E-Infrastructures

H2020- INFRAEDI-2018-2020

INFRAEDI-01-2018: Pan-European High Performance Computing infrastructure and services (PRACE)

PRACE-6IP

PRACE Sixth Implementation Phase Project

Grant Agreement Number: INFRAEDI-823767

D8.4

Interim progress report: Public Software release (docs, testing, issue tracker) and integration in external codes (SSC feedback)

Final

Version: 1.0
Author(s): Fabio Affinito, Joost VandeVondele, Alex Upton
Date: 19.04.2021

Project and Deliverable Information Sheet

PRACE Project	**Project Ref. №: INFRAEDI-823767**	
	Project Title: PRACE Sixth Implementation Phase Project	
	Project Web Site: http://www.prace-project.eu	
	Deliverable ID: D8.4	
	Deliverable Nature: Report	
	Dissemination Level: PU	**Contractual Date of Delivery:** 30/04/2021
		Actual Date of Delivery:
	EC Project Officer: Leonardo Flores Añover	

Document Control Sheet

Document	**Title:** Interim progress report: Public Software release (docs, testing, issue tracker) and integration in external codes (SSC feedback)	
	ID: D8.4	
	Version: 1.0	**Status:** ***Final***
	Available at: http://www.prace-project.eu	
	Software Tool: Microsoft Word 2016	
	File(s): D8.4_v1.0	
Authorship	**Written by:**	Fabio Affinito, Joost VandeVondele, Alex Upton

	Contributors:	Fabio Affinito, Momme Allalen, Simone Bacchio, Vicenç Beltran, Marco Bettiol, Mauro Bianco, John Biddiscombe, Ricard Borrell, Fabian Bösch, David Brayford, John Brennan, Dirk Brömmel, Tomáš Brzobohatý, Mark Bull, Zahra Chitgar, Laurent Chôné, Olivier Coulaud, Tilman Dannert, Edoardo Di Napoli, Myles Doyle, Jacob Finkenrath, Christophe Geuzaine, Paul Gibbon, Luc Giraud, Aleksander Grm, Kenneth Hanley, Berk Hess, Koen Hillewaert, Victor Holanda, Guillaume Houzeaux, Luigi Iapichino, Alberto Invernizzi, Niclas Jansson, Joe Jordan, Prashanth Kanduri, Sebastian Keller, Leon Kos, Marcin Krotkiewski, Chiara Latini, Carlos Lopez, Martti Louhivuori, Georgios Markomanolis, Michele Martone, Michal Merta,Teodor Nikolov, Henrik Nortamo, Lee O'Riordan, Adam Peplinski, Janes Povh, Lara Querciagrossa, Cristóbal Samaniego, Mikael Simberg, Matthieu Simonin, Ujjwal Sinha, Raffaele Solcà, Thomas Toulorge, Alex Upton, Joost VandeVondele, Ivona Vasileska, Radim Vavřík, Jonathan Vincent, Xinzhe Wu, Shuhei Yamamoto, Artem Zhmurov
	Reviewed by:	
	Approved by:	MB/TB

Document Status Sheet

Version	Date	Status	Comments
0.1	01.04.2021	1st Draft	1st version with input from all projects
0.2	15/04/2021	2nd Draft	2nd version addressing reviewer comments
1.0	19/04/2021	Final version	Final version addressing comments from the two internal reviews

Document Keywords

Keywords:	PRACE, HPC, Research Infrastructure, Exascale, Forward-looking software solutions

Table of Contents

List of Figures

References and Applicable Documents

[1] https://bitbucket.org/lecad-peg/oopd1/src/
[2] https://bitbucket.org/lecad-peg/bit1/src
[3] https://gitlab.version.fz-juelich.de/SLPP/epoch/epoch-dev
[4] https://gitlab.version.fz-juelich.de/SLPP/epoch/test-cases
[5] https://www.score-p.org
[6] https://github.com/MoPHA/
[7] https://www.genecode.org/
[8] https://github.com/fmihpc/vlasiator/tree/openacc
[9] https://github.com/ursg/vlasiator/tree/openacc_hackathon_2020
[10] https://github.com/ursg/vlasiator/tree/openacc_taskbased
[11] https://github.com/MoPHA/sympife-vmax
[12] https://github.com/MoPHA/strugepic
[13] https://bitbucket.org/lecad-peg/simpic/
[14] https://manual.gromacs.org/documentation/2021/download.html
[15] https://gitlab.com/gromacs/gromacs
[16] https://gitlab.com/gromacs/nb-lib
[17] https://bioexcel.eu/webinar-nb-lib-a-performance-portable-library-for-computing-forces-and-energies-of-multi-particle-systems-2021-03-11/
[18] https://github.com/bsc-pm/ompss-2-releases
[19] https://github.com/bsc-pm/llvm
[20] https://github.com/bsc-pm/tampi
[21] https://github.com/ParaStation/psmpi
[22] https://github.com/Mantevo/miniAMR
[23] https://github.com/LLNL/LULESH
[24] https://github.com/Mantevo/HPCCG
[25] https://doi.org/10.1109/CLUSTER49012.2020.00017
[26] https://doi.org/10.1109/CLUSTER49012.2020.00042
[27] https://github.com/bsc-pm/tagaspi
[28] https://github.com/ludwig-cf/ludwig
[29] http://numbox.it4i.cz/
[30] https://github.com/It4innovations/espreso
[31] https://github.com/It4innovations/espreso/wiki
[32] http://www.msca-expertise.eu/
[33] http://www.netlib.org/scalapac
[34] https://elpa.mpcdf.mpg.de/software

[35] https://github.com/STEllAR-GROUP/hpx
[36] https://doi.org/10.1145/3313828
[37] https://github.com/eth-cscs/DLA-Future
[38] https://github.com/eth-cscs/DLA-interface
[39] https://chase-library.github.io/ChASE/index.html
[40] https://github.com/electronic-structure/SIRIUS
[41] http://schleife.matse.illinois.edu/
[42] http://www.flapw.de/MaX-5.0/
[43] https://www.quantum-espresso.org/
[44] https://github.com/GridTools/GHEX
[45] https://github.com/GridTools/ghexbench
[46] https://github.com/NordicHPC/hwcart
[47] https://github.com/GridTools/ghexbench
[48] https://github.com/Lyncs-API
[49] https://github.com/sy3394/DDalphaAMG/tree/multirhs
[50] http://librsb.sourceforge.net/
[51] https://github.com/michelemartone/pyrsb
[52] https://octave.sourceforge.io/sparsersb/
[53] https://gitlab.inria.fr/solverstack/fabulous
[54] https://github.com/Lyncs-API
[55] https://github.com/Lyncs-API/lyncs.setuptools
[56] https://github.com/Lyncs-API/lyncs.utils
[57] https://github.com/Lyncs-API/lyncs.cppyy
[58] https://github.com/Lyncs-API/lyncs.mpi
[59] https://github.com/Lyncs-API/lyncs.io
[60] https://github.com/Lyncs-API/lyncs.clime
[61] https://github.com/Lyncs-API/lyncs.DDalphaAMG
[62] https://github.com/Lyncs-API/lyncs.tmLQCD
[63] https://github.com/Lyncs-API/lyncs.quda
[64] https://github.com/Lyncs-API/tuneit
[65] https://dask.org/
[66] http://gitlab.inria.fr/
[67] https://gitlab.inria.fr/solverstack/spack-repo
[68] https://hpc.guix.info/
[69] https://github.com/sy3394/DDalphaAMG
[70] https://sourceforge.net/projects/librsb/files/
[71] https://savannah.gnu.org/search/?words=sparsersb&type_of_search=bugs&Search=Search&exact=1#options
[72] https://bugs.debian.org/cgi-bin/pkgreport.cgi?package=librsb-dev
[73] https://github.com/michelemartone/pyrsb
[74] https://pypi.org/project/pyrsb/
[75] https://octave.sourceforge.io/sparsersb/
[76] https://spack.readthedocs.io/en/latest/package_list.html#librsb
[77] https://gitlab.inria.fr/guix-hpc/guix-hpc/-/blob/master/lrz/librsb.scm

[78] https://github.com/Nek5000/Nek5000
[79] https://svn.cenaero.be/MAdLib/trunk
[80] https://gitlab.onelab.info/gmsh/gmsh
[81] https://gitlab.com/bsc-alya/projects/alya-ueabs
[82] https://gitlab.com/rickbp/gempa
[83] https://github.com/JuliaQX
[84] https://openhpc.github.io/cloudwg/tutorials/sc20/exercise4.html
[85] https://fosdem.org/2021/schedule/event/containerized_hpc/
[86] https://github.com/QuantumBFS/Yao.jl
[87] https://github.com/aspuru-guzik-group/tequila
[88] https://hpc.fau.de/research/tools/likwid/

List of Acronyms and Abbreviations

aisbl	Association International Sans But Lucratif (legal form of the PRACE-RI)
AMR	Adaptive-mesh refinement
BETI	Boundary element tearing and interconnecting
CoE	Centre of Excellence
CPU	Central Processing Unit
CHASE	Chebyshev Accelerated Subspace iteration eigensolver
CUDA	Compute Unified Device Architecture (NVIDIA)
DCCRG	Distributed Cartesian cell refinable grid
DoA	Description of Action (formerly known as DoW)
EC	European Commission
EuroHPC	European High-Performance Computing Joint Undertaking
FETI	Finite element tearing and interconnecting
FMM	Fast-multipole method
GASPI	Global Address Space Programming Interface
GB	Giga (= 2^{30} ~ 10^9) Bytes (= 8 bits), also GByte
Gb/s	Giga (= 10^9) bits per second, also Gbit/s
GB/s	Giga (= 10^9) Bytes (= 8 bits) per second, also GByte/s
GFlop/s	Giga (= 10^9) Floating point operations (usually in 64-bit, i.e. DP) per second, also GF/s
GHz	Giga (= 10^9) Hertz, frequency =10^9 periods or clock cycles per second
GPU	Graphic Processing Unit
HPC	High Performance Computing; Computing at a high performance level at any given time; often used synonym with Supercomputing

HPL	High Performance LINPACK
KB	Kilo (= 2^{10} ~10^3) Bytes (= 8 bits), also KByte
LINPACK	Software library for Linear Algebra
MB	Management Board (highest decision making body of the project)
MB	Mega (= 2^{20} ~ 10^6) Bytes (= 8 bits), also MByte
MB/s	Mega (= 10^6) Bytes (= 8 bits) per second, also MByte/s
MFlop/s	Mega (= 10^6) Floating point operations (usually in 64-bit, i.e. DP) per second, also MF/s
MoU	Memorandum of Understanding.
MPI	Message Passing Interface
NIH	US National Institutes of Health
PFC	Plasma-facing component
PIC	Particle-in-cell
PM	Person-month
PRACE	Partnership for Advanced Computing in Europe; Project Acronym
QCD	Quantum chromodynamics
RI	Research Infrastructure
SIMD	Single instruction multiple data
SOL	Scrape-off layer
SPMD	Single program multiple data
SSC	Scientific Steering Committee
SVD	Singular value decomposition
TAMPI	Task-aware MPI
TAGASPI	Task-aware GASPI
TB	Tera (= 2^{40} ~ 10^{12}) Bytes (= 8 bits), also TByte
TFlop/s	Tera (= 10^{12}) Floating-point operations (usually in 64-bit, i.e. DP) per second, also TF/s
Tier-0	Denotes the apex of a conceptual pyramid of HPC systems. In this context the Supercomputing Research Infrastructure would host the Tier-0 systems; national or topical HPC centres would constitute Tier-1
WP8	Work Package 8 of PRACE-6IP

List of Project Partner Acronyms

BADW-LRZ	Leibniz-Rechenzentrum der Bayerischen Akademie der Wissenschaften, Germany (3rd Party to GCS)

BILKENT	Bilkent University, Turkey (3rd Party to UHEM)
BSC	Barcelona Supercomputing Center - Centro Nacional de Supercomputacion, Spain
CaSToRC	The Computation-based Science and Technology Research Center (CaSToRC), The Cyprus Institute, Cyprus
CCSAS	Computing Centre of the Slovak Academy of Sciences, Slovakia
CEA	Commissariat à l'Energie Atomique et aux Energies Alternatives, France (3rd Party to GENCI)
CENAERO	Centre de Recherche en Aéronautique ASBL, Belgium (3rd Party to UANTWERPEN)
CESGA	Fundacion Publica Gallega Centro Tecnológico de Supercomputación de Galicia, Spain, (3rd Party to BSC)
CINECA	CINECA Consorzio Interuniversitario, Italy
CINES	Centre Informatique National de l'Enseignement Supérieur, France (3 rd Party to GENCI)
CNRS	Centre National de la Recherche Scientifique, France (3 rd Party to GENCI)
CSC	CSC Scientific Computing Ltd., Finland
CSIC	Spanish Council for Scientific Research (3rd Party to BSC)
CYFRONET	Academic Computing Centre CYFRONET AGH, Poland (3rd Party to PNSC)
DTU	Technical University of Denmark (3rd Party of UCPH)
EPCC	EPCC at The University of Edinburgh, UK
EUDAT	EUDAT OY
ETH Zurich (CSCS)	Eidgenössische Technische Hochschule Zürich – CSCS, Switzerland
GCS	Gauss Centre for Supercomputing e.V., Germany
GÉANT	GÉANT Vereniging
GENCI	Grand Equipement National de Calcul Intensif, France
GRNET	National Infrastructures for Research and Technology, Greece
ICREA	Catalan Institution for Research and Advanced Studies (3rd Party to BSC)
INRIA	Institut National de Recherche en Informatique et Automatique, France (3rd Party to GENCI)
IST-ID	Instituto Superior Técnico for Research and Development, Portugal (3rd Party to UC-LCA)
IT4I	Vysoka Skola Banska - Technicka Univerzita Ostrava, Czech Republic
IUCC	Machba - Inter University Computation Centre, Israel
JUELICH	Forschungszentrum Jülich GmbH, Germany

