H2020-INFRAEDI-2018-2020

The European Centre of Excellence for Engineering Applications

Project Number: 823691

D3.2
Report on Exa-enabling enhancements and benchmarks

The EXCELLERAT project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 823691

Workpackage:	WP3	Driving Exa-HPC Methodologies and Technologies
Author(s):	Daniel Mira	BSC
	Ricard Borrell	BSC
	Ivan Spisso	CINECA
	Gabriel Staffelbach	CERFACS
	Thomas Gerhold	DLR
	Niclas Jansson	KTH
	Janik Schussler	SSC
	Gavin Pringle	EPCC
	Nicholas Brown	EPCC
Approved by	Executive Centre Management	
Reviewer	Andreas Ruopp	HRLS
Reviewer	Claudio Arlandi	CINECA
Dissemination Level	Public	

Date	Author	Comments	Version	Status
29/10/2020	Daniel Mira	First full version	V0.0	
30/10/2020	Ivan Spisso	complete section 5, check formatting	V0.1	
30/10/2020	Ricard Borrell	First full revision	V0.2	
30/10/2020	Daniel Mira	Deliver to internal review within consortium	V1	
10/11/2020	Claudio Arlandi	Full revision	V2.1	
12/11/2020	Andreas Ruopp	Full revision	V2.2	
24/11/2020	Daniel Mira	Corrected version	V3	
30/11/2020	Daniel Mira	Final updates	V4	

List of abbreviations

AMR	*Adaptive Mesh Refinement*
CFD	*Computational Fluid Dynamics*
CLI	*Command Line Interface*
CPU	*Central Processing Unit*
CRM	*NASA Common Research Model*
CoE	*Center of Excellence*
CUDA	*Compute Unified Device Architecture*
DLB	*Dynamic Load Balancing*
DMA	*Direct Memory Access*
FPGA	*Field-Programmable Gate Array*
GASPI	*Global Address Space Programming Interface*
GMRES	*Generalized Minimal Residual*
GPU	*Graphics Processing Unit*
HPC	*High-Performance Computing*
HLS	*High Level Synthesis*
HDL	*Hardware Description Language*
HBM	*High Bandwidth Memory*
MPI	*Message Passing Interface*
PARRSB	*Parallel graph partitioning using recursive spectral bisection (RSB)*
PE	*Parallel efficiency*
PETSc	*Portable, Extensible Toolkit for Scientific Computation*
PRACE	*Partnership for Advanced Computing in Europe*
RCB	*Recursive Coordinate Bisection*
RCT	*Reduced computation time*
SFC	*Space Filling Curve*
SEM	*Spectral Element Method*
Spliss	*Sparse Linear Systems Solver*
TDP	*Thermal Design Power*
UDF	*User Defined Function*
UPC	*Unified Parallel C*
VE	*Vector Engines*
VPN	*Virtual Private Network*
ISA	*Instruction Set Architecture*

Executive Summary

The progress on the development of Exascale enabling technologies on the EXCELLERAT core codes is presented for the second year of the project. The developments have been driven by the definition of an individual code development roadmap in collaboration with WP2 and WP4 to demonstrate Exascale simulations for the use-cases.

From this roadmap, several requirements were identified (see D2.1, and D2.2 on" Reference_Applications_Roadmap and Challenges") and a summary of the activities conducted to address these requirements is presented here. Two fundamental activities are associated with these developments: i) Task 3.1 focused on node-level performance and ii) Task 3.2 on system-level performance engineering. Note that main changes in the evolution of HPC systems are occurring at node level. This is a major reason to have a specific task focused on this topic.

In this second year, the activities carried out by the partners on these tasks have focused on the development of the application demonstrators of the use-cases. At node level (Task 3.1), the partners' developments have focused on porting the codes to GPUs and the new vectorial architecture SX-Aurora from NEC. The memory features of the modern AMD Epyc 2 have also been investigated. Finally, aspects related to intra-node parallelization, such as load balancing and OpenMP threading optimizations, have been considered. At the system level (Task 3.2), the focus has been on strong scaling analyses and on the optimization of the communication kernels. Regarding the advanced meshing techniques (Task 3.3), most of the work has been performed on the core code Alya, where the AMR workflow has been completed, and on the code AVBP, where an in-house implementation using the TREE PART domain decomposition library has been performed. For AVBP, different error estimators for AMR have been tested. Regarding emerging technologies, the focus has been placed on FPGAs and advanced developments for GPUs and ARM-based architectures.

The advances in HPC algorithms and computational methodologies presented here are part of the expertise of the EXCELLERAT consortium and conform services that EXCELLERAT is delivering to the engineering community.

Table of Contents

Table of Figures

Table of Tables

1 Introduction

The present document is a summary of the progress of Exa-enabling enhancements and benchmarks performed on the EXCELLERAT core codes Alya and AVBP in year two of the project. The report is divided into different sections that refer to the different tasks of the EXCELLERAT WP3. This deliverable is made from the different contributions of the partners, which have been compiled and linked to the requirements of the use-cases defined in WP2.

2 Node-level performance optimization - Task 3.1

Important changes in the evolution of HPC systems are occurring at node level. Consequently, the complexity associated with unlocking the intra-node performance of computing systems has increased substantially. This task addresses all the aspects related to performance at node level, including code porting and algorithm refactoring on various architectures. Subsequently, the activities carried out in T3.1 for the second year of the EXCELLERAT project are presented.

2.1 Nek5000

2.1.1 Improvement of the small matrix-matrix multiplication kernel for Nek5000 on a single GPU

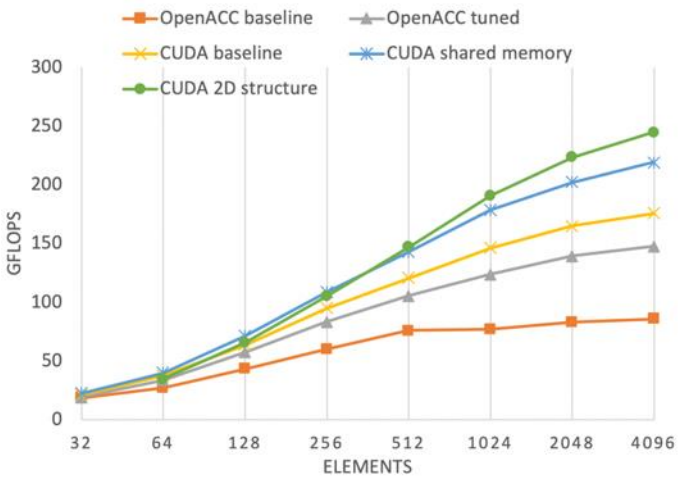


Figure 1: Performance results for Nekbone on a single GPU using 9th-order polynomials.

In the second year, the work focused on further optimizing the GPU performance of Nekbone. The work is based on the initial porting efforts reported in deliverable D3.1 "Report Exa-enabling enhancements and benchmarks", where tuned OpenACC and CUDA kernels were presented. The tuned CUDA kernels utilized a three-dimensional structure in shared memory, where one thread was allocated for each nodal point in the element. Thus, this approach was limited by the amount of shared memory available in a device and could not be used for large polynomial orders. Therefore, our main new contribution has been to utilize a new, two-dimensional thread structure, assigning threads in slices of an element, and progressing each slice in lockstep with each other. In Fig .1, how this new 2D structure improves performance with up to 10% compared to the previous shared memory implementation is shown.

2.1.2 Porting Nek5000 to SX-Aurora

In the course of the second-year porting and tuning of the entire spectral element code Nek5000 to the SX-Aurora TSUBASA, the recent vector computer from NEC, has started. Therefore, a similar approach as for the porting work to GPUs has been followed. The mini-app Nekbone

that was used for the initial porting and tuning of key kernels has been used. Particularly for node-level optimization the CEED Bake-off has been used for the Kernel BK5 [1] as a model problem to find optimal loop reordering for TSUBASA. Similar to a GPU, TSUBASA needs a large amount of work per element in order to achieve good performance. The initial OpenACC port of Nekbone [2] faced similar problems with the CPU version of the code. Therefore, several function calls were merged into one large loop nest per element, as illustrated in Fig .2. Albeit this formulation performed well on a GPU, the loop nest was too large, and suffered from too different access patterns for all three variables for the compiler to generate efficient vector code.

```
do k = 1, n
  do i = 1, n
    do j = 1, n
      do e = 1, nelt
        ws = 0d0
        !NEC$ unroll_completely
        do l = 1, n
          ws = ws + dxm1(j,l)*u(i,l,k,e)
        end do
        us(i,j,k,e) = ws
      end do
    end do
  end do
end do
```

Figure 2: An illustration of the merged loop nests in the OpenACC version.

In order to increase vectorization, the loop nest was split into three different loops, one for each variable, as shown in Fig .3, allowing the compiler to vectorize each loop.

```
do e = 1, nelt
  do k,j,i = 1, n ! tripel loop nest
    ur = 0
    us = 0
    ut = 0
    do l = 1, n
      ur = ur + dxm1(i,l)*u(l,j,k,e)
      us = us + dxm1(j,l)*u(i,l,k,e)
      ut = ut + dxm1(k,l)*u(i,j,l,e)
    end do
    wr(i,j,k,e) = g(i,j,k,1,e)*ur
                + g(i,j,k,2,e)*us
                + g(i,j,k,3,e)*ut
    ws(i,j,k,e) = g(i,j,k,2,e)*ur
                + g(i,j,k,4,e)*us
                + g(i,j,k,5,e)*ut
    wt(i,j,k,e) = g(i,j,k,3,e)*ur
                + g(i,j,k,5,e)*us
                + g(i,j,k,6,e)*ut
  end do ! end tripel loop nest k,j,i
end do
```

Figure 3: An example of the new vectorizable loop constructs for one of the variables.

For evaluation purposes, the BK5 was executed on a single Aurora node for a set of 128-8192 elements for 9^{th}-order polynomials. In Fig .4, the total GFLOPS/s when running the BK5 benchmark on one to eight cores is presented. The new transformations achieved more than 40% of the theoretical peak performance of a single core and almost 20% of the theoretical peak performance of a TSUBASA when using all eight cores.

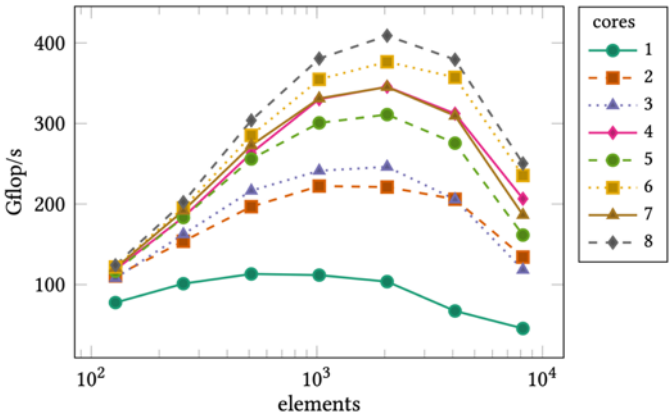


Figure 4: Performance of BK5 on a single SX-Aurora node for various numbers of elements and cores.

2.1.3 Preconditioner of the Nek5000 pressure calculation on GPUs

After improving small matrix-matrix multiplication kernels, the focus was set on increasing the performance of the pressure calculation as this is one of the most time-consuming simulation phases of the OpenACC implementation of the Nek5000. It adapts a relatively complex algorithm as this linear sub-problem is closely related to a divergence-free constraint and is usually very ill-conditioned. For this reason, it requires special preconditioning techniques that fit spectral element methods (SEM).

Nek5000 uses two possible preconditioning strategies taking advantage of existing domain decomposition: additive overlapping Schwarz and additive Schwarz-multigrid methods. Both methods split the preconditioner operator in two summed parts: the local and global solvers.

Additive overlapping Schwarz acts within each spectral element and uses a fast diagonalization method to reduce short-wavelength errors. Although information is exchanged between boundary elements during the local solve, the error propagation is rather slow and an additional operator is required to reduce long-wavelength errors. This is achieved by the global solver acting on a reduced number of degrees of freedom covering the whole computational domain. The main difference between simple Schwarz and Schwarz-multigrid methods is the local solver. While in the former case a single solve is performed on all elements, in the latter case a geometrical multigrid step reducing gradually the number of degrees of freedom is executed.

At present the simple Schwarz preconditioner is analyzed to identify possible modifications for improvements. In the first step the iterative solver (which uses the GMRES algorithm) was optimized by merging OpenACC kernels, e.g. for the *vlsc2* routine several kernels have been combined. Fig .5 shows that this leads to a reduction in total time for the single kernel compared to the time required by the many kernels that the new kernel replaces.

Figure 5: Trace for the GMRES implementation without merging of OpenACC kennels (top) with merging (bottom).

Subsequently, the preconditioner itself was adapted. The local solve executes mostly matrix-matrix multiplication and is performed on the GPU, whereas the global solve is performed on the CPU as it acts on the element vertices only and is quite communication intensive. Overlapping these two operations could provide a noticeable time reduction.

Unfortunately, both the local and global solve currently use the *gslib* communication library, which is not thread safe. This makes it impossible to overlap all of the global solve and local solve in separate threads. Therefore, the global solver was overlapped with fast diagonalization calculations only.

