Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-18
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 8 March 2018

The VOLNA-OP2 Tsunami Code (Version 1.0)

Istvan Z Reguly[1], Devaraj Gopinathan[2], Joakim H Beck[3], Michael B Giles[4], Serge Guillas[2], and Frederic Dias[5]

[1]Pázmány Péter Catholic University, Faculty of Information Technology and Bionics, Prater u 50/a, 1088 Budapest, Hungary
[2]Department of Statistical Science, University College London, London, UK
[3]Computer, Electrical and Mathematical Science and Engineering Division (CEMSE), King Abdullah University of Science and Technology (KAUST), Thuwal, 23955-6900, Saudi Arabia
[4]Math Institute, University of Oxford, Oxford, UK
[5]School of Mathematics and Statistics, University College Dublin, Dublin, Ireland

Correspondence: Istvan Z Reguly (reguly.istvan@itk.ppke.hu)

Abstract. In this paper, we present the VOLNA-OP2 tsunami model and implementation; a finite volume non-linear shallow water equations (NSWE) solver built on the OP2 domain specific language for unstructured mesh computations. VOLNA-OP2 is unique among tsunami solvers in its support for several high performance computing platforms: CPUs, the Intel Xeon Phi, and GPUs. This is achieved in a way that the scientific code is kept separate from various parallel implementations, enabling easy maintainability. It has already been used in production for several years, here we discuss how it can be integrated into various workflows, such as a statistical emulator. The scalability of the code is demonstrated on three supercomputers, built with classical Xeon CPUs, the Intel Xeon Phi, and NVIDIA P100 GPUs. VOLNA-OP2 shows an ability to deliver productivity to its users, as well as performance and portability on a number of platforms.

1 Introduction

After the Indian Ocean tsunami of 26 December 2004, Bernard et al. (2006) emphasized that one of the greatest contributions of science to society is to serve it purposefully, as when providing forecasts to allow communities to respond before a disaster strikes. In the last twelve years, the numerical modelling of tsunami has experienced great progress (Behrens and Dias (2015)). There is a variety of mathematical models, such as the shallow-water equations, the Boussinesq equations, or the 3D Navier-Stokes equations, and there exist a huge number of implementations - yet there are only a handful codes that are suitable for integration into a workflow, aimed at forecasting or statistical exploration.

For widespread use therefore three key ingredients are needed; first, the stability and robustness of the numerical approach, that gives a confidence in the results produced, second, the computational performance of the code, which allows for getting the right results quickly, efficiently utilising the available computational resources, and third, the ability to integrate into a workflow, allowing for simple pre- and post-processing, efficiently supporting the kinds of use cases that come up - for example large numbers of different initial conditions.

In the Related Work section we discuss a number of codes currently being used in production, and as such are trusted and reliable codes, already being used as part of a workflow. Yet, the computational performance of most of these codes is "good enough"; they were written by domain scientists, and may have been tuned to one architecture or an other, but for example, GPU support is almost non-existent. In today's and tomorrow's quickly changing hardware landscape however, "future-proofing" numerical codes is of exceptional importance for continued scientific delivery. Domain scientists can not be expected to keep up with architectural advances, and spend a significant amount of time re-factoring code to target new hardware. *What* to compute must be separated from *how* it is computed - indeed in a recent paper by Lawrence et al. (2017), leaders in the weather community chart the ways forward, and point to Domain Specific Languages (DSLs) as a potential way to address this issue.

OP2, by Mudalige et al. (2012), is such a DSL, embedded in C/C++ and Fortran; it has been in development since 2009: it provides an abstraction for expressing unstructured mesh computations at a high-level, and then provides automated tools to translate scientific code written once, into a range of high-performance implementations targeting multi-core CPUs, GPUs, and large heterogeneous supercomputers. The original VOLNA model (Dutykh et al. (2011)) was already discussed and validated in detail - was used in production for small-scale experiments and modelling, but was inadequate for targeting large-scale scenarios and statistical analysis, therefore it was re-implemented on top of OP2; this paper describes the process, challenges and results from that work.

As VOLNA-OP2 delivered a qualitative leap in terms of possible uses, its users have started integrating it into a wide variety of workflows; one of the key uses is for uncertainty quantification; for the stochastic inversion problem of the 2004 Sumatra tsunami in Gopinathan et al. (2017), for developing Gaussian process emulators which help reduce the number of simulation runs (Beck and Guillas (2016); Liu and Guillas (2017)), applications of stochastic emulators to a submarine slide at the Rockall Bank (Salmanidou et al. (2017)), a study of run-up behind islands (Stefanakis et al. (2014)), the durability of oscillating wave surge converters when hit by tsunamis (O'Brien et al. (2015)), tsunamis in the St. Lawrence estuary (Poncet et al. (2010)), a study of the generation and inundation phases of tsunamis (Dias et al. (2014)), and others. These applications present a number of challenges in integration into the workflow, as well as scalable performance: the need for extracting snapshots of state variables on the full mesh, or at a number of specified locations, capturing the maximum wave elevation or inundation - all in the context of distributed memory execution.

As the above references indicate, VOLNA-OP2 has already been key in delivering scientific results in a range of scenarios, and through the collaboration of the authors, it is now capable of efficiently supporting a number of use cases, making it a versatile tool to the community, therefore we have now publicly released it: it is freely available at github.com/reguly/volna.

The rest of the paper is organised as follows: Section 2 discusses related work, Section 3 presents the OP2 library, upon which VOLNA-OP2 is built, Section 4 discusses the VOLNA simulator itself, its structure and features, Section 5 discusses performance and scalability results on CPUs and GPUs, and finally Section 6 draws conclusions.

2 Related Work

Tsunamis have long been a key target for scientific simulations. Behrens and Dias (2015) give a detailed look at various mathematical, numerical, and implementational approaches to past and current tsunami simulations. The most common set of equations solved are the shallow water equations, and most codes use structured and nested meshes. A popular discretisation is finite differences, such codes include: NOAA's MOST (Titov and Gonzalez (1997)), COMCOT (Liu et al. (1998)), CENALT (Gailler et al. (2013)). On more flexible meshes many use the finite element discretisation, such as SELFE (Zhang and Baptista (2008)) and TsunAWI (Harig et al. (2008)), ASCETE (Vater and Behrens (2014)), Firedrake-Fluids (Jacobs and Piggott (2015)) or the finite volume discretisation, such as the VOLNA code (Dutykh et al. (2011)), GeoClaw (George and LeVeque (2006)) or HySEA (Macías et al. (2017)). Another model is described by the Boussinesq equations - these equations and the solver are more complex than shallow-water solvers. Since there is no consensus as to their advantage over it, they are used less commonly, examples include FUNWAVE Kennedy et al. (2000) and COULWAVE (Lynett et al. (2002)). Finally, the 3D Navier-Storkes equations provide the most complete description, but they are significantly more complex than other models - examples include SAGE (Gisler et al. (2006)) and the work of Abadie et al. (2012).

