Optimization of a Coupled Simulation with Delft3D-FLOW and SWAN for Informed Decision Making

J. Donners\textsuperscript{a*}, M. Guarrasi\textsuperscript{b}, A. Emerson\textsuperscript{b}, M. Genseberger\textsuperscript{c}

\textsuperscript{a}SURFsara, Amsterdam, The Netherlands
\textsuperscript{b}CINECA, Bologna, Italy
\textsuperscript{c}Deltares, Delft, The Netherlands

\section*{Abstract}

The applications Delft3D-FLOW and SWAN are used to simulate respectively water flow and water waves. These two applications have been coupled with Delft3D-WAVE and the combination of these three executables has been optimized on the Bull cluster “Cartesius”. The runtime could be decreased by a factor 4 with hardly any additional hardware. Over 80\% of the total runtime consists of unnecessary I/O operations for the coupling, of which 70\% could be removed. Both I/O optimizations and replacement with MPI were used. The Delft3D-FLOW application has also been ported to and benchmarked on the IBM Blue Gene/Q system “Fermi”.

\section*{Introduction}

Where scientific discovery is usually realised by isolating one effect and keeping all other factors constant, real-life applications often need to balance the effects of different factors to get as close as possible to the desired result. This approach requires the coupling of different models, where each model can adequately describe one factor. As computer applications are usually developed in isolation, the coupling of multiple applications requires a huge effort even to get the right functionality. As a result, performance of the coupling might not always be optimal.

Here, we analyse a coupled software application to simulate the different aspects of a lake, ranging from the optical properties of the water to safety concerns. The performance of the application is relevant to enable decision makers to make informed decisions within a reasonable timescale.

Results of this project will be presented at the World Lake Conference in Perugia, Italy, September 2014.

\section*{Applications}

\textbf{Delft3D}

The Delft3D FLOW user manual \cite{1} describes the DELFT3D software package as follows:

“Deltares has developed a unique, fully integrated computer software suite for a multi-disciplinary approach and 3D computations for coastal, river and estuarine areas. It can carry out simulations of flows, sediment transports, waves, water quality, morphological developments and ecology. It has been designed for experts and non-experts alike. The Delft3D suite is composed of several modules, grouped around a mutual interface, while being capable to interact with one another. Delft3D-FLOW is one of these modules.”

* Corresponding author. \textit{E-mail address:} john.donners@surfsara.nl
Delft3D-FLOW is a multi-dimensional (2D or 3D) hydrodynamic (and transport) simulation program which calculates non-steady flow and transport phenomena that result from tidal and meteorological forcing on a rectilinear or a curvilinear, boundary fitted grid. In 3D simulations, the vertical grid is defined following the sigma co-ordinate approach.

The Delft3D-FLOW module uses MPI decomposition with a structured grid and solves the flow equations using a finite difference approach. It uses an alternation direction implicit (ADI) method to solve the shallow water equations (elliptic equations for the free surface). More details and extensive profiling of Delft3D-FLOW can be found in a PRACE white paper [2].

The Delft3D-WAVE module is used to simulate the evolution of wind-generated waves in coastal waters [3]. It forms an interface between different, external wave models and the other Delft3D modules. The standard option as wave model is the SWAN model, which is described in more detail in the next section. Data are exchanged using a so-called communication file (com-file), which contains the most recent data of the flow and wave computations.

The Delft3D suite consists of several more modules for water quality, ecological modelling and sediment transport. The information exchange between modules is provided automatically by means of a so-called communication file; each module writes results required by another module to this communication file and reads from the file the information required from other modules. Other, module-specific, files contain results of a computation and are used for visualisation and animation of results [1].

SWAN

SWAN is a stand-alone application to simulate the evolution of random, short-crested wind-generated waves [4] in estuaries, tidal inlets, lakes, etc. The name SWAN stands for Simulating WAves Nearshore and it was developed at Delft University of Technology in the Netherlands. It has an independent development team apart from Delft3D. However, it can be coupled to the suite of Delft3D modules through the Delft3D-WAVE module.