KIFÜ (NIIFI)	Governmental Information Technology Development Agency, Hungary
KTH	Royal Institute of Technology, Sweden (3rd Party to SNIC-UU)
KULEUVEN	Katholieke Universiteit Leuven, Belgium (3rd Party to UANTWERPEN)
LiU	Linkoping University, Sweden (3rd Party to SNIC-UU)
MPCDF	Max Planck Gesellschaft zur Förderung der Wissenschaften e.V., Germany (3rd Party to GCS)
NCSA	NATIONAL CENTRE FOR SUPERCOMPUTING APPLICATIONS, Bulgaria
NTNU	The Norwegian University of Science and Technology, Norway (3rd Party to SIGMA2)
NUI-Galway	National University of Ireland Galway, Ireland
PRACE	Partnership for Advanced Computing in Europe aisbl, Belgium
PSNC	Poznan Supercomputing and Networking Center, Poland
SDU	University of Southern Denmark (3rd Party to UCPH)
SIGMA2	UNINETT Sigma2 AS, Norway
SNIC-UU	Uppsala Universitet, Sweden
STFC	Science and Technology Facilities Council, UK (3rd Party to UEDIN)
SURF	SURF is the collaborative organisation for ICT in Dutch education and research
TASK	Politechnika Gdańska (3rd Party to PNSC)
TU Wien	Technische Universität Wien, Austria
UANTWERPEN	Universiteit Antwerpen, Belgium
UC-LCA	Universidade de Coimbra, Labotatório de Computação Avançada, Portugal
UCPH	Københavns Universitet, Denmark
UEDIN	The University of Edinburgh
UHEM	Istanbul Technical University, Ayazaga Campus, Turkey
UIBK	Universität Innsbruck, Austria (3rd Party to TU Wien)
UiO	University of Oslo, Norway (3rd Party to SIGMA2)
UL	UNIVERZA V LJUBLJANI, Slovenia
ULIEGE	Université de Liège; Belgium (3rd Party to UANTWERPEN)
U Luxembourg	University of Luxembourg
UM	Universidade do Minho, Portugal, (3rd Party to UC-LCA)
UmU	Umea University, Sweden (3rd Party to SNIC-UU)
UnivEvora	Universidade de Évora, Portugal (3rd Party to UC-LCA)
UnivPorto	Universidade do Porto, Portugal (3rd Party to UC-LCA)

UPC	Universitat Politècnica de Catalunya, Spain (3rd Party to BSC)
USTUTT-HLRS	Universitaet Stuttgart – HLRS, Germany (3rd Party to GCS)
WCSS	Politechnika Wroclawska, Poland (3rd Party to PNSC)

Executive Summary

Work Package 8 of PRACE-6IP successfully runs ten projects developing forward-looking software solutions. Eight of these projects started in April 2019, whilst two projects started in January 2020, after selection in a second call for proposals. This deliverable reports on the public release of production quality software by all projects. This deliverable follows-up on the earlier deliverable D8.3, which documented the prototype releases, their availability on GitHub and similar services, and information on the use of a modern development infrastructure, including version control, automated continuous integration (CI), and standard documentation formats. This deliverable furthermore describes the outreach the projects have done, how they interacted with European actors such as CoEs and EuroCC, and how they have integrated their efforts in community codes, providing availability of these results to users of HPC infrastructure. Each of the projects reports an outlook for pre-exascale and projections of benchmark performance. Additional developments and results require the availability of the node architectures of the European pre-exascale systems themselves and are planned for the remaining period. All projects have made good progress towards or exceeded the project goals so far, and some impressive results have been delivered. The second phase of this Work Package can be considered successful.

1 Introduction

Work Package 8 (WP8) of PRACE-6IP focuses on 'Forward-looking Software Solutions' and has the objective to deliver high quality, transversal software that addresses the challenge posed by the rapidly changing HPC pre-exascale landscape. These challenges include the diversity of hardware and software complexity. It will advance strategic and long-term projects, allowing for disruptive approaches to modernise HPC software. The main outcome is open source software in the form of libraries or significantly refactored codes. All of the projects aim to provide software solutions that enable the use of modern HPC systems, such as the planned EuroHPC pre-exascale systems.

The ten projects within WP8 have been selected based on competitive, peer reviewed calls, as reported on in deliverables D8.1 and D8.2. This includes eight projects funded from the start of PRACE-6IP, and two projects funded via a second call, with a starting date of January 2020. These projects cover a wide range of scientific domains, from fundamental topics such as tasking runtimes, halo-exchange libraries, to mathematical libraries including sparse and dense linear algebra, to application domain related software targeted at science and engineering like plasma physics, biophysics, finite elements, and fluid dynamics, or emerging domains such as quantum computing.

The ten projects work independently, following their roadmaps as presented in the project proposals. In an earlier deliverable D8.3, a report on a public prototype release of the software, as well as an update on the development infrastructure used was provided. This early release helped to ensure that software sustainability is taken in serious consideration, using industry standard tools, issue tracking, continuous integration, validation and verification, documentation, etc. This deliverable D8.4 goes one step further and documents a next stage, namely production-quality software release, with the aim of bringing this software into the hands of users for the European HPC infrastructure. It provides information on availability, and performance of the software, and the outreach that has been performed by the projects. Highlights of the results obtained include the QuantEx project that enables distributed quantum circuit simulation on HPC systems, the exascale enabling of the whole software stack required for sparse linear solvers in the LyNcs project, the demonstrated portability and performance of a library for halo exchanges on different architectures and grids (GHEX), the higher efficiency of the EPOCH code at high core numbers, and iterative eigensolvers with a performance exceeding that of the industry standard. Several more results are described in detail in the sections below. This document is structured per project, providing a brief introduction for each of them, a description of the production-quality releases made, a section on community outreach and integration, and an outlook for (pre-)exascale and benchmark projections.

2 PiCKeX- Particle Kinetic codes for Exascale plasma simulation

2.1 Introduction and summary

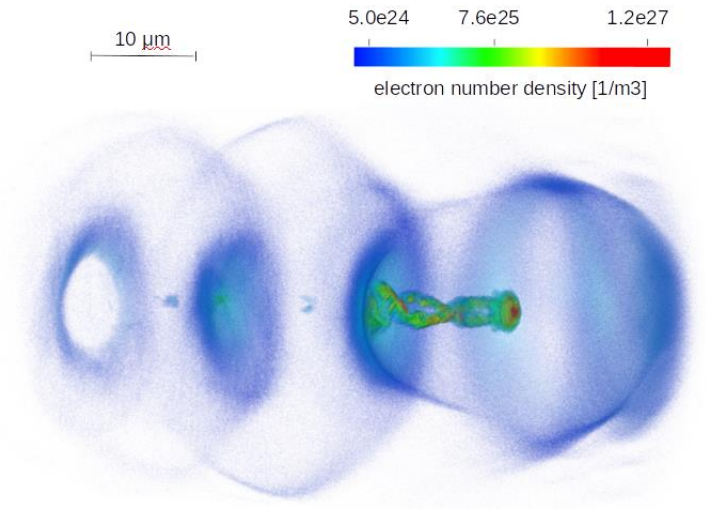

Figure 1: Compact laser-plasma particle and x-ray sources

Particle in Cell (PIC) codes have become one of the main tools for many areas in plasma physics, for example for modelling particle acceleration with high-power lasers, or to understand detailed dynamics and transport processes near the edge, the so called scrape-off layer (SOL) of magnetised plasma confinement vessels. The PicKeX project focuses on two important community codes: **EPOCH**, a fully relativistic, electromagnetic model and **BIT1/OOPD1**, a sophisticated PIC/Monte-Carlo model.

For both codes the project has enabled substantial refactoring work to be performed which would have been difficult to realise for a conventional research team utilising the code for scientific investigation. As a result, enhanced versions of both codes are now publicly available for rigorous testing by user groups. In particular, this includes OOPD1, a new GPU version of BIT1. The new version of EPOCH incorporates a significantly faster moving window algorithm, which is extensively used for an important class of problems based on laser-based particle accelerator schemes.

2.2 Production software release

The PIC codes OOPD1 and BIT1 have complex structures consisting of many interdependent sources that also include hard-coded atomic physics (cross-sections, particle collisions, etc.). Both use a particle mover and field solver as main algorithms and additionally a Monte Carlo algorithm for particle collisions. They share the same roots while OOPD1 is an object-oriented rewrite of PD1 from which BIT1 enhanced the physics part. The refactored version of OOPD1 and BIT1 have the following changes:

- Based on the work done on the prototype SIMPIC code for a full CPU to GPU port, a GPU version of the particle mover algorithm of OOPD1 using a global kernel was implemented.
- The extensive refactoring of OOPD1 for better modularisation (delineating algorithms from data) including the GPU particle mover is available under [1] in the 'feature/refactoring' branch.
- The refactored BIT1 code builds upon the same ideas as the OOPD1 refactoring under the refactoring branch on [2]. The BIT1 code is intended to be a basis for a OOPD1 physics upgrade and not a final product and therefore at this stage the BIT1 repository is not public.

The test environment for OOPD1 and BIT1 codes is based on numerous simulation cases inside OOPD1 that can be matched to BIT1 input format with the same physics. Enhanced physics and data parts from BIT1 require OOPD1 input update along with upgraded physics refactored from BIT1.

The refactored version of EPOCH contains the following major changes:

- The moving window code has been changed to be invoked after the window has traversed a minimum number of cells instead of for individual, single cells.
- The MPI communication to exchange the field boundaries has been restructured to reduce the number of MPI_SendRecv calls by a factor of 60.

These two changes together have reduced the overall amount of time spent in the moving window by 90%. The refactored version of EPOCH is available as the default branch at [3].

The test environment for EPOCH includes a number of test cases in 2D and 3D that have been created from community input to represent a number of common use cases of the code. They also include pure benchmarking cases to focus on specific aspects. All of those can be run from JUBE. The JUBE environment and test cases used for the EPOCH development are available at [4].

2.3 Community outreach and integration

The OOPD1 is an open source PIC/MC code, which is used for plasma tokamak simulations. Since we are working directly with the PIC/MC codes, its refactoring and GPU improvement, will directly benefit all people who are using these codes. As an example, more than 2 months are required for reaching the steady state of the plasma using a CPU version of the codes. With this implementation and providing a GPU version of the PIC codes we will reduce the time needed for plasma simulations.

EPOCH is an open source plasma physics simulation code with over 1200 registered users worldwide and has been extensively used for high energy density physics (HEDP), LWFA, and plasma astrophysics studies. To date it has not yet been extensively used within the PRACE framework, but does commonly feature in national Tier1 compute time allocations. The refactored version is already being field tested within local projects at FZJ and external users have been directly approached to provide feedback. The Warwick developer team is expected to test and integrate the enhancements into the production version before the end of the project lifetime.

2.4 Outlook for pre-exascale and benchmark projections

The original proposal foresaw the implementation of a task-based programming model for both community codes. For the BIT1 application this has been partially achieved through a StarPU version of the reduced physics mini-app SIMPIC (see Section 3.2), which will serve as a template for future optimisation of the BIT1 framework. In the PicKeX project however, priority was given to creating a GPU-enabled version OOPD1, which has been successfully implemented for the computationally intensive particle mover component, and a GPU version of the field solver is currently undergoing testing. Currently the BIT1/OOPD1 code is ported to GPU to about 50%. To see how fast the code works, a comparison between a CPU OOPD1 code and partly GPU OOPD1 was done (Figure 2 below). In this benchmark the same test case was used. The OOPD1 which has GPU parts is approximately 30% faster than the purely CPU version. By the end of the project a complete GPU version of the BIT1/OOPD1 will be made available.