Most of these codes described above work on CPUs, and while there has been some work on GPU implementations by Satria et al. (2012) and Liang et al. (2009), which use structured meshes and finite differences, it is unclear whether these are used in production. As far as we are aware, only Tsunami-HySEA (Macías et al. (2017)), also using finite volumes, is using GPU clusters in production - that code however only supports GPUs, and is hand-written in CUDA.

3 The OP2 Domain Specific Language

The OP2 library (Mudalige et al. (2012)) is a domain specific language embedded in C and Fortran that allows unstructured mesh algorithms to be expressed at a high level, and provides automatic parallelisation and a number of other features. It provides an abstraction that lets the domain scientist describe a mesh using a number of sets (such as quadrilaterals or vertices), connections between these sets (such as edges-to-nodes), and data defined on sets (such as x, y coordinates on vertices). Once the mesh is defined, an algorithm can be implemented as a sequence of parallel loops, each over all elements of a given set applying different "kernel functions", accessing data either directly on the iteration set, or indirectly through at most one level of indirection. This abstraction enables the implementation of a wide range of algorithms, such as the finite volume algorithms that VOLNA uses, but it does require that for any given parallel loop, the order of execution must not affect the end result (within machine precision) - this precludes the implementation of e.g. Gauss-Seidel iterations.

OP2 enables its users to write an application only once using its API, which is then automatically parallelised to utilise multi-core CPUs, GPUs, and large supercomputers through the use of MPI, OpenMP and CUDA. This is done in part through a code generator that parses the parallel loop expressions and generates boilerplate code around the computational kernel to facilitate parallelism and data movement, and in part through different back-end libraries that manage data, including MPI halo exchanges, or GPU memory management, as shown in Figure 1. Fore more details see Giles et al. (2011); Mudalige et al. (2012); op2.

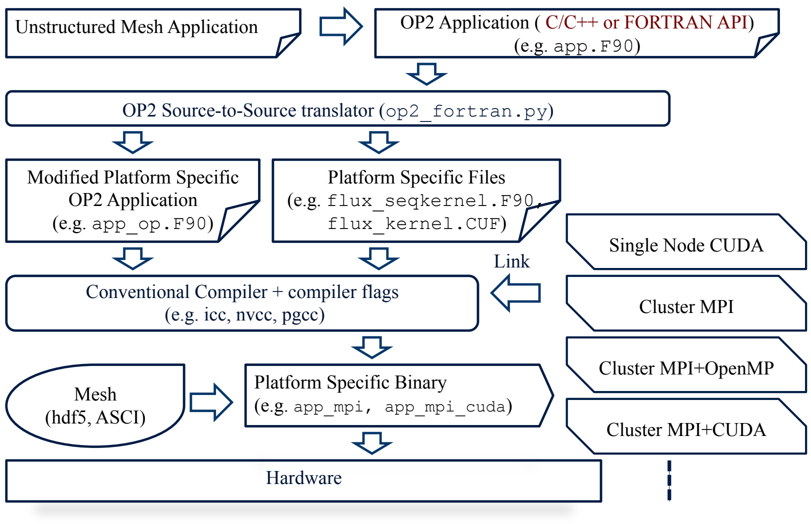

Unstructured Mesh Application
OP2 Application (C/C++ or FORTRAN API)
(e.g. app.F90)
OP2 Source-to-Source translator (op2_fortran.py)
Modified Platform Specific OP2 Application (e.g. app_op.F90)
Platform Specific Files (e.g. flux_seqkernel.F90, flux_kernel.CUF)
Single Node CUDA
Link
Cluster MPI
Conventional Compiler + compiler flags (e.g. icc, nvcc, pgcc)
Cluster MPI+OpenMP
Mesh (hdf5, ASCI)
Platform Specific Binary (e.g. app_mpi, app_mpi_cuda)
Cluster MPI+CUDA
Hardware

Figure 1. Build system with OP2

3.1 Parallelisation Approaches in OP2

OP2 takes full responsibility for orchestrating parallelism and data movement - from the user perspective, the code written looks and feels like sequential C code. To utilise clusters and supercomputers, OP2 uses the Message Passing Interface (MPI) to parallelise in a distributed memory environment; once the mesh is defined by the user, OP2 automatically partitions and distributes it among the available resources. It uses the standard owner-compute approach with halo exchanges, and overlaps computations with communications. In conjunction with MPI, OP2 uses a number of shared-memory parallelisation approaches, such as CUDA and OpenMP.

A key challenge in the fine-grained parallelisation of unstructured mesh algorithms is the avoidance of race conditions when data is indirectly modified. For example, in a parallel loop over edges, when indirectly incrementing data on vertices, multiple edges may try to increment the same vertex, leading to race conditions. OP2 uses a colouring approach to resolve this; elements of the iteration set are grouped into mini-partitions, and each element within these mini-partitions is coloured, so no two elements of the same colour access the same value indirectly. Subsequently mini-partitions are coloured as well. For CUDA, we assign mini-partitions of the same colour to different CUDA thread blocks, and for OpenMP to different threads. There is then a global synchronisation between different colours. In case of CUDA, threads processing elements within each thread block use the first level of colouring to apply increments in a safe way, with block-level synchronisation in-between.

3.2 Input and Output

OP2 supports parallel file I/O through the HDF5 library (The HDF Group (2000-2010)), which is critically important to its integration into VOLNA's workflow: reading in the input problem and writing out data required for analysis simultaneously on multiple processes.

4 The VOLNA simulator

4.1 Model, numerics, previous validation

The finite volume (FV) framework is the most natural numerical method to solve the non-linear shallow water equations (NSWE). It belongs to a class of discretisation schemes that are highly efficient in the numerical solution of systems of conservation laws, which are very common in compressible and incompressible fluid dynamics. Finite Volume methods are preferred over finite differences and often over finite elements because they intrinsically address conservation issues, improving their robustness: total energy, momentum and mass quantities are conserved exactly, assuming no source terms, and appropriate boundary conditions. The code was validated against the classical benchmarks in the tsunami community.