SWAN is parallelised using either MPI or OpenMP, but not both at the same time in a hybrid approach. SWAN uses a wave-front approach, where it computes sweeps across the domain in a diagonal fashion. The first thread calculates the results for the corner point, after which the second thread can jump in and start calculation of the next row and the first thread starts the next column. The MPI parallelization splits the domain in a number of columns. The second MPI process can only start when it receives the information from the first MPI process that has calculated the last column of the first row. The scalability of the MPI approach is therefore somewhat less good than OpenMP.

Setup

Delft3D has been ported to and tested on the following systems:

- Bull x86-64 system “Cartesius” running Linux at SURFsara (Intel compilers + Intel MPI)
- IBM Blue Gene/Q system “Fermi” at CINECA (IBM compilers + IBM MPI)

Extensive porting efforts have been reported in the PRACE white paper about Delft3D [2]. The porting to the Cartesius system was straightforward with the latest development release of the Delft3D software stack. Porting to the Fermi system was a huge effort and will be reported in a separate section.

The SWAN application supports a large range of platforms and compilers and is therefore easy to port to the platforms mentioned here.

Porting Delft3D to Fermi

We have also ported the flow module on the FERMI [5][6] Blue Gene/Q [7] system at CINECA. As described in a previous PRACE white paper [2], the FLOW module of Delft3D [1] is a multi-dimensional (2D or 3D) hydrodynamic (and transport) simulation program which calculates non-steady flow and transport phenomena resulting from tidal and meteorological forcing on a curvilinear, boundary fitted grid or spherical coordinates. As a preliminary step we have first ported the code also on other two PRACE Tier-1 systems: the IBM IDataPlex DX360M3 PLX[8], and the Eurotech hybrid cluster EURORA [9]. This first phase was quite straightforward and similar to the porting on CURIE cluster [10] [2], but it is necessary to make a comparison with the results obtained on more standard architectures.
Configuration and setup

Since the code was created for small and medium size Windows machines, and later adapted for Linux x86 systems, the porting of the code on a massively parallel supercomputer such as the FERMI Blue Gene/Q cluster was particularly complicated. In fact, many of the compilation, link and run-time problems resulted from the inability of the default build procedure to select appropriate options for the Blue Gene/Q (PowerPC) installation. It was also necessary to make a number of changes to the source code itself: the IBM XL compilers in general adhere very strictly to the FORTRAN and C standards, unlike say the GNU or Intel suites which are more flexible, with the result that some non-standard code constructs needed to be modified.

Some of the more important changes are listed below – for a complete list please contact the authors:

• First of all, many of the files were written in DOS format. In general for FORTRAN and C source files this does not cause problems for the IBM compilers but some auxiliary files used in the build process (e.g. headers or include files) needed conversion with the dos2unix UNIX utility.

• The program uses automake, autoconf and libtool to create a configure script, which then configures the whole package for a particular system. In order to avoid all possible compatibility problems, we have installed on FERMI the last version of autotools currently available (libtool 2.4.2, autoconf 2.69, automake 1.13.2). We also needed to install the expat library in order to enable some I/O procedures of the code (in particular those used for reading XML files).

• The code was compiled and tested using both IBM XL and GNU compilers. In order to ensure a sufficient compatibility with some platform specific options it was necessary to write a more complex configure procedure.

• One of the major issues related to the compilation of Delft3D on a Blue Gene/Q system is that on these systems for performance reasons a static compilation is preferred over a dynamic one. However, for many scientific packages, Delft3D included, the default linking mode is dynamic and it is quite challenging to create a statically linked version. For this reason we modified our script to use a static linking procedure. This change was particularly complicated because the use of libtool creates a series of bugs in linking that we eventually resolved using the --all-static flag (only during the linking procedure).

• It was necessary to declare some new environment variables in order to prevent crashing of the compilation procedure (e.g. ac_cv_func_malloc_0_nonnull=yes ; ac_cv_func_realloc_0_nonnull=yes ;).

• It has been also necessary to remove all the C-style comments in the config.h files included in pre-processable FORTRAN files, since they are not recognized by IBM FORTRAN compilers.

• We also had to pay special attention to the compilation of the version_number.exe utility which is used during the package build to update the version information of the compiled executable. This is due to the fact that on Blue Gene/Q two different versions of both GNU and XL compilers are available: one for the front-end nodes and the other one for the back-end nodes. Although it is necessary to compile the code for the execution on the back-end nodes (the nodes on which the code will run), the version_number.exe executable must be compiled using the front-end tool chain.

