Science and Technology Facilities Council

Technical Report
DL-TR-2020-001

Porting and Optimising TELEMAC-MASCARET for OpenPOWER

Judicaël Grasset, Stephen M. Longshaw, Charles Moulinec, and David R. Emerson

January 2020

Enquiries concerning this report should be addressed to:

Chadwick Library
STFC Daresbury Laboratory
Sci-Tech Daresbury
Keckwick Lane
Warrington
WA4 4AD

Tel: +44(0)1925 603397
Fax: +44(0)1925 603779
email: librarydl@stfc.ac.uk

Science and Technology Facilities Council reports are available online at:
https://epubs.stfc.ac.uk

ISSN 1362-0207

Innovation Return on Research (IROR) Technical Report - EDF (Work Package 4): Porting and Optimising TELEMAC-MASCARET for OpenPOWER

Judicaël Grasset[1], Stephen M. Longshaw[1], Charles Moulinec[1], and David R. Emerson[1]

[1]UKRI-STFC Daresbury Laboratory, Scientific Computing Department, Computational Engineering & Environment Group

January 24, 2020

1 TELEMAC-MASCARET Overview

TELEMAC-MASCARET is an open-source suite of hydrodynamic solvers for free-surface flow modelling, originally developed by EDF R&D in the 1990s[1]. Development is now pursued through the TELEMAC-MASCARET Consortium. The software can be used to simulate 2-D or 3-D flows, sediment transport, water quality, wave propagation in coastal areas and rivers. More details on the possible applications and other capabilities can be found on the software's website [2]. It is primarily written in Fortran 2003 (with a few parts in C) and currently parallelised using MPI.

2 Porting TELEMAC-MASCARET to POWER

The first part of this work was to make TELEMAC-MASCARET work on the IBM POWER platform. To the author's knowledge this has not been achieved before. For this port we considered three different compilers, firstly GCC because it is the best known and most widely used, next the IBM XL compiler as it is the native choice for the POWER architecture and finally the community version of the PGI compiler as the most likely to support GPU offloading fully.

Upon initial inspection it was determined that with GCC 8.2 (and previous versions), while the codebase compiled without error, it failed during execution on the POWER8 platforms provided by the Hartree Centre (Panther and Paragon). After deep examination it was found that this was due to a bug in the compiler itself. This was reported [3] and the fix incorporated into GCC at version 9.1. Using this version of GCC (and above), TELEMAC-MASCARET now compiles and executes correctly on Power platforms.

With IBM's XL compiler, a number of minor bugs were found both in TELEMAC-MASCARET and the compiler itself. These have all been reported upstream to both sets of developers and fixed. It is therefore now possible to run TELEMAC-MASCARET on Power platforms as of version v8p1r0 and using XL compiler version 16.1.1.1.

Using PGI 19.10, there is one specific bug in the compiler itself which prevents the code from compiling. As the code is old, EDF is always trying to improve it, in this case one number used to define the return value of a Fortran function has been replaced by a proper variable in a module, but the PGI 19.10 compiler is unable to compile it. A small reproducible test case has been created and sent to the PGI group [4] but

the problem is not fixed at the time of writing this report. In the meantime, a workaround has been created to allow compilation and execution, this has been included in the latest version of TELEMAC-MASCARET.

Following porting TELEMAC-MASCARET to the three named compilers, it was submitted to the OpenPOWER foundation for official OpenPOWER certification, which was granted in June 2019. The software is now featured on the official website of the OpenPOWER foundation as a direct result of this work [5].

	Minimum Compiler Version Required		
	GCC	PGI	XL
TELEMAC V8P1R0	9.1	Not working	16.1.1.1

Table 1: Summary of the requirements to compile TELEMAC-MASCARET on POWER

3 Experiments with GPUs

Many modern HPC clusters and particularly clusters based on the POWER architecture have GPUs integrated into their nodes, for example the POWER8 cluster primarily used in this work (Paragon) contains 4 NVIDIA P100 GPUs per node. Certainly the expectation is that future exascale systems will incorporate even more of this type of design. Like many legacy codes, TELEMAC-MASCARET has no current capability to exploit this. As a result, in the next part of the work detailed in this report we have looked to analyse whether there are parts of the code that can benefit from GPU acceleration.

