VOLNA-OP2 documentation

Istvan Reguly - istvan.reguly@oerc.ox.ac.uk

December 17, 2014

1 Introduction

This document describes the structure, compilation and usage of VOLNA-OP2.

2 Installation

The installation and compilation of VOLNA requires the following libraries

1. MPI - Any MPI 2.0+ distribution
2. Parallel HDF5 - for parallel file I/O
3. ParMetis version 3 or 4
4. PT-Scotch version 5
5. OP2 library
6. optionally CUDA

A number of environment variables have to be set for the compilation of OP2 and VOLNA to work, these usually point to the root directory of the library, which contains the `lib` and `include` directories: `MPI_INSTALL_PATH, HDF_INSTALL_PATH, PARMETIS_INSTALL_PATH, PTSCOTCH_INSTALL_PATH, OP2_INSTALL_PATH, CUDA_INSTALL_PATH`.

2.1 Emerald

On the Emerald system, relying on the Intel toolchain, these can be set by loading the appropriate modules:
```
module load intel/14.0
module load cuda/6.5.14
module load contrib/HDF5-1.8-parallel-intel
module load contrib/ParMetis-3.2.0-intel
module load contrib/Scotch-5.1-intel
```
as well as setting
```
export OP2_INSTALL_PATH=/home/oxford/eisox069/OP2-Common-3/op2/
```

3 VOLNA directory structure

The source code is organised as follows:

1. `volna_init`: contains dependencies from the old VOLNA source required for initialisation
2. `sp`: contains the main VOLNA-OP2 code, when built, generates the executables to be used on different platforms
   (a) volna2hdf5: a tool for converting input files to HDF5 files to be used for the VOLNA-OP2 executable
   (b) hdf52vtk: a tool for converting the ouput of VOLNA-OP2 in HDF5 format to VTK files.

4 volna2hdf5

This tool is key for setting up a simulation, it takes a `vln` file as input and generates a hdf5 file that will serve as the input for the actual simulation. Several of these are considered *events* that are triggered during the simulation at prescribed intervals, they are generally characterised by three parameters { `istart = ..  istep = ..  iend = ..` } ; these specify the when the event occurs, starting at a given timestep and ending at an other - if no start is given, then 0 is assumed, if no end is given then it will keep happening until the simulation ends. If no step is given, it defaults to every step.

The `vln` file will have the following structure:

1. `Time { end = ...  dtmax = ...  }`
   which specifies the simulation time and *optionally* the maximum timestep.
2. `Mesh "path/mesh.msh"` or
   `Mesh Rectangle p1x p1y p2x p2y ncellx ncelly`
   which specifies the simulation mesh, either as a *Gmsh* file, or as a rectangle, defined by two points `p1, p2` and divided into equally-sized triangles: `2*ncellx*ncelly`.
3. `PhysicalParams { g = 9.81 }` and
   `NumericalParams { cfl = .45 }`
   define physical and numerical parameters of the simulation
4. `Init {istart = 0 istep = 1 iend = 10} TYPE { return f(x,y,t);}` or
   `Init {istart = 0 istep = 1 iend = 10} TYPE "filename"`
   Defines initial conditions where `TYPE` is one of the following: `H, U, V, Bathymetry` depending on which state variable is to be set. The initial/boundary condition is imposed beginning at simulation step defined by `istart` every `istep` until `iend` is reached.

   If a fuction is defined, it has to be a one-line expression returning a single value, that may be a function of `x,y,t`, which is going to be computed for each cell, with barycenter `x,y` at timestep `t`. *Important*: with formulas, a header file will be generated, one directory up from where volna2hdf5 is run, if it's not in the default place, you will have to copy those files to the `sp` directory and recompile VOLNA. A more complicated expression can be used by editing the generated header file (e.g. `initBathymetry_formula.h` for Bathymetry events), and then recompiling VOLNA.

   If a file is defined, and `istart=iend` (i.e. it's only applied once at the beginning, then that file will contain the value for each and every cell, one cell per line. If `istart<iend`, then a

sequence of input files are required, the filename has to contain the letters *%i* which will be substituted with the iteration number (padded with 0s). E.g. for a filename `"bathy%i.txt"`, it will look for files named as `bathy0000.txt, bathy0001.txt`, etc. For the formatting of these files, please see Section 4.1.

Important: when using files, the deformations are generated with a time interval given by the generator. By default VOLNA chooses the correct timestep to satisfy the CFL condition, but this will be different from the time intervals used by the generation process, therefore the `dtmax` parameter has to be added to the `Time {}` field. AN implementation is currently being worked on where the two can be decoupled.