4.2 Numerical model

In a typical megatsunami (say, with characteristic wavelength $\sim 100\,km$, average ocean depth $\sim 4\,km$ and tsunami wave amplitude of $\sim 0.5\,m$), the non-linearity and dispersion are found to be weak (Dutykh et al. (2011)). Although frequency dispersion as the tsunami approaches the shore (or shallow depths) may precipitate local effects, they are overwhelmed by the inundation/run-up caused by the longer waves. Accordingly, the following non-dispersive NSWEs (in Cartesian coordinates) form the physical model of VOLNA:

$$H_t + \nabla \cdot (H\boldsymbol{v}) = 0 \tag{1}$$

$$(H\boldsymbol{v})_t + \nabla \cdot \left(H\boldsymbol{v} \otimes \boldsymbol{v} + \frac{g}{2} H^2 \boldsymbol{I}_2 \right) = -gH\nabla b \tag{2}$$

Here, $b(\boldsymbol{x},t)$ is the time-dependent bathymetry, $\boldsymbol{v}(\boldsymbol{x},t)$ is the horizontal component of the depth-averaged velocity, g is the acceleration due to gravity and $H(\boldsymbol{x},t)$ is the total water depth. Further, $\boldsymbol{I}_2$ is the identity matrix of order 2. The tsunami wave height or elevation of free surface $\eta(\boldsymbol{x},t)$, is computed as,

$$\eta(\boldsymbol{x},t) = H(\boldsymbol{x},t) - b(\boldsymbol{x},t) \tag{3}$$

where the sum of static bathymetry $b_s(\boldsymbol{x})$ and the dynamic seabed uplift $u_z(\boldsymbol{x},t)$ constitute the dynamic bathymetry,

$$b(\boldsymbol{x},t) = b_s(\boldsymbol{x}) + u_z(\boldsymbol{x},t) \tag{4}$$

b_s is usually sourced from bathymetry datasets pertaining to the region of interest (say, global datasets like ETOPO1/GEBCO or regional bathymetries). The vertical component $u_z(\boldsymbol{x},t)$ of the seabed deformation is calculated depending on the physics of tsunami generation, *e.g.* via co-seismic displacement for finite fault segmentations by Gopinathan et al. (2017), submarine sliding by Salmanidou et al. (2017, 2018) *etc.* The time-dependency in $u_z(\boldsymbol{x},t)$ enables the tsunami to be actively generated (Dutykh and Dias (2009)). This is a step-forward from the common passive mode of tsunamigenesis that utilises an instantaneous rupture. The active mode is particularly important for tsunamigenic earthquakes with long and slow ruptures, *e.g.* the 2004 Sumatra-Andaman event (Lay et al. (2005); Gopinathan et al. (2017)) and submerged landslides (Løvholt et al. (2015)), *e.g.* the Rockall Bank event (Salmanidou et al. (2017)).

In addition to the capabilities of employing active generation and consequent tsunami propagation, VOLNA also models the run-up/run-down (*i.e.* the final inundation stage of the tsunami). These three functionalities qualify VOLNA to simulate the entire tsunami life-cycle. The ability of the NSWEs (1-2) to model both propagation, as well as run-up and run-down processes was validated in Kervella et al. (2007) and Dutykh et al. (2011), respectively. Thus, the use of uniform model for the entire life-cycle obviates many technical issues such as the coupling between the sea bed deformation and the sea surface deformation and the use of nested grids.

VOLNA uses the cell-centered approach for control volume tesselation, meaning that degrees of freedom are associated with cell barycenters. A Harten-Lax-van Leer (HLL) numerical flux was used to ensure that the standard conservation and consistency properties are satisfied: the fluxes from adjacent triangles that share an edge exactly cancel when summed and the numerical flux with identical state arguments reduces to the true flux of the same state. Details can be found in Dutykh et al. (2011).

4.3 Code structure

The structure of the code is outline in Algorithm 1; the user inputs a configuration file (.vln), which specifies the mesh to be read in from *gmsh* files, as well as initial/boundary conditions of state variables, such as the bathymetry deformation starting the tsunami, which can be defined in various ways (mathematical expressions or files, or a mix of both). We use a variable timestep second-order Runge-Kutta method for evolving the solution in time. In each iteration, events may be triggered; e.g. further bathymetry deformations, displaying the current simulation time, or outputting simulation data to VTK files for visualisation.

The original VOLNA source code was implemented in C++, utilising libraries such as Boost (Schling (2011)). This gives a very clear structure, abstracting data management, event handling and low level array operations for the higher level algorithm - an example is shown in Figure 2. While this coding style was good for readability, it had its limitations in terms of performance: there was excessive amounts of data movement and certain operations could not be parallelised - indirect increments with potential race conditions in particular. Some features - such as describing the bathymetry lift with a mathematical formula - were implemented with functionality and simplicity, not performance, in mind.

To better support performance and scalability, and thus allow for large-scale simulations, we have re-engineered the VOLNA code to use OP2 - the overall code structure is kept similar, but matters of data management and parallelism are now entrusted to OP2. To support parallel execution we separated the pre-processing step from the main body of the simulation: first the mesh

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-18
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 8 March 2018

Algorithm 1 Code structure of VOLNA

```
Initalise mesh from gmsh file
Initialise state variables
while $t < t_{final}$ do
    Perform pre-iteration events
    Second-order Runge-Kutta time stepper
        Update state variables on cell faces, compute boundary conditions
        Determine fluxes across cell faces and compute timestep
        Apply fluxes and bathymetric source terms to state variables on cells
    Perform post-iteration events
end while
```

```
outConservative.H  *= dt;
outConservative.U  *= dt;
outConservative.V  *= dt;

outConservative.H  += MidPointConservative.H;
outConservative.U  += MidPointConservative.U;
outConservative.V  += MidPointConservative.V;

outConservative.H  += inConservative.H;
outConservative.U  += inConservative.U;
outConservative.V  += inConservative.V;

outConservative.H  *= .5;
outConservative.U  *= .5;
outConservative.V  *= .5;

outConservative.H  =
     ( outConservative.H.cwise()  <= EPS )
     .select( EPS, outConservative.H  );
outConservative.Zb  = inConservative.Zb;
ToPhysicalVariables( outConservative, out );
//Implementation of ToPhysicalVariables:
ScalarValue TruncatedH  =
     ( outConservative.H.cwise()  < EPS )
     .select( EPS, outConservative.H  );
out.H = outConservative.H;
out.U = outConservative.U.cwise()  / TruncatedH;
out.V = outConservative.V.cwise()  / TruncatedH;
out.Zb  = outConservative.Zb;
```

Figure 2. Code snippet from the original VOLNA

```
inline void EvolveValuesRK2_2(const float *dT,
              float *outConservative,
              const float *inConservative,
              const float *midPointConservative,
              float *out)
{
  outConservative[0] *= (*dT);
  outConservative[1] *= (*dT);
  outConservative[2] *= (*dT);

  outConservative[0] += midPointConservative[0];
  outConservative[1] += midPointConservative[1];
  outConservative[2] += midPointConservative[2];

  outConservative[0] += inConservative[0];
  outConservative[1] += inConservative[1];
  outConservative[2] += inConservative[2];

  outConservative[0] *= 0.5f;
  outConservative[1] *= 0.5f;
  outConservative[2] *= 0.5f;

  outConservative[0] = MAX(outConservative[0],EPS);
  outConservative[3] = inConservative[3];

  //call to ToPhysicalVariables inlined
  float TruncatedH = outConservative[0];
  out[0] = outConservative[0];
  out[1] = outConservative[1] / TruncatedH;
  out[2] = outConservative[2] / TruncatedH;
  out[3] = outConservative[3];
}
...
op_par_loop(EvolveValuesRK2_2, "EvolveValuesRK2_2", cells,
  op_arg_gbl(&dT,1,"float", OP_READ),
  op_arg_dat(outConservative,-1,OP_ID,4,"float",OP_RW),
  op_arg_dat(inConservative,-1,OP_ID,4,"float",OP_READ),
  op_arg_dat(midPointConservative,-1, OP_ID,4,"float",OP_READ),
  op_arg_dat(values_new,-1,OP_ID,4,"float", OP_WRITE));
```

Figure 3. Corresponding code in OP2-VOLNA

and simulation parameters are parsed into a HDF5 data file, which can then be read in parallel by the main simulation, which also uses HDF5's parallel file I/O to write results to disk.

