D fails, corresponding main kernel restarts failed subordinates (T_1 restarts S_1, master kernel restarts T_1 etc.). When node B fails, master kernel is recovered from backup copy.

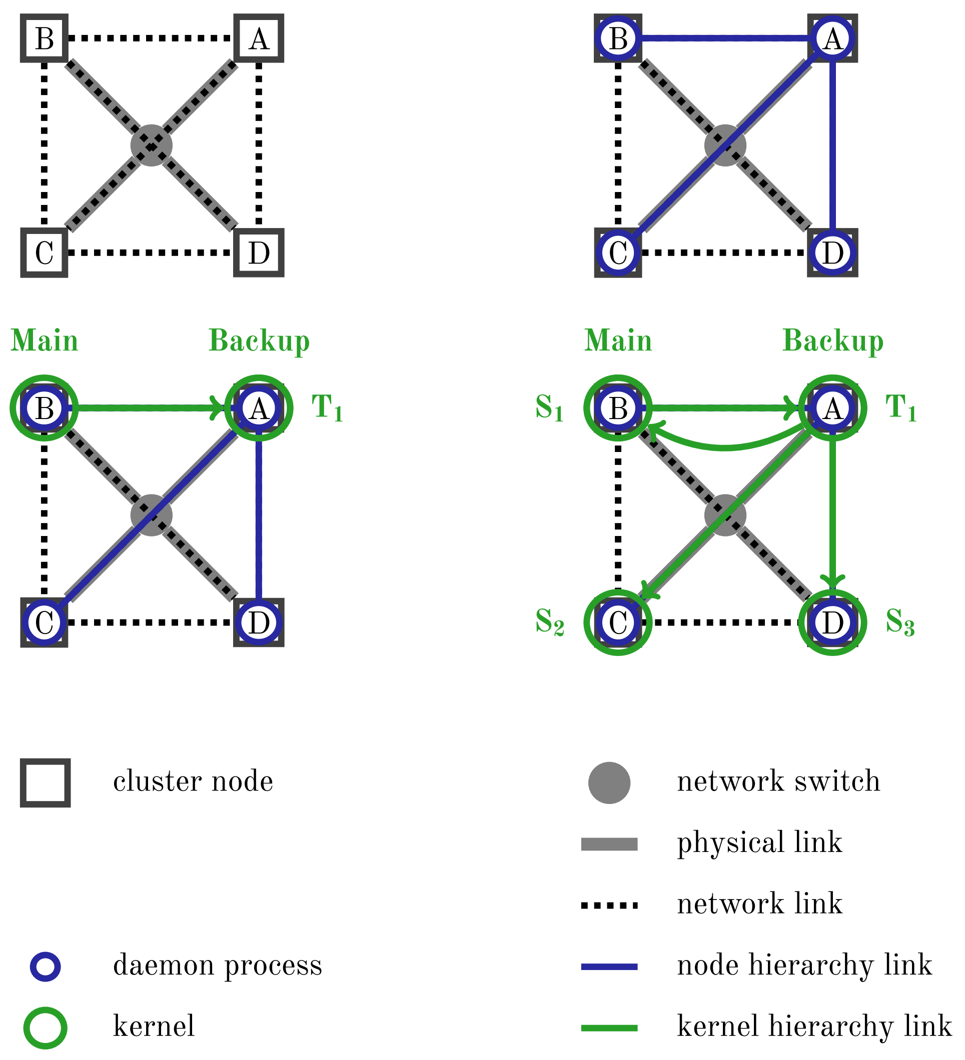


Figure 17: Working scheme of fail over algorithm.

Evaluation results. Fail over algorithm was evaluated on physical cluster (table 11) on the example of distributed AR model application, which is described in detail in section 5.3. The application consists of a series of functions, each of which is applied to the result of the previous one. Some of the functions are computed in parallel, so the programme is written as a sequence of steps, some of which are made internally parallel to get better performance. In the programme only the most compute-intensive step (the surface generation) is executed in parallel across all cluster nodes, and other steps are executed in parallel across all cores of the master node.

The application was rewritten for the distributed version of the framework which required adding read/write methods to each kernel which is sent over the network and slight modifications to handle failure of a node with the main kernel. The kernel was marked so that the framework makes a replica and sends it to some subordinate node along with its subordinate kernel. Other code changes involved modifying some parts to match the new API. So, providing fault tolerance by means of kernel hierarchy is mostly transparent to the programmer: it demands explicit marking of the main kernel and adding code to read and write kernels to the byte buffer.

In a series of experiments performance of the new version of the application in the presence of different types of failures was benchmarked:

- no failures,
- failure of a slave node (a node where a copy of the main kernel is stored),
- failure of a master node (a node where the main kernel is run).

Only two directly connected cluster nodes were used for the benchmark. Node failure was simulated by sending `SIGKILL` signal to the daemon process on the corresponding node right after the copy of the main kernel is made. The application immediately recognised node as offline, because the corresponding connection was closed; in real-world scenario, however, the failure is detected only after a configurable TCP user timeout [61]. The execution time of these runs were compared to the run without node failures. Results are presented in fig. 19, and a diagram of how kernels are distributed between two nodes is shown in fig. 18.

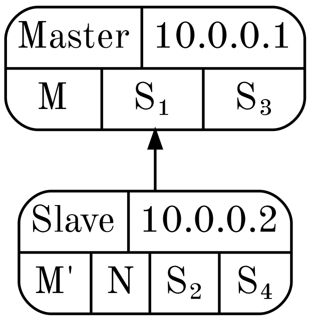


Figure 18: Master and slave nodes and mapping of main kernel M, a copy of main kernel M', current step kernel N and subordinate kernels $S_{1,2,3}$ on them.

As expected, there is considerable difference in application performance for different types of failures. In case of slave node failure the main kernel as well as some subordinate kernels (that were distributed to the slave node) are lost, but master node has a copy of the main kernel and uses it to continue the execution. So, in case of slave node failure nothing is lost except performance potential of the slave node. In case of master node failure, a copy of the main kernel as well as the subordinate kernel, which carried the copy, are lost, but slave node has the original main kernel and uses it to restart execution of the current sequential step, i.e. send the subordinate kernel to one of the survived cluster nodes (in case of two directly connected nodes, it sends the kernel to itself). So, application performance is different, because the number of kernels that are lost as a result of a failure as well as their roles are different.

Slave node failure needs some time to be discovered: it is detected only when subordinate kernel carrying a copy of the main kernel finishes its execution and tries to reach its parent. Instant detection requires forcible termination of the subordinate kernel which may be inapplicable for programmes with complicated logic.