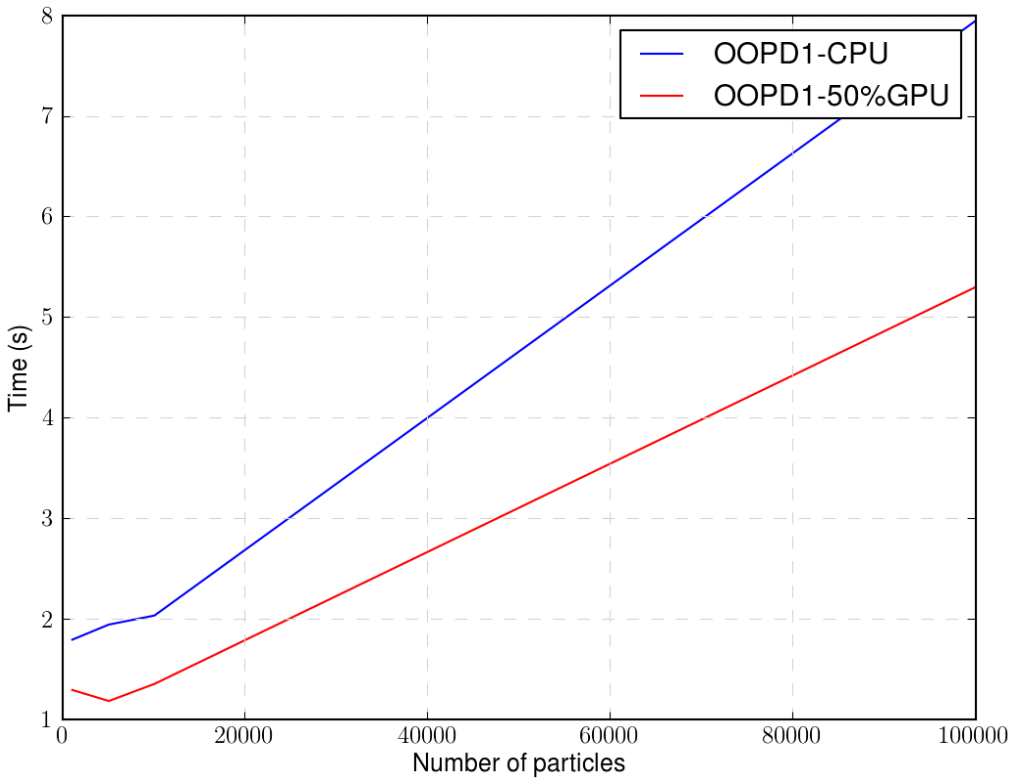

Figure 2: OOPD1 CPU and OOPD1 50% GPU comparison with ngrid=10000, dtfactor= 0.001, run for 200 timesteps

Concerning EPOCH, an early strategic decision was made to leave the data structures intact in order to maintain compatibility with other development branches and facilitate integration of the performance enhancements with the main production version. A rigorous analysis of EPOCH using the Score-P toolset [5] allowed two major bottlenecks to be identified: i) the 'moving window' algorithm, in which the entire grid is rezoned to follow the dynamics of a plasma wake just behind a laser pulse; and ii) the particle pusher with its integrated current deposition. Both of these components consume a large fraction of the simulation time and represent good candidates for improvement. A comparison between the refactored version and current production version shows the new version to be 30-40% faster at high core numbers for LWFA studies (see Figure 3), which

should lead to significant savings in production runs. Further reduction of MPI redundancies and full realisation of vectorisation and multi-threading potential should lead to an improvement of overall scalability to 10^4 - 10^5 cores on currently available Tier-0 machines.

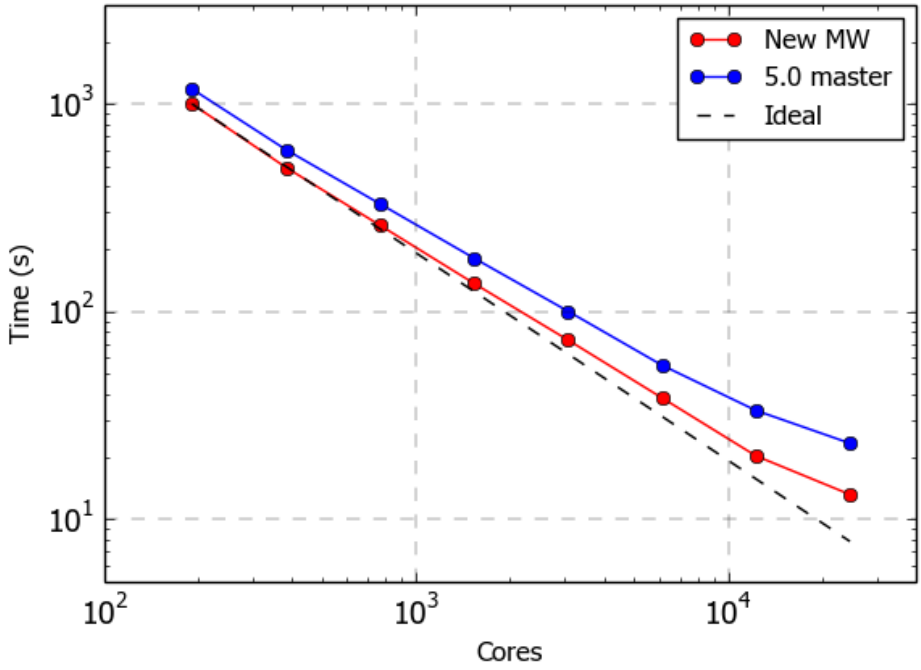

Figure 3: Strong scaling comparison between EPOCH refactored version and the latest official version. The benchmark used a 2D moving window with 5 million cells and 10 particles/cell

3 MoPHA – Modernisation of Plasma Physics Simulation Codes for Heterogeneous Exascale Architectures

3.1 Introduction and summary

Code modernisation efforts are needed for many scientific software applications to fully benefit from the upcoming heterogeneous exascale systems. This is true also for plasma simulation codes, such as ELMFIRE, GENE, and Vlasiator. Task-based parallelism potentially offers better scalability and portability than traditional approaches by abstracting hardware-specific optimisations away from the scientific algorithms. Some frameworks, such as StarPU or AMReX, even offer a relatively easy way to achieve both task-based parallelism and support for GPUs.

In the MoPHA project, we have explored task-based parallelism for plasma simulations and tested ways to add support for GPUs or other accelerators to plasma simulation codes. The aim has been to pave the way for the plasma simulations codes to be ready for the upcoming pre-exascale systems.

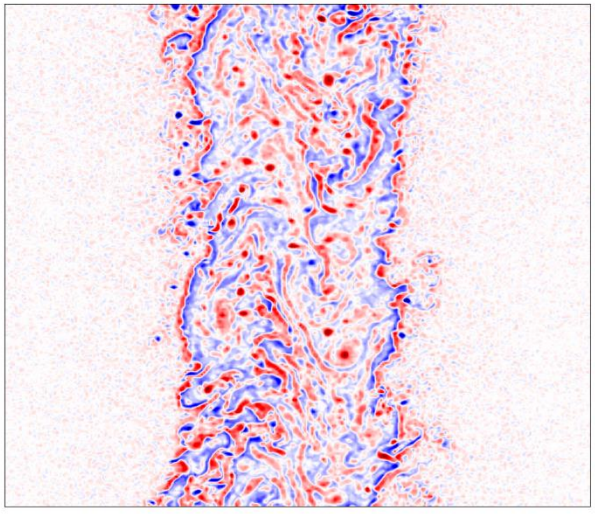

Figure 4: Turbulent flow in a fusion plasma simulation

3.2 Production software release

In the MoPHA project we have developed a number of different codes, some of which are hosted in their own repositories, but the main site for publishing prototype mini-apps and documentation related to the project is on GitHub [6]

GENE: Experimental GENE version with partial support for task-based parallelism with StarPU. The main goal to achieve in this code version is the taskification of the computationally intensive parts, such as the right-hand side in the Vlasov equations and the solution of the fields, with StarPU.

At this point, we provide an implementation that computes the right-hand side of a time step where we partially exploit the inherent task-based parallelism of the application using StarPU for a specific test scenario. The computation of a time step consists of 4 stages of a Runge-Kutta method

and in each stage several terms of the right-hand side are computed. The individual terms are computed through a cache-optimised block loop and were taskified so that the execution of these individual tasks can overlap, provided that the data dependencies allow it. Additional parallelism was added in the calculation by grouping terms and using local buffers which are sum reduced at the end of the computation to form the actual RHS. In an earlier version, the execution of the tasks was done only on the CPU (on a single node, one MPI rank per node). In the current version, it is possible to compute some of the terms on the GPU thanks to the integration, refactoring and code reuse from the GENE's GPU branch. GENE can be accessed via [7]

VLASIATOR: Experimental Vlasiator version with partial support for GPUs using OpenACC directives. Implemented a set of the main computational algorithms for offloading to the GPUs using a directive based approach, focusing on the semiLag solver in the velocity space acceleration update.

Initial results are promising, but further improvements are needed to optimise data movement between host and device memory. For this purpose, two improved prototype versions were developed for comparison, one based on managed memory to alleviate the concerns with the data transfers and one based on a tasks concept, where each OpenMP thread creates an asynchronous queue and sends OpenACC calls, to partially overlap computation and communication. Initial results are promising, but some concerns remain, e.g. with load-balance and the fact that the CPU cannot send enough requests to fully use the GPU. Further improvements and changes in the data structures are still needed.

The code is available as a separate branch in the main Vlasiator git repository: [8]. The managed memory version is available at: [9]. The task-based version is available at: [10]

SYMPIFE-VMAX/ELMFIRE: Mini-app for particle-in-finite-elements Vlasov-Maxwell systems with multiple species. The mini-app serves as a basis for the refactoring of the ELMFIRE code. The prototype code uses the MFEM finite elements framework from which it leverages versatile mesh-handling and refining infrastructure, and arbitrary order mixed-elements spaces. The prototype implements symplectic integrators of order 1, 2 and 4 based on Lie-Trotter splitting for the VM system. The MFEM infrastructure allows the use of complex meshes and automatic domain decomposition, as well as hybrid parallelism [11]

STRUGEPIC/ELMFIRE: Mini-app for structure preserving PIC simulations using AMReX. It demonstrates the use of the scalable framework AMReX for creating PIC plasma simulations. The main purpose of the mini-app is to serve as an example of features and functionality provided by AMReX for plasma simulations. In addition, it also performs well enough that it can be used for proper plasma simulations by itself. Together with the SYMPIFE-VMAX mini-app, it serves as a basis for the refactoring of the ELMFIRE code [12]

SIMPIC: Mini-app for simple PIC simulations using StarPU. It demonstrates the use of the StarPU framework for task-based parallelism in plasma simulations. Due to its general applicability, SIMPIC can serve as a how-to guide for other PIC codes.

At this stage, we provide a full GPU version of the SIMPIC mini-app, using CUDA. The code has two main algorithms, the first is a particle mover and the second a field solver. To make a CPU to GPU transfer for the particle mover and field solver, a moverparticlesGPU() function with global_kernel and a fieldsolverGPU() function with cuSPARSE solver were created. In addition, benchmarking on different cluster architectures and integration with the StarPU simulator was done. For better visualisation of the memory transfer between the host and the device, profiling

and tracing were done with the TAU and NVIDIA profilers. The code can also serve as base for CPU to GPU transfer in a more complex PIC code. In this version the code works only with one GPU, but the idea is that the last version of this mini-app will work on multiple GPUs.

The GPU version and documentation of the profiling and tracing results are available at the SIMPIC repository [13]

3.3 Community outreach and integration

GENE: GENE is a well-established open source plasma micro turbulence code that is widely used by the community of plasma physics. Since we are working directly with the GENE code, its modernisation and any performance improvement achieved by the task-based parallelisation with StarPU and the use of heterogeneous hardware, will directly benefit all the members of the large and growing user base of GENE around the world. As an example of code integration, we would like to mention the Whole Device Modeling application (WDMApp) of the Exascale Computing Project (ECP), of which GENE is part and is coupled to another gyrokinetic code for the simulation of the edge of a magnetic confinement device. Transfer of knowledge is also continuously done in our monthly meetings and slack channel.

VLASIATOR: Vlasiator simulates the Earth's magnetosphere in kinetic physics. It solves the Vlasov equation (6D advection equation) with a sparse phase space simulation and implements semi-Lagrangian advection solvers coupled with a constrained transport field solver. It has proven to be really accurate and is a well-known tool that has also provided the first full global 6D hybrid-Vlasov simulation of the magnetosphere. In light of the upcoming pre-exascale European HPC systems, porting of some of the kernels to GPUs is an active line of development. The Vlasiator team organises and participates in hackathons and presents their science in various conferences. All the efforts are public and are actively communicated to the whole community. All code developed in this project is publicly available as separate branches in the main code repository of Vlasiator or its main developer's code repository.

SYMPIFE-VMAX/STRUGEPIC/ELMFIRE: STRUGEPIC and SYMPIFE-VMAX are both being used by the ELMFIRE research group at Aalto University for prototyping new structure-preserving algorithms for the full and reduced Vlasov-Maxwell-Landau systems. Their use is planned within the EUROfusion Theory, Simulation, Verification and Validation (TSVV) activity for probing the limits of gyrokinetic simulations when the gyrokinetic ordering may be violated. Both these activities are part of a collaboration with the Numerical Methods for Plasma Physics division at the Max-Planck Institute for Plasma Physics. Development of space-physics relevant features is also ongoing in collaboration with the Esa Kallio group at Aalto University.