Performance-critical parts of the code, essentially any operations on the computational mesh, are re-implemented using OP2: they are written with an element-centric approach and grouped for maximal data reuse. Calculations that were previously a sequence of operations, each calculating all partial results for the entire mesh, now apply only to single elements (such as cells or edges), and OP2 automatically applies these computations to each element - this avoids the use of several temporaries and improves computational density. This process involves outlining the computational "kernel" to be applied at each set element (cell or edge) to a separate function, and writing a call to the OP2 library - a matching code snippet is shown in Figure 3.

The workflow of VOLNA is made of a few sources of information being created and given as inputs to the code. The first is the merged bathymetry and topography over the whole computational domain, i.e. the seafloor and land elevations, over which the flow will propagate. This is given through an unstructured triangular mesh. This is then transformed into a usable input to VOLNA via the *volna2hdf5* code to generate compact HDF5 files. The mesh is also renumbered with the Gibbs-Poole-Stokmeyer algorithm to improve locality.

The second is the dynamic source of the tsunami. It can be an earthquake or a landslide. To describe the temporal evolution of seabed deformation, either a function can be used, or a series of files. When a series of files is used (typically when another numerical model provides the spatio-temporal information of a complex deformation), there is a need to define the frequency of these updates in the so-called *vln* generic input file to VOLNA. A recent improvement has been the ability to define these series of files for a sub-region of the computational domain, and at possibly lower resolution. Performance is better when using a function for the seabed deformation, since I/O requirements for files can generate large overheads - VOLNA-OP2 allows for describing the initial bathymetry with an input file, and then specifying relative deformations using arbitrary code that is a function of spatial coordinates and time. In a similar manner, one can also define initial conditions for wave elevation and velocity.

The generic input file of VOLNA, includes information about the frequency of the updates in the seabed deformation, the virtual gauges where time series of outputs will be produced and possibly some options to output time series of outputs over the whole computational domain in order to create movies for instance. These I/O requirements obviously affect performance: the more data to output and the slower the file system, the larger the effect.

To simulate tsunami hazard for a large number of scenarios is computationally expensive, so VOLNA has been replaced in past studies by a statistical emulator, i.e. a cheap surrogate model of the simulator (Sarri et al. (2012); Beck and Guillas (2016); Liu and Guillas (2017); Salmanidou et al. (2017); Guillas et al. (2018)). To build the emulator, input parameters are varied in a design of experiments, and the runs are submitted with these inputs to collect input-output relationships. The output of interest could for example be the waveforms, free surface elevation, and velocity, among others. The increase in flexibility in the definition of the region over which the earthquake source of the tsunami is defined reduces the size of the series of files used as inputs: this is really helpful when a set of simulations needs to be run. Similarly, the ability to specify the relative deformation using an arbitrary code that is a function of spatial coordinates and time also reduces the computational and memory overheads when running a set of simulations.

5 Results

5.1 Running VOLNA

A key goal of this paper is to demonstrate that by utilising the OP2 library, VOLNA delivers scalable high performance on a number of common systems. Therefore we take a testcase simulating tsunami propagation in the Indian Ocean, and run it on three different machines: NVIDIA P100 Graphical Processing Units (specifically a DGX-1 system in the UK's JADE supercomputer), a classical CPU architecture in the UK's Archer supercomputer (specifically dual-socket Intel Xeon 12-core Ivy Bridge CPUs), and Intel's Xeon Phi platform (64-core Knights Landing-generation chips, configured in cache mode).

There are four key computational stages that make up 90% of the total runtime: a stage evolving time using the second-order Runge-Kutta scheme (*RK*), a stage that computes the physical across the edges of the mesh (*fluxes*), a stage that computes the minimum timestep (*dT*), and a stage that applies the fluxes to the cell-centered state variables (*applyFluxes*). Each of these stages consist of multiple steps, but for performance analysis we study them in groups.

Table 1. Details of the triangular meshes

Mesh	Name	Size	Vertices	Edges	Triangles
	$(h \sim 0.9\,km)$	n_V	n_E	n_T	
M_1	23.1M	h	11545882	34627495	23081614
M_2	5.7M	$2h$	2897099	8681146	5784048
M_3	1.4M	$4h$	732710	2187979	1455270

The *RK* stage is computationally fairly simple, no indirect accesses are made, cell-centered state variables are updated using other cell-centered state variables, and therefore parallelism is easy to exploit, and the limiting factor to performance will be the speed at which we can move data; achieved bandwidth. The *fluxes* stage is computationally complex, and the results are edge-centered, therefore large amounts of data accessed indirectly through an edges-to-cells mapping. The *dT* stage moves significant amounts of data to compute the appropriate timestep for each cell, and then carries out a global reduction to calculate the minimum - particularly over MPI this can be an expensive operation. The *applyFluxes* stage, while computationally simple, is complex due to its indirect increment access patterns; per-edge values have to be added onto cell-centered values, and in parallelising this operation, OP2 needs to make sure to avoid race conditions.