To investigate the effects of overlapping a stenotic pipe flow was considered. This test case has attracted many experimental and computational studies because of its geometric similarities with industrial applications such as Venturi pipes. The GPU version of the code was executed for simulations of a stenotic pipe flow, consisting of 3200 elements with 7^{th} polynomial order, on a single GPU node of the Swan system. Swan is a Cray in-house XC40 system and each of GPU nodes has one NVIDIA P100 GPU and 18 Intel Xeon Broadwell cores. Overlapping the GPU calculation of the local solve with the global solve on this system results in a reduction of the execution of both solves from 92 to 70 seconds.

2.2 AVBP

In the last period, efforts have focused on the comparison of the node level performance of AVBP on multiple architectures: Intel, AMD, and ARM. CERFACS was granted access to the IRENE Joliot Curie AMD partition from PRACE at TGCC during its testing phase and to a PRACE allocation (racoe006). The AMD Epyc 2 architecture offers a unique topology were process placement can impact cache and memory bandwidth access. The IRENE AMD system

uses the novel AMD Epyc 2 architecture with two sixty-four core processors per nodes. An admin access to the nodes furthermore allowed to measure the impact of the CPU clocking frequency on the simulation. Using these features, it was possible to measure the impact of the architecture's characteristics on an AVBP execution of simulation of a von Karman street simulation. Using the AMDuProf tool the bandwidth usage at runtime on a single node was analyzed. Both results are shown below in Fig. 6.

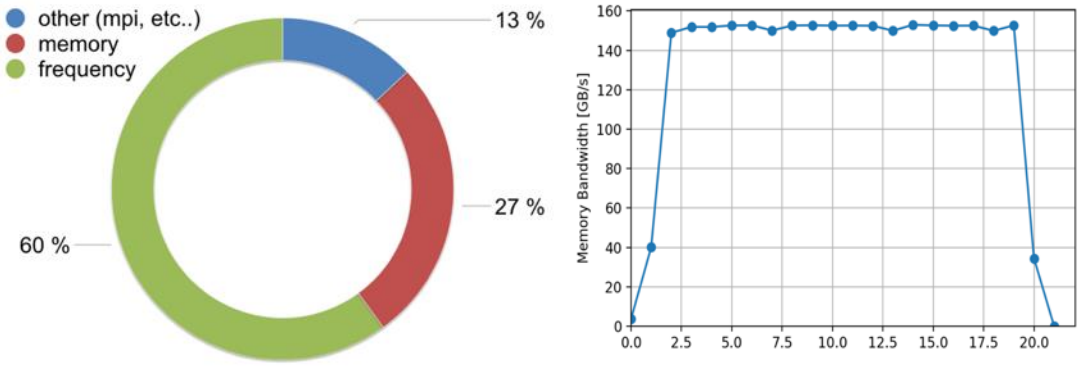


Figure 6: Code characterization for AVBP on a single AMD node (left). Measured bandwidth usage for a single node running AVBP at runtime using AMDuProf (right).

The code characterization reveals that AVBP remains compute bound with a 60% dependency of the frequency of the CPU (turboboost was disabled for these tests, but improved performance by 20% on latter cases), see Fig. 6. The high core count of the Epyc architecture (64 cores) has some but limited impact on simulation time (here 27%). This is due to bandwidth restrictions. This is verified by the bandwidth usage measurement at runtime on a single node: the code uses up to 150GB/s out of the 400GB/s that are available.

2.3 Alya

The application of a dynamic load balance strategy for the integration of stiff chemical source terms in combustion simulations with detailed chemistry was addressed. Chemical reactions occur in thin layers, which are usually characterized by highly non-linear and stiff chemical reaction rates that are very costly to evaluate. In fact, this problem is of relevance when more realistic fuels or surrogates are to be considered. In Alya, the source terms are integrated using an implicit first-order backward Euler scheme based on the library CVODE [3]. The computational costs of the chemical integration take a large share of the overall timestep for complex fuel. Therefore, strategies for increasing the performance are of paramount relevance. The high imbalance is inherent to the problem being solved. The chemical reaction is only solved at the elements containing the flame and for a certain temperature range. Therefore, the imbalance cannot be addressed by repartitioning the domain, because the problem is unsteady and the flame undertakes some dynamics that cannot be predicted. In collaboration with POP (https://pop-coe.eu), the DLB library (Dynamic Load Balancing library) [4] has been integrated into Alya to reduce the imbalance and to increase the computational performance in combustion simulations with stiff chemistry.

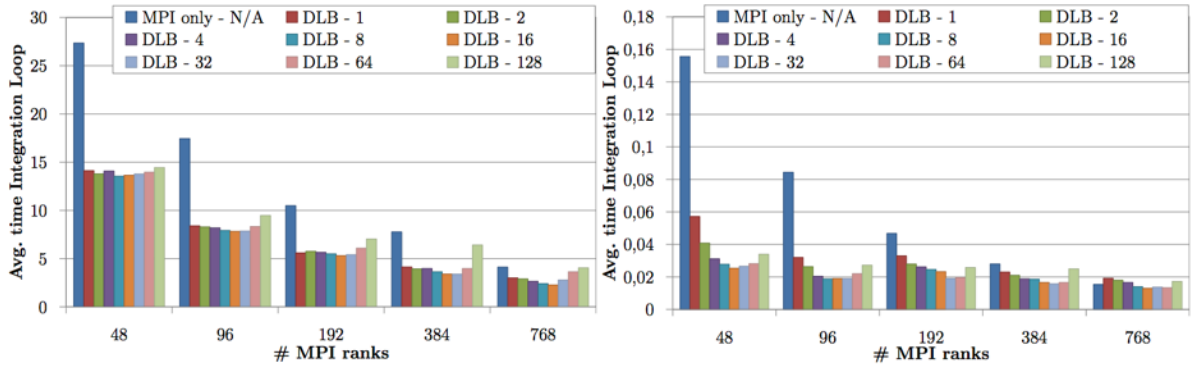


Figure 7. Scalability of the chemical integration loop: detailed chemistry (left) and reduced chemistry (right).

In Fig. 7, the duration of the integration loop for the detailed and reduced chemistry cases, where the x-axis represents the different numbers of MPI ranks corresponding to 1, 2, 4, 8, and 16 nodes on Marenostrum IV supercomputer, is plotted. Obviously, the DLB integration improves the pure MPI code in all cases. It is furthermore observable that the impact of the grain size becomes more important when the number of MPI processes is increased. A high grain size value has a negative impact on the performance when DLB is used and the number of MPI ranks is increased. This is due to the fact that the increase of the number of MPI ranks leads to a lower load per rank, and packing the load in big chunks does not allow for malleability to balance the load by DLB. The optimum grain size in all the cases is found to be around 32. In the reduced chemistry case using small or large grain sizes has a negative impact on the performance. That is, in the case of large grain sized the same situation as in the detailed chemistry use case appears, i.e., large grain sizes do not offer sufficient flexibility to load balance the computation. As it can be seen in Fig. 7, using small grain sizes in the reduced chemistry case leads to a reduced impact of DLB on the performance due to the smaller relative weight of the integration loop in the whole time step.

The speed up obtained in the integration loop using DLB with different grain sizes is compared to results of a pure MPI version running with different numbers of MPI ranks in Fig. 8. For clarity, only the largest and smallest grain sizes (i.e., 1 and 128) and the ones that delivered best performance (i.e., 16, 32 and 64) are shown. From the results it becomes obvious that with DLB using a grain size of 16 the execution can be accelerated by a factor of 2x in all the cases for the detailed chemistry case, except when using 16 nodes (768 MPI ranks), where a speed up of 1.8x is achieved. It is observed that the more nodes are used the less speed up DLB is able to obtain. This is due to DLB not featuring load balancing across nodes and also because the amount of computation per MPI rank is reduced. This leads to less granularity hindering optimal load balancing. For the detailed chemistry case, the speed up using 48 and 96 MPI ranks (1 and 2 nodes) is more impressive than in the detailed chemistry case. Here, the speed up reaches factors 6x and 4.5x. This is because the load balance in this case is fairly low, leaving a lot of space for improvement by applying DLB. When the problem is partitioned among more MPI ranks, the load is more distributed, leaving less load imbalance to address by DLB.

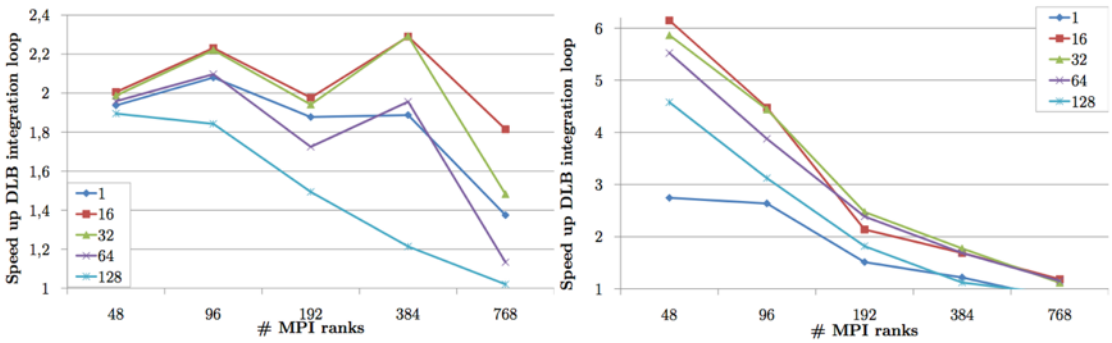


Figure 8. Speed up of DLB compared to the pure MPI execution for the integration loop: detailed chemistry (left) and reduced chemistry (right).

2.4 CODA

During the second year of the project, for CODA two main tasks were carried out in regard to node-level performance optimization. First, the linear solver library Spliss to run the computationally intensive linear solver on GPUs was integrated into CODA. Second, the performance of CODA on two different state-of-the-art CPU architectures from AMD and Intel was evaluated and compared to CODA's performance and threading capabilities.

2.4.1 Executing the Linear Solver on GPUs

Solving linear equation systems is an integral part of implicit methods in computational fluid dynamics (CFD). The efficient solving of large linear systems that result from the discretization of the Reynolds-averaged Navier-Stokes equations (RANS) in CFD methods requires algorithms that are well adapted to the specific numerical problems, which is usually not covered by generic solver libraries.

CODA introduced the Spliss (Sparse Linear System Solver) library developed for CODA, although not exclusively. Spliss aims to provide a library for linear solvers that, on the one hand, is tailored to the wide requirements of CFD applications but, on the other hand, is independent of the particular CFD solver. A key design aspect of Spliss is computational efficiency and parallel scalability for current and emerging HPC technologies. Spliss not only leverages all available parallelization levels of contemporary HPC platforms, but also offers different approaches on these levels. On the distributed level, Spliss provides the classical two-sided communication approach using MPI as well as a modern one-sided communication approach using the GASPI library [5]. Furthermore, hybrid parallelization using OpenMP and heterogeneous parallelization on GPU devices are available.

Focusing on the specific task of linear-system solving allows for integrating more advanced, but also more complex, hardware-adapted optimizations, while at the same time hiding this complexity from the CFD solver CODA. One example is the usage of GPUs. Spliss enables the execution of the computationally intensive linear solver on GPUs. However, the Spliss interface design provides this capability to a user in a transparent way. By that means, CODA can leverage GPUs without the necessity of any code adaption in CODA.

The work carried out focused on the porting of Spliss to GPUs, which has been achieved. After DLR's GPU cluster was put into operation during this period (FLUCS-R6, see D.2.1 and D2.2 for reference), first performance results of Spliss on GPUs were evaluated and an up to 25x runtime improvement for initial benchmarks was achieved. Currently, Spliss is extended to support the efficient usage of multiple GPUs per compute node. Spliss is now ready to be used by CODA, whereas the computation in the linear solver can be transparently switched between CPU and GPU. Upcoming work will focus on testing and evaluating the entire workflow of CODA with the linear solver running on GPUs.

2.4.2 Comparing Current CPU Architectures

After getting access to DLR's new HPC system CARA, which is based on AMD's Epyc architecture and an Intel Cascade Lake test system during the period (FLUCS-R4, FLUCS-R8, both delayed), both systems were evaluated with the Use Case C6U1 (FLUCS-T7, FLUCS-T9, both in ongoing). See D.2.1 and D2.2 for reference in Requirements and Tasks for FLUCS-TX and FLUCS-RX. The initial results were compared and the ideal hybrid MPI-OpenMP setup for both architectures were identified. Furthermore, a limitation in the AMD Epyc 2 architecture that limits the efficient hybrid usage to four OpenMP threads per MPI was found. This limitation restricts CODA's hybrid capabilities and hence also its scalability, since CODA relies on using as many OpenMP threads per MPI rank as possible. The Intel Cascade Lake architecture did not impose those limitations.

3 System-level performance optimization - Task 3.2

This task is focused on identifying and overcoming bottlenecks at system level. Load balancing and communication/synchronization reductions are key aspects to achieve parallel performance. Advanced features of MPI will be considered throughout the project. The developments carried out in this task include both implementation optimizations and algorithm refactoring. In the following the activities carried out in T3.2 for the second year of the EXCELLERAT project are presented.