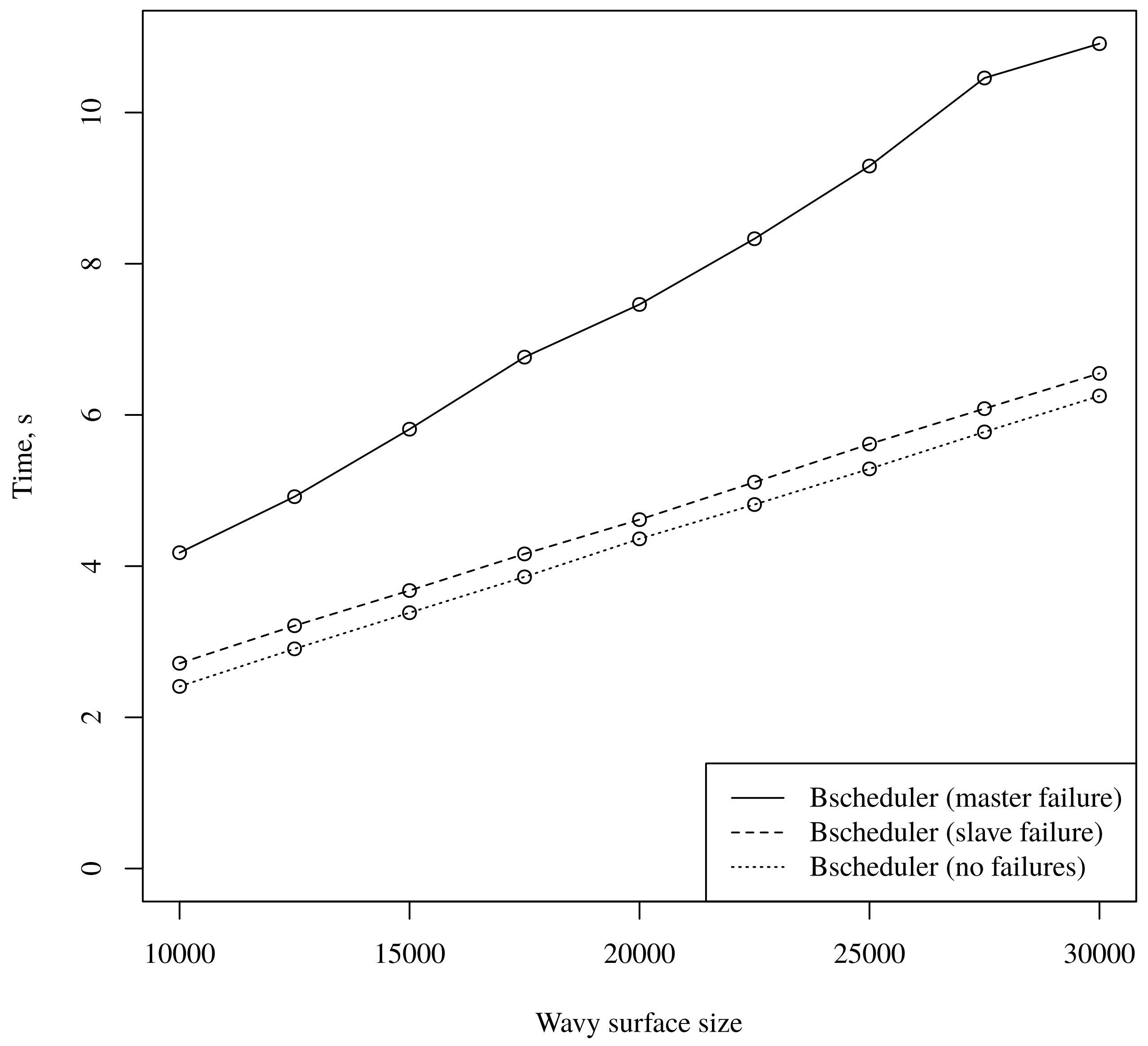


Figure 19: Performance of AR model in the presence of node failures.

To summarise, if failure occurs right after a copy if the main kernel is made, only a small fraction of performance is lost no matter whether the main kernel or its copy is lost.

Discussion of test results. Since failure is simulated right after the first subordinate kernel reaches its destination (a node where it is supposed to be executed), slave node failure results in a loss of a small fraction of performance; in real-world scenario, where failure may occur in the middle of wavy surface generation, performance loss due to *backup* node failure (a node where a copy of the main kernel is located) would be higher. Similarly, in real-world scenario the number of cluster nodes is larger, hence less amount of subordinate kernels is lost due to master node failure, so performance penalty is lower. In the benchmark the penalty is higher for the slave node failure, which is the result of absence of parallelism in the beginning of AR model wavy surface generation: the first part is computed sequentially, and other parts are computed only when the first one is available. So, the loss of the first subordinate kernel delays execution of every dependent kernel in the programme.

Fail over algorithm guarantees to handle one failure per sequential programme step, more failures can be tolerated if they do not affect the master node. The algorithm handles simultaneous failure of all subordinate nodes, however, if both master and backup node fail, there is no chance for a programme to continue the work. In this case the state of the current computation step is lost, and the only way to restore it is to restart the application from the beginning (which is currently not implemented in Bscheduler).

Kernels are means of abstraction that decouple distributed application from physical hardware: it does not matter how many cluster nodes are currently available for a programme to run without interruption. Kernels eliminate the need to allocate a physical live spare node to tolerate master node failures: in the framework of kernel hierarchy any physical node (except master) can act as a live spare. Finally, kernels allow to handle failures in a way that is transparent to a programmer, deriving the order of actions from the internal state of a kernel.

The experiments show that it is essential for a parallel programme to have multiple sequential steps to make it resilient to cluster node failures, otherwise failure of backup node, in fact triggers recovery of the initial state of the programme. Although, the probability of a master node failure is lower than the probability of a failure of any of the slave nodes, it does not justify loosing all the data when the long-running programme is near completion. In general, the more sequential steps one has in a parallel programme the less time is lost in an event of a backup node failure, and the more parallel parts each sequential step has the less time is lost in case of a master or slave node failure. In other words, the more nodes a programme uses the more resilient to their failures it becomes.

Although it is not shown in the experiments, Bscheduler does not only provide tolerance to cluster node failures, but allows for new nodes to automatically join the cluster and receive their portion of kernels from the already running programmes. This is trivial process in the context of the framework as it does not involve restarting failed kernels or copying their state, so it has not been studied experimentally in this work.

Theoretically, hierarchy-based fault tolerance based can be implemented on top of the message-passing library without loss of generality. Although it would be complicated to reuse free nodes instead of failed ones in the framework of this library, as the number of nodes is often fixed in such libraries, allocating reasonably large number of nodes for the programme would be enough to make it fault-tolerant. At the same time, implementing hierarchy-based fault tolerance inside message-passing library itself is not practical, because it would require saving the state of a parallel programme which equals to the total amount of memory it occupies on each cluster node, which in turn would not make it more efficient than checkpoints.

The weak point of the proposed method is the period of time starting from a failure of master node up to the moment when the failure is detected, the main kernel is restored and new subordinate kernel with the parent's copy is received by a slave node. If backup node fails within this time frame, execution state of a programme is completely lost, and there is no way to recover it other than restarting the programme from the beginning. The duration of the dangerous period can be minimised, but the probability of an abrupt programme termination can not be fully eliminated. This result is consistent with the scrutiny of *impossibility theory*, in the framework of which it is proved the impossibility of the distributed consensus with one faulty process [74] and impossibility of reliable communication in the presence of node failures [75].