5. `OutputTime { istart = 0 istep = 1 iend = 10 } "cout"`
   displays the current iteration number and timestep on the console, `istart` and `iend` are usually omitted.

6. `OutputSimulation { istart = 0 istep = 1 iend = 10 } "output_%i.vtk"`
   Saves all the state variables as specified by `istart, istep, iend`, in order (H,U,V,Zb). Outputs either VTK files or HDF5 files, for configuration, please see Section 5.

7. `OutputMaxElevation { istart = 0 istep = 1 iend = 10 } "output_%i.vtk"`
   Saves the value of state variables when H is maximum so far during the simulation. If only `istep=-1` is defined, it will save all the sate variables at the very first step and when H (wave elevation from the reference 0 elevation) reaches the maximum overall during the whole simulation before exiting.

8. `OutputLocation { istep = 1 } "output.txt" { x = ... y = ...}` Saves total elevation (H+Zb), at a particular location in the simulation domain into the output file. All location events are aggregated into a gauges.h5 file, please see Section 5.2.

4.1 Bathymetry files

The original VOLNA used input files, where each file listed the absolute bathymetry value for each cell in the simulation mesh, however this results in an excessive amount of data for long simulations. This input method can still be used, just specify the argument `old-format` when running `volna2hdf5` and VOLNA itself.

The new file format (v3 files), is as follows: if your string is something like "bathy%i.txt" then it will look for a bathy_init.txt, which specifies the initial bathymetry for each cell center (same format as before), a bathy_geom.txt which describes a coarse mesh which will define the bathymetry deformations, this file is structured as follows: first line number of cells and number of points, next #(number of cells) number of lines listing the indices of the points of each cell, then #(number of points) number of lines, each with 3 double precision values for the x and y and z coordinates of the points. Then, it will read the actual bathymetry deformations on each point on the coarse mesh, from a sequence of files, as defined by the description of the event. The coarse mesh is then used during the simulation to compute bathymetry deformations from the initial fine-resolution bathymetry, computing for each simulation cell which coarse cell it lies in, and interpolation the deformation from the coarse cell's three vertices.

4.2 Running volna2hdf5

To run volna2hdf5, type the following command:
./volna2hdf5 input_vln_file.vln

Optionally defining `old-format` if old format bathymetry files are used.

5 Running VOLNA

If you used a formula for one of the `Init` events, you will have to recompile VOLNA. Then, select which implementaion you want to use (CPU, GPU, MPI, etc), for example with a multi-GPU run:
`mpirun ... ./volna_mpi_cuda input_file.h5 0 OP_BLOCK_SIZE=128 OP_PART_SIZE=128`
The first argument is always the input generated by volna2hdf5, the second argument is the format for outputting OutputSimulation and OutputMaxElevation files (0 - HDF5, 1 -ASCII VTK, 2 - Binary VTK). For distributed memory execution, you can only use HDF5 files. The third and fourth arguments set CUDA specific parameters for OP2, the values above are usually best. Optionally you may have to add `old-format` if your input hdf5 file uses old format bathymetry files.

5.1 VOLNA executables

Compilation will generate a number of executables, to be used in different situations and on different hardware. The parameters described above are the same.

Single process versions are: `volna_seq, volna_openmp, volna_cuda` for single-threaded CPU execution, multi-threaded CPU execution and single GPU execution respectively.

Distributed memory versions are `volna_mpi, volna_mpi_openmp, volna_mpi_cuda`: for plain MPI, hybrid MPI+OpenMP and multi-GPU execution respectively.

5.2 OutputLocation files

OutputLocation events are bundled together, if they all use the same timings, into a gauges.h5 file, which is compressed to save space. To read the file, you can use any hdf5 tool, there are two datasets written: /dims which has 2 integer fields, the first indicating the size in the contiguous direction (number of OutputLocation events + 1 for timestaps) and the second in the strided dimension (number of timestamps). Data is stored under /gauges, a 1D float array of size $dims[0] * dims[1]$. To get the timestamp at iteration T and the value for gauge N (indexed $1...dims[0]-1$), access $gauges[T * dims[0]]$ for the timestamp and $gauges[T * dims[0] + N] - T$ has to be less than $dims[1]$. A python script is attached (read_gauges.py) that shows an example of this.