SIMPIC: SIMPIC is an open source prototype PIC code, which contains only two algorithms without any cross sections and particle collisions. For this reason, the GPU version of the code was implemented in a more complex PIC code. With this transfer we speed up the code and we reduce the memory transfer. This mini-app helped to reduce the time needed for plasma simulations in the other PIC codes. At this stage of the project, most of the work has been done. All updates of the code were disseminated in our monthly meetings.

3.4 Outlook for pre-exascale and benchmark projections

GENE: Although there is still work to do and issues to solve, results show overlaps in the execution of tasks on homogeneous and heterogeneous hardware (CPU and GPU). This gives us the confidence that we are on the right path towards an implementation that efficiently uses all the available computing resources. The objective of MoPHA for GENE is to modernise, modularise, refactor, and implement a task-based parallelisation of GENE as well as to transfer any knowledge gained in the process. Cross-review of the code structures and modularisation of GENE has been done. Development of simplified test code, test of StarPU with test code and test of StarPU in the GENE code were also completed. At this stage of the project, we are implementing computational kernels using StarPU, which in some cases require further refactoring of the modules.

We expect that the code will perform better/faster than the current version of GENE in pre-exascale systems for problems with large computationally intensive kernels, for example nonlinear problems. In these cases, the tasks are large enough that they can be broken into smaller tasks to be distributed for execution across the available hardware but still large enough so that the task scheduling and StarPU internals can be neglected. For problems with small size tasks, the scheduling and StarPU internal operations add up to the entire runtime and make the solution of the problem often slower. From our initial results, we can infer that the code will be able to use all of the computational resources. However, additional work is still necessary to investigate the different schedulers and how to further break or merge the available tasks so that the schedulers can make an optimal distribution of the tasks to the hardware.

VLASIATOR: Although there is still work to be done on the GPU version, the CPU code scales to hundreds of thousands of CPUs. Since the code has potential to scale really well, we expect that with a proper porting of the code to GPUs, it will scale also really well to a large number of GPUs. Of course, there are many challenges and further improvements are still needed e.g. to optimise some of the data structures and to verify which programming model is the most efficient and portable.

SYMPIFE-VMAX/STRUGEPIC/ELMFIRE: SYMPIFE-VMAX is GPU-ready (both CUDA and HIP through the MFEM backends), but scaling tests and benchmarks on multiple GPUs are necessary. A multi-GPU CUDA version of the STRUGEPIC has also been developed. Further scaling tests and optimisation is required for scale-up on many nodes. Porting to HIP is desirable in the future, but HIP support in AMReX is only experimental at the moment. The physics benchmarks and further optimisation will be pursued in particular within the EUROfusion TSVV activity, and with support from the Advanced Computing Hubs of the EUROfusion Theory and Advanced Simulation Coordination.

SIMPIC: Benchmarking of the code with a single GPU was done and looks promising, but after the final version of the code with multi-GPU support is ready, thorough benchmarking should be done. Based on the results from implementing this mini-app into other PIC codes such as BIT1 in the PiCKeX project, it is evident that that our code is already fully workable. Also a StarPU version of the code was done and was tested on different HPC clusters.

4 NB-LIB: Performance portable library for N-body force calculations at the Exascale

4.1 Introduction and summary

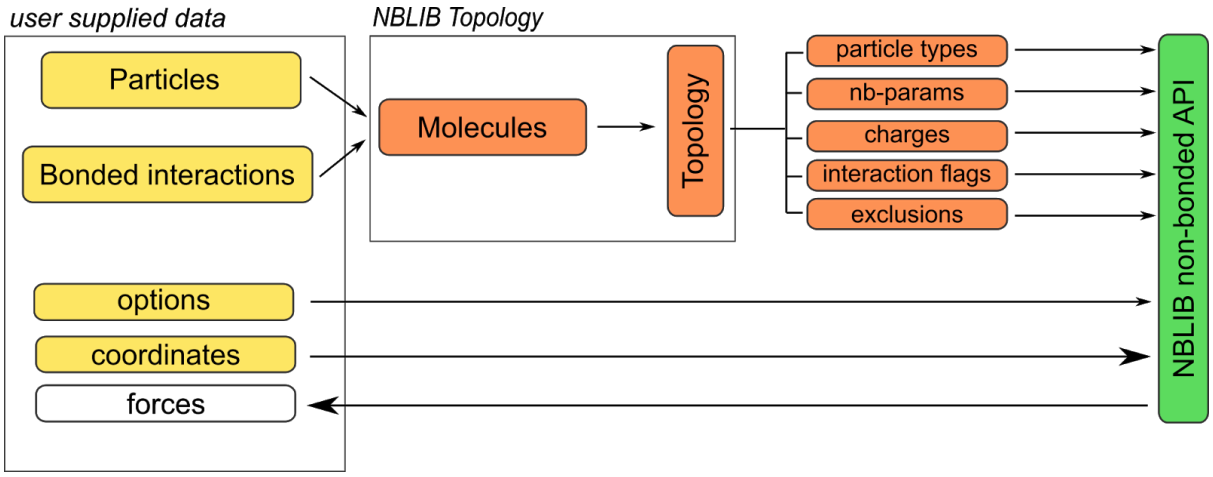

Figure 5: Overview of NB-LIB

A large number of scientific applications use particle interactions (e.g. Molecular Dynamics, Monte Carlo or multiscale simulations in life sciences or materials), and several smaller codes or combinations of codes have unique features that allow novel workflows. However, while computers have become more specialised, many codes are not optimised for GPUs or other accelerators and it is increasingly hard to achieve parallelisation. This will make these codes increasingly difficult to use on next-generation, exascale systems.

One of those codes currently undergoing exascale optimisation efforts is GROMACS, also among the benchmark codes for pre-exascale machines coming online in 2020. While it has a long track record as a widely used and highly performant HPC code, it is very difficult to offer in a single application all the unique features and niche use-cases that the various many-body codes combined support. The goal of the NonBonded-LIBrary (NB-LIB) is therefore to make the cutting-edge performance of GROMACS available through a high-level C++ API to its non-bonded force kernels. In combination with the system setup functionality that NB-LIB offers in addition, users will then be able to implement arbitrary workflows that might be required for their special use case while leveraging the performance of GROMACS for the force calculations. This way, future acceleration, porting, and library features will benefit all applications.

4.2 Production software release

The NB-LIB project is bundled with the public release of GROMACS 2021. This means that NB-LIB is installed by default on HPC resources all over the world because GROMACS is one of the most popular scientific codes in the HPC landscape. There are a number of ways for users to get access to the bundled version of NB-LIB included in the GROMACS 2021 release, including through the public GROMACS documentation [14] as well as via the GROMACS GitHub page

[15]. Additionally, for users who desire access to the most up-to-date version of NB-LIB, it is also possible to access via the NB-LIB GitHub page [16].

The NB-LIB repository includes extensive documentation as well as sample scripts that demonstrate library usage. Though NB-LIB is written in state of the art C++, care has been taken to make it simple for end users to write scripts using NB-LIB, allowing sample scripts that read like Python code.

The core functionality of NB-LIB is to allow performant nonbonded calculations, and this is the central feature of the public release. Nonbonded forces can be calculated on both CPU and GPU, and since GROMACS has backends for CUDA, OpenCL, and SYCL, users with diverse hardware can expect good performance. In addition to the core nonbonded force calculation, NB-LIB provides a molecular topology API allowing users to easily construct the input data for NB-LIB calculations. However, since the public methods of NB-LIB only take elementary types, users can call NB-LIB with data from other sources as well. Finally, NB-LIB also provides a listed forces API allowing calculation of bond, angle, dihedral, etc. forces, as well as a simple integrator. This means that users can write a complete MD code using NB-LIB.

4.3 Community outreach and integration

The NB-LIB project is closely aligned with a number of longer term development goals of the GROMACS project, which is currently funded to a large extent by the BioExcel CoE. This has meant that there has been a near-constant dialogue between core GROMACS developers and NB-LIB developers. This dialogue, combined with the rigorous code review standards for new code being accepted into GROMACS has helped ensure that NB-LIB has high code quality standards in the GROMACS 2021 release of which NB-LIB is a part. The NB-LIB topology specification, and the corresponding system setup API were designed after discussions with members of the OpenMM and OpenFF consortiums. This means that NB-LIB should in principle be compatible with those popular molecular simulation codes. Discussions have also happened with the core developers of the HADDOCK docking code, and some work is ongoing to enable the types of use cases needed for drug docking workflows. Given that this is not a core goal of NB-LIB; it is not certain that all functionality needed for such workflows would be completed before the project end.

One very successful outreach effort was a recent webinar [17], hosted by the BioExcel CoE. This had an audience of more than 40 participants from all over the world. Many participants had questions about how to enable the workflows that would be meaningful to them, and discussions are ongoing to see how and if effort can be prioritised to help meet any of these use cases.

4.4 Outlook for pre-exascale and benchmark projections

Overall, NB-LIB has exceeded many of the project goals. The primary goal of NB-LIB is to expose a performant API for non-bonded force calculation. Since NB-LIB relies on and exposes the known performance of the GROMACS non-bonded force calculation engine, this goal can reasonably be said to have been achieved. Additionally, NB-LIB provides a convenient API for system setup, a force calculator for bonds, angles, etc., and a simple integrator. This means that users wanting good performance in particle simulations can potentially use NB-LIB for entire projects, as NB-LIB can be used to write molecular simulation mini-apps. Currently, to run NB-LIB across multiple nodes, for instance using MPI, the user would need to manage all data transfer themselves. Work is ongoing to expose the GROMACS domain decomposition code in order to reduce this data

management burden on users. Unlike the achieved objectives of NB-LIB, which mostly entailed designing and implementing APIs and glue code, the work on domain decomposition requires significant refactoring of core GROMACS functionalities. This means that there is some chance that at the completion of the current NB-LIB project, it may still be up to users to manage the complexity of domain decomposition, but significant effort is being put into the necessary GROMACS refactoring that would allow exposing a domain decomposition API.

5 LoSync – Synchronisation reducing programming techniques and runtime support

5.1 Introduction and summary

The LoSync project aims to improve the scalability of applications by removing unnecessary synchronisation and serialisation, and by fully exploiting the potential for overlapping computations and communications. To do this, we make use of modern features of well-standardised APIs, to ensure portability and relevance.

Efficiently implementing a pure task-based programming on distributed memory systems is very challenging. Instead, we propose a hybrid model which uses task-based programming inside a node and traditional message-passing between nodes. To minimise synchronisation and expose as much parallelism as possible, our experience has shown that communications as well as computations should be expressed as tasks. However, standard communication libraries are difficult to use like this without encountering the risk of deadlocks, for example where all threads are executing tasks containing blocking communication calls. In LoSync we are developing and evaluating task-aware versions of MPI and GASPI libraries (called TAMPI and TAGASPI) which are integrated with OmpSs and OpenMP task-based runtimes. In TAMPI and TAGASPI, tasks blocked on communication calls are paused, freeing their executing threads to process other tasks, until the communications complete and the paused tasks can be resumed.

Figure 6 below shows the state transition diagram for tasks in this model. Whenever a running task is blocked in MPI (for example), its status is changed to paused, and the executing thread is released to be able to execute other tasks (either computation or communication). When the blocking MPI call completes, the task becomes ready again, and can be rescheduled for execution when resources are available. TAMPI also supports another mode which allows the task to complete but only release its dependencies when the MPI call completes: this mode is easier to integrate with existing OpenMP implementations, but requires extensions to the MPI interface.

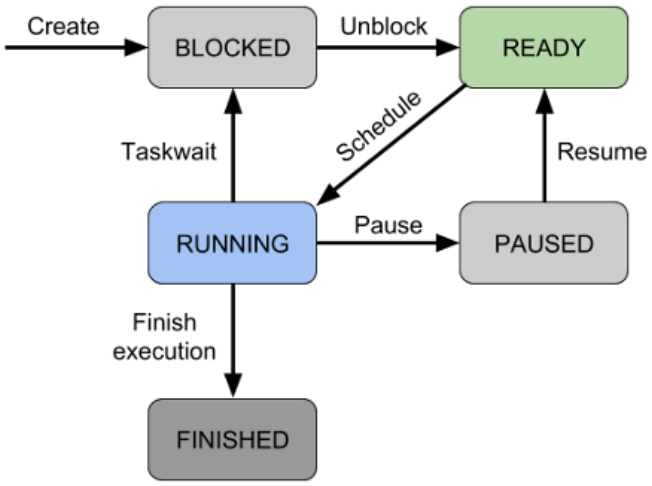

Figure 6: State transition diagram for tasks (blocking mode)

5.2 Production software release

The OmpSs-2 programming model [18] is released twice a year coinciding with ISC and SC events. In the latest release (20.11) we have included our new LLVM-based OmpSs-2 compiler, a lightweight instrumentation system based on CTF format and a user-friendly configuration file to manage all runtime options. It is worth noting that our LLVM distribution [19] also includes a modified OpenMP LLVM runtime that can be used with TAMPI and TAGASPI libraries.