5.2.4 Summary

Current state-of-the-art approach to developing and running parallel programmes on the cluster is the use of message passing library and job scheduler, and despite the fact that this approach is highly efficient in terms of parallel computation, it is not flexible enough to accommodate dynamic load balancing and automatic fault-tolerance. Programmes written with message passing library typically assume

- equal load on each processor,
- non-interruptible and reliable execution of batch jobs, and
- constant number of parallel processes throughout the execution which is equal to the total number of processors.

The first assumption does not hold for sea wave simulation programme because AR model requires dynamic load balancing between processors to generate each part of the surface only when all dependent parts have already been generated. The last assumption also does not hold, because for the sake of efficiency each part is written to a file asynchronously by a separate thread. The remaining assumption is not related to the programme itself, but to the job scheduler, and does not generally hold for very large computer clusters in which node failures occur regularly, and job scheduler restores the failed job from the checkpoint greatly increasing its running time. The idea of the proposed approach is to give parallel programmes more flexibility:

- provide dynamic load balancing via pipelined execution of sequential, internally parallel programme steps,
- restart only processes that were affected by node failure, and
- execute the programme on as many compute nodes as are available in the cluster.

In this section advantages and disadvantages of this approach are discussed.

In comparison to portable batch systems (PBS) the proposed approach uses lightweight control flow objects instead of heavy-weight parallel jobs to distribute the load on cluster nodes. First, this allows to have per-node kernel queues instead of several cluster-wide job queues. The granularity of control flow objects is much higher than of the batch jobs, and despite the fact that their execution time cannot be reliably predicted (as is execution time of batch jobs [76]), objects from multiple parallel programmes can be dynamically distributed between the same set of cluster nodes, thus making the load more even. The disadvantage is that this requires more RAM to execute many programmes on the same set of nodes, and execution time of each programme may be greater because of the shared control flow object queues. Second, the proposed approach uses dynamic distribution of master and slave roles between cluster nodes instead of their static assignment to the particular physical nodes. This makes nodes interchangeable, which is required to provide fault tolerance. Simultaneous execution of multiple parallel programmes on the same set of nodes increases throughput of the cluster, but also decreases their performance taken separately; dynamic role distribution is the base on which resilience to failures builds.

In comparison to MPI the proposed approach uses lightweight control flow objects instead of heavy-weight processes to decompose the programme into individual entities. First, this allows to determine the number of entities computed in parallel based on the problem being solved, not the computer or cluster architecture. A programmer is encouraged to create as many objects as needed, guided by the algorithm or restrictions on the size of data structures from the problem domain. In sea wave simulation programme the minimal size of each wavy surface part depends on the number of coefficients along each dimension, and at the same time the number of parts should be larger than the number of processors to make the load on each processor more even. Considering these limits the optimal part size is determined at runtime, and, in general, is not equal the number of parallel processes. The disadvantage is that the more control flow objects there are in the programme, the more shared data structures (e.g. coefficients) are copied to the same node with subordinate objects; this problem is solved by introducing another intermediate layer of objects, which in turn adds more complexity to the programme. Second, hierarchy of control flow objects together with hierarchy of nodes allows for automatic recomputation of failed objects on surviving nodes in an event of hardware failure. It is possible because the state of the programme execution is stored in objects and not in global variables like in MPI programmes. By duplicating the state to a subordinate node, the system recomputes only objects from affected processes instead of the whole programme. Transition from processes to control flow objects increases performance of a parallel programme via dynamic load balancing, but may inhibit its scalability for a large number of nodes and large amount of shared structures due to duplication of these structures.

Decomposition of a parallel programme into individual entities is done in Charm++ framework [77] (with load balancing [78]) and in actor model [79, 80], but none of the approaches uses hierarchical relationship to restart entity processing after a failure. Instead of using tree hierarchy of kernels, these approaches allow exchanging messages between any pair of entities. Such irregular message exchange pattern makes it impossible to decide which object is responsible for restarting a failed one, hence non-universal fault tolerance techniques are used instead [81].

Three building blocks of the proposed approach — control flow objects, pipelines and hierarchies — complement each other. Without control flow objects carrying programme state it is impossible to recompute failed subordinate objects and provide fault tolerance. Without node hierarchy it is impossible to distribute the load between cluster nodes, because all nodes are equal without the hierarchy. Without pipelines for each device it is impossible to execute control flow objects asynchronously and implement dynamic load balancing. These three entities form a closed system, in which programme logic is implemented in kernels or pipelines, failure recovery logic — in kernel hierarchy, and data transfer logic — in cluster node hierarchy.

5.3 MPP implementation

Distributed AR model algorithm. This algorithm, unlike its parallel counterpart, employs copying of data to execute computation on different cluster nodes, and since network throughput is much lower than memory bandwidth, the size of data that is sent over the network have to be optimised to get better performance than on SMP system. One way to accomplish this is to distribute wavy surface parts between cluster nodes copying in the coefficients and all the boundary points, and copying out generated wavy surface part. Autoregressive dependencies prevent from creating all the parts at once and statically distributing them between cluster nodes, so the parts are created dynamically on the first node, when points on which they depend become available. So, distributed AR model algorithm is a master-slave algorithm [82] in which the master dynamically creates tasks for each wavy surface part taking into account autoregressive dependencies between points and sends them to slaves, whereas slaves compute each wavy surface part and send them back to the master.

In MPP implementation each task is modelled by a kernel: there is a principal kernel that creates subordinate kernels on demand, and subordinate kernels that generate wavy surface parts. In `act` method of principal kernel a subordinate kernel is created for the first part — a part that does not depend on any points. When this kernel returns, the principal kernel in `react` method determines which parts can be computed in turn, creates a subordinate kernel for each part and sends them to the pipeline. In `act` method of subordinate kernel wavy surface part is generated and then the kernel sends itself back to the principal. The `react` method of subordinate kernel is empty.

Distributed AR algorithm implementation has several advantages over the parallel one.

- Pipelines automatically distribute subordinate kernels between available cluster nodes, and the main programme does not have to deal with these implementation details.
- There is no need to implement minimalistic job scheduler, which determines execution order of jobs (kernels) taking into account autoregressive dependencies: the order is fully defined in `react` method of the principal kernel.
- There is no need in separate version of the programme for SMP machine, the implementation works transparently on any number of nodes, even if job scheduler is not running.

Performance of distributed AR model implementation. Distributed AR model implementation was benchmarked on the two nodes of "Ant" system (table 11). To increase network throughput these nodes were directly connected to each other and maximum transmission unit (MTU) was set to 9200 bytes. Two cases were considered: with one Bscheduler daemon process running on the local node, and with two daemon processes running on each node. The performance of the programme was compared to the performance of OpenMP version running on single node.