During the Telemac User Conference of 2018, it was highlighted that the specific subroutine *qnlin3* is particularly slow and problems that use TELEMAC-MASCARET and make use of this part of the code suffer because of this in terms of computational time to solution. Following analysis we were able to determine that *qnlin3* consists entirely of computation and memory accesses, this particular function has no reliance on MPI communications at all. Which makes it a good fit for GPU offloading.

Pragma based GPU acceleration using OpenACC and OpenMP were chosen for this work rather than re-writing the *qnlin3* function completely using CUDA. A goal of this work was to make it likely that it would be acceptable for inclusion in future releases, by simply *decorating* the existing function with pragmas, rather than writing a separate CUDA kernel, this improves code portability as well as maintainability.

For the experiments presented, the publicly available test-case used is *tomawac/fetch_limited/tom_test6* which is the only one delivered with TELEMAC-MASCARET that directly uses the *qnlin3* function. Figure 1 shows the results of the GPU offloading with PGI. Specific details on how it has been achieved and more benchmarks can be found in [6, 7].

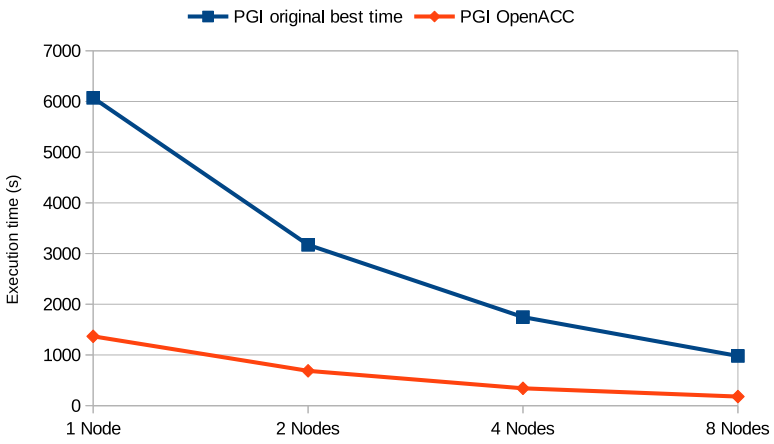

Figure 1: Comparison of the execution time of the original code and the OpenACC version when compiled with PGI and run on POWER8

4 EDF Supplied Test Case

As part of this work, EDF provided a specific test-case, henceforth EDF_TC. This was chosen as it has been used in previous studies [8] and has particular strategic importance. The domain of EDF_TC goes 60 km offshore and along the beach of *La Baie de la Somme*, in the north of France. Some areas of interest are the *Penly* and *Paluel* nuclear plants and the *Fécamp* offshore wind farm, all of which are encapsulated in this case.

EDF_TC uses a number of different modules of the TELEMAC-MASCARET suite, namely: Telemac2d, Tomawac and Sisyphe. And provides a mesh comprised of 41 084 elements.

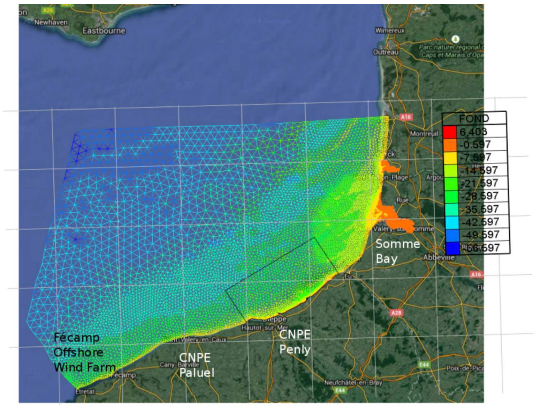

Figure 2: Superposition of the mesh and the map of *La Baie de la Somme* and surroundings. From [8]

4.1 Profiling of the EDF Test Case

As the tools required for this profiling task were available on the UKs national supercomputer, ARCHER, it was used here [9]. It was undertaken using the Arm MAP profiler [10]. This work has allowed performance bottlenecks and hot spots in the code to be found quickly.