5.2 Tsunami demonstration case

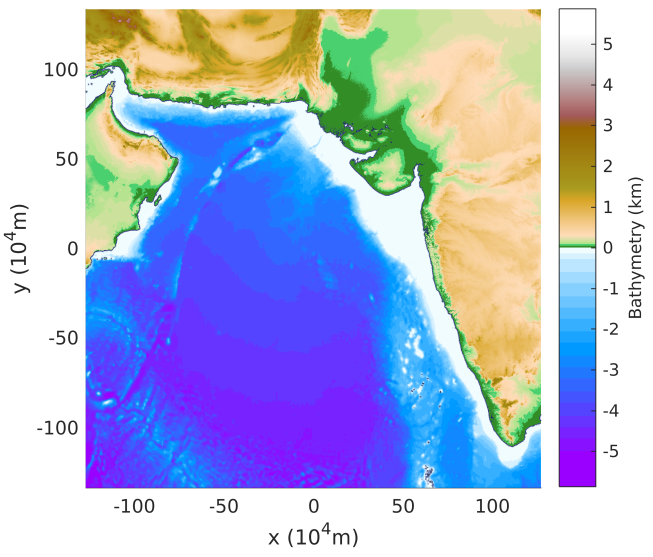
For performance and scaling analysis, we employ the Makran subduction zone as the tsunamigenic source for the numerical simulations. Our region of interest extends from $55°E$ to $79°E$ and from $6°N$ to $30°N$. The bathymetry (Fig. 4a) is obtained from GEBCO (www.gebco.net). The region of interest is projected about the center latitude (*i.e.* $18°N$) to form the rectangular computational domain for VOLNA in Cartesian co-ordinates (Fig. 4b). This translates to a region of approximately $2500\,km \times 2700\,km$ in area. The calculation of the sea-floor deformation or uplift (assumed instantaneous) is modeled via the Okada solution (Okada (1992)). This deformation is generated by the earthquake source which is modeled as a 4-segment finite fault model (Table 2) with a uniform slip of 30m. Although non-uniform meshes are essential for resolution of certain effects like inundation, port vortices and velocities, for the purpose of ascertaining the speed-up we restrict ourselves to uniform meshes. These uniform triangular meshes for the simulations are generated using Gmsh (Geuzaine and Remacle (2009)). The characteristic mesh sizes are h, $2h$, $4h$ (Table 1), where $h \sim 0.9\,km \sim$ GEBCO resolution of 30". Finally, the total number of virtual gauges is around 10^4 distributed in a uniform 100×100 grid.

5.3 Performance and Scaling on classical CPUs

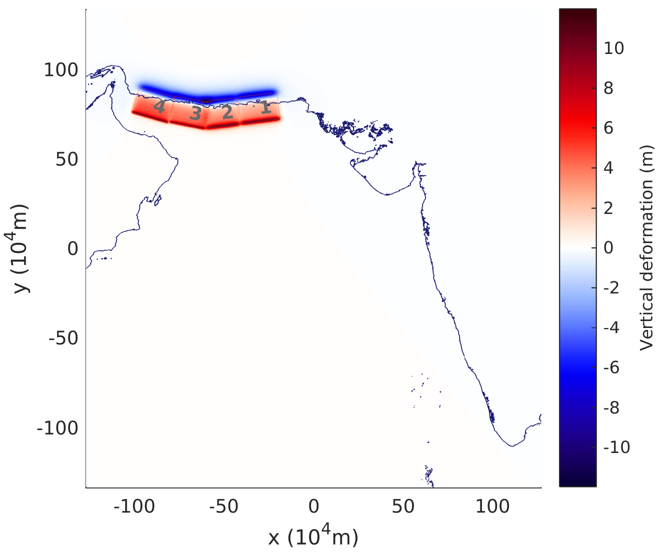
As the most commonly used architecture, we first evaluate performance on a classical CPUs in the Archer supercomputer: dual-socket Xeon E5-2697 v2 CPU, with 12 cores each, supporting the AVX instruction set. We use a hybrid MPI+OpenMP configuration, with 2 MPI processes per node (1 per socket), and 12 OpenMP threads each.

Table 2. Finite fault parameters of the 4-segment tsunamigenic earthquake source

Segment	Length	Down-dip width	Longitude	Latitude	Depth	Strike	Dip	Rake
i	(km)	(km)	(°)	(°)	(km)	(°)	(°)	(°)
1	220	150	65.23	24.50	10	263	6	90
2	188	150	63.08	24.23	10	263	7	90
3	199	150	61.25	24.00	5	281	8	90
4	209	150	59.32	24.32	5	286	9	90



(a) Bathymetry



(b) Seabed deformation

Figure 4. (a) Bathymetry from GEBCO's geodetic grid is mapped onto a Cartesian grid for use in VOLNA. (b) Uplift caused by a uniform slip of 30m in the 4 segment finite fault model (given in Table 2).

First, we compared two code variants generated by OP2: a 'plain' version, and one with extra code generated to enable AVX vectorisation. The *RK* stage performs the same in both variants, however the *fluxes* and *dT* stages saw significant performance gains - the compiler did not automatically vectorise computations in these stages, it had to be forced to do so. The *applyFluxes* stage could not be vectorised due to race conditions.

On a single node, running the largest mesh, 26% of time was spent in the *RK* stage, achieving 96 GB/s throughput on average, 29% of time was spent in the *fluxes* stage, achieving 68 GB/s, 13% of time was spent in the *dT* phase, achieving 76 GB/s, and 30% of time was spent in the *applyFluxes* stage, achieving 76 GB/s. The maximum bandwidth on this platform is 120 GB/s. The time spent in MPI communications ranged from 7% on the smallest mesh to 1.9% on the largest mesh.

When scaling to multiple nodes as shown in Figure 5, it is particularly evident on the smallest problem that the problem size per node needs to remain reasonable, otherwise MPI communications will dominate the runtime: for the 1.4M mesh, at

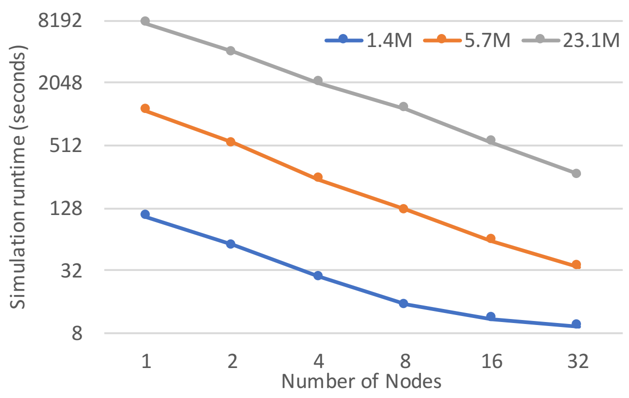


Figure 5. Performance scaling on Archer (Intel Xeon CPU) at three different mesh sizes with 1.4, 5.7, and 23.1 million triangles

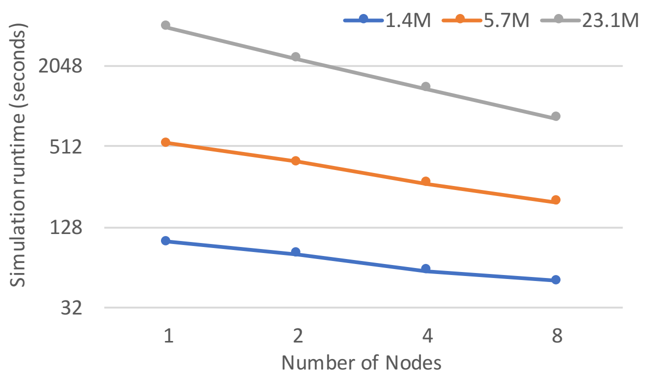


Figure 6. Performance scaling on Archer-KNL (Intel Xeon Phi) at different mesh sizes

32 nodes 5.8 seconds out of 9.36 total (62%). This can be characterised by the *strong scaling efficiency*; when doubling the number of computational resources (nodes), what percentage of the ideal 2× speedup is achieved. For the smallest problem, scaling efficiency is above 90% up to 8 nodes, but at 16 and 32 it falls rapidly, to 69% and 58% respectively. On M_2 efficiency is above 95% up to 16 nodes, and falls to 87% at 32 nodes, and for the largest mesh, it remains above 95%.