3.1 Nek5000

3.1.1 Communication kernels for Nek5000 on SX-Aurora:

To extend the work on BK5 on the SX-Aurora presented in Section 2.1.2 to a full Nekbone implementation, an efficient gather-scatter operation is needed, both for summation between elements on a single core and global summation between elements on different cores sharing degrees of freedom.

The gather-scatter implementation in Nekbone (gslib) heavily relies on pre-processor macros and complex pointer arithmetic, which prevents vectorization by the NEC compiler. This has also been observed when porting Nek5000 to GPUs, in particular for the electromagnetics solver where gslib was one of the main bottlenecks [6].

One of the objectives in the Horizon 2020 project EPiGRAM-HS (https://epigram-hs.eu) is to carry out a refactoring of Nek5000 to modern Fortran, and to enable the code to use large-scale heterogenous systems. Of particular interest for the current work is the refactored version's newly developed gather-scatter kernel, implemented directly in Fortran, with more vectorizable loop constructs. Together with EPiGRAM-HS, the new gather-scatter kernel has been optimized for the SX-Aurora.

To obtain good performance on TSUBASA, each gather (and scatter) operation needs to be injective, otherwise loops cannot be vectorized, and the optimized machine instructions cannot be utilized. Compared to *gslib*, the EPiGRAM-HS gather-scatter (*ehs-gs*) retains information about the underlying mesh topology. Based on this information, the gather-scatter operations can be divided into a non-injective part, for degrees of freedom on edges and corners with an arbitrary number of neighbors, and an injective part for degrees of freedom on a facet with a single neighbor. In Fig. 9, the performance of the new *ehs-gs* is compared to *gslib* when

running the full Nekbone, using one and eight cores respectively. With the more vectorizable loops, *ehs-gs* is overall always faster, achieving up to more than twice the performance.

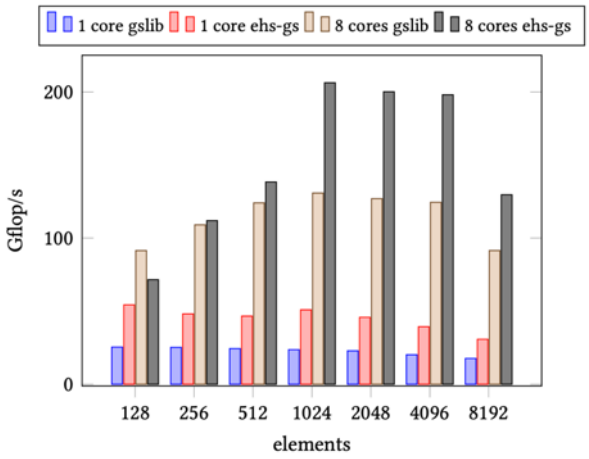


Figure 9: Performance of the tuned Nekbone using two different gather-scatter kernels, standard (gslib) and tuned (ehs-gs), evaluated on a single VE using one and eight cores respectively.

Furthermore, the new gather-scatter implementation also allows to overlap communication. In the old *gslib*, first all gather-scatter operations between local elements were performed, before shared entities were exchanged and a final gather-scatter of shared entities finalized the entire operation. The new implementation (*ehs-gs*) first gathers all the shared entities and initiates the non-blocking communication. While messages are in flight, the local gather-scatter operation is performed before each rank is waiting for messages to arrive. Once all expected messages have been received, all final gather-scatter operations are performed on the received data. The performance of the new overlapping formulation was evaluated using up to eight SX-Aurora cards, Vector Engines (VEs), running Nekbone for three different problem sizes with 1024, 4096, and 8192 elements. In Fig. 10 it can be seen that good scalability is only obtained if there are sufficient elements assigned to each VE. For the largest case, a slightly super-linear speed up on up to four VEs is achieved.

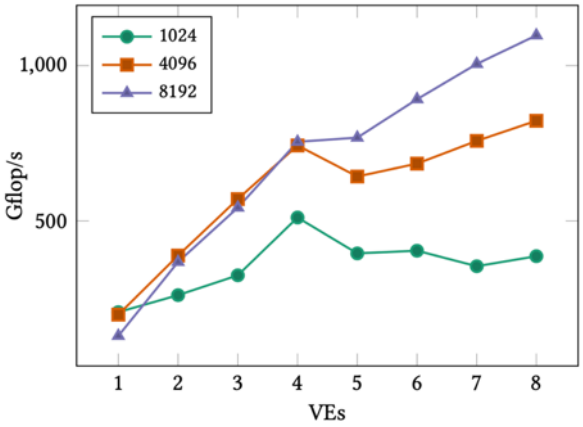


Figure 10: Performance results for Nekbone running across multiple Vector Engines (VEs).

3.1.2 Communication kernel for local solver in Nek5000 pressure preconditioner:

In section 2.1.3, the Nek5000 pressure preconditioner, which is based on the additive overlapping Schwarz method, is briefly described. Focusing on a local solver in this algorithm possible ways of refactoring the communication kernels were analyzed.

The local solve performs two exchanges of element face data: The first is used to assemble the sub-domain for the fast diagonalization method, and the second redistributes the solution. In both cases, Nek5000 applies a standard gather-scatter strategy utilizing the *gslib* library. However, in this specific situation the operations performed by *gslib* are far more complex actually needed. The library sums up all element's external degrees of freedom including faces, edges, and vertices. The local solver, however, only requires the interior face information. That is, the sum of the edges and vertices, as calculated by *gslib*, is not required for the calculation. Solely communicating and calculating the required information hence simplifies the calculation and reduces the necessary communication overhead.

Another possible improvement can be obtained by overlapping the communication and the calculation, which is not supported by *gslib*. At present, an experimental local solve communicator for conforming meshes is implemented and tests on non-AMR cases have started. The important difference between the new approach and what is done by *gslib* is the treatment of faces as objects and not as a collection of separate grid points, where each point possesses a unique global identification number. In the new communicator no global numbering that would require a local ordering is used. The consequence of this design is the explicit formulation of a transformation operator related to relative face alignment. In *gslib*, the face alignment is hidden by the global numbering of grid points. The code is continuously be developed, non-conforming mesh support will be added soon, and tests will be performed on AMR test cases.

3.2 AVBP

Strong scaling experiments of AVBP have been executed on the IRENE AMD system on up to 131072 cores using a 1.4 Billion element mesh. The results are shown in Fig. 11. Similar to the case presented in D3.1, domain decomposition was performed using Treepart [PR-PA-20-115] that maps the local hardware topology of the node to the domain decomposition to improve data locally and to reduce communications. The test was performed using the Intel 19 compiler suite and OpenMPI 4.0.2 on full nodes using flat MPI. For these tests it was not possible to use MPI3's one sided communication capabilities as the OpenMPI implementation did not support shared windows at that time (this has been resolved since, but tests have not been performed a second time yet).

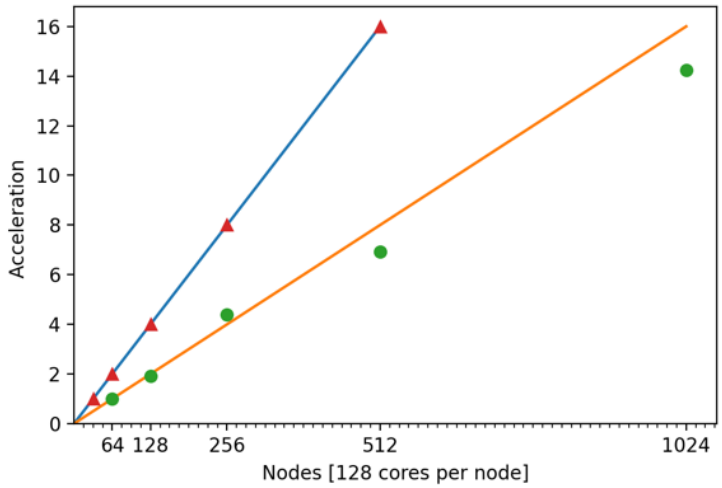


Figure 11: Strong scaling performance of the AVBP code on the IRENE Joliot Curie system (AMD architecture). Circle: 1.4B rocket engine demonstrator (PRACE collaboration); Continuous line represents the ideal acceleration.

Similar to the previous tests presented in D3.1, it seems that for AVBP the domain decomposition technique is key to reach exascale performance at the system level. Most issues encountered so far have been related to global communication in the domain decomposition phase. Using, however, TREEPART these issues have been resolved.

Weak scaling experiments have been performed to compare the full to half node performance of AVBP and the Intel MPI 19 to OpenMPI 4.0.2 performance. The results are show in Figure 12. Since the AVBP code uses fully unstructured grids with a 0 cell to 1 node halo communication, it is impossible to generate perfectly balanced weak scaling tests. As a reference for the tests the un-partitioned total number of cells on a simple turbulent channel case starting at 1M cells for 128 Cores / 1 node up to 256 nodes has been used. A performance test shows a very large performance gap between Intel MPI and OpenMPI. As the system is AMD in nature and new, it is probable that optimizations are required to account for the strange core count per node. Nevertheless, beside the shift in performance both implementations behave in the same manner when using a full node.

Half-node tests were performed to verify the influence on node bandwidth and if the load per node has an impact on the scalability. The scalability remains largely the same albeit shifted. A 20% improvement in reduced computation time (RCT), which is the time required to do an iteration and a cell of the mesh on one core. RCT [us/iteration/node,core] is observed when using half the cores per node showing a limited but real impact of bandwidth on the performance of the code. However, this performance improvement is not sufficient to compensate the 2x factor in computational costs from using only half the cores per node. A paper was accepted in the PASC '20 conference containing more details [21].

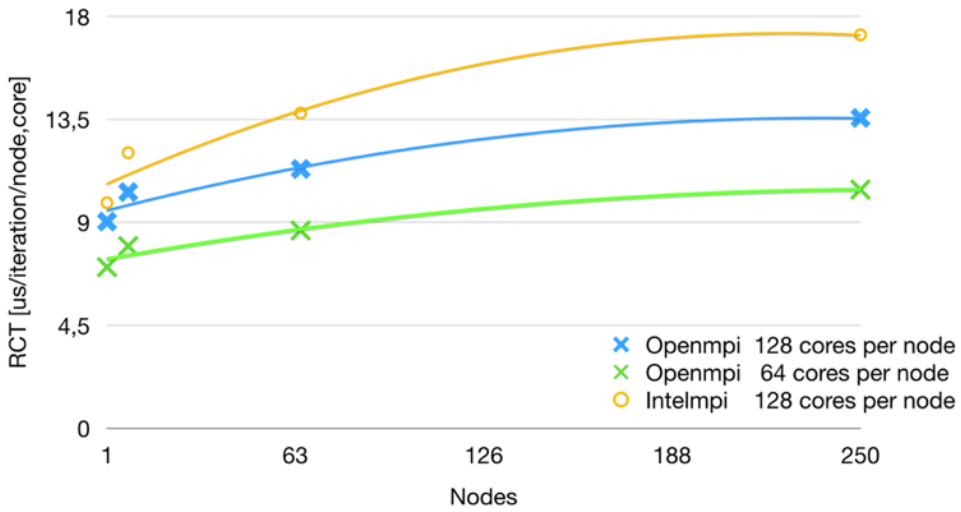


Figure 12: Weak scaling performance comparison: full to half node performance and Intel MPI 19 to OpenMPI 4.0.2

3.3 Alya

Regarding system level performance optimization, investigations on the strong scaling performance of the second use case U1C2 have been in focus for Alya. In more detail, in U1C2 the multi-phase reacting flow field of a double-swirl airblast concept spray flame of a test rig being constructed at TU Berlin is simulated. The performance of a Eulerian-Lagrangian framework, where the disperse phase is represented by Lagrangian droplets and the gas phase is described by the Eulerian phase, has been analyzed. The mesh consists of 1 billion cells and about 200k particles exist in the domain. The tests aim at showing the acceleration achieved from using 100 up to 400 nodes of the MareNostrum IV supercomputer. The parallel efficiency (PE) achieved is 91%. The MareNostrum nodes are composed of 48 CPU-cores, i.e., the maximum number of CPU-cores considered is 19200. Note that this is a multi-physics case that includes more physical phenomena than the flow simulation of the use case U1C1 presented in the first-year deliverable D3.1, where only the scaling of the Navier-Stokes solver was assessed. In the present case, the algorithm solves for the particle transport, heating and evaporation, the Navier-Stokes and energy equations, and the transport equations for the controlling variables used in the flamelet method.