For example to print the timestamps and values at gauge number 10:

```
#!/usr/bin/env python
import sys
import re
import datetime
import h5py

f = h5py.File("gauges.h5","r")
dims = f['/dims']
data = f['/gauges']
print 'timestamp','gauge 10'
for i in range(0,dims[1]):
print data[i*dims[0]], data[i*dims[0]+11]
f.close()
```

5.3 OutputSimulation files

When running VOLNA, the second argument specifies the file format for outputting these files (0 - HDF5, 1 -ASCII VTK, 2 - Binary VTK). If HDF5 output is chosen (which is strongly recommended for performance), these files will have to be converted into VTK files with the hdf52vtk tool provided. To convert all the output h5 files in a directory, use the following script:
`hdf52vtk_convert_files.sh simluation_input.h5`
which will iterate through any files in the directory, and use the input file to the original execution of VOLNA to combine geometry information with the state variables stored in the output files to generate VTK files. The HDF5 files can be used by themselves as well, they contain a single dataset `/values`, which stores the four state variables (H, U, V, Zb) in this order, for each cell, the cells are ordered in the same way as in the input simulation file. Note however that the ordering (numbering) of cells is usually different from the input Gmsh file; to improve locality volna2hdf5 reorders them. To access the matching coordinates of the cell centers, you can open the input file and read the `/cellCenters` dataset, which contains the x-y coordinate pairs for each cell, and is ordered the same way, so for a given cell index idx, the coordinates are $(cellCenters[idx][0], cellCenters[idx][1]$ and the (H, U, V, Zb) values are $(values[idx][0], values[idx][1], values[idx][2], values[idx][3]$.

6 Performance considerations

1. Refrain from using OutputSimulation too often because (compared to the actual simulation) it may take a lot of time. Use HDF5 files for output. At the end of the simulation, the time taken for file I/O can be calculated by subtracting the total computation time from the total simulation time (which are reported in the output file).

2. When running on multiple GPUs, leave at least 100000 cells per GPU, below that you will not see much performance increase with more GPUs.

3. When moving large h5 files, consider compressing them first, e.g. h5repack -i gaussian.h5 -o gaussian_compressed.h5 -f GZIP=9

7 Examples

A self-contained example with a grid generated by VOLNA itself

```
#End simulation time is 5 seconds
Time { end = 5 }
#A rectangular domain between the points (0,180) and (-30,30) consisting of 2*360*120
triangles
Mesh Rectangle 0 180 -30 30 360 120
#Usual constants
PhysicalParams { g = 9.81 }
NumericalParams { cfl = .45 }
# beginning at the first timestep, upon every step, apply the following bathymetry
deformation Init { istart = 0 istep = 1 } Bathymetry { return .2*(-5-x)*(x<0)-(x>=0)+.2*(t<1)
exp(-(x+3-2*t)*(x+3-2*t)-y*y)+.2*(t>=1)*exp(-(x+1)*(x+1)-y*y);}
# Display iteration number and timestamp every 10 iterations on the console
OutputTime { istep = 10 } "cout"
# Save the state space (H,U,V,Zb) every 10 iterations to the following file (%i is
```

```
substituted for the current iteration number)
OutputSimulation { istep = 10 } "result_sim_lands/snapshot%i.vtk"
# Save the state space (H,U,V,Zb) at iteration 0 into max0000.vtk and then at the end
of the simulation the values when H reached its maximum in different cells
OutputMaxElevation { istep = -1 } "result_elev_lands/max%i.vtk"
```

An example that uses text files for bathymetries:

```
#End simulation time is 5 seconds, with a fixed timestep of 0.05 to match the generated
bathymetry files
Time { end = 5 dtmax = 0.05 }
#A mesh described in a GMSH file
Mesh "test.msh"
#Usual constants
PhysicalParams { g = 9.81 }
NumericalParams { cfl = .45 }
# beginning at the first timestep, upon every step, up until the 10th timestep apply
the bathymetry deformation specified in the following files:  Init { istart = 0 istep
= 1 iend 10} Bathymetry "bathy/bathy%i.txt"
# Save total water height (H+Zb) every timestep at a prescribed location, the filename
is unused
OutputLocation { istep=1 } " gauge_x0_y0.txt" { x = 0.00001 y = 0.0001 }
```

Given the above initial bathymetry specification, VOLNA expects 2 types of files in the `bathy` directory: `bathy_init.txt` an initial bathymetry value for all cells in the simulation mesh (one value each)

`bathy_geom.txt` which describes the coarse mesh which will define the bathymetry deformations, this file is structured as follows: first line number of cells and number of points, next #(number of cells) number of lines listing the indices of the points of each cell, then #(number of points) number of lines, each with 3 double precision values for the x and y and z coordinates of the points.

`bathy%i.txt` a sequence of files (in this case 10, bathy000.txt, bathy001.txt, ... bathy010.txt) each with the bathymetry deformation, relative to the original bathymetry defined in `bathy_init.txt`, for each vertex of the coarse mesh. VAlues are interpolated to all cells within any of the triangles defined in the coarse mesh.