• The code has also a series of compilation problems related to the presence of the DelftOnLine module. In order to avoid these problems the common.am file was modified removing all reference to DelftOnLine.

• Some modifications are necessary also on the configure.ac file in order to resolve a compilation problem related to pre-processing.

• The program includes a function (rdtsc) to provide some architecture-dependent information for Windows or x86 systems. Since the appropriate code for PowerPC was not present, we had to modify this function for the program to compile and work correctly on Fermi.

• It was discovered that the default version of DELFT3D can use only up to 1000 cores (or MPI tasks). This limit is due to the fact that temporary output files created by individual tasks have filenames related to the task number and the algorithm allows only 3 characters to represent this number. By modifying the files involved in this algorithm (iniid.f90, rdgrid.f90, strgrd.F90 inicut.f90 and inigrd.f90) to allow 4 characters for this field the task limit was increased to 10,000 cores.

• Another issue encountered during the porting the code on Blue Gene/Q systems was related to the non-temporary I/O files. Using default compilation options, the NEFIS library (the library that handles the non-XML I/O for Delft3D) on FERMI produces files which, according to the UNIX ls command, are larger than 4
exabytes (presumably due to corrupt i-node data). The problem was solved by adding the –DPTR8 flag to the compilation to enforce the use of 8 byte file pointers.

**Benchmarking test case: Sediment transport**

The code was compiled with both the XL and GNU toolchains using typical non-aggressive optimization options (i.e. –g –O2 compilation flags) and double precision to represent a real production environment. Initial tests using the test case described below (i.e. sediment transport) demonstrated nearly identical performances for both the XL and GNU compiled versions of the code and so for the benchmarks only the XL version was used. Moreover, attempts to use more aggressive optimizations for the XL compilation (e.g. at –O3 level) did not result in any significant performance benefits.

This model computes hydrodynamic evolution sediment transport and morphology updates. Due to these extra processes, different parts of the code are activated, which are highly compute-intensive. This model has 243 horizontal grid-points in each dimension (even if it has only one layer in the Z dimension). This is a first simple step to enable the use of the code on Blue Gene/Q systems. Since Delft3D uses 1D domain decomposition and each domain needs a minimum number of columns, the maximum numbers of processes that we use is only 128 cores.

In figure 2 we plot the total execution time for a 2000 time step integration. It can be seen that the benchmark does not scale well, even if the execution time decreases up to 128 cores. The overhead for performing sediment transport is significant and sensitive to the number of processes, since the solution of the ADI-solver depends on the process count which cascades into changes in sediment transport and resulting height changes that feed back to the circulation, even for short simulations like these. This affects the reliability of the results, although we do not know to what extent. The Scalasca output is given in figure 1. As we can see for 64 processes, the CPU time is dominated by two routines of sediment transport:

- **erosed** (compute sediment fluxes): 34.8%
- **bott3d** (update depth due to changes in bottom sediment): 39.9%

The routine **erosed** calls several functions for each grid-point with active sediment transport and depth changes (bedbc1993, calseddf1993, bedtr1993, comparealldouble, getsedthick_1point and more). These functions are called tens of billions of times, even for these short benchmark runs.

Other important routines are:

- **adi** (uzd + sud. Hydrodynamic alternate direction integration): 10.5%
- **tritra** (compute transports for conservative constituents): 5.8%
- **taubot** (compute new chezy values (including wave effects): 1.6%

It is also possible to evaluate the computational imbalance of these souboutines using the Scalasca profiling tool. Indeed, the underload heuristic of Scalasca identifies processes/threads where the exclusive execution time spent for a particular call-path was below the average value. Using this heuristic on our data we obtain:

- **erosed**: 35.4%
- **bott3d**: 17.1%
- **adi**: 6.1%
- **tritra**: 2.9%
- **taubot**: 1.6%

Where the load unbalance is expressed as a percentage of the inclusive time (execution+MPI) spent by all processes in that same routine.
As you can see the large part of the load imbalance is related to the sediment transport. If we repeat the same simulation without considering this part of the code, Delft3D scales up to its physical limit (128 cores for this example).