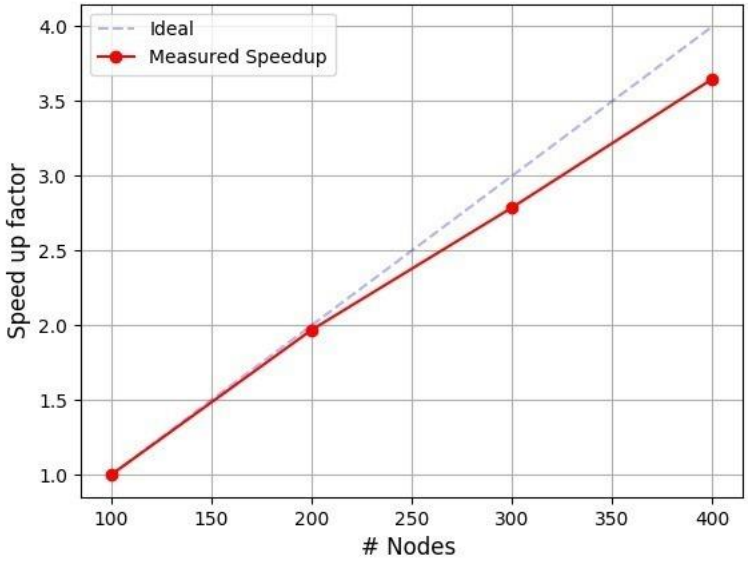


Figure 13: Strong scaling performance of the Alya code on the MareNostrum IV supercomputer for the case U1C2.

3.4 CODA

During the second year of the project, two main tasks were carried out for CODA in regard to system-level performance optimization. First, CODA and the surrounding workflow were adapted and tuned to DLR's new HPC cluster CARA. Second, the Use Case C6U1 was assessed on the new HPC cluster and CODA's scalability was evaluated.

3.4.1 Adapting and Tuning CODA on the New DLR HPC System CARA

After DLR's new HPC cluster CARA went operational in February 2020 (FLUCS-R4, delayed), CODA and the surrounding workflow were installed and intensively tested (FLUCS-T4). The AMD Epyc 2 processor introduces new architecture features that need to be considered in CODA, such as two-level NUMA domains. CODA uses classical domain decomposition to make use of distributed-memory parallelism (MPI) and additional sub-domain decomposition to make use of shared-memory parallelism (OpenMP) resulting in a hybrid two-level parallelization. Each sub-domain is processed by a dedicated software thread that is mapped one-to-one to a hardware thread to maximize data locality. Therefore, the performance of CODA was evaluated on the new architecture with particular focus on the hybrid setup of MPI ranks and OpenMP threads. CODA's performance depends on a) the size of the NUMA domains and b) the introduced overhead for thread operation across multiple NUMA domains. CODA was evaluated with different hybrid setups that correspond to different architectural characteristics of the AMD Epyc 2 processor.

It was found that thread operation across the second-level NUMA domains (across sockets) introduces a significant overhead in the range of 200% to 300% slower runtimes. In addition, thread operation across the first-level NUMA domain, i.e., using more than 8 OpenMP threads per MPI rank, introduces a small overhead (3%-12%) in comparison to using 8 OpenMP threads per MPI rank. Next to the NUMA domains, the CPU grouping by last-level cache had a noticeable impact. On the AMD Epyc 2 architecture each four CPUs share a last-level cache (L3 cache). Thread operation across multiple shared last-level caches, i.e., using more than 4 OpenMP threads per MPI rank, introduces an additional overhead of up to 20% in comparison to 4 OpenMP threads per MPI rank. Furthermore, using the two-way simultaneous

multithreading (SMT), i.e., using two hardware threads per physical CPU core, allows an addition performance gain of up to 20% to the according setup with only one hardware thread per CPU core. Consequently, the most efficient hybrid setup is 4 OpenMP threads per MPI rank or 8 OpenMP threads per MPI rank with two-way simultaneous multithreading.

3.4.1 Scalability Evaluation on the New DLR HPC System CARA

After identifying the ideal hybrid setup and adapting all workflow components to CARA (FLUCS-T4), efforts were focused on evaluating the scalability of CODA on CARA using the use-case C6U1 (part of FLUCS-T7).

The use-case solves the Reynolds-averaged Navier-Stokes equations (RANS) with a Spalart-Allmaras turbulence model (SA-neg). It uses finite volume spatial discretization with an implicit Euler time integration. The input of the use case is an unstructured prism mesh from the NASA Common Research Model (CRM) [7] with about 5 million points and 10 million volume elements. The mesh is a rather small mesh chosen for strong scalability analysis of CODA at reasonable scales. Production meshes are at least 20 times larger and accordingly achieve a good efficiency on much higher scales. For the small mesh, the Use Case C6U1 achieves about 60% parallel efficiency on the largest available partition on CARA with 512 nodes and 32k cores.

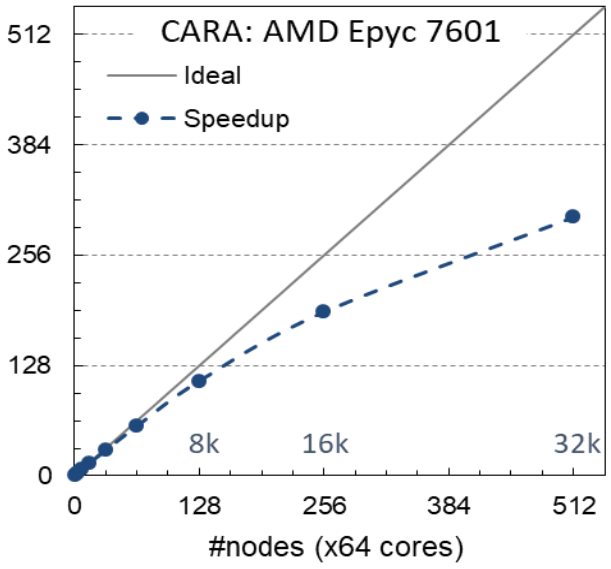


Figure 14: Scalability of CODA on CARA, DLR's HPC cluster based on AMD Eypc 2 processors.

4 Implementation of advanced meshing techniques - Task 3.3

This section describes the activities related to the meshing techniques that have been developed during the second year of the project. There are two partners (BSC and CERFACS) involved in the development of the adaptive mesh refinement (AMR) with two codes, Alya and AVBP. The third partner involved in this task is KTH, as Nek5000 has already this capability and the effort is only directed to optimizing this tool for certain conditions, so the details of this effort is not reported here, but on WP2 in particular for the deliverable D2.3.

4.1 Management Report

We maintain a living document which summarizes each Reference Application in turn, including details of ownership, contacts, meshes and meshing software currently employed, associated Use Cases, work done to date under T3.3, and work planned for their T3.3 effort. This will give all members of T3.3 an overview of each Reference Applications status which can, in turn, inform other Reference Application owner to possible synergies regarding meshing techniques in general but also, and more importantly, the topic of adaptive mesh refinement in the realm of Exascale.

As part of our living document, we have a section on the Best Practice in Meshing, describing the state-of-the-art and the future of meshing software in general, and for CFD in particular. This has improved our understanding of FEM techniques in general. This will form input to our CoE's D4.5 Best Practice Guide. The new UK Exascale programme, namely ExCALIBUR, contains an Exascale Mesh Network named ELEMENT, which ran a 2-day online workshop in October 2020. EXCELLERAT was in attendance, and discussions were presented and held regarding parallel mesh generation, end user stories, geometry definition, CAD interaction, and mesh adaptivity. This workshop has proved invaluable to inform the Section on Best Practice in Meshing.

4.2 CAD software mesh workflow in FEniCS

A workflow has been created to use CAD software meshes, including STLs, as input to a FEniCS simulation.

4.3 Mesh adaptivity in Nek5000

AMR simulations are now possible with Nek5000, thanks to work performed by CINECA. This was done using resources from WP2, and is reported in detail in D2.3. Nek5000 now employs a conforming, high order (suitable for SEM), hex-based mesh; surface representation and mesh surface projection.

4.4 Mesh adaptivity in Alya

The mesh adaptation corresponds to the requirement Alya-R3. The work in mesh adaptivity started in the second year of the project, being supported by some developments of the first year, namely Alya-R1 (fully parallel workflow) and Alya-R2 (dynamic load balance),

Indeed, the implementation of dynamic load balancing, based on mesh re-partitioning, solved one of the critical features required for mesh adaptivity: restarting a simulation online with a new partitioning. This capability requires dynamic data structures and the migration of data between parallel processes.

To complete the mesh adaptivity workflow, two steps within the simulation restart process had to be interleaved: i) the generation of the new mesh according to the properties of the physics solution and ii) the interpolation of the simulation fields from the original mesh to the new mesh.

Regarding the new mesh generation, we have taken advantage of the work carried out in the ParSec project from the WP8 of PRACE 6IP [8], where Alya has been linked with the mesh generator *gmsh*. In EXCELLERAT, we have focused on evaluating meaningful error

estimators, and the corresponding mesh size estimators, to be provided to *gmsh*. This process is illustrated in Fig. 15.

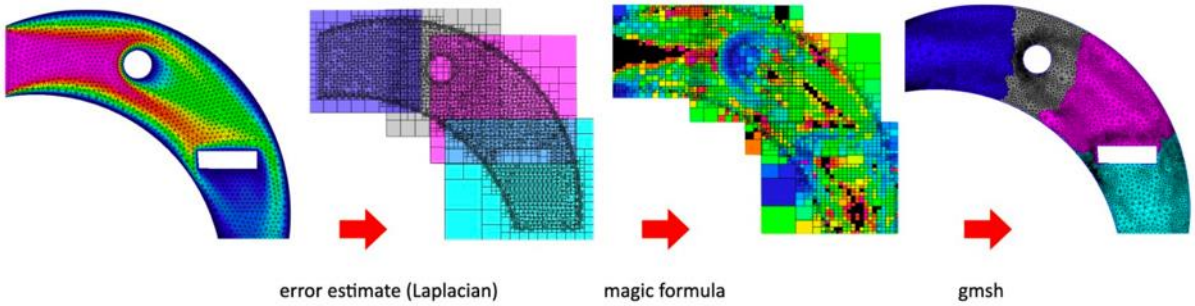


Figure 15: Solution error and mesh size estimation

For error estimation, different strategies have been implemented. Tests revealed basing the error evaluation on a Laplacian filter that is applied to the solution field to be the most promising approach. Then, a sizing formula, provided by *gmsh*, was used to evaluate a size field as an input for *gmsh* to generate the new mesh.

There are several options to provide the size field to *gmsh*. Figure 16 illustrates the variants that we have implemented in Alya. The top image shows the background mesh, distributed into two subdomains, and in the bottom, from left to right, the bin, the octree, and the octbin strategies are depicted. The octbin method is the most flexible way to provide a zonal distribution of the sizing to *gmsh*. It can achieve the bin's accuracy with fewer memory requirements and allows for smoother transitions between refinements than the octree approach.

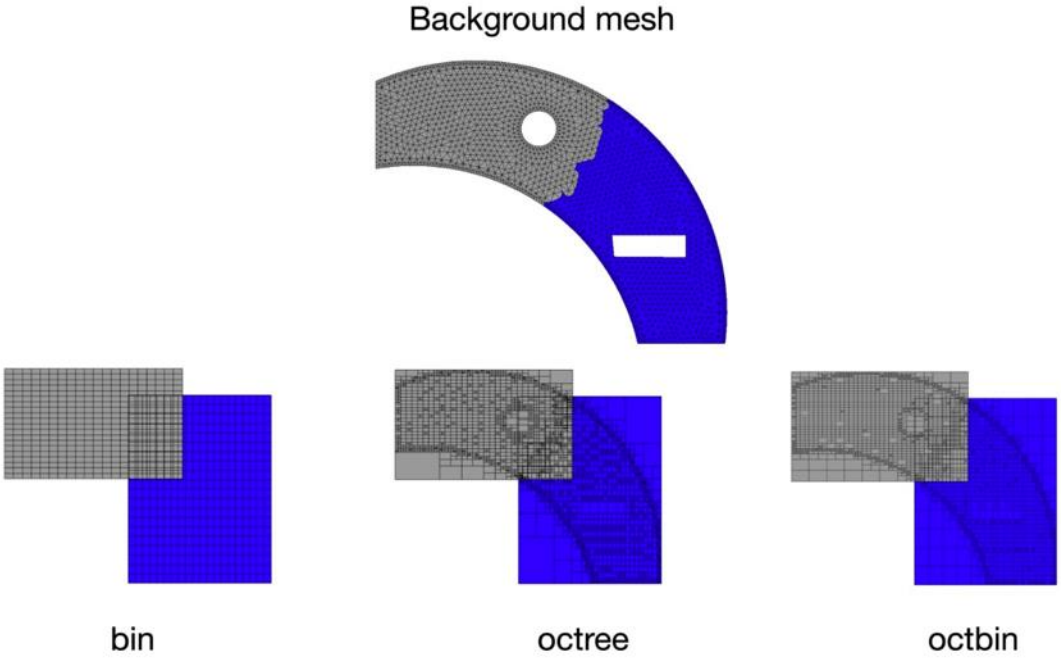


Figure 16: Options for size field representation.

The other input that needs to be provided to *gmsh* is the surface mesh, which defines the sub-domains boundaries. Each process generates a new mesh within the boundaries of its subdomain's surface. The main issue with the parallelization of the re-meshing is to deal with the mesh generation in the subdomains border. Since a conformal approach has been implemented, the interface elements between two subdomains need to match.

To solve this problem, an *interface freezing* approach has been adopted. In this approach, the interface elements are not changed such that each process can adapt the rest of its subdomain without falling into incoherent mesh configurations at the interface. However, this approach requires an iterative process, interleaving displacement of the interface and local remeshing, to ensure that the overall domain is re-meshed. This *interface freezing* method is illustrated in Fig. 17.