The Task-Aware MPI (TAMPI) library is also publicly available [20] and periodically updated with new features, performance improvements and bug-fixes. Since the last reporting period, we have extended TAMPI to support MPI one-sided operations. Now it is possible to safely use MPI_Get, MPI_Put and MPI_Win_Fence operations inside tasks. To implement this new feature in TAMPI we have first extended ParaStationMPI [21] with a new MPI_Win_Ifence operation, that is the non-blocking counterpart of the standard MPI_Win_Fence. TAMPI leverages this non-blocking MPI_Win_Ifence to implement the support of one-sided MPI operations.

The work on applications has also progressed since the last report. We have ported, evaluated and analysed three well-known mini-apps that combine OmpSs-2/OpenMP and TAMPI: miniAMR [22], Lulesh [23] and HPCCG [24], and the results have been included in two research papers [25], [26].

The work on the Task-Aware GASPI (TAGASPI) library, which leverage GASPI one-sided operations and notifications, has been finished and a complete evaluation using the miniAMR, Gauss-Seidel, IFSKer and HPCCG benchmarks will be submitted to the SC'21 conference. The initial version of this library is publicly available on GitHub [27].

5.3 Community outreach and integration

BSC has developed training material which covers the OmpSs + TAMPI programming model combination, including hands-on programming exercises. This material forms part of the "Heterogeneous Programming on GPUs with MPI + OmpSs" course, which was run most recently as part of the PATC training programme in March 2021. A basic guide to TAMPI is available online [20] and we will continue to develop a more comprehensive best-practice guide to this style of hybrid programming.

In the next phase of the project, we plan to implement a larger application code using the OmpSs + TAMPI - we are currently evaluating Ludwig [28], a Lattice-Boltzmann simulation code as the target for this. There are some potential synergies with other WP8 projects, such as NB-LIB and MoPHA which we will explore in the near future.

5.4 Outlook for pre-exascale and benchmark projections

So far in the LoSync project we have shown, by implementing a range of mini-apps from a variety of computational science domains, that the model of using tasks within a node and conventional communication (e.g. with MPI) between nodes can be very successful. The ability to wrap the communication calls inside tasks has been demonstrated to be very valuable. As well as avoiding unnecessary serialisation, it provides a much more flexible and effective mechanism for overlapping computation and communication than can be achieved with standard non-blocking MPI calls. This is because it allows any available task to be executed while waiting for the

communication to complete, and the polling for completion can be undertaken by threads inside the runtime.

Getting the granularity of tasks right is critical for good performance. If we have too few large tasks, cores will be left idle. If we have too many small tasks, the overheads of scheduling and managing task dependencies may become a bottleneck. With more complex applications we can have many different types of task, each with their own "natural" granularity. However, it is important to be able to chunk up small tasks into larger ones. This can lead to a significant number of tuneable parameters (i.e. the grain sizes for each type for task) which results in an optimisation problem which can be hard to solve. Some heuristic methods have been developed to deal with this, but a more comprehensive mechanism might be beneficial. Another technique which has been explored with some promising results is the use of parallel loop tasks: a set of independent loop iterations can be treated as a single task for dependency analysis and scheduling purposes, but the iterations can be executed in parallel by multiple threads.

For applications with fairly regular data access patterns, converting from a conventional MPI data decomposition, and/or statically scheduled parallel loops, to tasks can result in a loss of locality. This can be addressed, to some extent, by the runtime by preferring to schedule dependent tasks on the same core as their parent, but the runtime may lack sufficient information about the amount of data movement implied by the dependencies. Further work is required to address this limitation.

Some applications have a clear and obvious correspondence between data structures and computational tasks, while others do not. In the latter case, this can lead to synchronisation points where all tasks acting on a shared data structure must be completed before any subsequent tasks can run. In molecular dynamics, for example, we may find that all force updates to particles must be completed before any velocity and position updates may occur. A technique to overcome this is to apply data decomposition method inside a node – this may be simply the existing decomposition method already used between processes in the application, or it may exploit additional levels of parallelism. Communication between decomposed data structures becomes explicit (as read/writes), and this enables the synchronisation to be relaxed.

6 FEM/BEM based domain decomposition solvers

6.1 Introduction and summary

The aim of the project is to extend the existing domain decomposition library ESPRESO [29] to support highly scalable efficient solution of harmonic analysis and sound scattering problems. The distributed memory parallelisation of the code is based on the FETI method and its variants.

In order to achieve this, a number of activities are on-going. These include the development of suitable preconditioners for the parallel harmonic analysis solver – currently, three types of preconditioners are implemented in the code, each suitable for different kinds of problems, as the preconditioners differ in the way that they construct an artificial coarse space for the FETI method.

In addition, GPU acceleration of computationally intensive parts of the code has been undertaken, and tested using NVIDIA V100 and A100 GPUs. Work has also continued on the Solver as a Service platform, with the ultimate goal to enable users without a HPC background to use the application. Further improvements to the harmonic analysis solver have also continued, which for example now also includes the pre-stress harmonic analysis in rotor dynamics solver.

Coupled to this, testing and documentation is on-going, with continuous updates to the documentation in project repository. Finally, development has also begun on the module for the parallel solution of acoustic problems.

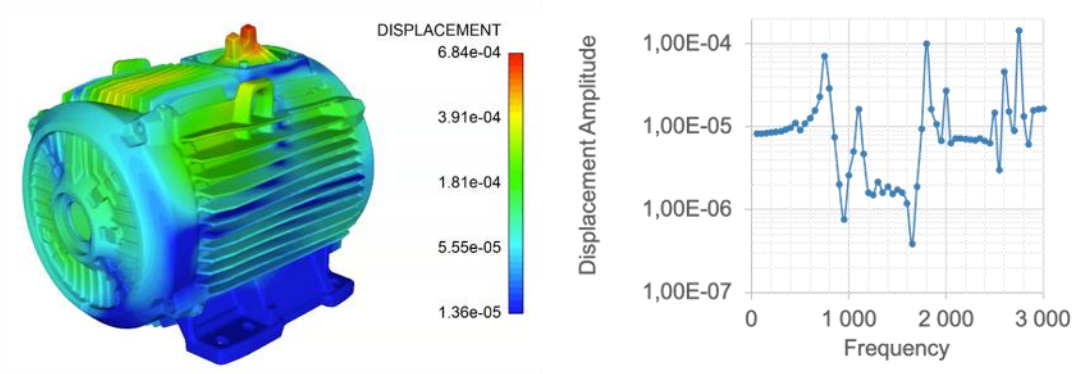

Figure 7: Frequency response of the electric motor case computed using 450 nodes of the Salomon cluster at IT4Innovations in 714 s (15 million degrees of freedom, 60 frequency samples).

6.2 Production software release

The developed code is publicly available within the master branch of the ESPRESO library in its official repository at GitHub [30]. Most of the functionalities described in the previous section are available in the public code. Currently, the code supports solution of the harmonic analysis problems in real domain parallelised across both frequency and spatial domains. Three preconditioners based on an artificial coarse space for the FETI method are available. The code in the master branch now supports the GPU acceleration of the computationally intensive sections of the code. Moreover, besides the installation instructions and generic information about the solver,

the manual available in the GitHub repository now contains solver-specific documentation with examples of usage of the harmonic analysis solver (see the Wiki section of the repository [31]).

6.3 Community outreach and integration infrastructure

The developed ESPRESO library is being integrated into the Solver as a Service online platform at IT4Innovations. The goal is to make the supercomputing resources available to engineers without HPC background. The user will be able to submit a job via web interface either by directly uploading the ESPRESO configuration file or by using an online GUI to specify the problem to be solved. The front-end of the service is now implemented, while development of the back-end is currently in progress.

The above-described developed parallel solver for harmonic analysis has already been used in two significant projects. First of them is focused on the development of the digital twin of an electromotor in cooperation with the Siemens company. The second project (EXPERTIZE [32]) aims to create a European training network that trains the experts in nonlinear structural mechanics of turbomachinery and high-performance computing.

Moreover, the ESPRESO library provides an API that can be used to call the solver from external software. In the past, it has been used, e.g. within the Elmer library.

6.4 Outlook for pre-exascale and benchmark projections

Although the goal of the project was to develop a highly scalable solver for both harmonic analysis and sound scattering problems, due to the initial problems with the efficiency of the preconditioners for the harmonic analysis, the development of the sound scattering module has been delayed. Therefore, the currently available benchmarks only describe the scalability of the harmonic analysis module. From the implementation point of view, we have been able to achieve most of the goals stated in the project proposal. The structure of the ESPRESO library has been refactored and several parts of the code, e.g. system matrix assembly, have been optimised and significantly accelerated. Due to the heterogeneous design of current and future pre-exascale machines, a GPU acceleration is also a crucial and unavoidable part of modern code. We have successfully accelerated the most time-consuming parts of the code and tested the acceleration using the modern NVIDIA V100 and A100 architectures (see Figure 8 below). For the distributed memory parallelisation, we use the combination of parallelisation in spatial and frequency domain, which enables us to further extend the scalability. So far, the code has been tested on up to 450 nodes (10800 cores) of the Salomon cluster solving a problem with 15 million spatial degrees of freedom and 60 frequencies. Further tests are planned and the ESPRESO team has submitted a proposal for a preliminary access to the LUMI supercomputer for extended testing. Furthermore, a PRACE Preparatory Access project will be submitted to further optimise and benchmark the code on large machines.

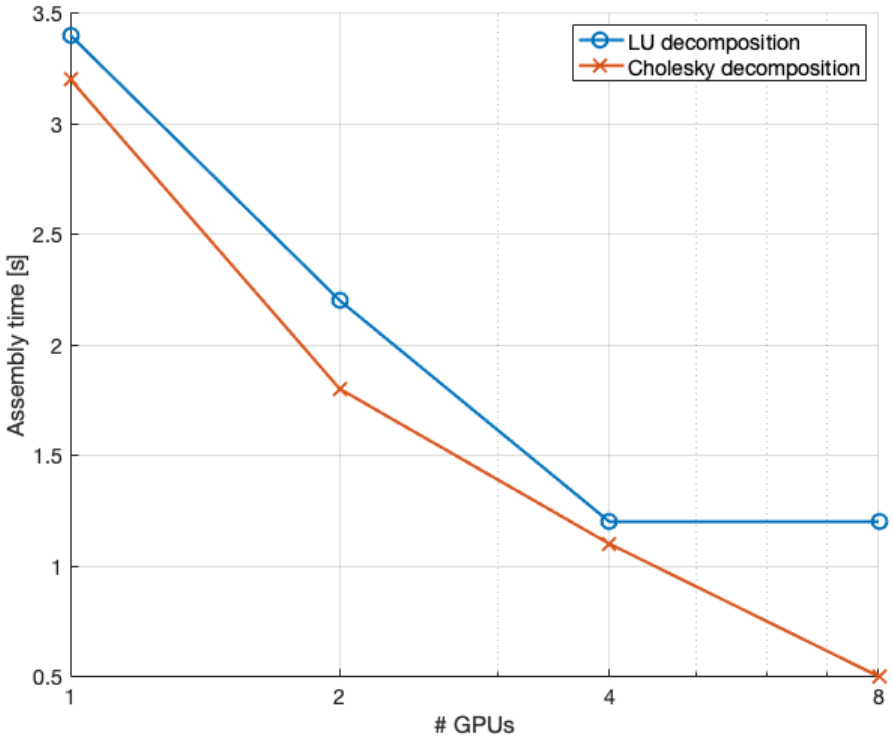

Figure 8: Scalability of the multi-GPU assembly of Schur complement matrices on NVIDIA DGX A100 machine.

7 Performance portable linear algebra

7.1 Introduction and summary

Linear algebra algorithms play a central role in scientific applications, for example, in the particular case of Materials Science, many applications rely heavily on linear algebra to solve complex tasks. Overall, the diversity of linear algebra operations together with the large size of the operands motivates the necessity of high-performance implementations of distributed algorithms. For example, modern electronic structure methods rely on the Density Functional Theory (DFT) method, which highly depends on the solution of either dense or sparse eigenvalue problems.

To solve the dense eigenvalue problems, applications are mainly using the ScaLAPACK [33] or ELPA [34] libraries. ScaLAPACK is the de-facto standard for distributed dense linear algebra since 1992. However, the fork-join approach used for its implementation is not suitable for modern node architectures.

A more modern approach consists in the task based implementation of the algorithms and the goal of this project is to deliver a modern and efficient distributed linear algebra package (DLA-Future) based on HPX [35] (a C++ tasking library which tests the proposals of C++ standard about tasks), that can replace ScaLAPACK in scientific applications.