5.4 Performance and Scaling on the Intel Xeon Phi

Second, we evaluate Intel's latest many-core chip, the Xeon Phi, which integrates 64 cores, each equipped with AVX-512 vector processing units and supporting 4 threads, and built with a 16GB on-chip high-bandwidth memory, here used as a cache for off-chip DDR4 memory. The chips were configured in the "quad" mode, and all 16GB as cache. We use 4 MPI processes, one per quadrant, and 32 OpenMP threads each. Vectorisation on this platform is paramount for achieving high performance - every stage with the exception of *applyFluxes* was vectorised - the latter was not due to compiler issues.

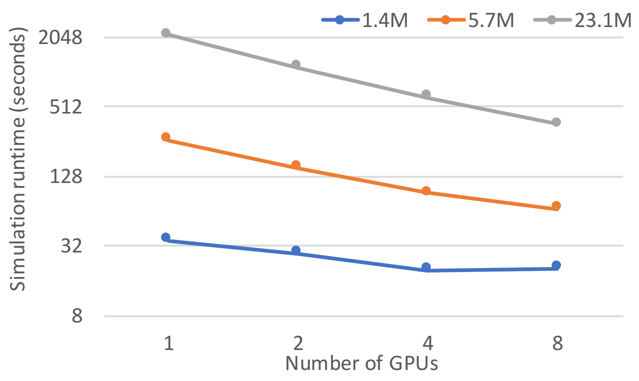


Figure 7. Performance scaling on JADE (P100 GPU) at different mesh sizes

On a single node, the straightforward computations of the *RK* stage can utilise the available high bandwidth very efficiently: only 15% of time spent here, achieving a 352 GB/s. The *fluxes* stage takes 25% of time and achieves 156 GB/s, *dT* 12.5% and achieves 160 GB/s, and the *applyFluxes* stage takes 47% and achieves only 92 GB/s - largely due to the lack of vectorisation and irregular memory access patterns. On the largest mesh, it is 1.92× faster than a single node of the classical CPU system, but on the smallest it is only 8% faster - due to underutilisation.

Performance when scaling to multiple nodes is shown in Figure 6: it is quite clear that scaling is worse than on the classical CPU architecture - the Xeon Phi requires a considerably larger problem size per node to operate efficiently. Strong scaling efficiency is particularly poor on the smallest mesh at 58-61%, but even on the largest mesh it's only between 82-87%. At scale therefore the performance advantage over the classical CPU system becomes lower.

5.5 Performance and Scaling on P100 GPUs

Third, we evaluate performance on GPUs - an architecture that has continually been increasing its market share in high performance computing thanks to its efficient parallel architecture. The P100 GPUs are hosted in NVIDIA's DGX-1 system, they are interconnected by high-speed NVLink connections. Each chip contains 60 Scalar Multiprocessors, with 64 CUDA cores each, giving a total of 3840 cores. There is also 16 GB of high-bandwidth memory on-package. To utilise these devices, we use CUDA code generated by OP2, and compiled with CUDA 8. Similarly to Intel's Xeon Phi, high vector efficiency is required for good performance on the GPU.

On a single GPU, running the largest mesh, 25% of runtime is spent in the *RK* stage, achieving 281 GB/s, *fluxes* takes 15%, achieving 528 GB/s thanks to a high degree of data re-use in indirect accesses, *dT* takes 11.4% and achieves 490 GB/s, and finally *applyFluxes* takes 49% of time, achieving 223 GB/s. Indeed, this last phase has the most irregular memory access patterns, which is commonly known to be degrading performance on GPUs. Nevertheless, even a single GPU outperforms a classical CPU node by a factor of 3.6, and the Xeon Phi by 1.89×. On the smallest problem, this ratio is only 2.9 - more parallelism is required for efficient operation.

Performance when scaling to multiple GPUs is shown in Figure 7: similarly to the Xeon Phi, GPUs are also sensitive to the problem size and the overhead of MPI communications, even more so than the Phi. On the smallest problem efficiency drops from 64% to 48%, and on the largest problem from 95% to 85%. Even so, at 8 GPUs it is still 2.3/3.3× faster than 8 nodes of the Phi or the CPU system respectively.

5.6 Power Consumption Considerations

When running large numbers of long simulations, the cost-to-solution is an important metric, and a primary driver of this is energy consumption - admittedly the cost of core-hours or GPU-hours varies significantly, here we do not look at specific prices. However, in terms of energy efficiency, a dual-socket CPU consumes up to 260 Watts, which is then roughly tripled when looking at the whole node, due to memory, disks, networking, etc. In comparison, the Intel Xeon Phi CPU has a TDP of 215 W, roughly 750 W for the node. A P100 GPU has a TDP of 300 W, but has to be hosted in a CPU system - the more GPUs in a single machine, the better amortised this cost is: the TDP of a DGX-1 system is 3.2 KW (8x300 for the GPUs, plus 800 for the rest of the system) - which averages to 400 W per GPU. Thus in terms of power efficiency GPUs are by far the best choice for VOLNA. Nevertheless, a key benefit of VOLNA-OP2, is that it can efficiently utilise any high performance hardware commonly available.

6 Conclusions

In this paper we have introduced and described the VOLNA-OP2 code; a tsunami simulator built on the OP2 library, enabling execution on CPUs, GPUs, and heterogeneous supercomputers. By building on OP2, the science code of VOLNA itself is written only once using a high-level abstraction, capturing *what* to compute, but not *how* to compute it. This approach enables OP2 to take control of the data structures and parallel execution; VOLNA is then automatically translated to use sequential execution, OpenMP, or CUDA, and by linking with the appropriate OP2 back-end library, these are then combined with MPI. This approach also future-proofs the science code: as new architectures come along, the developers of OP2 will update the back-ends and the code generators, allowing VOLNA to make use of them without further effort. This kind of ease-of-use and portability makes VOLNA-OP2 unique between the tsunami simulation codes.

We have described the key features of VOLNA, the discretisation of the underlying physical model (*i.e.* NSWE) in the finite volume context and the second-order Runge-Kutta timestepper, as well as the input/output features that allow the integration of the simulation step into a larger workflow; initial conditions, and bathymetry in particular, can be specified in a number of ways to minimise I/O requirements, and efficient parallel output is used to write out simulation data on the full mesh or specified points.

We have discussed how VOLNA-OP2 is currently being used in different of ways, with a particular focus on its use within a statistical emulator; there are already published papers on this: Beck and Guillas (2016); Liu and Guillas (2017); Salmanidou et al. (2017).