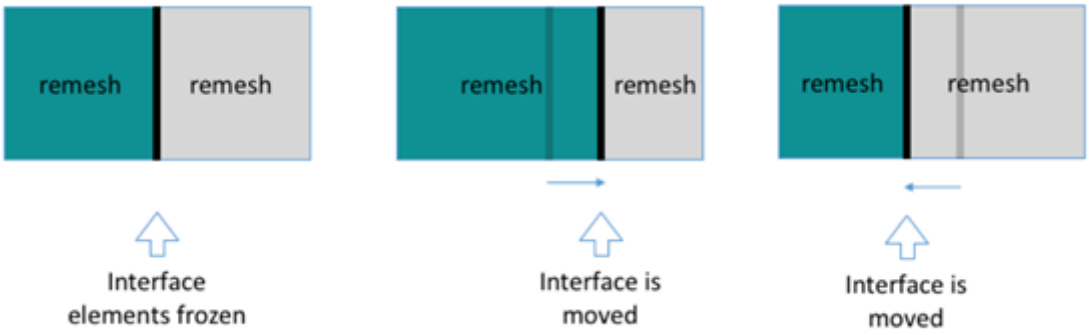


Figure 17. Parallelization of mesh adaptivity based on the *interface freezing* approach.

Finally, the load balancing functionalities, already implemented in Alya in the first year of the project, are used to balance the new mesh distribution that may have become unbalanced due to different re-meshing requirements on different domain zones.

The parallel interpolation tools already available into the Alya kernel for the interpolation step have also been used. Therefore, the overall parallel mesh adaptation workflow has been completed in the second year of the project. At present, the implementations in different examples are verified. For instance, Fig. 18 presents a snapshot of the mesh adaptation for the simulation of the flow around a cylinder at Re=120. It can be observed that the mesh is concentrated around the vortex structures of the velocity field.

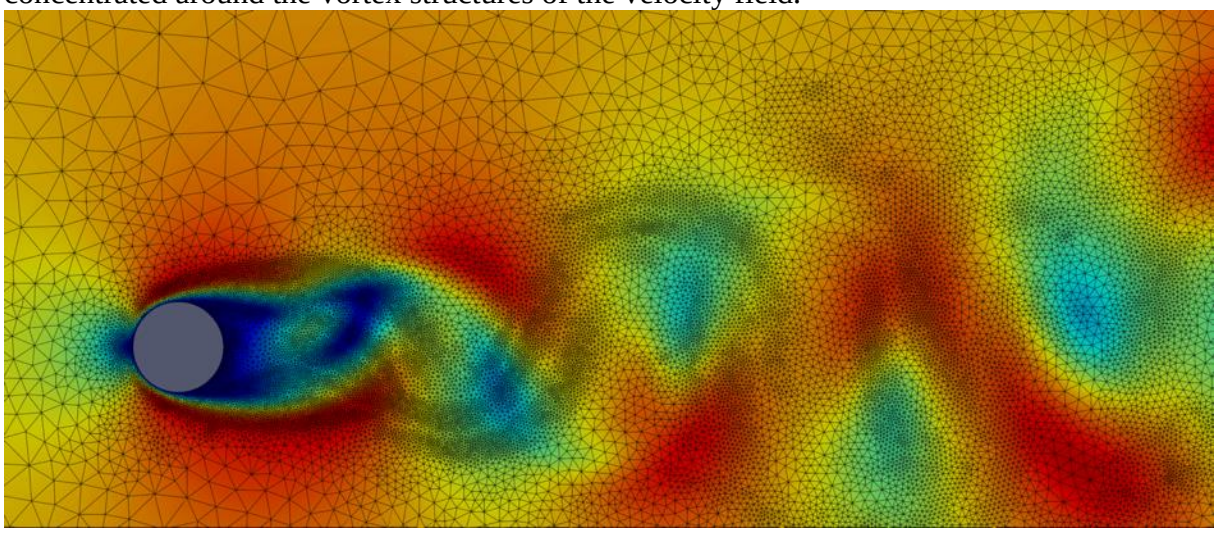


Figure 18: Mesh adaptation for the flow around cylinder at Re=120.

The next steps for the third year of the project regarding mesh adaptivity will be: i) optimize the workflow to make it efficient on large scale simulations, and ii) apply mesh adaptivity in the use cases related to the Alya code: U1C2 and U2C2.

4.5 Mesh adaptivity in AVBP

For AVBP, the work on mesh adaptivity has recently been completed. It was based on the requirements specified in D2.1 and D.2.2. AVBP-R1: dynamic mesh structure at runtime, AVBP-R2: accurate interpolation methods, AVBP-R4: Incorporate automatic mesh refinement and AVBP-R5: remeshing. All requirements have been completed recently.

A full AMR workflow is now available in AVBP using the CORIA-YALES2 library (CORIS-CNRS) [9] based on the MMG library (INRIA) [10]. The mesh adaptation strategy is shown in Fig. 19. In this workflow, mesh adaptation is handled as a black box by AVBP, data structures introduced in AVBP-R1 are shared in memory with the Yales2 library and it handles remeshing via MMG. The adaptation cycles begin with an initial partitioned mesh with a dual cache blocking decomposition. The cache blocks are merged to create a single domain per process and then, MMG performs the mesh adaptation while freezing the parallel interfaces. Subsequently, interpolation and splitting are performed as well as constrained load balancing using METIS to ensure previous domain interfaces become interior edges for the next adaptation loop. This cycle repeats until the target metric is verified.

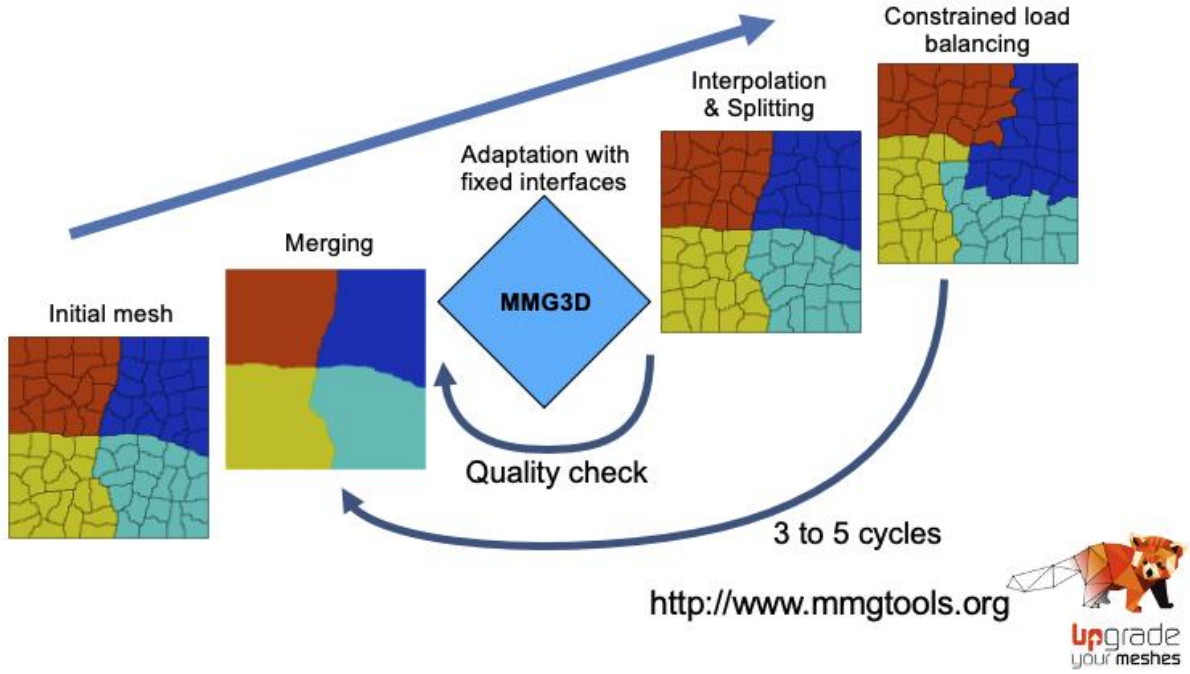


Figure 19: Mesh adaptation strategy of YALES2 (courtesy of V. Moreau)

To single out the desired refinement zones, AVBP provides a refinement mask to YALES2: 0 where the current mesh is not supposed to be modified and 1 where the mesh should be refined and a user-defined target edge size is verified.
For the current uses cases two physical phenomena need to be tracked by the AMR method: the turbulence and the flame. To handle each, a vorticity-based sensor and a flame presence sensor is employed. The vorticity sensor is a threshold on the magnitude of vorticity above which the

metric is set to 1. Current tests suggest that a level of 1000 to 3000 is sufficient for this case. This is of course highly case-dependent. Secondly, to refine the flame, the flame sensor for the thickened flame model is used to detect the zone where the flame is located and to improve the corresponding resolution locally. This sensor is based on the reaction rate with a threshold derived from a 1D flame simulation.

In the course of the project, it turned out that the CORIA-YALES2 library is not fully open source and its usage is currently reserved to research activities. This could limit the application for this workflow in future, more industrial-oriented applications. Therefore, a new open source library called TREEADAPT was implemented, see Fig. 20 for example. Taking advantage of recent improvements in mesh decomposition and load balancing in AVBP from the EU project EPEEC [11], CERFACS has extended the TREEPART domain decomposition library to feature mesh adaptation using MMG. TREEPART performs an initial domain decompositioning of the mesh and MMG adapts each domain while freezing parallel interfaces. Then, the resulting mesh is interpolated and rebalanced until the initial metric target is met. TREEADAPT takes advantage of the structure of TREEPART and is able to switch partitioning methods on the fly improving convergence of the mesh adaptation. So far, TREEADAPT has only been tested and used for static mesh refinement.

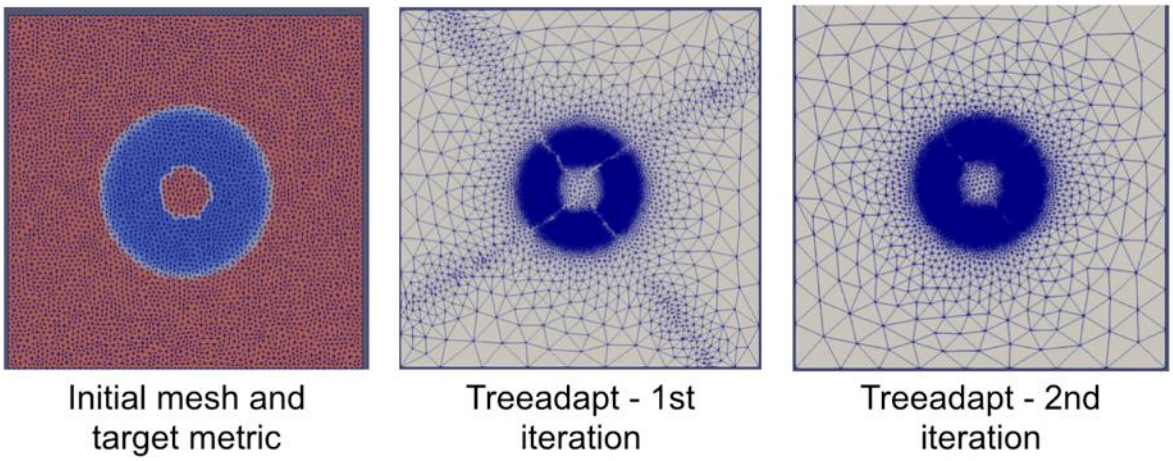


Figure 20: Mesh adaptation of an inviscid convective vortex using TREEADAPT.

TREEADAPT has been tested up to 4096 Epyc 2 AMD cores and 1.4B elements in a collaboration with the RockDyn PRACE project from Centrale Supelec (T. Schmitt), see Fig. 21 for a scaling test. It allowed to reduce mesh generation time from 3 days (using the standard meshing tools) to 30 minutes.

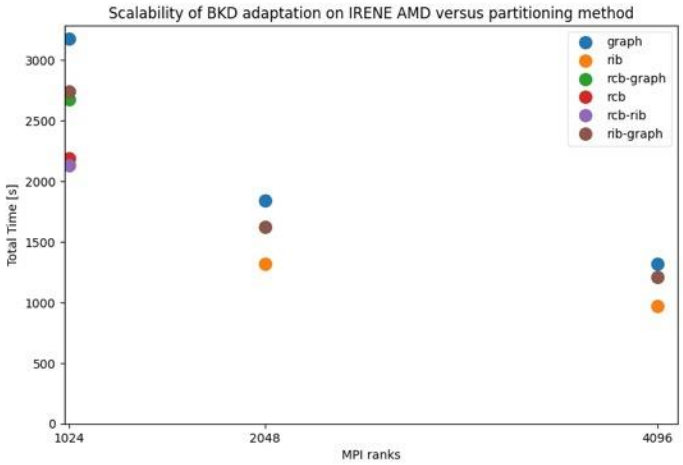


Figure 21: Scaling of TREEADAPT on the RockDyn BKD configuration on the IRENE AMD system.