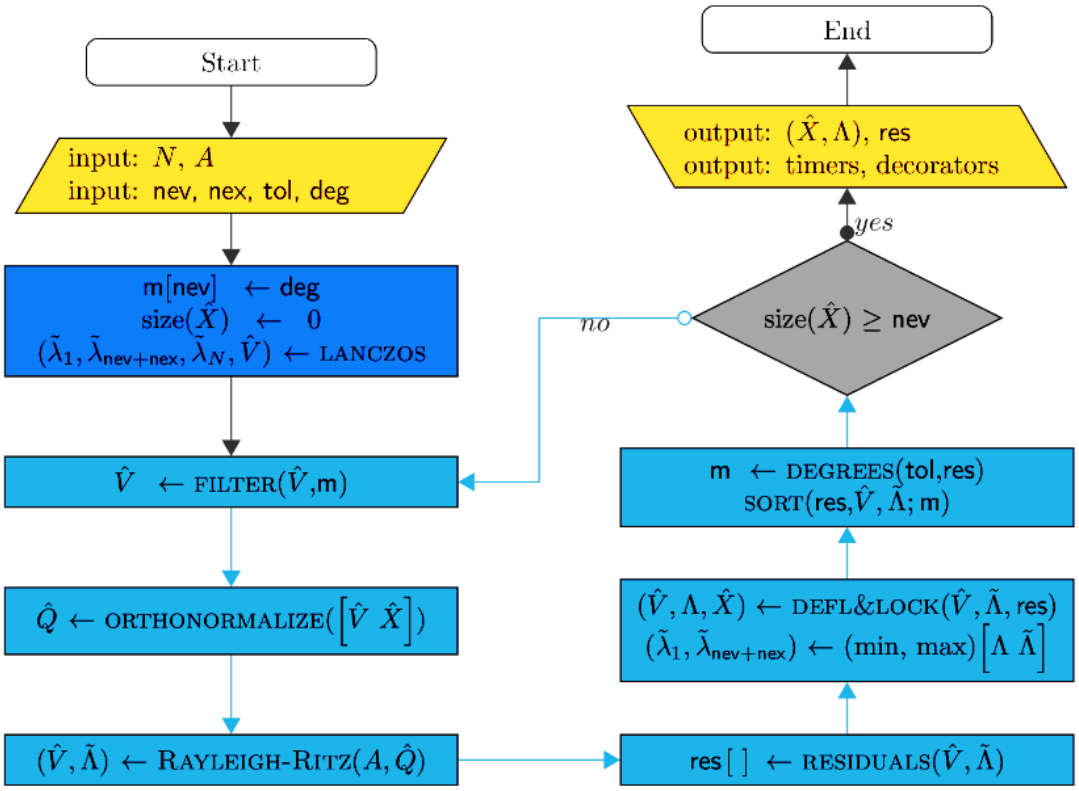

Figure 9: Eigensolver workflow

An alternative strategy in the development of an eigensolver is to leverage well-known and well-established iterative algorithms such as subspace iteration. A modern example of such an algorithm has recently been implemented in the Chebyshev Accelerated Subspace iteration Eigensolver (ChASE) library [36]. When tackling sequences of Hermitian eigenproblems, as they often appear in electronics structure codes, ChASE takes advantage of the distinctive features connecting adjacent problems in a sequence.

7.2 Prototype software release

7.2.1 DLA-Future and DLA-Interface

Currently, DLA-Future functionalities include the Cholesky decomposition and the solution of the triangular system of equations for distributed multi-core and NVIDIA GPU (CUDA) systems. The library includes the transformation of a generalised eigenvalue problem to a standard eigenvalue problem as well for multi-core systems. DLA-Future sources are available in GitHub at the following link [37].

The team has invested a lot of time to make asynchronous MPI functions work efficiently with HPX, and therefore the implementation of some of the algorithms of the eigensolver has been delayed.

We performed a weak scaling analysis of the library comparing it to other libraries on CSCS' Piz Daint (Figure 10). DLA-Future outperforms ScaLAPACK and Slate and the performances are comparable with DPLASMA, but with a slightly worse scaling (which will be fixed in the next release, see Section 7.4.1).

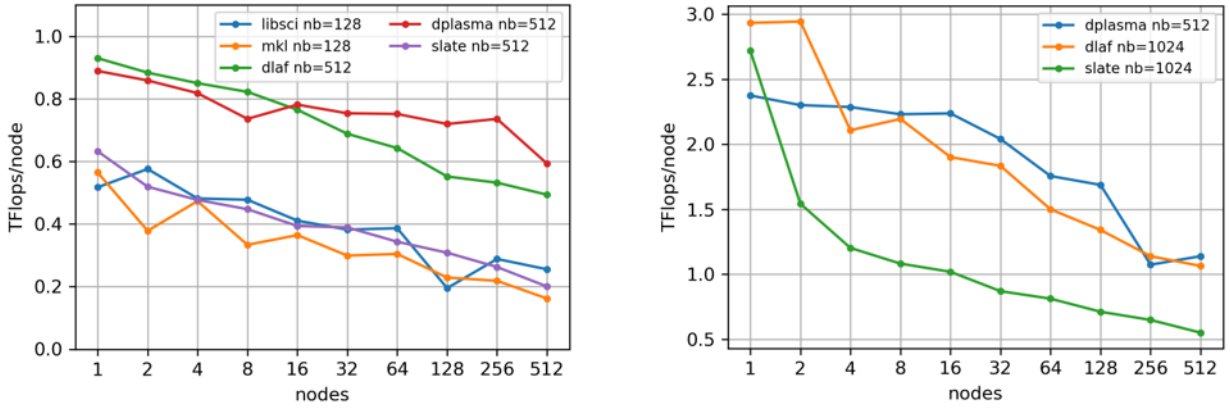

Figure 10: Comparison of the weak-scaling performance of DLA-Future Cholesky decomposition compared with other libraries (left: multicore implementations, right: CUDA implementations)

We decided to develop DLA-Future as a pure C++ library, and to provide ScaLAPACK like C and Fortran bindings through DLA-interface, which is available at the following link [38]

Currently DLA-Interface provides the binding for the Cholesky factorisation. The plan is to add the eigensolver and generalised eigensolver bindings when they are ready in DLA-Future.

7.2.2 ChASE library

The ChASE library is a modern C++ numerical library implementing one of the oldest and most respected iterative algorithms. With the advent of massively parallel architectures, the algorithm at the base of ChASE can leverage very efficient low-level BLAS-like kernels and achieve a very competitive parallel efficiency. Being an iterative eigensolver, ChASE can solve only for up to 20% of the extremal spectra of Standard and Generalized Complex Hermitian and Real Symmetric Eigenvalue problems. Among its most salient functionalities, the ChASE library features multiple data distribution geometries: custom-block, block-cyclic, and element-wise cyclic (Elemental). Thanks to its templated data type, it can solve problems in both single and double precision. ChASE is effective for solving sequences of eigenvalue problems for which it can take advantage of approximate solutions and experience speedups up to 3x. In addition, the algorithm of the Chebyshev filter has been optimised to minimise the number of FLOPs necessary to converge the desired eigenpairs.

ChASE comes in several different parallel flavours. It can be executed with a parallel MPI-OpenMP over distributed multi-core clusters. It also features a parallel MPI-CUDA hybrid execution on distributed many-core clusters with multiple GPUs per node. A tailored implementation using MPI+HPX is currently under testing and it will be available by the end of the project. The library is available in the newly launched GitHub repository [39].

7.3 Community outreach and integration

7.3.1 DLA-Future and DLA-Interface

As discussed in 7.2.1 the problems of integrating the communication layer in HPX delayed the development of some of the parts of the eigensolver.

Therefore, the current DLA-Future effort is in implementing and optimising the algorithms for the eigensolver, which, when ready, will be tested in the SIRIUS [40] library.

7.3.2 ChASE library

The ChASE library has been successfully integrated in a code for Optoelectronic simulations developed and maintained at the University of Illinois Urbana-Champaign (UIUC) and NCSA [41]. There is an ongoing effort to fully integrate ChASE within the FLEUR [42] code for Material simulations based on Density Functional Theory. There is also a preliminary agreement to test the use of ChASE both in the Yambo and Quantum Espresso [43] (QE) codes. Both FLEUR and the QE suite are part of the MaX-II Center of Excellence and have a large base of users in a wide community.

7.4 Outlook for pre-exascale and benchmark projections

7.4.1 DLA-Future and DLA-Interface

As shown in section 7.2.1, the weak scaling behaviour of DLA-Future is not in line with the performance of DPLASMA. However, a couple of optimisations in the communication code (use of MPI asynchronous functions and better memory usage) are showing promising results.

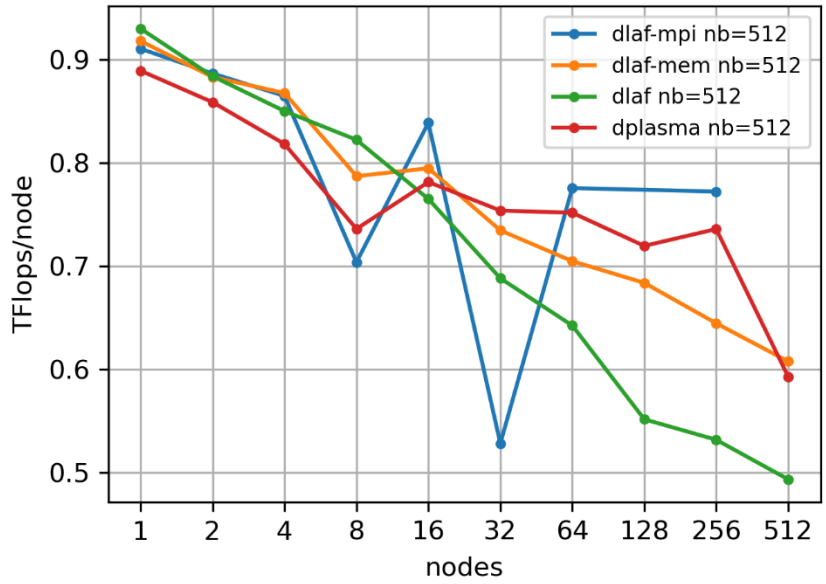

Figure 11: Comparison of the weak-scaling performance of the unreleased optimisations of DLAF vs best performing libraries in Figure 10

Unfortunately, the poor performance on square MPI grids (DLA-Future is running two ranks per node), is still preventing these optimisations from being released.

7.4.2 ChASE library

Particularly when executed on the newest AMD cluster with multiple A100 Nvidia GPUs (JUWELS Booster), ChASE is performing well and the relative performance of the various kernels relative to each other is in line with the computational cost that is expected. This result is achieved thanks to the extensive use of cuSOLVER.

On the other hand, when run in parallel on CPU-only, ChASE shows a growing bottleneck due to the QR decomposition. We are in the process of designing a new distributed algorithm for such a kernel that would balance again the relative cost among the main kernels of ChASE. Ultimately we plan to benchmark the code of eigenproblems with size > O(100k).

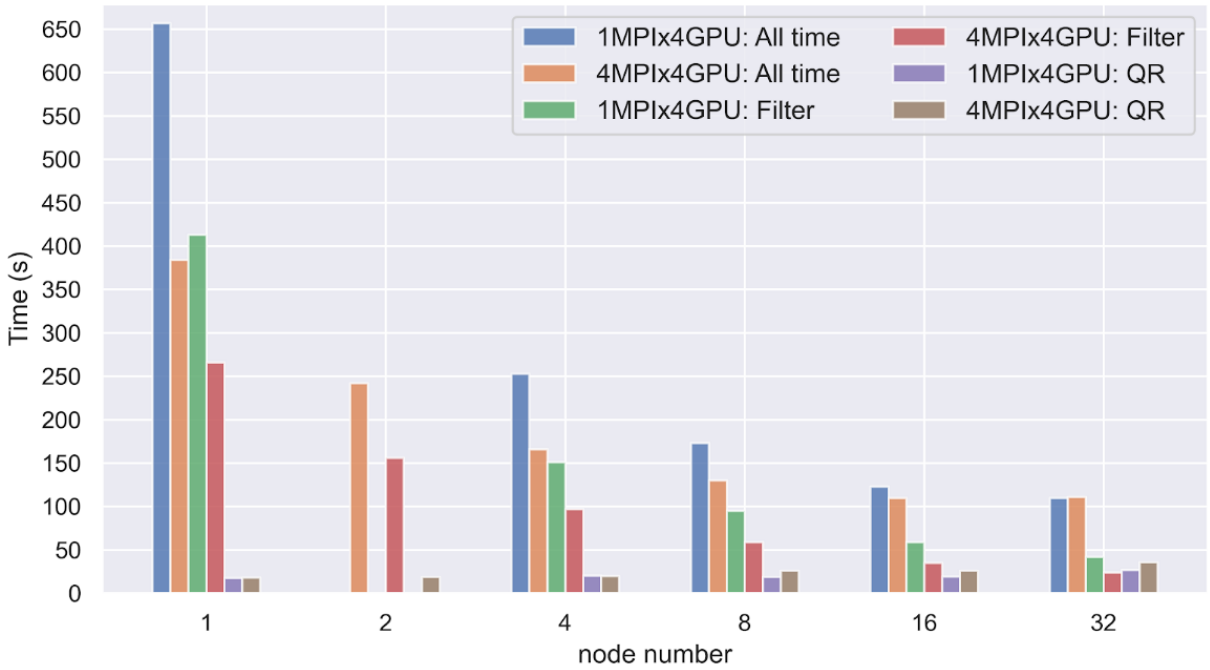

Figure 12: Performance of ChASE on JUWELS Booster (matrix BSE 76k , OMP = 12, nev=2350, nex=200)

8 GHEX: Generic Halo-Exchange for Exascale

8.1 Introduction and summary

GHEX is a library for halo-update in scientific applications. While halo-update is ubiquitous in HPC, its implementation (typically using MPI) is highly dependent on the details of the problems being solved. The main characteristics that distinguish the different applications are: the grid type (Cartesian, structured, unstructured), the periodicity of boundaries, the types of the values, the dimensionality and the number of fields being updated, the number of halo-lines exchanged, and more.