Through performance scaling and analysis of the code on traditional CPU clusters, as well as GPUs and Intel's Xeon Phi, we have demonstrated that VOLNA-OP2 indeed delivers near-optimal performance on a variety of platforms and, depending on problem size, scales well to multiple nodes.

There is still a need for even more streamlined and efficient workflows. For instance, we could integrate within VOLNA, the finite fault source model for the slip with some assumptions on the rupture dynamics, we could also integrate the bathymetry-based meshing (the mesh needs to be tailored to the depth and gradients of the bathymetry to optimally reduce computational time). Indeed, there would be even less exchanges of files and more efficient computations, especially in the context of uncertainty quantification tasks such as emulation or inversion.

In the end, the gain in computational efficiency will allow higher resolution modelling, such as using 2 m topography and bathymetry collected from LIDAR, i.e. a greater capability. It will allow greater capacity by enabling more simulations to be performed. Both of these enhancements will subsequently lead to better warnings more tailored to the actual impact on the coast as well as better urban planning since hazard maps will gain in precision geographically and probabilistically, due to the possibility of exploring a larger number of more realistic scenarios.

Code availability. The code is available at https://github.com/reguly/volna/ It depends on the OP2 library, which is also available at: https://github.com/OP-DSL/OP2-Common, and depends on an MPI distribution, parallel HDF5, and a partitioner, such as ParMetis or PT-Scotch. For GPU execution, the CUDA SDK and a compatible device is required.

Competing interests. The authors declare that they have no conflict of interest.

Acknowledgements. We would like to thank Endre Lśzló, formerly of PPCU ITK, who worked in the initial port of Volna to OP2. István Reguly was supported by the János Bólyai Research Scholarship of the Hungarian Academy of Sciences. The authors would like to acknowledge the use of the University of Oxford Advanced Research Computing (ARC) facility in carrying out this work http://dx.doi.org/10.5281/zenodo.22558. SG gratefully acknowledges support through the NERC grants PURE (Probability, Uncertainty and Risk in the Natural Environment) NE/J017434/1, and "A demonstration tsunami catastrophe risk model for the insurance industry" NE/L002752/1. SG and DG acknowledge support from the NERC project (NE/P016367/1) under the Global Challenges Research Fund: Building Resilience programme. DG acknowledges support from the Royal Society, UK and Science and Engineering Research Board (SERB), India for the Royal Society-SERB Newton International Fellowship (NF151483).

References

OP2 github repository, https://github.com/OP2/OP2-Common.

Abadie, S. M., Harris, J. C., Grilli, S. T., and Fabre, R.: Numerical modeling of tsunami waves generated by the flank collapse of the Cumbre Vieja Volcano (La Palma, Canary Islands): Tsunami source and near field effects, Journal of Geophysical Research: Oceans, 117, n/a–n/a, https://doi.org/10.1029/2011JC007646, http://dx.doi.org/10.1029/2011JC007646, c05030, 2012.

Beck, J. and Guillas, S.: Sequential Design with Mutual Information for Computer Experiments (MICE): Emulation of a Tsunami Model, SIAM/ASA Journal on Uncertainty Quantification, 4, 739–766, https://doi.org/10.1137/140989613, https://doi.org/10.1137/140989613, 2016.

Behrens, J. and Dias, F.: New computational methods in tsunami science, Phil. Trans. R. Soc. A, 373, 20140 382, 2015.

Bernard, E., Mofjeld, H., Titov, V., Synolakis, C., and González, F.: Tsunami: scientific frontiers, mitigation, forecasting and policy implications, Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, 364, 1989–2007, https://doi.org/10.1098/rsta.2006.1809, http://rsta.royalsocietypublishing.org/content/364/1845/1989, 2006.

Dias, F., Dutykh, D., O'Brien, L., Renzi, E., and Stefanakis, T.: On the Modelling of Tsunami Generation and Tsunami Inundation, Procedia IUTAM, 10, 338 – 355, https://doi.org/https://doi.org/10.1016/j.piutam.2014.01.029, http://www.sciencedirect.com/science/article/pii/S2210983814000303, mechanics for the World: Proceedings of the 23rd International Congress of Theoretical and Applied Mechanics, ICTAM2012, 2014.

Dutykh, D. and Dias, F.: Tsunami generation by dynamic displacement of sea bed due to dip-slip faulting, Mathematics and Computers in Simulation, 80, 837 – 848, https://doi.org/http://dx.doi.org/10.1016/j.matcom.2009.08.036, http://www.sciencedirect.com/science/article/pii/S0378475409002821, 2009.

Dutykh, D., Poncet, R., and Dias, F.: The VOLNA code for the numerical modeling of tsunami waves: Generation, propagation and inundation, European Journal of Mechanics-B/Fluids, 30, 598–615, 2011.

Gailler, A., Hébert, H., Loevenbruck, A., and Hernandez, B.: Simulation systems for tsunami wave propagation forecasting within the French tsunami warning center, Natural Hazards and Earth System Sciences, 13, 2465, 2013.

George, D. L. and LeVeque, R. J.: Finite volume methods and adaptive refinement for global tsunami propagation and local inundation, 2006.

Geuzaine, C. and Remacle, J.-F.: Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities, International Journal for Numerical Methods in Engineering, 79, 1309–1331, https://doi.org/10.1002/nme.2579, http://dx.doi.org/10.1002/nme.2579, 2009.

Giles, M. B., Mudalige, G. R., Sharif, Z., Markall, G., and Kelly, P. H.: Performance analysis and optimization of the OP2 framework on many-core architectures, The Computer Journal, 55, 168–180, 2011.

Gisler, G., Weaver, R., and Gittings, M. L.: SAGE calculations of the tsunami threat from La Palma, Sci. Tsunami Hazards, 24, 288–312, 2006.

Gopinathan, D., Venugopal, M., Roy, D., Rajendran, K., Guillas, S., and Dias, F.: Uncertainties in the 2004 Sumatra–Andaman source through nonlinear stochastic inversion of tsunami waves, Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, 473, https://doi.org/10.1098/rspa.2017.0353, http://rspa.royalsocietypublishing.org/content/473/2205/20170353, 2017.

Guillas, S., Sarri, A., Day, S., Liu, X., and Dias, F.: Functional emulation of high resolution tsunami modelling over Cascadia, Annals of Applied Statistics, to appear, 2018.

Harig, S., Pranowo, W. S., and Behrens, J.: Tsunami simulations on several scales, Ocean Dynamics, 58, 429–440, 2008.

Jacobs, C. T. and Piggott, M. D.: Firedrake-Fluids v0.1: numerical modelling of shallow water flows using an automated solution framework, Geoscientific Model Development, 8, 533–547, https://doi.org/10.5194/gmd-8-533-2015, https://www.geosci-model-dev.net/8/533/2015/, 2015.