When using multiple nodes to calculate EDF_TC (see Table 2), time spent in direct computation quickly decreases and is replaced by MPI overhead. When running on 4 nodes approximately 50% of the overall wall-clock time is spent in communications, which rises to approximately 80% when running on 8 nodes. The majority of the MPI overhead comes from the *parcom* and *post_interp* subroutines that are called in the *propa* subroutine of the Tomawac module.

# nodes	#1 subroutine (%)	#2 subroutine (%)	#3 subroutine (%)	CPU(%)	MPI(%)
1	qbrek (22)	qnlin1 (17)	mpi_allreduce (12)	83	16
2	mpi_allreduce (20)	qbrek (19)	qnlin1 (8)	66	33
4	mpi_allreduce (38)	qbrek (9)	mpi_alltoall (7)	42	56
8	mpi_allreduce (45)	mpi_alltoall (13)	mpi_alltoallv (10)	20	79

Table 2: Top three time-consuming routines for EDF_TC

The main part of TELEMAC-MASCARET used by EDF_TC is the *qnlin1* subroutine and not the more computationally intensive *qnlin3* (which has been ported to GPU as part of this work). As *qnlin1* is less computationally intensive than *qnlin3* but gets called more frequently, the benefit of porting this to GPU is not clear due to the significant amount of data transfer from host to GPU that would need to happen when compared to the benefits of the function being parallelised for GPUs. Furthermore, the computational overhead of this subroutine is overshadowed by the cost of the MPI communications when running on more than 2 nodes.

4.2 Protocol

Please note that further work in this section was completed using UKRI-STFC Hartree Centre's Paragon POWER8 cluster.

It has been noted that when running on 1 or 2 nodes, the execution time is regular (about 1% difference between runs). But when running on 4 nodes or more the difference increases significantly (up to 36% in our observations). In order to reduce irregularities between timing data, we ran the same simulation a number of times and present the lowest execution times.

The compilers used for these tests are: GCC 9.1 and PGI 19.10. The MPI library used is Spectrum MPI 10.3.0.

Num. of nodes	1	2	4	8	16
Execution time (s) (PGI)	712	401	284	212	205
Execution time (s) (GCC)	782	432	300	220	209

Table 3: Execution time of TELEMAC-MASCARET using EDF_TC

4.3 Fixing *qbrek*

As seen in table 2, on 1 node the main time consuming subroutine is *qbrek*. This is due to it containing a loop written in non-column order, which then prevents coherent use of the CPU cache. To fix this problem the loop was re-written.

Table 4 shows the results of this modification. Between 1 and 8 nodes there is a performance increase which starts to disappear at 16 nodes because the amount of data per MPI rank is too small due to the inherently small mesh associated with EDF_TC. This means the data per rank fits entirely in cache, therefore making little difference if the loop is written in a cache-friendly order or not. It is expected however that this work will benefit problems that use larger mesh sizes in the future.

Num. of nodes	1	2	4	8	16
Execution time (s) (PGI)	688	386	271	205	211
Execution time (s) (GCC)	742	405	281	212	207
Speedup (PGI)	1.03	1.04	1.05	1.03	0.97
Speedup (GCC)	1.05	1.07	1.07	1.04	1.01

Table 4: Execution time with *qbrek* patch applied

4.4 Reducing MPI communication

This section describes the structure of the post_interp subroutine, including where its bottlenecks are and how these were mitigated.

4.4.1 Description of post_interp

Listing 1 describes the main structure of the post_interp subroutine, the most time consuming user of MPI related overhead for EDF_TC. It can be seen that MPI calls are numerous and inside a loop, furthermore most of the calls are global communications (MPI_All*), which means that each processors used for the simulation will send or receive data for that MPI call. One of the problems with synchronous global communications is that they need all of the processors to do the communication at the same time. It also generates a lot of network traffic (when running on more than one node).