Probably using a larger array and disabling the sediment transport we will be able to reach a good scalability also using several thousands of cores. We are considering now to run a large scale hydrodynamic model of Lake Garda to verify this.

**Test case: Lake Marken**

A multidisciplinary coupled model of Lake Marken in the Netherlands has been developed to help the design process of a lake and its environment. Different stakeholders are involved to comment on a variety of aspects of the design, ranging from economical, engineering and safety concerns to recreational and ecological aspects.
Interactive sessions where measures can be drawn and their effects can be calculated almost instantly would be a welcome addition, but are not feasible yet. The runtime for a simulation of 1 year takes about 4 days.

Both Delft3D-FLOW and SWAN use a computational grid with 197 by 206 grid points in the horizontal direction. In the vertical direction Delft3D-FLOW use 7 boundary fitted layers. The simulation runs 214848 steps (each time step represents 2.5 minutes of simulated time) with Delft3D, interleaved with a SWAN run called after every 24 steps (corresponding to one SWAN run per simulated hour). This totals to 8952 calls to SWAN. The coupling is bi-directional: the flow in the lake computed by Delft3D-FLOW is input to SWAN and the significant wave heights computed by SWAN are input to Delft3D-FLOW. All the coupled runs have been performed on the Cartesius system.

**Benchmarking**

Initially, the coupled model was run with three separate executables in the following configuration:

1. Delft3D-FLOW is run with 6 MPI processes,
2. Delft3D-WAVE is run as a separate, serial process and
3. SWAN is run as an OpenMP executable with 24 threads (equal to the nr. of cores per node)

The coupled model is run for 648 time steps and the estimated remaining runtime is taken as an indicator for the performance. The initial estimated remaining runtime is 4 days and 18 hours. All tests are summarized in table 1.

<table>
<thead>
<tr>
<th>Test</th>
<th>Estimated remaining runtime</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0. Initial performance</td>
<td>4 days 18 hours</td>
<td></td>
</tr>
<tr>
<td>1. Delft3D-FLOW with 12 MPI processes (disc.)</td>
<td>4 days 8 hours</td>
<td>Crashes after 880 time steps</td>
</tr>
<tr>
<td>2. Use of ramdisk (/dev/shm) for I/O</td>
<td>3 days 2 hours</td>
<td></td>
</tr>
<tr>
<td>3. export FORT_BUFFERED=true</td>
<td>3 days 0 hours</td>
<td></td>
</tr>
<tr>
<td>4. export KMP_AFFINITY=compact,0,0</td>
<td>2 days 19 hours</td>
<td></td>
</tr>
<tr>
<td>5. SWAN with NetCDF for hotstart data (disc.)</td>
<td>2 days 16 hours</td>
<td></td>
</tr>
<tr>
<td>6. SWAN with MPI for hotstart data</td>
<td>2 days 3 hours</td>
<td></td>
</tr>
<tr>
<td>7. Delft3D-FLOW with 11 MPI processes</td>
<td>2 days 0 hours</td>
<td></td>
</tr>
<tr>
<td>8. SWAN with 32 threads on fat node</td>
<td>1 day 21 hours</td>
<td></td>
</tr>
</tbody>
</table>
9. WAVE-FLOW coupling through MPI

Table 1: Performance tests of different improvements. Each test includes the modifications from the previous tests, unless the modification has been discontinued (indicated by 'disc.').