At present, an extension to account for periodicities and further optimisations of the code to reduce load balancing time is investigated.

5 Test lab for emerging technologies - Task 3.4

5.1 Emerging technologies in Nekbone

5.1.1 Porting of Nekbone to FPGA

The current activity aims to accelerate Nekbone on a Xilinx Alveo U280 FPGA [11], which is the state-of-the-art FPGA architecture. There are two potential benefits of doing this: firstly, power efficiency, and secondly performance. The second is important, as whilst the FPGA can likely not match a GPU on raw performance alone, by tuning the hardware memory access then potentially some of the memory bottlenecks on the CPU can be ameliorated. The Xilinx's latest Vitis platform [12] is used for current work, where the code is written in C++ and then synthesised down to the hardware level. This builds on work done previously looking at advection kernels, and the work described here will be presented at international conference [13].

Description	Performance (GFlops)	% CPU performance	% theor. performance
24 core Xeon Platinum CPU	65.74	-	-
Initial FPGA version	0.020	0.03	0.29
Optimized for dataflow	0.28	0.43	4.06
Optimized memory access	0.42	0.63	6.09
Optimize matrix multiplication	12.72	19.35	20.85
ping-pong buffering	27.78	42.26	45.54
remove pipeline stalls	59.14	89.96	96.95
increase to 400 Mhz	77.73	118	5.73

Table 1. Steps taken to gain optimal performance for FPGA kernel.

The table above illustrates the steps taken to gain optimal performance for FPGA kernel. It is important to note that this is a single kernel, and it is possible to include multiple kernels (see later in this section.) The CPU we compare against is a 24 core Xeon Platinum Cascade Lake (8260M) and it can be seen how there are numerous steps needed to translate the initial Von-Neumann code to the dataflow perspective, but crucially one must undertake these steps if they

are going to get good performance. The dataflow approach is illustrated in the diagram reported in Table 1, where the kernel is split up into its constituent components and these are able to run concurrently. Each element consists of 4096 data points, and there was an issue with dependencies between the dataflow stages, causing stalling between them for a single element. Therefore, the approach adopted consist of dataflow pipelines decomposed into three stages, the middle one working on the current element, *e*, the previous one on the next element, *e+1*, and the next stage on the previous element, *e-1*. In High Level System terminology (HLS), the tool which takes C/C++ and synthesises this down into the Hardware Description Language (HDL) used to program the device, ping-pong buffers are used to store intermediate results and then forward them onto the next stages. This pipelines the elements themselves and increases the amount of concurrency, which is important as all the functionality here is laid out on the electronics of the chip. Furthermore, initially Xilinx' Vitis open source scientific library is used and specifically their matrix multiplication implementation. However, there were numerous overheads here when it came to streaming and again it resulted in dataflow stages stalling. This was replaced with a modified, streaming matrix multiplication stage, which significantly improved performance – both in terms of generation of results much sooner than previously, and also increasing the number of floating-point operations that can occur per cycle (vectorisation). Lastly, the clock frequency has been increased to 400Mhz (the default is 300Mhz); this should not be seen as a silver bullet because, without a well performing kernel in the first place, then a clock frequency increase might improve performance slightly, but won't address any underlying issues. Furthermore, an increased clock frequency impacts the overall complexity of the kernel, and by increasing to 400Mhz the depth of our matrix multiplication increased to 61 cycles. The theoretical performance, that is the percentage of theoretical performance achieved by a specific configuration based on the design of the algorithm, has been included in such analysis. The higher this number, the closer the dataflow algorithm stages are to being fully occupied and running efficiently (fully optimal). Lower than this, the code is stalling due to inefficiencies. As new designs of the algorithm are developed, this theoretical performance increases and a benefit of FPGAs is the transparency about how code is physically executed at the electronics level. For instance, on the CPU, and to a less extent on GPUs too, there is a significant disconnect between the ISA (the programmer's view) and the micro-architecture (how it actually runs). Thus, understanding this theoretical performance is next to impossible on many classical architectures, whereas it is far more transparent on FPGAs.

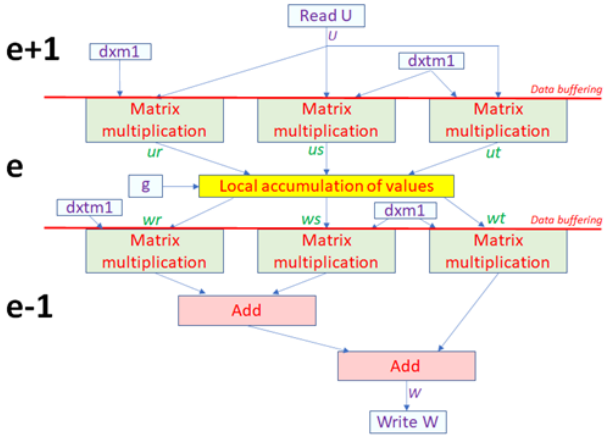


Figure 22: Dataflow used for performance's optimization

By optimizing a single kernel on the U280, only a fraction of the overall FPGA's resources is utilized. Therefore, there was further opportunity to increase performance by leveraging

multiple kernels and as each element is independent from any other element, these can be distributed across the kernels.

Description	Performance (GFlops)	Energy usage (Watts)	Energy Efficiency (GFlops/watt)
1 core of CPU	5.38	65.16	0.08
24 cores of CPU	65.74	176.65	0.37
V100 GPU	407.62	173.63	2.34
1 kernel	74.29	45.61	1.63
2 kernels	146.94	52.47	2.80
4 kernels	289.02	71.98	4.02

Table 2: Performance and energy efficiency comparison of multiple kernels versus other technologies.

Table 2 above contains a performance and energy efficiency comparison of multiple kernels against other technologies. The CPU is the same 24 core Xeon Platinum Cascade Lake (8260M) compared against previously, and running over all 24 cores resulted in an energy efficiency of 0.37 GFLOPS/Watt. For comparison, it is included a single core CPU run, which resulted in 5.38 GFLOPS and energy efficiency of 0.08 GFLOPS/Watt.

Nekbone has mature support for GPU acceleration of the kernel via CUDA. NVIDIA Tesla V100 GPU has been tested, compiling with the Portland Group Compiler version 20.5-0, and CUDA 10.2. This resulted in 407 GFLOPS and, due to the high performance, an energy efficiency of 2.34 GFLOPS/Watt. The GPU's performance is impressive, although it should be noted that the bespoke GPU acceleration in Nekbone has been developed and tuned over many years and GPU generations. Table 2 also reports the performance for different numbers of our *ax* kernels on the Alveo U280 FPGA. One kernel draws 45.61 Watts (the FPGA idle with the bitstream loaded draws 39 Watts), and whilst the energy efficiency of 1.63 GFLOPS/Watt of a single kernel is significantly higher than the CPU, it is somewhat disappointing when compared against the GPU. However, the advantages of FPGAs start to become more apparent as the number of kernels is scaled. It is possible to fit up to four of our kernels on the U280, and at this configuration 289 GFLOPS are achieved. This over four times the performance of the 24 core CPU, and 71% of the performance of the V100 GPU. The energy consumption of four kernels is 72 Watts and it can be observed that, on average, adding an extra kernel requires approximately an additional 7 Watts, with a performance increase close to 74 GFLOPS per kernel. With four kernels, the energy efficiency is over 4 GFLOPS/Watt, which is significantly higher than that of the GPU. We were pleasantly surprised with how well FPGA performance scaled as we added kernels. In part this is because the ports of each kernel connect to different High Bandwidth Memory (HBM) banks, so there is no contention. We found that if HBM banks were shared between kernels then it resulted in conflicts, excessive time in the additional hold fix phase of routing (over 12 hours) and reduced performance. Effectively, by keeping the kernels separate they can then run independently and scale better. Therefore, whilst FPGA considerably beat the CPU in terms of raw performance, against the GPU performance is more challenging to match. However, power efficiency is very important and the FPGA is way ahead of the CPU when it comes to this metric and almost double the power efficiency compared with the GPU. This is a very important aspect to highlight for future exa-scale architectures.

5.1.2 Nekbone on hybrid CPU-GPU clusters

Nekbone [14,15] is a proxy app for Nek5000 that illustrates important computational and scaling aspects of the entire solver. Nekbone solves a standard Poisson equation using a conjugate gradient iteration with a simple or spectral element multigrid preconditioner on a

block or linear geometry (parameters are set within the test directory of the simulation). Nekbone exposes the principal computational kernel to reveal the essential elements of the algorithmic architecture coupling that is pertinent to Nek5000. In particular, the evaluation of the Poisson operator through a tensor-product operation is the most time-consuming part of both Nekbone and Nek5000 [16]. The purpose of this kernel is to be able to carry out tests, using new and emerging architectures, without having to involve the larger suite. This work is based and it is a follow-up of the GPU tests reported in [17]. The Nekbone code has been the subject of testing on various HPC clusters up to Tier-0 class, in particular:

- **Galileo cluster** (CINECA); equipped with about 1000 Intel Broadwell nodes (2x18-core Intel Xeon ES-2697 at 2.30 GHz), and OmniPath interconnection.
- **Marconi A3 partition** (CINECA): It is a Lenovo NeXtScale platform. A3 partition features Intel Xeon Skylake (SKL) processors.
- **Marconi 100** (CINECA): it is based on 980 nodes; each node is equipped with 2x16 cores IBM POWER9 AC922 processors and with 4 NVIDIAV100 GPUs connected with a high-speed internal network Mellanox Infiniband EDR DragonFly+.
- **ARMIDA** (E4): 8 Marvell TX2 compute node (per node: 64 cores Marvell TX2@2,2/2.5 GHz, 256 GB RAM) Mellanox IB 100Gb EDR, Nvidia Tesla V100-PCIE 32GB).

Nekbone has run with 100 CG (Conjugate Gradient) iterations, polynomial degree 9 and for 10 GLL points in each dimension. The reference test is “example2” from the official repository [18]. It will run without the multigrid preconditioner and without user-provided decompositions of the processor counts and the elements. Tests with V100 GPUs report different GPU programming models (OpenAAC, CUDA Fortran, CUDA shared memory, CUDA Fortran optimized). Figure 23 shows the weak scaling, by keeping constant the number of elements per core (128). Using a patch about MPI tag, Nekbone ran well over 152,160 cores on Marconi A3, with a peak performance of 35,9 Teraflop/s, using OpenMPI (3.0).

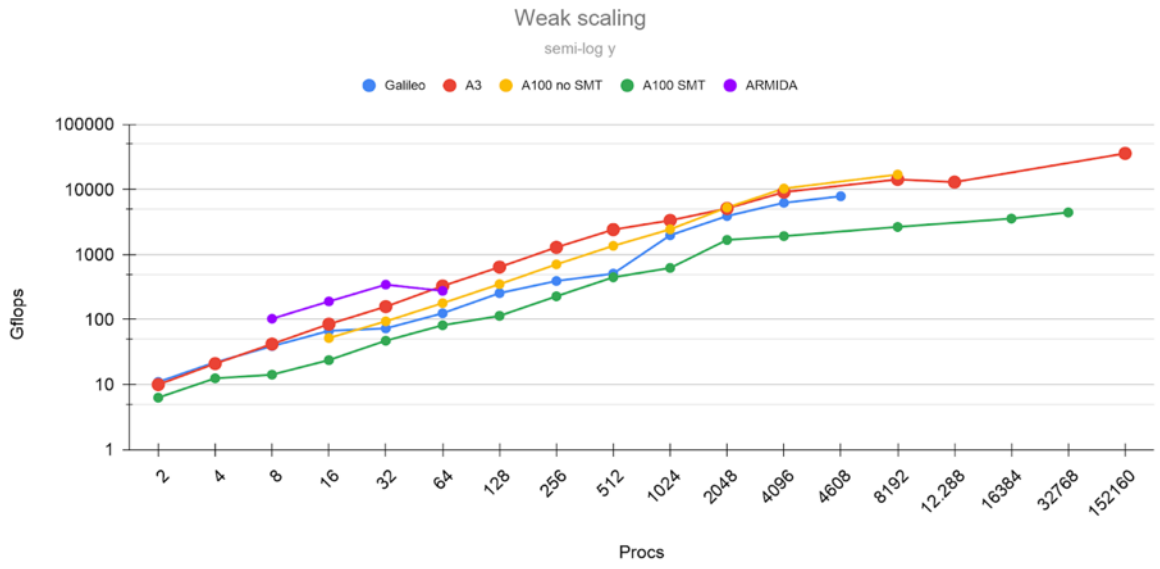


Figure 23 Weak scaling of Nekbone on different architectures, only cpus; semi-log y scale