Listing 1: Simplified post_interp subroutine

```
do if=1,nfrequency
    ! 1/ Check if there are data to interpolate with other ranks
    call MPI_Allreduce()
    ! 2/ Exchange data for computing interpolation
    call MPI_Alltoallv()
    ! 3/ Check for errors
    call MPI_Allreduce()
    ! 4/ Check for errors
    call MPI_Allreduce()
    ! 5/ Send interpolated values
    call MPI_Alltoallv()
    ! 6/ Unknown
    do j=1,n
      call MPI_Irecv()
    enddo
    do j=1,n
      call MPI_Isend()
    enddo
enddo
```

4.4.2 MPI Patch: 1

It has been noticed that the values used in step 1 of the *post_interp* subroutine are modified every 60 iterations of Telemac2d, therefore, instead of exchanging data via MPI at every time-step, these values can be saved locally and only updated when necessary. The iteration frequency of 60 comes from the parameter *"Coupling period for Tomawac"* in the EDF_TC input files.

Also noted is that steps 3 and 4 are used for error checking but use an expensive method to achieve this. Each processor sets its error value to one if there is a problem and then an MPI allreduce is used to sum the error value across all MPI ranks. If the final total is greater than 0 it means that one processor had an error and the program should exit. A less expensive approach is proposed where each processor first checks if it's error value is non-zero, and if it is then it calls *mpi_abort*, causing all other ranks to also exit. This adds a minor change to the *post_interp* subroutine but allows a nominal reduction in MPI communication overhead.

Num. of nodes	1	2	4	8	16
Execution time (s) (PGI)	666	357	235	174	151
Execution time (s) (GCC)	697	378	247	179	154
Speedup (PGI)	1.07	1.12	1.21	1.22	1.36
Speedup (GCC)	1.12	1.14	1.21	1.23	1.36

Table 5: Execution time with *qbrek* patch and *MPI patch: 1* applied

4.4.3 MPI Patch: 2

This second patch to the *post_interp* subroutine incorporates the modifications of *MPI Patch: 1* and also removes the need for step 2 of the subroutine at every iteration. The data exchanged in the first *MPI_Alltoallv* is only updated every 60 time-steps, it is less costly to save them locally and only update them when necessary.

Step 5 is also simplified as part of this patch. The second *MPI_alltoallv* in the algorithm transfers a significant amount of data but very little of this is actually used, MPI communication was therefore modified to only send the relevant data. This optimisation is specific to this test case at the moment and so needs generalising to allow similar optimisations to be applied to all problem types. This patch adds significant modification to the *post_interp* subroutine.

Num. of nodes	1	2	4	8	16
Execution time (s) (PGI)	609	338	220	162	176
Execution time (s) (GCC)	664	355	227	168	140
Speedup (PGI)	1.17	1.19	1.29	1.31	1.16
Speedup (GCC)	1.18	1.22	1.32	1.31	1.49

Table 6: Execution time with qbrek patch and the MPI patch 2

4.4.4 MPI Patch: 3

This patch uses the modifications of patches 1 and 2 and also modifies the way step 5 in the *post_interp* subroutine performs its communication. Instead of using an *MPI_alltoallv*, which requires all MPI ranks to communicate at the same time, even when they do not have any data to share, the MPI sparse communication routine *MPI_Neighbor_alltoallv* is used in its place.

Neighbourhood communication is shown to be beneficial when using more than 2 nodes, however using between 1 and 2 nodes execution times are longer than using the original *MPI_alltoallv* method. This is likely because this configuration had less potential for improvements when using this communication pattern due ot the low number of peers and also because setting up this new communication type has an overhead cost.

This patch adds significant modifications to the code and increases its complexity, as MPI neighbourhood communicators are not widely used and therefore not widely understood. It also require an MPI library which implements the MPI 3.0 standard, while the original TELEMAC-MASCARET is able to work with previous MPI standards.

Num. of nodes	1	2	4	8	16
Execution time (s) (PGI)	635	347	218	157	115
Execution time (s) (GCC)	724	365	223	161	117
Speedup (PGI)	1.12	1.16	1.30	1.35	1.78
Speedup (GCC)	1.08	1.18	1.35	1.37	1.79

Table 7: Execution time with qbrek patch and the MPI patch 3

4.4.5 Scalability

Although TELEMAC-MASCARET is still far from achieving ideal scaling, Figure 3 shows that it has been improved noticeably for this test-case through the application of these three patches to the *post_interp* subroutine.