Most of the existing halo-update solutions have been designed for MPI-based applications, with extensions to MPI+X, where X is usually OpenMP. With the advent of many-cores architectures, like accelerator-based and hybrid, the need for modern programming models becomes more and more pressing. For instance, exploring multithreading architectures more efficiently than the typical MPI+X approaches could improve the effect of some latency-hiding techniques, and hence scalability.

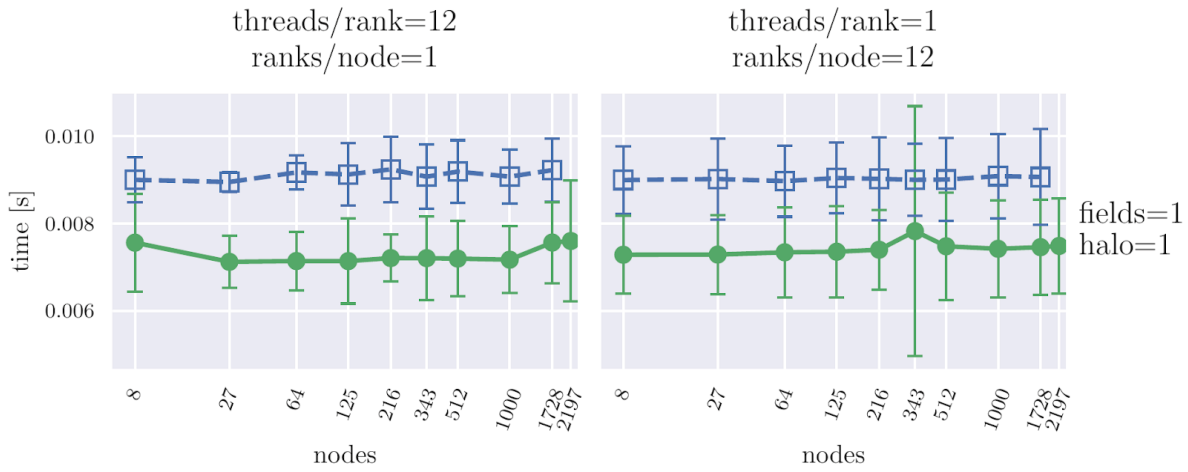

Figure 13: Halo Exchange weak scaling on Piz Daint using in-node direct remote memory access (RMA) through system/shared memory

From the design point of view there are two distinct aspects of GHEX. On the high level is the user-facing, unified and intuitive halo exchange API. Instead of accepting data as arguments, GHEX API accepts functions with defined interfaces. These functions allow the users to pass the required information, while GHEX remains oblivious to the specifics (e.g. the data layout) of the application. On the lower level GHEX defines a future- and callback-based APIs, which is close to the hardware and the computing platform. They provide access to low-level transport layers (in addition to MPI, for instance UCX and Libfabric), and exploit shared memory interprocess communication mechanisms (e.g. using XPMEM and CUDAIPC). These lower level transport layers have more flexible interfaces that can in some cases result in better performance than MPI. GHEX uses these interfaces in the high level halo exchanges to enable latency-hiding and to overlap computations with communication. In addition, GHEX users can also directly access the transport layer API, which is especially well suited for highly asynchronous, task-based computing

models. Both the interfaces are portable, so that the applications can be easily run on different platforms with different transport layers.

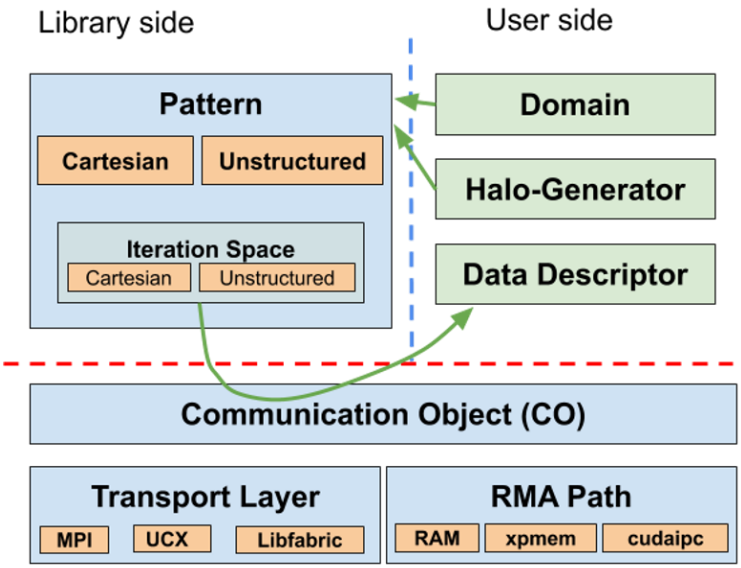

Figure 14: Overview of GHEX

8.2 Production software release

The code is publicly available on GitHub at [44] under the BSD-3-Clause license. CI is implemented through GitHub Actions, with runs tests on every pull-request and on demand. Tests are run on a single node virtualised environment and cover: high level exchange API, low level MPI / UCX / libfabric transport layers (both with future-based and callback-based API), single / multi-threaded modes, structured / unstructured grids (including support for inflated-cube grids and Atlas bindings) and RMA on shared-memory regions.

The current set of supported functionalities, in addition to the one tested with CI/CD, includes also NVIDIA and AMD GPU and hybrid CPU/GPU communications, support for XPMEM, CUDA-IPC to perform remote memory accesses to neighbour ranks, GridTools bindings, and Fortran bindings. Clearly, the testing space, in terms of functionality and performance is huge, and the current CI/CD frameworks are incapable of supporting automatic testing that covers all of the available features, which would require access to a wide variety of different computers. We plan to gradually add CI/CD features in the future, especially in collaboration with CSCS CI/CD initiatives. To compensate for this difficulty, we are actively collecting benchmarking results and scripts in another repository [45].

During code optimisation for AMD EPYC CPUs it became clear that the multi-level, hierarchical memory architecture poses a substantial challenge to performance. Data exchange speed between ranks and threads within the same compute node varies a lot depending on the memory domain they are bound to. To address this, we have developed HWCART, a hardware-aware Cartesian MPI communicator, which allows the user to map ranks to cores in a way that maximises

communication speed between the neighbours and minimises off-node communication. The code is available under BSD-3 license from a separate repository [46], since this is a useful utility not only for GHEX users.

8.3 Community outreach and integration

GHEX is part of the GridTools ecosystem, and available in the GitHub organisation of the same name. GridTools is a set of libraries and tools aimed at the weather and climate community to develop models using modern technologies.

GHEX has been ported to the weather model COSMO, and improvements on the performance of 9% have been observed. Since COSMO went into a feature-freeze by the time we were able to demonstrate the advantage of GHEX, it was not possible to include it in the COSMO code. Further integration with climate models, in this case by ECMWF, is facilitated by Atlas bindings. Atlas is currently developed at ECMWF for providing parallel data structures for Earth System models.

GHEX has been included in a benchmark application, called GTBench, which is currently used to benchmark architectures against the typical weather and climate computation patterns, and was part of the benchmarks for the procurement process for the computer acquired by the LUMI consortium. We are currently integrating GHEX as a transport layer in a Python-driven weather application framework named GT4Py. GT4Py is a strategic tool adopted by several partners, especially the Center for Climate Systems Modeling (C2SM) at ETH Zurich, in a multi-year project.

At University of Oslo we have worked to integrate GHEX with two community codes:

- Bifrost - a stellar atmosphere simulation code, a classical MPI-based single-threaded code, where each rank runs on a single CPU core and computes on a regular, cubic sub-domain. After each timestep all ranks exchange halos with their neighbours. GHEX's ability to handle multiple data fields and large halos more efficiently than native MPI does improve the overall Bifrost performance by up to 8%.
- DISPATCH is a task-based framework, in which each task handles a grid, which is not necessarily aligned with all other grids (e.g. grids can overlap and be rotated). DISPATCH is heavily multi-threaded, with threads picking up tasks as they become free. In this case we have integrated the code with GHEX transport layer API. It provides direct access to UCX and libfabric, and proves to be more efficient than MPI when it comes to multi-threaded applications.

Both codes are implemented in Fortran, and hence use GHEX Fortran bindings. In both cases integration with GHEX was done in a non-invasive way, i.e. the original codes do not require GHEX to compile and run. Instead, they can choose to use the legacy MPI communication implementation, or the GHEX backend with a compile-time option. This demonstrates that GHEX can be used as a plug-and-play solution with relatively little effort.

GHEX presentations at international meetings

- Fifth Workshop on Programming Abstractions for Data Locality (PADAL'19), September 2019, Halo-Update Communication Layer for Hybrid Computing, by Mauro Bianco, ETH Zurich
- PRACE Inter-WP Topical Session “Exascale for European Datacentres”, Design for Portability of Performance in Halo-Exchanges, by Mauro Bianco, ETH Zurich

- LUMI researcher workshop, 31st March, 2020: GHEX: Generic Halo Exchange for Exascale, by Marcin Krotkiewski, University of Oslo
- Whole Sun Virtual Meeting 2021, GHEX: Generic Exascale-ready library for halo-exchange operations on various grids/meshes by Mauro Bianco, ETH Zurich

8.4 Outlook for pre-exascale and benchmark projections

From the hardware point of view, we focused on modern CPU and GPU architectures. The benchmark suite [47] suite enables direct performance evaluation of pure halo-exchange operations for a 3-dimensional cartesian grid. It features different backends (pure MPI, gridtools::gcl, and ghex), which allow comparison to different halo-exchange libraries. The number of halo points, the number of fields, as well as the number of threads per rank are some of the inputs we can vary. This benchmark suite was instrumental in evaluating different exchange strategies presented in this section.

In systems based on AMD processors, the cores share memory resources on multiple levels (L3 cache, NUMA-node, socket). To optimise performance on such systems we developed a memory domain aware Cartesian communicator for MPI (HWCART), which allows the user to fine-tune the rank placement. In Figure 15 below we show the impact of HWCART and GHEX on performance and scalability on Betzy, a system with 1300+ dual-socket compute nodes with AMD EPYC 7742 and connected with HDR100 InfiniBand.

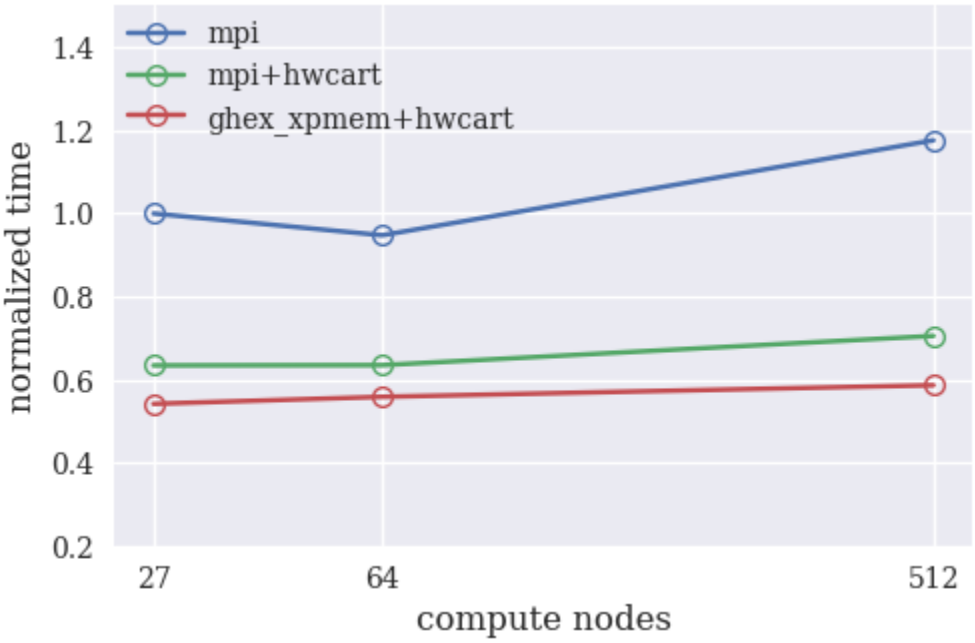

Figure 15: Comparison of halo exchange on Cartesian grids, with GHEX and HWCART, and simple MPI implementation on 65536 cores. 1 data field and 5 halo lines. GHEX with HWCART gives 2x improvements in execution times (the baseline is 27 compute nodes)

On GPU architectures, we focused on the one hand on efficiently gathering the data to be exchanged in specialised kernels, and on the other hand on using GPU-aware MPI

implementations or the UCX transport layer, which are actively developed by GPU vendors, making both NVIDIA and AMD GPUs supported.