Bscheduler outperforms OpenMP in both cases (fig. 20). In case of one node the higher performance is explained by the fact that Bscheduler does not scan the queue for wavy surface parts for which dependencies are ready (as in parallel version of the algorithm), but for each part updates a counter of completed parts on which it depends. The same approach can be used in OpenMP version, but was discovered only for newer Bscheduler version, as queue scanning can not be performed efficiently in this framework. In case of two nodes the higher performance is explained by greater total number of processor cores (16) and high network throughput of the direct network link. So, Bscheduler implementation of distributed AR model algorithm is faster on shared memory system due to more efficient autoregressive dependencies handling and its performance on distributed memory system scales to a larger number of cores due to small data transmission overhead of direct network link.

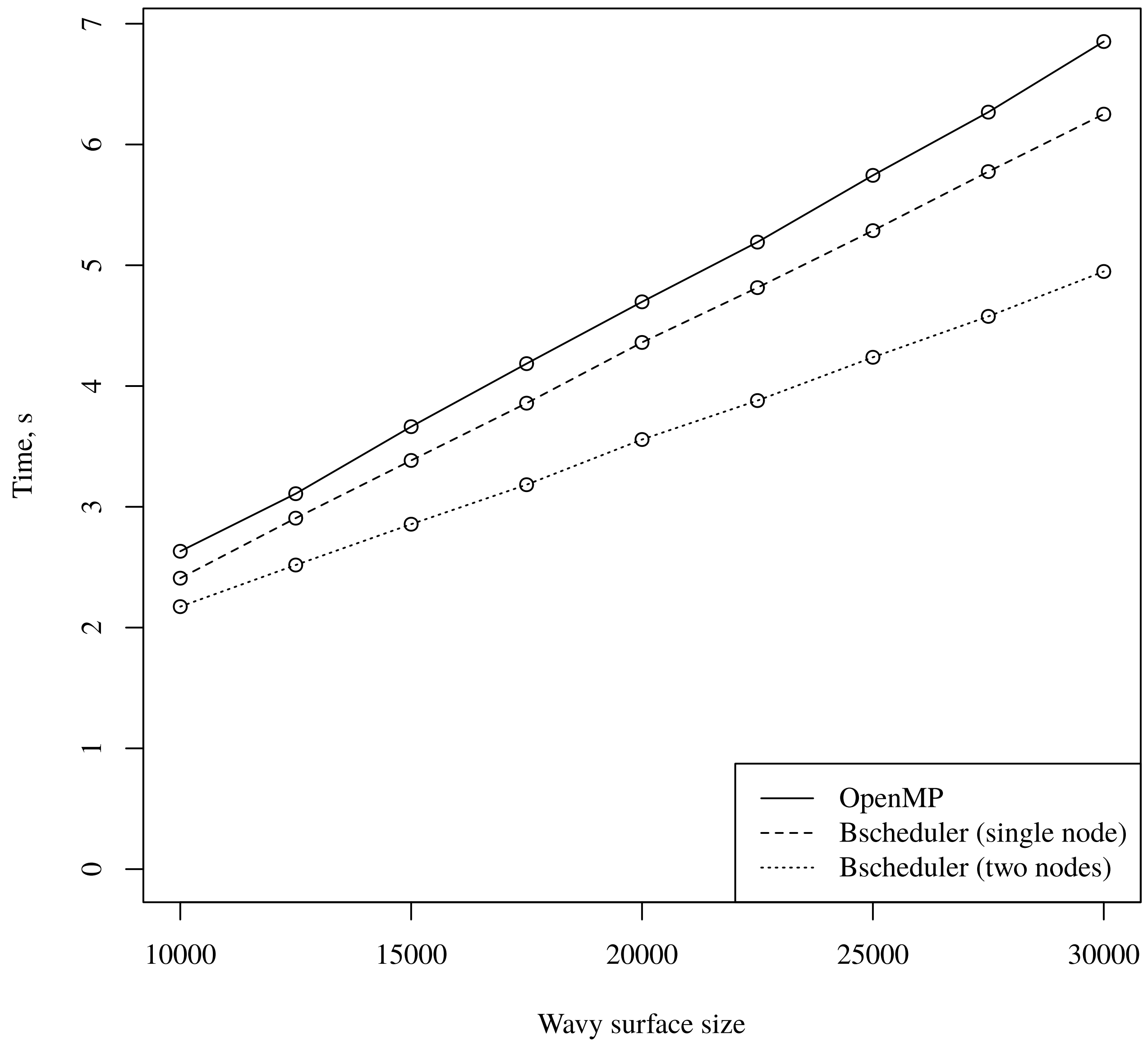


Figure 20: Performance comparison of Bscheduler and OpenMP programme versions for AR model.

6 Conclusion

In the study of mathematical apparatus for sea wave simulations which goes beyond linear wave theory the following main results were achieved.

- AR and MA models were applied to simulation of sea waves of arbitrary amplitudes. Integral characteristics of wavy surface were verified by comparing to the ones of real sea surface.

- New method was applied to compute velocity potentials under generated surface. The resulting field was verified by comparing it to the one given by formulae from linear wave theory for small-amplitude waves. For large-amplitude waves the new method gives a reasonably different field. The method is computationally efficient because all the integrals in its formula are written as Fourier transforms, for which there are high-performance implementations.

- The model and the method were implemented for both shared and distributed memory systems, and showed near linear scalability for different number of cores in several benchmarks. AR model is more computationally efficient on CPU than on GPU, and outperforms LH model.

One of the topic of future research is studying generation of waves of arbitrary profiles on the basis of mixed ARMA process. Another direction is integration of the developed model and pressure calculation method into existing application software packages.

7 Summary

A problem of determining pressures under sea surface is solved explicitly without assumptions of linear and small-amplitude wave theories. This solution coupled with AR and MA models, that generate sea waves of arbitrary amplitudes, can be used to determine the impact of wave oscillations on the dynamic marine object in a sea way, and in case of large-amplitude waves gives more accurate velocity potential field than analogous solutions obtained in the framework of linear wave theory and theory of small-amplitude waves.

Numerical experiments show that wavy surface generation as well as pressure computation are efficiently implemented for both shared and distributed memory systems, without computing on GPU. High performance in case of wavy surface generation is provided by the use of specialised job scheduler and a library for multi-dimensional arrays, and in case of velocity potential computation — by the use of FFT algorithms for computing integrals.

The developed mathematical apparatus and its numerical implementation can become a base of virtual testbed for marine objects dynamics studies. The use of new models in virtual testbed would allow to

- conduct long-time simulations without the loss of efficiency,
- obtain more accurate pressure fields due to new velocity potential field computation method, and
- make software complex more robust due to high convergence rate of the models and by using fault-tolerant batch job scheduler.

8 Acknowledgements

The graphs in this work were prepared using R language for statistical computing [83, 84] and Graphviz software [85]. The manuscript was prepared using Org-mode [86–88] for GNU Emacs which provides computing environment for reproducible research. This means that all graphs can be reproduced and corresponding statements verified on different computer systems by cloning thesis repository[2], installing Emacs and exporting the document.

[2]`https://github.com/igankevich/arma-thesis`

9 List of acronyms and symbols

MPP Massively Parallel Processing, computers with distributed memory.