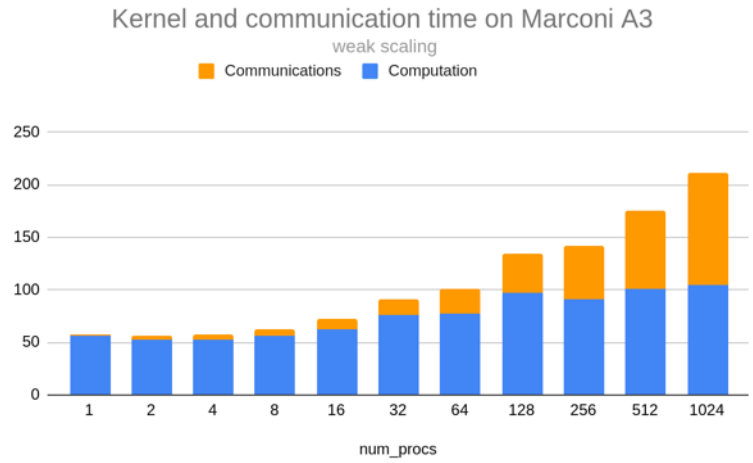
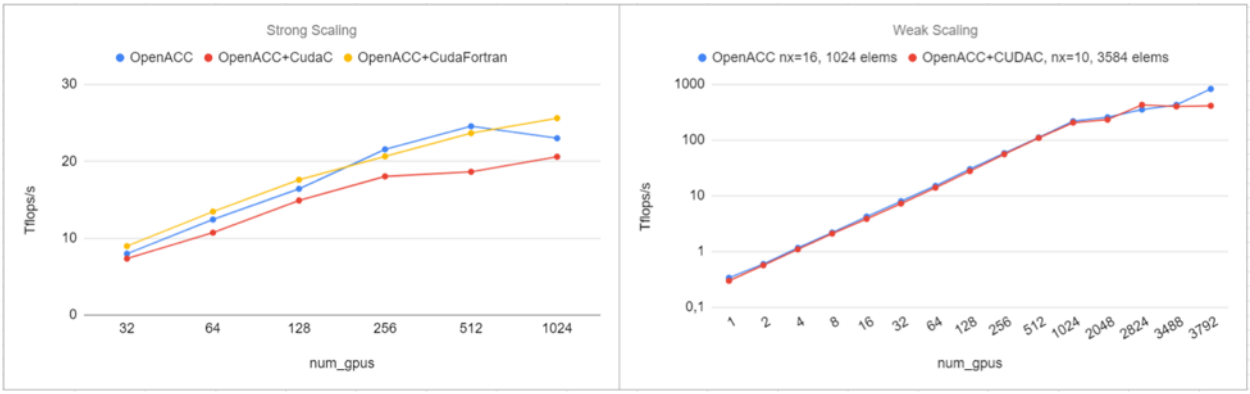


Figure 24: Kernel and communication time using Marconi A3.

Figure 24 shows the kernel and communication time (using Intel APS). It is clearly reported the MPI boundness of Nekbone by increasing the number of processes using Marconi A3. Nekbone has three different GPU porting's flavour: i) OpenACC ii) OpenACC + Cuda Fortran (PGi compiler only) iii) OpenACC + CUDAC. All versions use 1 MPI process per GPU model, so the maximum number of usable MPI processes is four per node. The protocol 1 MPI-N GPU model is not supported. All three versions were intensively tested.



Figures 25: Strong (left) and weak (rigth) scaling of Nekbone on M100 GPU cluster, semi-log y scale.

Initially, for the MultiGPU mode, the performance turned out to be very poor, even two orders of magnitude less than the single GPU one. This performance was due to incorrect MPI-GPU mapping during the execution on Marconi100. In fact, the code relies on external directives for mapping. By making small changes to the code (adding the *cudaSetDevice*), the mapping now takes place directly from the source code, without the need to know the configurations of the environment. By doing so, performance around petaflops (1 PFlop= 1000 TFlops) is achieved. Strong Scaling on Gpus shows in Figure 25 (top) the best performance using OpenACC + CudaFortran. For this activity, the relevant figure is the weak scaling on the full machine. Weak scaling of Fig. 25 (bottom) shows the weak scaling of two different set-ups: the pure OpenACC version (blue line) has a polynomial of order 15 with 1024 elems/GPU, whereas the CUDAC one (red line) has fewer degree (nx=10) but more elements (3584 elems/GPU). Therefore, the first case is more computationally intensive. The best performance has been achieved recently with 835 TeraFlop/s in OpenACC configuration, out of a total of 3792 GPUs. We should stress that the comparison is not fair when comparing cases using different *nx*. We want to compare the "best run" for each configuration that it is possible to run. This has been reached thanks to

the new GPU Nvidia driver (v 440.64.00) with cuda v.10 and fine tuning of the kernel. These results, is as far as we know, the best peak performance achieved by Nekbone into HPC cluster; it overcomes the results reported in Fig. 5 of [17], by running the same test-case on similar architecture (Titan with V100 GPus configuration used in ref [16]).

5.2 Emerging technologies in AVBP

5.2.1 Porting to GPUs

In the first year, AVBP was heavily modified to account for a dynamic mesh structure required for AMR. Using the experience on a first GPU port, we have ported again the code with this new version and released version 7.7 (Sept. 2020). The directive-based language OpenACC was used to ensure a portable and maintainable source code. AVBP 7.7 supports GPU usage for standard combustion cases with basic boundary conditions. Since, we have continued the port and are currently working on supporting the workflow for use case C3U2.

Meanwhile, performance tests have been carried out in the PIZ DAINT system as show in figure 26. These tests were performed using use case C3U1 with a static mesh. Scalability up to 32 GPUs is excellent. However, they revealed a very strong dependency of time to solution on the cache block size. Indeed, AVBP had been optimized originally for scalar processors and uses cache blocking to improve performance on standard architectures. This feature in return hinders performance on GPU accelerators. In our case the bigger the cache block, the best the performance on GPU.

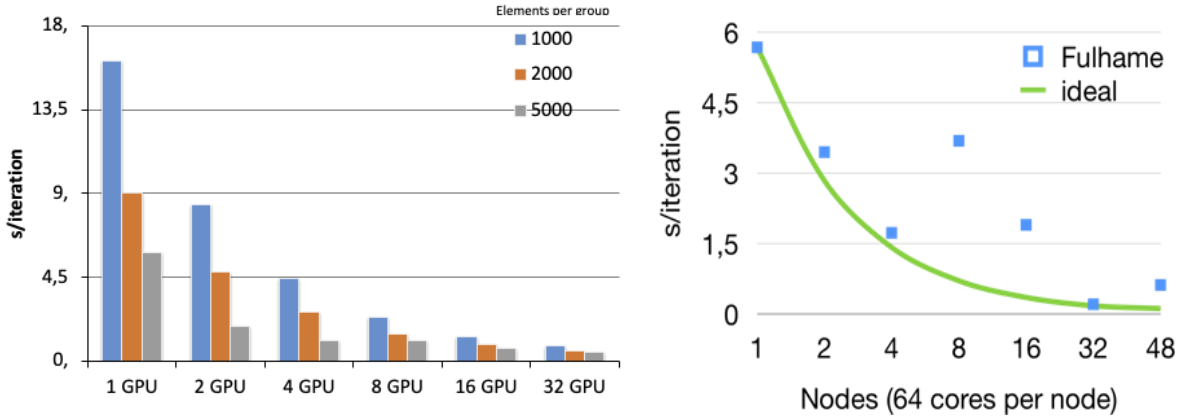


Figure 26. Strong scaling performance using use case C3U1 (static mesh) on the CSCS Piz Daint system (left) and on the FULHAME system (right).

Currently the GPU workflow is being extended to support C3U2 by accelerating Wall Law boundary conditions as well as a new combustion model and a new diffusion operator. Additional tests are being performed on V100 system (CTE POWER at BSC and JEANZAY at IDRIS).

5.2.2 Porting to ARM based architectures

Another target architecture for AVBP is ARM. The success of the A64FX architecture foretells a plethora of ARM equipped processors for 2021 as well as the European Processor Initiative which has chosen ARM as one of the alternative designs for a European processor.
During the last period AVBP has been ported to three ARM systems:

- INTI: thunderx2 system from the GENCI (Tier 0-Tier 1 French coordinator).
- FULHAME: thunderx2 system from EPCC.
- hi16164: huwai processor (early silicon) from EPI at JSC.

Porting to this architecture was mostly straightforward as the code already support GNU compilers natively and the ARM compiler is mostly based on clang.

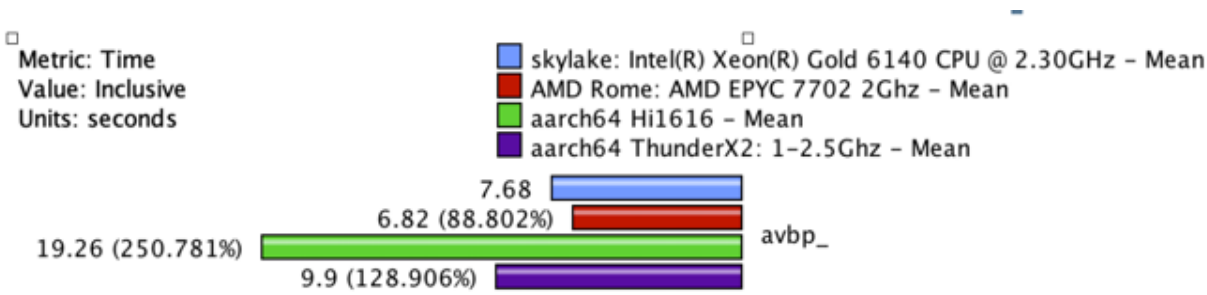


Figure 27: Gprof results for a simple combustion simulation on a single node versus architecture.

Figure 27 highlights the comparative performance results versus AMD and Intel processors. Details on frequency were not available for all systems. This suggest the competitiveness of the ThunderX2 architecture since performance is comparable while not supporting any vectorisation. However, the Hi1616 early silicon seems to be behind in performance. This might be due to the lack of compiler optimisations for this architecture or defaults on the early silicon. Strong scaling on the FULHAME system is reported on the right hand-side of Fig. 26. These are early tests performed on a new system but they already show a good scaling on the architecture. Some artefact can be seen (for example using 8 nodes), they have been identified as early system issues not related with the code or the architecture. During the next period it is expected to have access to a64fx architectures to see the benefits from vectorisation on the ARM architecture.

5.3 Emerging technologies in Alya

In the first year of EXCELLERAT project, Alya's matrix assembly phase has been ported to GPUs using OpenACC. The matrix assembly alternates with the linear solver on each time step of a simulation. We presented some tests carried out on the POWER9 cluster at BSC, composed of nodes with 4 NVIDIA Volta V100 GPUs and 2 POWER9 8335 CPUs (40 cores in total). For the numerical experiments, we used the flow around a full airplane case (U1C2). Roughly speaking, we observed that, in terms of performance, a single GPU performance is equivalent to the 2 CPUs. Therefore, as the load of running with CPUs is not negligible, we developed a co-execution strategy to simultaneously use both the CPU and the GPU. The 20% performance boost achieved was coherent with the ratio of the performance of the two devices.

The directives-based language, OpenACC, is our choice for the GPU because it is easy to implement, and the resulting code is readable for the developer. The matrix assembly phase is where the equations are discretized, and this is a part of the code that is maintained and developed by application scientists. Using a low-level language, such as CUDA, should improve the performance but pose difficulties in maintaining and further developing the code. This situation is not the same in other parts of the code. For instance, application scientists generally use the linear solver as a black box, so it does not require the same readability level. Consequently, for the linear solver, we use accelerated libraries based on CUDA.

In this second year of the project, we have carried out a study to measure the performance that could be achieved by i) using CUDA instead of OpenACC and ii) creating specific kernels for each type of mesh element - extending both the length and complexity of the code. In the left hand-side of Fig. 28 some numerical experiments carried out at the CTE POWER9 cluster for the flow around airplane case (U1C2) are reported. For the three types of elements evaluated

(tetrahedral, pyramids and hexahedra), the OpenACC generic kernel's elapsed time is about twice of the specific kernel implemented in CUDA. In the right-hand side of Figure 29, the same analysis has been carried out for the basic kernels algebraic of the linear solver: SpMV, AXPY and DOT. In this case the language has been changed, but the kernel has not been restructured as in the assembly. For the vector operations (AXPY and DOT) the performance obtained is almost the same, while a slowdown of 15% is observed for the SpMV with the OpenACC implementation. This study provides a sort of ideal performance reference for the GPU version of the matrix assembly. The next steps will be to analyse the trade-off between maximal performance versus software maintenance and development considerations. Nevertheless, having established a reference will allow to quantify the potential benefits of the corresponding developments.

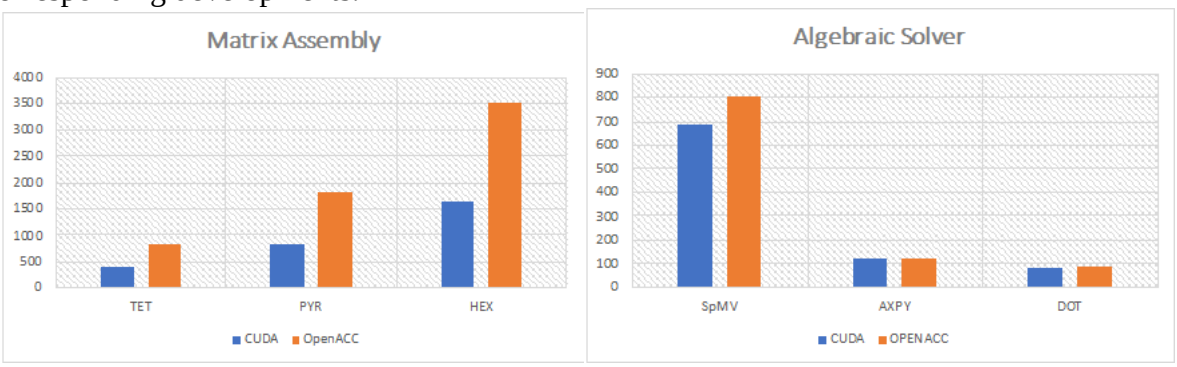


Figure 28. Elapsed time (microsec) of the matrix assembly for different element types (left) and basic linear algebra solver kernels for OpenACC and CUDA implementations (right).