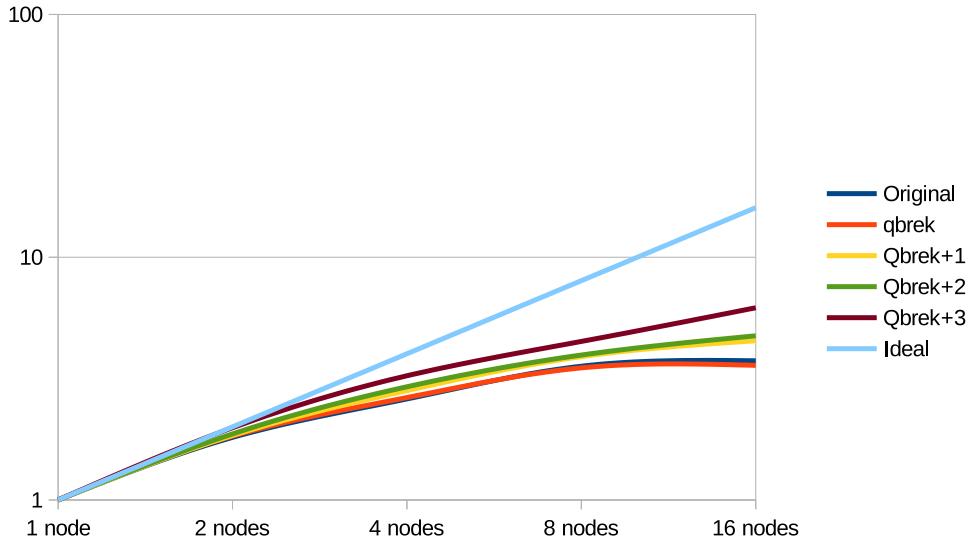

Figure 3: Scalability of the different versions of the code when compiled with GCC on POWER8

5 Acknowledgements

This work was supported by the UKRI-STFC Hartree Centre's Innovation Return on Research (IROR) programme and this work also used the ARCHER UK national Supercomputing Service.

References

[1] Galland, Jean-Charles, Goutal, Nicole, Hervouet, Jean-Michel, TELEMAC: A new numerical model for solving shallow water equations. Advances in Water Resources, 1991, vol. 14, no 3, p. 138-148.

[2] TELEMAC-MASCARET, www.opentelemac.org (accessed 2019-01-24)

[3] GCC bug report, gcc.gnu.org/bugzilla/show_bug.cgi?id=87689 (accessed 2019-01-24)

[4] PGI bug report, www.pgroup.com/userforum/viewtopic.php?f=4&t=6429 (accessed 2019-01-23)

[5] TELEMAC-MSCARET V8 OpenPOWER Certification, www.openpowerfoundation.org/?resource_lib=stfc-daresbury-laboratory-telemac-mascaret-v8 (accessed 2019-01-24)

[6] Grasset, Judicaël, Audouin, Yoann, Longshaw, Stephen M., Moulinec, Charles, Emerson, David R., *Porting and Optimising TELEMAC-MASCARET for the OpenPOWER Ecosystem*, in Proceedings of the 2019 Emerging Technology Conference, Editors: M.K. Bane and V. Holmes, ISBN 978-0-9933426-4-6, 9th-10th April 2019, Huddersfield, UK.

[7] Grasset, Judicaël, Longshaw, Stephen M., Moulinec, Charles, Emerson, David R., Audouin, Yoann, Tassi, Pablo, *Porting TELEMAC-MASCARET to OpenPOWER and experimenting GPU offloading to accelerate the TOMAWAC module*, in Proceedings of the 2019 Telemac User Conference, 15th-17th October 2019, Toulouse, France.

[8] Jia, Meng, Tassi, Pablo, Huybrechts, Nicolas, *Numerical study of the influence of waves and tidal currents on the sediment dynamics in the vicinity of the Somme Bay area (France)*, in Proceedings of the 2015 Telemac User Conference, 15th-16th October 2019, Daresbury, United Kingdom.

[9] UK National supercomputer ARCHER, http://archer.ac.uk (accessed 2019-01-24)

[10] Arm MAP profiler, www.arm.com/products/development-tools/server-and-hpc/forge/map (accessed 2019-01-24)