SMP Symmetric Multi-Processing, computers with shared memory.

GPGPU General-purpose computing on graphics processing units.

ACF auto-covariate function.

FFT fast Fourier transform.

PRNG pseudo-random number generator.

BC boundary condition.

PDE partial differential equation.

NIT non-linear inertialess transform which allows to specify arbitrary distribution law for wavy surface elevation without changing its original auto-covariate function.

AR auto-regressive process.

ARMA auto-regressive moving-average process.

MA moving average process.

LH Longuet—Higgins model, formula of which is derived in the framework of linear wave theory.

LAMP Large Amplitude Motion Programme, a programme that simulates ship behaviour in ocean waves.

CLT central limit theorem.

PM Pierson—Moskowitz ocean wave spectrum approximation.

YW Yule—Walker equations which are used to determine autoregressive model coefficients for a given auto-covariate function.

LS least squares.

PDF probability density function.

CDF cumulative distribution function.

BSP Bulk Synchronous Parallel.

OpenCL Open Computing Language, parallel programming technology for hybrid systems with GPUs or other co-processors.

OpenGL Open Graphics Library.

OpenMP Open Multi-Processing, parallel programming technology for multi-processo systems.

MPI Message Passing Interface.

FMA Fused multiply-add.

DCMT Dynamic creation of Mersenne Twisters, an algorithm for creating pseudo-random number generators that produce uncorrelated sequences when run in parallel.

GSL GNU Scientific Library.

BLAS Basic Linear Algebra Sub-programmes.

LAPACK Linear Algebra Package.

DNS Dynamic name resolution.

HPC High-performance computing.

GCS Gram—Charlier series.

SN Skew normal distribution.

PBS Portable batch system, a system for allocating and distributing cluster resources for parallel programmes.

Transcendental functions non-algebraic mathematical functions (i.e. logarithmic, trigonometric, exponential etc.).

List of Figures

List of Tables

Publications on the subject of thesis

[1] Degtyarev A., Gankevich I. Evaluation of hydrodynamic pressures for autoregression model of irregular waves // Proceedings of 11th International Conference “Stability of Ships and Ocean Vehicles”. — Athens, Greece, 2012. — P. 841–852.

[2] Дегтярев А. Б., Ганкевич И. Г. Вычисление гидродинамических давлений под реальной морской поверхностью на основе авторегрессионной модели нерегулярного волнения // Труды XLV НТК “Проблемы мореходных качеств судов, корабельной гидромеханики и освоения шельфа” (Крыловские чтения). — 2013. — С. 25–29.

[3] Gankevich Ivan, Degtyarev Alexander. Model of distributed computations in virtual testbed // Proceedings of IX International Conference on Computer Science and Information Tecnologies (CSIT’2013). — Yerevan, Armenia, 2013. — P. 240–244.

[4] Degtyarev A., Gankevich I. Calculation Scheme for Wave Pressures with Autoregression Method // 14th International Ship Stability Workshop. — 2014. — P. 135–139.

[5] Ганкевич И. Г., Дегтярев А. Б. Методы распределения нагрузки на многопроцессорную систему // Процессы управления и устойчивость. — 2014. — Т. 1, № 17. — С. 295–300.

[6] Degtyarev A., Gankevich I. Hydrodynamic Pressure Computation under Real Sea Surface on Basis of Autoregressive Model of Irregular Waves // Physics of Particles and Nuclei Letters. — 2015. — Vol. 12, no. 3. — P. 389–391. — URL: `http://dx.doi.org/10.1134/S1547477115030073`.

[7] Novel Approaches for Distributing Workload on Commodity Computer Systems / Ivan Gankevich, Yuri Tipikin, Alexander Degtyarev, Vladimir Korkhov // Computational Science and Its Applications – ICCSA 2015 / Ed. by Osvaldo Gervasi, Beniamino Murgante, Sanjay Misra et al. — Springer International Publishing, 2015. — Vol. 9158 of Lecture Notes in Computer Science. — P. 259–271. — URL: http://dx.doi.org/10.1007/978-3-319-21410-8_20.

[8] Gankevich Ivan, Degtyarev Alexander. Efficient processing and classification of wave energy spectrum data with a distributed pipeline // Computer Research and Modeling. — 2015. — Vol. 7, no. 3. — P. 517–520. — URL: http://crm-en.ics.org.ru/journal/article/2301/.

[9] Gankevich Ivan, Tipikin Yuri, Gaiduchok Vladimir. Subordination: Cluster management without distributed consensus // International Conference on High Performance Computing Simulation (HPCS). — 2015. — P. 639–642. — Outstanding poster paper award.

[10] Running applications on a hybrid cluster / A. Bogdanov, I. Gankevich, Gayduchok V., N. Yuzhanin // Computer Research and Modeling. — 2015. — Vol. 7, no. 3. — P. 475–483. — URL: http://crm-en.ics.org.ru/journal/article/2295/.

[11] Applications of on-demand virtual clusters to high performance computing / I. Gankevich, S. Balyan, S. Abrahamyan, V. Korkhov // Computer Research and Modeling. — 2015. — Vol. 7, no. 3. — P. 511–516. — URL: http://crm-en.ics.org.ru/journal/article/2300/.

[12] Degtyarev Alexander, Gankevich Ivan. Balancing Load on a Multiprocessor System with Event-Driven Approach // Transactions on Computational Science XXVII / Ed. by L. Marina Gavrilova, Kenneth C. J. Tan. — Berlin, Heidelberg : Springer Berlin Heidelberg, 2016. —

P. 35–52. — ISBN: 978-3-662-50412-3. — URL: http://dx.doi.org/10.1007/978-3-662-50412-3_3.

[13] Factory: Non-stop batch jobs without checkpointing / I. Gankevich, Y. Tipikin, V. Korkhov, V. Gaiduchok // International Conference on High Performance Computing Simulation (HPCS'16). — 2016. — July. — P. 979–984.

[14] Factory: Master Node High-Availability for Big Data Applications and Beyond / Ivan Gankevich, Yuri Tipikin, Vladimir Korkhov et al. // Computational Science and Its Applications – ICCSA 2016: 16th International Conference, Beijing, China, July 4-7, 2016, Proceedings, Part II / Ed. by Osvaldo Gervasi, Beniamino Murgante, Sanjay Misra et al. — Springer International Publishing, 2016. — P. 379–389. — ISBN: 978-3-319-42108-7. — URL: http://dx.doi.org/10.1007/978-3-319-42108-7_29.

[15] Acceleration of Computing and Visualization Processes with OpenCL for Standing Sea Wave Simulation Model / Andrei Ivashchenko, Alexey Belezeko, Ivan Gankevich et al. // Computational Science and Its Applications – ICCSA 2017: 17th International Conference, Trieste, Italy, July 3-6, 2017, Proceedings, Part V / Ed. by Osvaldo Gervasi, Beniamino Murgante, Sanjay Misra et al. — Cham : Springer International Publishing, 2017. — P. 505–518. — ISBN: 978-3-319-62404-4. — URL: https://doi.org/10.1007/978-3-319-62404-4_38.