Kennedy, A. B., Chen, Q., Kirby, J. T., and Dalrymple, R. A.: Boussinesq modeling of wave transformation, breaking, and runup. I: 1D, Journal of waterway, port, coastal, and ocean engineering, 126, 39–47, 2000.

Kervella, Y., Dutykh, D., and Dias, F.: Comparison between three-dimensional linear and nonlinear tsunami generation models, Theoretical and Computational Fluid Dynamics, 21, 245–269, https://doi.org/10.1007/s00162-007-0047-0, https://doi.org/10.1007/s00162-007-0047-0, 2007.

Lawrence, B. N., Rezny, M., Budich, R., Bauer, P., Behrens, J., Carter, M., Deconinck, W., Ford, R., Maynard, C., Mullerworth, S., Osuna, C., Porter, A., Serradell, K., Valcke, S., Wedi, N., and Wilson, S.: Crossing the Chasm: How to develop weather and climate models for next generation computers?, Geoscientific Model Development Discussions, 2017, 1–36, https://doi.org/10.5194/gmd-2017-186, https://www.geosci-model-dev-discuss.net/gmd-2017-186/, 2017.

Lay, T., Kanamori, H., Ammon, C. J., Nettles, M., Ward, S. N., Aster, R. C., Beck, S. L., Bilek, S. L., Brudzinski, M. R., Butler, R., DeShon, H. R., Ekström, G., Satake, K., and Sipkin, S.: The Great Sumatra-Andaman Earthquake of 26 December 2004, Science, 308, 1127–1133, https://doi.org/10.1126/science.1112250, http://science.sciencemag.org/content/308/5725/1127, 2005.

Liang, W.-Y., Hsieh, T.-J., Satria, M. T., Chang, Y.-L., Fang, J.-P., Chen, C.-C., and Han, C.-C.: A GPU-Based Simulation of Tsunami Propagation and Inundation, pp. 593–603, Springer Berlin Heidelberg, Berlin, Heidelberg, https://doi.org/10.1007/978-3-642-03095-6_56, https://doi.org/10.1007/978-3-642-03095-6_56, 2009.

Liu, P. L., Woo, S.-B., and Cho, Y.-S.: Computer programs for tsunami propagation and inundation, Cornell University, 1998.

Liu, X. and Guillas, S.: Dimension Reduction for Gaussian Process Emulation: An Application to the Influence of Bathymetry on Tsunami Heights, SIAM/ASA Journal on Uncertainty Quantification, 5, 787–812, https://doi.org/10.1137/16M1090648, https://doi.org/10.1137/16M1090648, 2017.

Løvholt, F., Pedersen, G., Harbitz, C. B., Glimsdal, S., and Kim, J.: On the characteristics of landslide tsunamis, Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, 373, https://doi.org/10.1098/rsta.2014.0376, http://rsta.royalsocietypublishing.org/content/373/2053/20140376, 2015.

Lynett, P. J., Wu, T.-R., and Liu, P. L.-F.: Modeling wave runup with depth-integrated equations, Coastal Engineering, 46, 89–107, 2002.

Macías, J., Castro, M. J., Ortega, S., Escalante, C., and González-Vida, J. M.: Performance Benchmarking of Tsunami-HySEA Model for NTHMP's Inundation Mapping Activities, Pure and Applied Geophysics, 174, 3147–3183, https://doi.org/10.1007/s00024-017-1583-1, https://doi.org/10.1007/s00024-017-1583-1, 2017.

Mudalige, G., Giles, M., Reguly, I., Bertolli, C., and Kelly, P.: OP2: An active library framework for solving unstructured mesh-based applications on multi-core and many-core architectures, in: Innovative Parallel Computing (InPar), 2012, pp. 1–12, IEEE, 2012.

Okada, Y.: Internal deformation due to shear and tensile faults in a half-space, Bulletin of the Seismological Society of America, 82, 1018–1040, http://www.bssaonline.org/content/82/2/1018.abstract, 1992.

O'Brien, L., Christodoulides, P., Renzi, E., Stefanakis, T., and Dias, F.: Will oscillating wave surge converters survive tsunamis?, Theoretical and Applied Mechanics Letters, 5, 160–166, 2015.

Poncet, R., Campbell, C., Dias, F., Locat, J., and Mosher, D.: A Study of the Tsunami Effects of Two Landslides in the St. Lawrence Estuary, pp. 755–764, Springer Netherlands, Dordrecht, https://doi.org/10.1007/978-90-481-3071-9_61, https://doi.org/10.1007/978-90-481-3071-9_61, 2010.

Salmanidou, D. M., Guillas, S., Georgiopoulou, A., and Dias, F.: Statistical emulation of landslide-induced tsunamis at the Rockall Bank, NE Atlantic, Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences, 473, https://doi.org/10.1098/rspa.2017.0026, http://rspa.royalsocietypublishing.org/content/473/2200/20170026, 2017.

Salmanidou, D. M., Georgiopoulou, A., Guillas, S., and Dias, F.: Rheological considerations for the modelling of submarine sliding at Rockall Bank, NE Atlantic Ocean, Physics of Fluids, 2018.

Sarri, A., Guillas, S., and Dias, F.: Statistical emulation of a tsunami model for sensitivity analysis and uncertainty quantification, Natural Hazards and Earth System Sciences, 12, 2003–2018, 2012.

Satria, M. T., Huang, B., Hsieh, T.-J., Chang, Y.-L., and Liang, W.-Y.: GPU acceleration of tsunami propagation model, IEEE Journal of Selected Topics in Applied Earth Observations and remote Sensing, 5, 1014–1023, 2012.

Schling, B.: The Boost C++ Libraries, XML Press, 2011.

Stefanakis, T. S., Contal, E., Vayatis, N., Dias, F., and Synolakis, C. E.: Can small islands protect nearby coasts from tsunamis? An active experimental design approach, in: Proc. R. Soc. A, vol. 470, p. 20140575, The Royal Society, 2014.

The HDF Group: Hierarchical data format version 5, http://www.hdfgroup.org/HDF5, 2000-2010.

Titov, V. V. and Gonzalez, F. I.: Implementation and testing of the method of splitting tsunami (MOST) model, Tech. rep., NOAA Technical Memorandum ERL PMEL-112, 11 pp UNIDATA, 1997.

Vater, S. and Behrens, J.: Well-balanced inundation modeling for shallow-water flows with discontinuous Galerkin schemes, in: Finite volumes for complex applications VII-elliptic, parabolic and hyperbolic problems, pp. 965–973, Springer, 2014.

Zhang, Y. J. and Baptista, A. M.: An Efficient and Robust Tsunami Model on Unstructured Grids. Part I: Inundation Benchmarks, Pure and Applied Geophysics, 165, 2229–2248, https://doi.org/10.1007/s00024-008-0424-7, https://doi.org/10.1007/s00024-008-0424-7, 2008.