6 Validation and benchmarking suites - Task 3.5

In order to quantify and evaluate the progress and evolution of the codes after the technical developments made in WP3, some benchmarks were defined for each code. This will permit to monitor the progress of the codes throughout the life of the project and evaluate the performance of the codes respect to the starting day. These benchmark cases are not expected to be as the use-cases of WP2, but they are defined in order to expose the bottlenecks of the codes when running the reference applications in WP2. The different cases and activities involved in the execution of these benchmarks are given below:

Partner	BSC
Code	Alya
Test case	JAXA high-lift configuration Standard Model
Linked use-case	C2U2 – External aerodynamics
Requirements (WP2)	Alya-R2: Dynamic load balancing Alya-R5: Portability to emerging technologies.
Objective	Co-execution on heterogeneous architectures
Short Description	A dynamic load balancing strategy was applied to efficiently run full aircraft simulations in the POWER9 heterogeneous architecture with NVIDIA V100 accelerators for medium-size meshes of the order of ~200 million elements. A parallelization strategy was implemented to fully exploit the different levels of parallelism, together with a novel co-execution method for the efficient utilization of heterogeneous CPU/GPU architectures.

Activities	Description	Starting date	End date
A1	Porting assembly phase to GPU	November 2018	May 2019
A2	GPU speedup analysis	March 2019	April 2019
A3	Load balancing for co-execution	June 2019	December 2019
A4	Load-balancing analysis	September 2019	January 2020

KPIs	April 2019[1]	January 2020[1]	…
SP of the accelerated vs. non accelerated assembly	3.48x	-	-
LB achieved with co-execution in P9 system	-	96%	
TS reduction achieved with co-execution vs. GPU only version in P9 system	-	23%	

Table 3. Benchmark suite Alya.

Partner	CERFACS
Code	AVBP
Test case	2D flame propagation
Linked use-case	C3U1 – Safety application
Requirements (WP2)	AVBP-R1: Dynamic mesh structure AVBP-R4 : efficient remeshing AVBP-R5: remeshing criteria
Objective	Parallel Dynamic mesh refinement
Short Description	A parallel remeshing strategy has been applied to a 2D flame propagation case derived from C3U1 to evaluate the potential acceleration of AMR and study remeshing criteria strategies

Activities	Description	Starting date	End date
A1	Dynamic mesh structure	November 2018	Jan 2020
A2	Remeshing criteria for 2D	Jan 2020	Dec 2020

KPIs	April 2019[2]	January 2020[2]	…
SP of the adapted case compared to the resolved	-	2x	-
TS reduction achieved with co-execution vs. GPU only version in P9 system	-	3%	

Table 4. Benchmark suite AVBP.

[1] Respect to baseline condition
[2] Respect to baseline condition

Partner	KTH
Code	Nek5000
Test case	AMR simulation of flow over NACA0012 aerofoil with 3D wing tip
Linked use-case	C1U1 – Wing with 3D wing tip
Requirements (WP2)	Nek5000-R3: Efficient strategies for hex-based meshing of complex geometries Nek5000-R4: Proper scheme for element's geometry description and projection of grid points on defined surface Nek5000-R5: High quality mesh partitioner based on graph bisection Nek5000-R10: Implementation and testing of UQ tools in Nek5000
Objective	Pre-processing stage: building hex-based coarse mesh for moderately complex geometries Code initialisation: testing initial AMR pipeline focusing on geometrical mesh consistency
Short Description	Performing AMR simulation starts with creating very coarse mesh, that would be later refined in the region with significant computational error. For hex-based meshes with complex geometries this is a challenging task. During a run the mesh is dynamically modified by adding/removing computational subdomains (elements) keeping external domain surfaces unchanged. This requires additional geometry correction step based on 3D projection.

Activities	**Description**	**Starting date**	**End date**
A1	3D projection routines for NACA0012 profile with rounded wing tip	Mar 2019	Apr 2019
A2	Coarse mesh of NACA0012 profile with rounded wing tip	Apr 2019	May 2019
A3	Initial refinement on wing surface (without use of error indicator)	May 2019	Sep 2019
A4	Full AMR run (including spectral error indicator) with v19 Nek5000 version	Sep 2019	Apr 2020
A5	Testing different refining strategies for h-type AMR	Apr 2020	
A6	UQ tools in Ne5000	May 202	
A5	Testing different partitioning tools (ParMETIS, PARRSB [19], TREEPART) and strategies for relatively complex meshes	Sep 2020	

KPIs[3]	Baseline	May 2019	Oct 2020	…
USR/USR	Coarse mesh generation with Nek5000 specific tools (to be computed)	Achieved		
LB	Two -level graph partitioning using ParMETIS (to be computed)		Testing and comparison of ParMetis and PARRSB on relatively big and complex meshes	

Table 5. Benchmark suite Nek5000 - 1.

Partner	KTH
Code	Nek5000
Test case	AMR simulation of flow over 3D periodic hill
Linked use-case	C1U1 – Wing with 3D wing tip
Requirements (WP2)	Nek5000-R5: High quality mesh partitioner based on graph bisection Nek5000-R6: Efficient pressure preconditioner for non-conforming, deformed elements
Objective	Code initialisation: testing mesh partitioning using graph bisection; testing initialisation of the coarse-grid solver for deformed elements Code executions: monitoring pressure iteration count for different element aspect ratio.
Short Description	A key aspect of the performance of the incompressible flow solver is efficient solution of pressure problem, as divergence-free constraint is a man source of stiffness in the set of equations. In this test we focus on the main performance issues e.g. work balance and efficient pressure preconditioner.

Activities	Description	Starting date	End date
A1	Merging/adapting existing AMR branch with official Nek5000 repository (v19)	May 2019	Feb 2020
A2	Testing different partitioning tools (ParMETIS, PARRSB) and strategies	May 2020	Oct 2020
A3	Improved pressure preconditioners for non-conformal meshes using AMG	Oct 2020	
A4	Improved pressure preconditioners for non-conformal meshes with deformed elements (two-level Schwarz and Schwarz-multigrid	Oct 2020	
A5	Refactoring of communication kernels	Sep 2020	

[3] KPI's need to be previously defined in "KPIs_Benchmark_Suite.docx"

KPIs[4]	Baseline	Oct 2020	Status (Date)	…
LB	Two -level graph partitioning using ParMETIS (to be computed)	PARRSB: adjusted to AMR branch and tested on small simple meshes		
CompE	Pressure preconditioner based on additive Schwarz and XXT (to be computed)	Implementation of simplified communication kernel for local solver in Schwarz preconditioner		

Table 6. Benchmark suite Nek5000 - 2.

7 Data dispatching through data transfer - Task 3.6.

Like in the previous deliverables, SSC is combining their two work package efforts “Data dispatching through data transfer” from Work Package 3 and “Data Management” from Work Package 4 into one deliverable, which also fits to their motivation in combining data transfer and data management into their newly designed data exchange platform. The detailed contribution can be found in deliverable D4.6.

[4] KPI’s need to be previously defined in “KPIs_Benchmark_Suite.docx”

8 Conclusions

As a conclusion, the progress on the development of Exascale enabling technologies on the EXCELLERAT core codes for the second year of the project has been presented. While in Year 1 most of the work was dedicated to node-level and system-level performance optimizations, this year substantial effort was dedicated to AMR and emerging technologies. The activities carried out by the partners on these tasks have been focused on porting to GPUs, use of the new vectorial architecture SX-Aurora from NEC and testing the memory features of the modern AMR Epyc 2. Finally, aspects related to intra-node parallelization, such as load balancing and OpenMP threading optimizations, have been considered. At the system level, the focus has been the strong scaling and optimizing the communication kernels. Additional focus has been given to improving the strong scaling of the codes and designing and implementing new distributed memory load balancing strategies. A benchmark suite to test and monitor the evolution of the codes was developed and it is fully operational. The data transfer and dispatching strategy has been extended outside the project consortium to the medical sector. Meshing activities have made a good progress and AMR has been tested and demonstrated on some use-cases for AVBP along with the in-house implementation using the TREE PART domain decomposition library.

The developments presented here along with the demonstrators based on the use-cases described in D2.3 evidence a clear progress to bringing the engineering world closer to exascale. These activities are the central part of the technical core of EXCELLERAT and intimately connected to the applications in WP2 and the services in WP4. These advances in HPC algorithms and computational methodologies are the building blocks not only for the use-cases described in EXCELLERAT, but also beyond and could be applied to other applications of the engineering realm. These advances in HPC technologies for Exascale are part of the expertise of the EXCELLERAT consortium and are ultimately defined as services that EXCELLERAT can deliver to the engineering community. This WP has contributed with the following services to the EXCELLERAT services portfolio, further details are given on the EXCELLERAT website [20]:

- Co-Design Engineering Software- and System-Design.
- Data management for large scale simulation result and input data.
- Efficient and modern implementation of Exascale ready engineering software.
- Efficient execution of large-scale engineering simulation workflows.
- Holistic Testing and Validation for the Engineering Workflow.
- Meshing and re-meshing techniques, methodologies and Software.
- Modelling of Engineering Problems.
- Numerical Solution methods for Engineering Problems.
- Performance Engineering for the Complete Large-Scale Engineering Workflow.
- Strategies for Load-Balancing and Data-distribution.

9 References

[1] libCEED. 2020. Development site. https://github.com/ceed/libceed.
[2] Stefano Markidis, JingGong, Michael Schliephake,Erwin Laure, Alistair Hart, David Henty, Katherine Heisey, and Paul Fischer. 2015. OpenACC acceleration of the Nek5000 spectral element code. The International Journal of High-Performance Computing Applications 29, 3 (2015), 311–319.
[3] A. C. Hindmarsh, P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward, "SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers," ACM Transactions on Mathematical Software, 31(3), pp. 363-396, 2005.
[4] https://www.bsc.es/research-and-development/software-and-apps/software-list/dlb-library-dynamic-load-balancing.
[5] http://www.gaspi.de/gaspi/.
[6] M. Otten, J. Gong, A. Mametjanov, A. Vose, J. Levesque, P. Fischer, and M. Min. 2016. An MPI/OpenACC implementation of a high- order electromagnetics solver with GPUDirect communication. The International Journal of High-Performance Computing Applications 30, 3 (2016), 320–334.
[7] https://commonresearchmodel.larc.nasa.gov.
[8] ParSeC: Parallel Adaptive Refinement for Simulations on Exascale Computers. Competitive project within PRACE-6IP (GA 823767).
[9] https://www.coria-cfd.fr/index.php/YALES2S
[10] Three-dimensional adaptive domain remeshing, implicit domain meshing, and applications to free and moving boundary problems – C. Dapogny, C. Dobrzynski and P. Frey – April 1, 2014 – JCP, 262, pp. 358–378.
[11] https://www.xilinx.com/products/boards-and-kits/alveo/u280.html
[12] https://www.xilinx.com/products/design-tools/vitis/vitis-platform.html
[13] Brown N.: *"Exploring the acceleration of Nekbone on reconfigurable architectures"*, Sixth International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC'20), 13 Nov. 2020
[14] P. F. Fischer, J. W. Lottes, and S. G. Kerkemeier. Nek5000 website.Accessed:.02 Sept. 20020 [Online]. Available: https://nek5000.mcs.anl.gov/
[15] H. M. Tufo and P. Fischer, "Terascale spectral element algorithms and implementations," in Proc. Supercomputing'99, 1999.
[16] Karp M, Jansenn N., Podobas A., Schlatter P., Markidis S.: *Optimization of Tensor-product Operations in Nekbone on GPUs*, Distributed, Parallel, and Cluster Computing (cs.DC) DOI https://arxiv.org/abs/2005.13425
[17] Gong J, Markidis S, Laure E, Otten M, Fischer P, Min M: *Nekbone performance on GPUs with OpenACC and CUDA Fortran implementations*, Journal of Supercomputing, 2016 vol: 72 (11) pp: 4160-4180, DOI 10.1007/s11227-016-1744-5
[18] https://nek5000.mcs.anl.gov.
[19] https://github.com/Nek5000
[20] https://www.excellerat.eu
[21] P. Mohanamuraly, G. Staffelbach , Hardware Locality-Aware Partitioning and Dynamic Load-Balancing of Unstructured Meshes for Large-Scale Scientific Applications PASC '20:

Proceedings of the Platform for Advanced Scientific Computing Conference, June 2020 Article No.: 7 Pages 1–10 https://doi.org/10.1145/3394277.3401851.