[16] Distributed Data Processing on Microcomputers with Ascheduler and Apache Spark / Vladimir Korkhov, Ivan Gankevich, Oleg Iakushkin et al. // Computational Science and Its Applications – ICCSA 2017: 17th International Conference, Trieste, Italy, July 3-6, 2017, Proceedings, Part V / Ed. by Osvaldo Gervasi, Beniamino Murgante, Sanjay Misra et al. — Cham : Springer International Publishing, 2017. — P. 387–398. — ISBN: 978-3-

319-62404-4. — URL: https://doi.org/10.1007/978-3-319-62404-4_28.

References

[1] Nonlinear time domain simulation technology for seakeeping and wave-load analysis for modern ship design / Y. S. Shin, V. L. Belenky, W. M. Lin et al. // Transactions of Society of Naval Architects and Marine Engineers. — 2003. — Vol. 111. — P. 557–583.

[2] van Walree F., de Kat J. O., Ractliffe A. T. Forensic research into the loss of ships by means of a time domain simulation tool // International shipbuilding progress. — 2007. — Vol. 54, no. 4. — P. 381–407.

[3] de Kat Jan O., Paulling J. Randolph. Prediction of extreme motions and capsizing of ships and offshore marine vehicles // 20th conference on Offshore Mechanics and Arctic Engineering (OMAE'01). — 2001.

[4] van Walree F. Development, validation and application of a time domain seakeeping method for high speed craft with a ride control system // Proceedings of the 24th symposium on Naval Hydrodynamics. — 2002.

[5] Wolfram Research, Inc. — Mathematica. — Champaign, Illinois, 2016.

[6] GNU Octave version 4.0.0 manual: a high-level interactive language for numerical computations / John W. Eaton, David Bateman, Søren Hauberg, Rik Wehbring. — 2015. — URL: `http://www.gnu.org/software/octave/doc/interpreter`.

[7] Kochin N., Kibel I., Roze N. Theoretical hydrodynamics [in Russian]. — FizMatLit, 1966. — P. 237.

[8] Бухановский А. В. Вероятностное моделирование полей ветрового волнения с учетом их неоднородности и нестационарности : Ph. D. thesis / А. В. Бухановский ; СПбГУ. — 1997.

[9] Degtyarev A. B., Reed A. M. Modelling of incident waves near the ship's hull (application of autoregressive approach in problems of simulation of rough seas) // Proceedings of the 12th International Ship Stability Workshop. — 2011.

[10] Longuet-Higgins Michael S. The statistical analysis of a random, moving surface // Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences. — 1957. — Vol. 249, no. 966. — P. 321–387.

[11] Huang Norden E., Long Steven R. An experimental study of the surface elevation probability distribution and statistics of wind-generated waves // Journal of Fluid Mechanics. — 1980. — Vol. 101, no. 01. — P. 179–200.

[12] Рожков В. А. Теория вероятностей случайных событий, величин и функций с гидрометеорологическими примерами // СПб, Прогресс-Погода. — 1996.

[13] Рожков В. А., Трапезников Ю. А. Вероятностные модели океанологических процессов. — Гидрометеоиздат, 1990.

[14] Spanos Pol D. ARMA Algorithms for Ocean Spectral Analysis. — University of Texas at Austin. Engineering Mechanics Research Laboratory, 1982.

[15] Spanos Pol D., Zeldin B. A. Efficient iterative ARMA approximation of multivariate random processes for structural dynamics applications // Earthquake engineering & structural dynamics. — 1996. — Vol. 25, no. 5. — P. 497–507.

[16] Fusco Francesco, Ringwood John V. Short-term wave forecasting for real-time control of wave energy converters // IEEE Transactions on sustainable energy. — 2010. — Vol. 1, no. 2. — P. 99–106.

[17] Box George E. P., Jenkins Gwilym M. Time series analysis: forecasting and control. — Holden-Day, 1976.

[18] Choi ByoungSeon. An order-recursive algorithm to solve the 3-D Yule-Walker equations of causal 3-D AR models // IEEE Transactions on Signal Processing. — 1999. — Sep. — Vol. 47, no. 9. — P. 2491–2502.

[19] Liew J., Marple S. L. Three-dimensional fast algorithm solution for octant-based three-dimensional Yule-Walker equations // IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'05). — Vol. 2. — 2005. — March. — P. 889–892.

[20] Degtyarev Alexander B., Reed Arthur M. Synoptic and short-term modeling of ocean waves // International Shipbuilding Progress. — 2013. — Vol. 60, no. 1-4. — P. 523–553.

[21] Longuet-Higgins Michael S. The effect of non-linearities on statistical distributions in the theory of sea waves // Journal of fluid mechanics. — 1963. — Vol. 17, no. 03. — P. 459–480.

[22] Owen Donald B. Tables for computing bivariate normal probabilities // The Annals of Mathematical Statistics. — 1956. — Vol. 27, no. 4. — P. 1075–1090.

[23] Wallace David L. Asymptotic approximations to distributions // The Annals of Mathematical Statistics. — 1958. — Vol. 29, no. 3. — P. 635–654.

[24] Boccotti Paolo. On wind wave kinematics // Meccanica. — 1983. — Vol. 18, no. 4. — P. 205–216.

[25] Degtyarev A., Gankevich I. Evaluation of hydrodynamic pressures for autoregression model of irregular waves // Proceedings of 11th International

Conference on Stability of Ships and Ocean Vehicles, Athens. — 2012. — P. 841–852.

[26] Дегтярев А. Б., Подолякин А. Б. Имитационное моделирование поведения судна на реальном волнении // Тр. II Межд. конф. по судостроению – ISC'98. — Vol. B. — Санкт-Петербург, 1998. — P. 416–423.

[27] Analysis of Peculiarities of Ship-Environmental Interaction : Rep. : 09-97-1AB-1VA / Strathclyde University ; Executor: A. Degtyarev, A. Boukhanovsky. — Ship Stability Research Center, Glasgow : 1997. — September.

[28] Oppenheim Alan V., Schafer Ronald W., Buck John R. Discrete-time signal processing. — Prentice hall Englewood Cliffs, NJ, 1989. — Vol. 2.

[29] Svoboda David. Efficient computation of convolution of huge images // Image Analysis and Processing (ICIAP'11). — Springer, 2011. — P. 453–462.

[30] Pavel Karas, David Svoboda. Algorithms for efficient computation of convolution. — INTECH Open Access Publisher, 2013.

[31] Matsumoto Makoto, Nishimura Takuji. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator // ACM Transactions on Modeling and Computer Simulation (TOMACS). — 1998. — Vol. 8, no. 1. — P. 3–30.

[32] Matsumoto Makoto, Nishimura Takuji. Dynamic creation of pseudorandom number generators // Monte Carlo and Quasi-Monte Carlo Methods. — 1998. — Vol. 2000. — P. 56–69.