<table>
<thead>
<tr>
<th>Test</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Using twice as many MPI processes for Delft3D-FLOW increases the performance, but destabilizes the model. The solution of the ADI method for the free surface depends on the number of MPI processes.</td>
</tr>
<tr>
<td>2</td>
<td>It is clear that a lot of time is spent in I/O. The coupled simulation has the following I/O-related stages:</td>
</tr>
<tr>
<td></td>
<td>- Delft3D-FLOW reads input</td>
</tr>
<tr>
<td></td>
<td>- Delft3D-FLOW writes output</td>
</tr>
<tr>
<td></td>
<td>- Delft3D-FLOW uses DelftIO to signal to Delft3D-WAVE that it can take over</td>
</tr>
<tr>
<td></td>
<td>- Delft3D-WAVE reads the Delft3D-FLOW output</td>
</tr>
<tr>
<td></td>
<td>- Delft3D-WAVE writes SWAN input</td>
</tr>
<tr>
<td></td>
<td>- SWAN reads input</td>
</tr>
<tr>
<td></td>
<td>- SWAN reads start dump ('hotstart files')</td>
</tr>
<tr>
<td></td>
<td>- SWAN writes output</td>
</tr>
<tr>
<td></td>
<td>- SWAN writes start dump</td>
</tr>
<tr>
<td></td>
<td>- Delft3D-WAVE reads SWAN output, converts it and</td>
</tr>
<tr>
<td></td>
<td>- Delft3D-WAVE writes Delft3D-FLOW input,</td>
</tr>
<tr>
<td></td>
<td>- Delft3D-WAVE uses DelftIO to signal to Delft3D-FLOW that it can continue.</td>
</tr>
<tr>
<td>3</td>
<td>Benchmarks show that most time in SWAN is spent in formatted reading and writing of the action density. A first optimization is therefore the use of a ramdisk, in this case /dev/shm, for the I/O. This approach effectively replaces disk I/O by memory copies and it therefore results in a 25% improvement over the original benchmark. Of course, the performance cost due to many I/O-related system calls is not lowered with this approach. Another optimization is the use of buffered I/O (test 3), which does lower the system overhead somewhat.</td>
</tr>
<tr>
<td>4</td>
<td>The SWAN model is run as an OpenMP application with 1 thread on each available core of the node. By default, threads can be moved around by the operating system, but this causes poor memory and cache performance. Threads need to be affinitized to maximize cache reuse and to keep threads as close as possible to their memory. Although the latest release 4.0 of the OpenMP standard provides mechanisms for thread placement, this was not yet available during these tests. Each compiler provides its own mechanism to define thread affinity. The Intel compiler provides this functionality through the KMP_AFFINITY environment variable. The setting KMP_AFFINITY=compact,0,0 places the threads as close as possible together. Hyperthreading does not provide a performance benefit for SWAN and is therefore disabled in all tests mentioned in this report.</td>
</tr>
</tbody>
</table>
| 5    | Tests have shown that the conversion from floating point to string and vice versa, that is required for the formatted I/O, is an expensive part in SWAN. Note that this is not related to the reading from or writing to disk, which is already optimized in previous tests using a ramdisk and buffering. Different options have been tested to reduce this conversion time. A recent development of SWAN introduced different options for the writing of hotstart files: using either the NetCDF file format or unformatted binary I/O. Both options have been tested: test 5 shows a small improvement when using NetCDF, while the unformatted binary I/O option didn't work correctly. A proof of concept was developed, where the largest part of the formatted I/O for the hotstart files was replaced with MPI, storing the action density inbetween 2 SWAN runs in a dummy array in Delft3D-WAVE. Test 6 shows that this results in a 25% performance increase, when compared to test 4. Note that not all of the hotstart file was converted to MPI communication and some conversion-related time is still required. Due to the use of MPI, the Delft3D-WAVE binary keeps waiting in a busy-wait loop while SWAN is computing, which
wastes resources in the default configuration. For most runs we use only 1 node and the simulation therefore runs with \texttt{I\_MPI\_WAIT\_MODE=1}, to give the CPUs to SWAN.

Further gains were obtained by scaling up the different components of the coupled model. Although it has been shown before that the use of 12 MPI tasks causes an unstable model, it is possible to run the Delft3D-FLOW model using 11 MPI tasks (test 7). The use of 32 threads on a fat node (4 sockets, 8-core SandyBridge CPUs) instead of 24 threads (2 sockets, 12-core IvyBridge CPUs) reduces the runtime again by a few hours (test 8).

The last development that was successfully implemented is the replacement of the DelftIO library with MPI. The DelftIO library uses files to signal between Delft3D-FLOW and Delft3D-WAVE in the current setup. Although a feature has been implemented in DelftIO to use shared memory segments for this signalling, the coupled model would hang when this was tried: strace shows that both processes, Delft3D-WAVE and the first MPI rank of Delft3D-FLOW, try to set and unset the same semaphore. The DelftIO calls were replaced with 2 calls to \texttt{MPI\_Send/MPI\_Recv} and an \texttt{MPI\_Barrier}. Furthermore, whereas in the original simulation the Delft3D-FLOW and Delft3D-WAVE binaries were started independently, the modified simulation only starts Delft3D-WAVE. The coupler Delft3D-WAVE starts both Delft3D-FLOW and SWAN using the \texttt{MPI\_Comm\_spawn} call. This modification increases the performance with another 30%.