[33] Veldhuizen Todd L., Jernigan M. Ed. Will C++ be faster than Fortran? // International Conference on Computing in Object-Oriented Parallel Environments / Springer. — 1997. — P. 49–56.

[34] Veldhuizen Todd. Techniques for scientific C++ // Computer science technical report. — 2000. — Vol. 542. — P. 60.

[35] GSL GNU. Scientific Library. — 2008.

[36] clFFT developers. clFFT: OpenCL Fast Fourier Transforms (FFTs). — `https://clmathlibraries.github.io/clFFT/`.

[37] Goto Kazushige, Van De Geijn Robert. High-performance implementation of the level-3 BLAS // ACM Transactions on Mathematical Software (TOMS). — 2008. — Vol. 35, no. 1. — P. 4.

[38] Goto Kazushige, Geijn Robert A. Anatomy of high-performance matrix multiplication // ACM Transactions on Mathematical Software (TOMS). — 2008. — Vol. 34, no. 3. — P. 12.

[39] Kilgard Mark J. The OpenGL Utility Toolkit (GLUT) Programming Interface. — 1996.

[40] Fabri Andreas, Pion Sylvain. CGAL: The computational geometry algorithms library // Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems / ACM. — 2009. — P. 538–539.

[41] GNU Scientific Library Reference Manual / M. Galassi, J. Davies, J. Theiler et al. ; Ed. by Brian Gough. — 3 edition. — Network Theory Ltd., 2009. — ISBN: 0954612078, 9780954612078.

[42] Parallel programming with migratable objects: Charm++ in practice / Bilge Acun, Arpan Gupta, Nikhil Jain et al. // International Conference

for High Performance Computing, Networking, Storage and Analysis / IEEE. — 2014. — P. 647–658.

[43] Oozie: towards a scalable workflow management system for Hadoop / Mohammad Islam, Angelo K. Huang, Mohamed Battisha et al. // Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies / ACM. — 2012. — P. 4.

[44] Dean Jeffrey, Ghemawat Sanjay. MapReduce: Simplified data processing on large clusters // Communications of the ACM. — 2008. — Vol. 51, no. 1. — P. 107–113.

[45] Apache Hadoop YARN: Yet Another Resource Negotiator / Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas et al. // Proceedings of the 4th Annual Symposium on Cloud Computing. — SOCC '13. — New York, NY, USA : ACM, 2013. — P. 5:1–5:16. — URL: http://doi.acm.org/10.1145/2523616.2523633.

[46] Valiant Leslie G. A bridging model for parallel computation // Communications of the ACM. — 1990. — Vol. 33, no. 8. — P. 103–111.

[47] Pregel: a system for large-scale graph processing / Grzegorz Malewicz, Matthew H Austern, Aart JC Bik et al. // Proceedings of the ACM SIGMOD International Conference on Management of data / ACM. — 2010. — P. 135–146.

[48] Hama: An efficient matrix computation with the mapreduce framework / Sangwon Seo, Edward J. Yoon, Jaehong Kim et al. // IEEE 2nd international conference on Cloud Computing Technology and Science (CloudCom'10) / IEEE. — 2010. — P. 721–726.

[49] ZooKeeper: Wait-free Coordination for Internet-scale Systems. / Patrick Hunt, Mahadev Konar, Flavio Paiva Junqueira, Benjamin Reed //

USENIX annual technical conference / Boston, MA, USA. — Vol. 8. — 2010. — P. 9.

[50] Lakshman Avinash, Malik Prashant. Cassandra: A decentralized structured storage system // ACM SIGOPS Operating Systems Review. — 2010. — Vol. 44, no. 2. — P. 35–40.

[51] Divya Manda Sai, Goyal Shiv Kumar. ElasticSearch: An advanced and quick search technique to handle voluminous data // An International Journal of Advanced Computer Technology. — 2013. — Vol. 2, no. 6. — P. 171.

[52] Boyer Eric B., Broomfield Matthew C., Perrotti Terrell A. GlusterFS One Storage Server to Rule Them All. — No. LA-UR-12-23586. United States, 2012. — July. — URL: `http://www.osti.gov/scitech/servlets/purl/1048672`.

[53] Ostrovsky David, Rodenski Yaniv, Haji Mohammed. Pro Couchbase Server. — Apress, 2015.

[54] Tel Gerard. Introduction to distributed algorithms. — Cambridge University press, 2000.

[55] Lantz Bob, Heller Brandon, McKeown Nick. A network in a laptop: rapid prototyping for software-defined networks // Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks / ACM. — 2010. — P. 19.

[56] Reproducible network experiments using container-based emulation / Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar et al. // Proceedings of the 8th international conference on Emerging networking experiments and technologies / ACM. — 2012. — P. 253–264.

[57] Heller Brandon. Reproducible Network Research with High-fidelity Emulation : Ph. D. thesis / Brandon Heller ; Stanford University. — 2013.

[58] Design and analysis of dynamic leader election protocols in broadcast networks / Jacob Brunekreef, Joost-Pieter Katoen, Ron Koymans, Sjouke Mauw // Distributed Computing. — 1996. — Vol. 9, no. 4. — P. 157–171.

[59] Stable leader election / Marcos K Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, Sam Toueg // Distributed Computing. — Springer, 2001. — P. 108–122.

[60] Romano Paolo, Quaglia Francesco. Design and evaluation of a parallel invocation protocol for transactional applications over the web // IEEE Transactions on Computers. — 2014. — Vol. 63, no. 2. — P. 317–334.

[61] TCP User Timeout Option : RFC : 5482 / RFC Editor ; Executor: Lars Eggert, Fernando Gont : 2009. — March. — P. 1–13. URL: `http://www.rfc-editor.org/rfc/rfc5482.txt`.

[62] A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems / Ifeanyi P. Egwutuoha, David Levy, Bran Selic, Shiping Chen // The Journal of Supercomputing. — 2013. — Vol. 65, no. 3. — P. 1302–1326.

[63] Meyer Hugo, Rexachs Dolores, Luque Emilio. RADIC: A Fault Tolerant Middleware with Automatic Management of Spare Nodes // Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA) / The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp). — Athens, 2012. — P. 1–7.

[64] Anderson J. Chris, Lehnardt Jan, Slater Noah. CouchDB: The definitive guide. — O'Reilly Media, Inc., 2010.

[65] Architecture of next generation Apache Hadoop MapReduce framework / Arun C. Murthy, Chris Douglas, Mahadev Konar et al. // Apache Jira. — 2011.

[66] Okorafor Ekpe, Patrick Mensah Kwabena. Availability of JobTracker machine in Hadoop/MapReduce ZooKeeper coordinated clusters // Advanced Computing: An International Journal (ACIJ). — 2012. — Vol. 3, no. 3. — P. 19–30.

[67] Uhlemann K., Engelmann C., Scott S. L. JOSHUA: Symmetric Active/Active Replication for Highly Available HPC Job and Resource Management // IEEE International Conference on Cluster Computing. — 2006. — September. — P. 1–10.