The optimization of the I/O has resulted in more than 69% performance improvement, without increasing the number of cores used for the simulation. Another 5% performance improvement was reached with scaling up the application to use as many cores as possible on one node of Cartesius.

Several other options could further improve the performance of the coupled simulation:

- Some time is also spent in SWAN pre- and postprocessing, e.g. the obstacles (routine \texttt{swobsto}). This can be multi-threaded or skipped after the first time, since these obstacles are probably static. The total time spent per call in pre- and postprocessing for SWAN is 2.7s, while the computation takes on average 4.9s.
- Use of SWAN with MPI on multiple nodes could give a gain of a factor of 2 for SWAN on multiple nodes, estimated from timings per iteration. Unfortunately, the coupling of SWAN with MPI and Delft3D-WAVE is not yet correctly implemented.
- Convert SWAN into a library that can be linked and called from any application. SWAN can then still be distributed as an executable, but it can also be linked to an executable for coupling purposes. That executable replaces the I/O with MPI calls and can run the whole period instead of being restarted hourly.

**Interactive design session of a lake environment**

An interactive design session was held in the Collaboratorium at SURFsara on 4 November 2013. The Collaboratorium is a visualization and presentation space for science and industry. The facility is of great use for researchers that are faced with exponential growth and complexity of data from observations, experiments and simulations. The comfortable room can accommodate up to seven people with connections for video, audio, network and power. One side of the room consists of a video wall to to display several high-resolution images, animations and notes. The session can be enhanced with video conferencing, touch interaction and 3D projection.
The goal of the interactive design session was to assess a.o. benefits of the approach, expectations by users at forehand and which aspects should be improved. For the session we used a practical case of Lake Loosdrecht. Lake Loosdrecht is a smaller lake than Lake Marken but it has some similar aspects which are of importance for the approach. We used a simplified model based on fetch length with wall-clock times of maximal 4 minutes. Several participants (from landscape design to local authorities) that are involved in current combined environmental and societal projects on Lake Marken joined the session. Although the model used for the session is much more simplified than the model of Lake Marken, to our surprise, the participants indicated that it really was of benefit for a better understanding of the physical and biological processes and to have a better dialogue and more efficient design process. Based on this main outcome of the session, we concluded that the more simplified fetch length method is of use for interactive design sessions to have a dialogue with stakeholders about possible measures. More details of these measures should than be studied later on with the more detailed model of Lake Marken.

Conclusions

The coupled Delft3D-SWAN simulation was benchmarked and optimized. The original goal of the project was to reduce the runtime of the simulation from 4 days to about 1 hour to enable the use of the model in an interactive design session. Over 80% of the runtime is spent in unnecessary I/O that is used to couple the different applications, of which 70% has been eliminated. The runtime has been reduced to about 1 day and 6 hours and it is expected that the runtime can be reduced to about 12 hours with further optimization of the I/O and scaling of SWAN. However, this requires more involvement from the main developers to explain the intricacies of the code and to adopt changes in the development of the software. Due to the low scalability of all profiled applications, the use of a PRACE Tier-0 system is not effective. As a result, we must conclude that it is not possible to use this coupled model as a tool in an interactive design session that cannot last more than 1 working day. However, the interactive design session with a more simplified model, taught us that the simplified model is of value for the design process. We think we can further improve this by making the transition for modelling with the simplified model and the more detailed model more flexible. This will be subject of further research. In addition, after a significant porting effort, the Delft3D-FLOW application has been benchmarked on the IBM Blue Gene/Q architecture – to our knowledge the first time that this has been achieved. Given the
current limited parallel scalability of the code with MPI this porting is only really relevant for models which are large enough to utilize at least 1024 cores but with further improvements in the OpenMP parallelisation it may be possible to extend the utility of the application to smaller datasets.

References
[1] Delft3D-FLOW user manual