[68] Symmetric active/active high availability for high-performance computing system services / Christian Engelmann, Stephen L. Scott, Chokchai Box Leangsuksun, Xubin Ben He // Journal of Computers. — 2006. — Vol. 1, no. 8. — P. 43–54.

[69] Virtual Router Redundancy Protocol : RFC : 2338 / RFC Editor ; Executor: S. Knight, D. Weaver, D. Whipple et al. : 1998. — April. — P. 1–26. URL: http://www.rfc-editor.org/rfc/rfc2338.txt.

[70] Virtual Router Redundancy Protocol (VRRP) : RFC : 3768 / RFC Editor ; Executor: Robert Hinden : 2004. — April. — P. 1–27. URL: http://www.rfc-editor.org/rfc/rfc3768.txt.

[71] Virtual Router Redundancy Protocol (VRRP) Version 3 for IPv4 and IPv6 : RFC : 5798 / RFC Editor ; Executor: S. Nadas : 2010. — March. — P. 1–39. URL: http://www.rfc-editor.org/rfc/rfc5798.txt.

[72] Cassen Alexandre. Keepalived: Health checking for LVS & high availability // URL http://www.linuxvirtualserver.org. — 2002.

[73] Armstrong Joe. Making reliable distributed systems in the presence of software errors : Ph. D. thesis / Joe Armstrong ; The Royal Institute of Technology Stockholm, Sweden. — Sweden, 2003.

[74] Fischer Michael J, Lynch Nancy A, Paterson Michael S. Impossibility of distributed consensus with one faulty process // Journal of the ACM (JACM). — 1985. — Vol. 32, no. 2. — P. 374–382.

[75] The impossibility of implementing reliable communication in the face of crashes / Alan Fekete, Nancy Lynch, Yishay Mansour, John Spinelli // Journal of the ACM (JACM). — 1993. — Vol. 40, no. 5. — P. 1087–1107.

[76] Zotkin Dinitry, Keleher Peter J. Job-length estimation and performance in backfilling schedulers // Proceedings of the 8th International Symposium on High Performance Distributed Computing / IEEE. — 1999. — P. 236–243.

[77] Charm++ for Productivity and Performance / Laxmikant V. Kale, Anshu Arya Abhinav Bhatele Abhishek Gupta, Nikhil Jain et al. — 2012.

[78] Adaptive load balancing for MPI programs / Milind Bhandarkar, Laxmikant V. Kalé, Eric de Sturler, Jay Hoeflinger // International Conference on Computational Science (ICCS'01). — Springer, 2001. — P. 108–117.

[79] Hewitt Carl, Bishop Peter, Steiger Richard. A universal modular actor formalism for artificial intelligence // Proceedings of the 3rd international joint conference on Artificial intelligence / Morgan Kaufmann Publishers Inc. — 1973. — P. 235–245.

[80] Actors: A model of concurrent computation in distributed systems. : Rep. / DTIC Document ; Executor: Gul A Agha : 1985.

[81] Scalable replay with partial-order dependencies for message-logging fault tolerance / Jonathan Lifflander, Esteban Meneses, Harshitha Menon et al. // IEEE International Conference on Cluster Computing (CLUSTER) / IEEE. — 2014. — P. 19–28.

[82] More scalability, less pain: A simple programming model and its implementation for extreme computing / Ewing L. Lusk, Steven C. Pieper, Ralph M. Butler et al. // SciDAC Review. — 2010. — Vol. 17, no. 1. — P. 30–37.

[83] R Core Team. — R: A Language and Environment for Statistical Computing. — R Foundation for Statistical Computing, Vienna, Austria, 2016. — URL: `https://www.R-project.org/`.

[84] Sarkar Deepayan. Lattice: Multivariate Data Visualization with R. — New York : Springer, 2008. — ISBN: 978-0-387-75968-5. — URL: `http://lmdvr.r-forge.r-project.org`.

[85] Gansner Emden R., North Stephen C. An open graph visualization system and its applications to software engineering // Software — Practice and Experience. — 2000. — Vol. 30, no. 11. — P. 1203–1233. — URL: `http://www.graphviz.org/`.

[86] Schulte E., Davison D. Active Documents with Org-Mode // Computing in Science Engineering. — 2011. — May-June. — Vol. 13, no. 3. — P. 66–73.

[87] A Multi-Language Computing Environment for Literate Programming and Reproducible Research / Eric Schulte, Dan Davison, Thomas Dye, Carsten Dominik // Journal of Statistical Software. — 2012. — 1. — Vol. 46, no. 3. — P. 1–24. — URL: `http://www.jstatsoft.org/v46/i03`.

[88] Dominik Carsten. The Org-Mode 7 Reference Manual: Organize Your Life with GNU Emacs. — UK : Network Theory, 2010. — with contributions by

David O’Toole, Bastien Guerry, Philip Rooke, Dan Davison, Eric Schulte, and Thomas Dye.

10 Appendix

10.1 Longuet—Higgins model formula derivation

In the framework of linear wave theory two-dimensional system of equations (1) is written as

$$\begin{aligned}&\phi_{xx} + \phi_{zz} = 0,\\ &\zeta(x,t) = -\frac{1}{g}\phi_t, \qquad\qquad \text{at } z = \zeta(x,t),\end{aligned}$$

where $\frac{p}{\rho}$ includes ϕ_t. The solution to the Laplace equation is sought in a form of Fourier series [7]:

$$\phi(x,z,t) = \int\limits_0^\infty e^{kz}\left[A(k,t)\cos(kx) + B(k,t)\sin(kx)\right]dk.$$

Plugging it in the boundary condition yields

$$\begin{aligned}\zeta(x,t) &= -\frac{1}{g}\int\limits_0^\infty \left[A_t(k,t)\cos(kx) + B_t(k,t)\sin(kx)\right]dk\\ &= -\frac{1}{g}\int\limits_0^\infty C_t(k,t)\cos(kx + \epsilon(k,t)).\end{aligned}$$

Here ϵ is white noise and C_t includes dk. Substituting integral with infinite sum yields two-dimensional form of (2).

10.2 Derivative in the direction of the surface normal

Directional derivative of ϕ in the direction of vector $\vec{n}$ is given by $\nabla_n \phi = \nabla \phi \cdot \frac{\vec{n}}{|\vec{n}|}$. Normal vector $\vec{n}$ to the surface $z = \zeta(x, y)$ at point (x_0, y_0) is given by

$$\vec{n} = \begin{bmatrix} \zeta_x(x_0, y_0) \\ \zeta_y(x_0, y_0) \\ -1 \end{bmatrix}.$$

Hence, derivative in the direction of the surface normal is given by

$$\nabla_n \phi = \phi_x \frac{\zeta_x}{\sqrt{\zeta_x^2 + \zeta_y^2 + 1}} + \phi_y \frac{\zeta_y}{\sqrt{\zeta_x^2 + \zeta_y^2 + 1}} + \phi_z \frac{-1}{\sqrt{\zeta_x^2 + \zeta_y^2 + 1}},$$

where ζ derivatives are calculated at (x_0, y_0).