Figure 16 shows results for GPU runs for unstructured grids, with 100 structured added, which is a typical number in weather/climate applications.

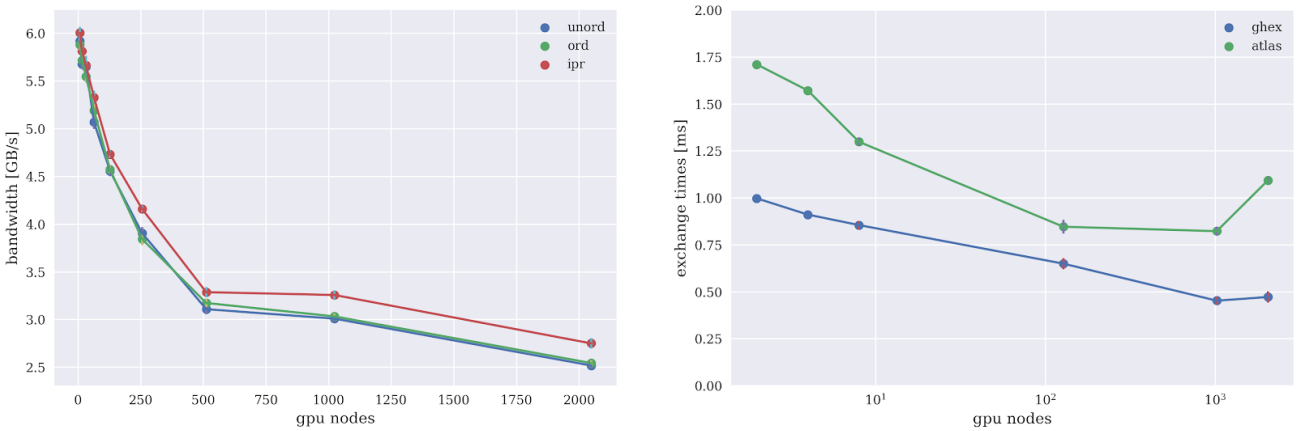

Figure 16: Left: per-node bandwidth on Piz Daint GPU partition, 1 rank per node, exchanging one single field on GPU using unstructured mesh. Right: exchange times on Piz Daint, GPU partition, GHEX halo exchange is compared with Atlas built-in halo exchange.

From these preliminary results, we are confident on the scalability of GHEX on pre-exascale computers, especially the one provided by the LUMI consortium. Future benchmarking plans will include more extensive investigations of application codes, and additional transport layers, like the libfabric API which will be well supported by future HPE-CRAY computers.

9 LyNcs: Linear Algebra, Krylov methods, and multi-grid API and library support for the discovery of new physics

9.1 Introduction and summary

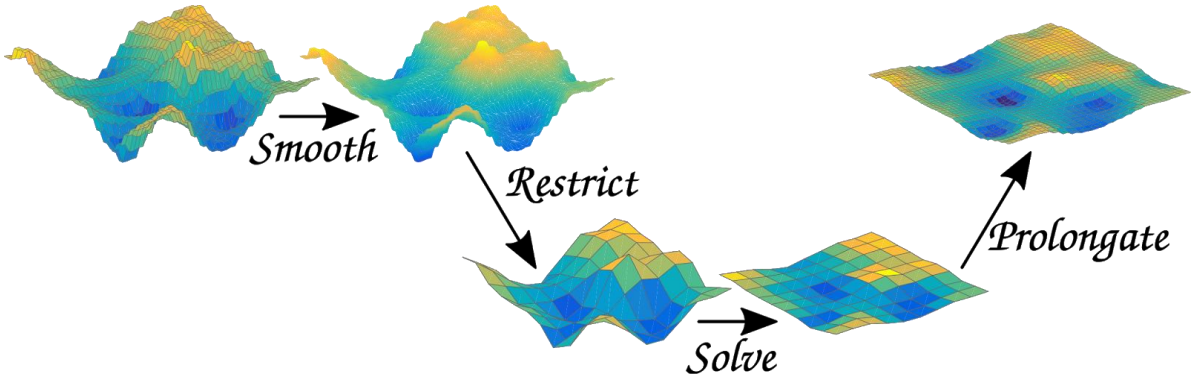

Figure 17: Important algorithmic steps in the Krylov accelerated multigrid solver developed in the LyNcs project

The project, Linear Algebra, Krylov-subspace methods, and multigrid solvers for the discovery of New Physics (LyNcs), is addressing challenges for iterative solvers with large sparse matrices which are arising on modern and upcoming architectures due to massive parallelisation. LyNcs is targeting efficient solutions for linear systems based on large sparse matrices by pooling together software development efforts across Europe. LyNcs will provide the European communities with the next generation of parallel libraries for solving sparse linear systems at the exascale. The project is led by the Computation-based Science and Technology Research Centre (CaSToRC) of The Cyprus Institute, which joins forces with partners from the French Institute for Research in Computer Science and Automation (INRIA) and the Leibniz Supercomputing Centre (LRZ).

Part of LyNcs is the development of an API that is targeting massive parallel machines to perform advanced task management with shared memory among huge parallel partitions. This LyNcs API together with implementing cutting-edge sparse linear solver algorithms, the development of novel block Krylov solvers and optimisation of existing parallel codes will enable community software to efficiently use the up-coming pre-exa and exascale machines. The software improvements target all levels of the scientific application software stack, from the basic Sparse BLAS library to fully-fledged simulation codes. Namely, LyNcs is targeting the Fast-Accurate Block Linear Krylov Solver (Fabulous), the Lattice QCD community solver library DDalphaAMG and at the lowest level the efficient sparse matrix support software Librsb.

9.2 Production software release

LyNcs is targeting software which spans all levels of the scientific software stack to ensure readiness for the up-coming massively parallel pre-exascale and exascale systems. Part of the developed software are:

- LyNcs -API: [48]

- DDalphaAMG: [49]
- Librsb: [50]
- PyRSB: [51]
- GNU Octave "sparsersb" [52]
- Fabulous: [53]

LyNcs-API: The LyNcs API, written in python, is available on GitHub under the organisation [54] and published under a BSD 3-Clause license. Within the last year major changes towards a first version are made based on separation of concerns creating a modular infrastructure of all LyNcs-API packages. Within this structure dedicated modules are created for general purpose [55], [56], [57], [58], [59] and for interfaces to lattice QCD libraries [60], [61], [62], [63] and a tool for optimisation and cross-checks of calculations [64]. Using the GitHub CI/CD capability new commits and merges are constantly checked to ensure coverage and usability of the package. To ensure a user friendly environment and visibility the source code is well documented and a user guide is at the moment given in the various readme of the packages. The task management of the API is based on the Dask package [65], which enables simple memory shared task management especially designed for large allocations for the next generation of High Performance Computing Systems. The LyNcs API is recently adapted by the Extended Twisted Mass Collaboration as the backend driver for porting the major Simulation code to GPUs. This guarantees the software support of LyNcs-API beyond PRACE-6IP.

Fabulous: Fabulous implements Block Krylov solver methods, is written in C++ and its development is ongoing. Within LyNcs new capabilities enabling flexible preconditioner are enabled though the implementation of the block GCR with inexact breakdown detection. The implementation of a novel Block Krylov solver, a flexible Block GCRO technique with deflation at restart and inexact breakdown detection has been achieved. This latter feature enables the detection of the convergence of some right-hand sides (or linear combination of right-hand sides) and automatically reduces the block size to reduce the computational effort. In addition, this solver allows to recycle spectral information between sequences of multiple right-hand-sides making it well suited for the coarse grid solve of a multigrid context. Integration of Fabulous in Maphys++ (Maphys' redesign using modern C++ genericity) is ongoing and will allow greater flexibility for the user and for software interconnection. Fabulous is distributed under CeCill license and available at the Inria gitlab [66], it should be noted that the GitLab page for Inria linear algebra software packages (Chameleon, Fabulous, Maphys) uses continuous integration, issue tracking, unitary testing, and complex scenarios testing. Integration is enabled through two high performance software distributions: spack [67] and guix-hpc [68]. A new release of Fabulous with the novel capabilities has been made and has been benchmarked using various Blas and sparse Blas, including librsb, libraries.

DDalphaAMG with multiple right-hand sides: Within LyNcs a new public version of DDalphaAMG is released, which enables multiple right-hand side together with the optional usage of advanced Block Krylov solver methods enabled in a linkage of Inria's solver library Fabulous. The major change within the development under LyNcs are the vector ordering which enables row major ordering. This adds flexibility capabilities by having vectorisation during compilation without explicitly using vector instructions. This guarantees portability without major performance lost to different CPU architectures, like ARM, Intel or AMD CPUs. The public released version is

available under GitHub [69] and is successfully tested on various systems, like SuperMUC-NG, Hawk and Frontera and the ARM prototype system BSC-CTR.

Librsb: The librsb library [50] is a complete Sparse BLAS for shared memory parallel computers. Its first public release dates to 2013. Originally it consisted of C99 and OpenMP, and has bindings in several languages. Is licensed under LGPLv3. Since the start of the LyNcs project, librsb has been developed further and regularly updated (with release tarballs on [70]). The recent updates have fixed many minor bugs and improved the documentation. The primary pillar of this activity has been a consistent expansion of the internal test suite. This has allowed reaching a very high coverage test rate: >90% code lines and >98% of functions. A good fraction of the bugs has been identified thanks to the aggressive coverage testing expansion. Other bugs have been identified after community reports (e.g. see [71] or [72]). The Python interface [73] has been brought to a production level (Linux binaries installable via [74]). The GNU Octave access layer [75] has been updated regularly. Thanks to the Debian and Cygwin volunteer community, Linux and Cygwin users can benefit from using pre-compiled binaries after each librsb and sparsersb release. Since 2020, librsb has been included in the "Spack" HPC software distribution [76], as well as GUIX-HPC [77]. The recent development activity in librsb has been in creating templated C++ kernels for multiplying sparse matrices by multiple right-hand-sides (SpMM). These changes are totally internal to librsb and benefit transparently any code using it, including the most recent Maphys, adapted by the Inria partners to use librsb. Preliminary benchmarks on SuperMUC-NG and AMD EPYC Rome 7742 reveal performance competitive or exceeding that of Intel MKL on large matrices by Inria and CaSToRC.

9.3 Community outreach and integration

Opportunities for outreach identified by the LyNcs partners include international and national community conferences, PRACE, EuroHPC JU, EuroCC and other dissemination events. Although most of the conferences were cancelled due to the pandemic, we could participate and engage with the community in various online events, which we will outline within this section.

LyNcs activities were disseminated through PRACE-6IP activities such as the WP8 virtual F2F in September 2020 and the virtual PRACE Inter WP meeting in October 2020. In addition, our activities were part of a poster presented at the EuroHPC Summit Week in March 2021, and Shuhei Yamamoto had the opportunity to contribute an invited virtual Seminar talk at Riken, Kobe in September 2020 and at the major community conference APLAT2020 on his work on multiple right-hand sides in DDalphaAMG. The community outreach to one of the major lattice QCD collaborations, Extended Twisted Mass Collaboration (ETMC), by Simone Bacchio resulted in the adaptation of the LyNcs-API as the backend for the new simulation code of the collaboration dedicated for GPU machines. This is a major success, not only for the LyNcs project but also for PRACE-6IP, since adoption of the API by a major European collaboration is a substantial step in ensuring sustainability of our efforts and allows for future extensions of the project.

In the current state, integration between the different software tasks and packages is complete. Those that are beyond prototype status have also been made part of the software releases. This includes the integration of the community software package tmLQCD, DDalphaAMG, QUDA, and c-lime within the LyNcs-API and integration of Fabulous with the DDalphaAMG-multi rhs. While the ETMC transitions to the LyNcs API, the integration of tmLQCD will help support testing and the CI/CD activities thus allowing tmLQCD to be maintained as a legacy code. All interfaces of the different software libraries are updated and in a ready-to-use stage. This includes Fabulous,

with the new parameters for the newly available IB-BGCRO-DR solver, DDalphaAMG-multi-rhs, which can now be called with or without Fabulous, and various interfaces for librsb, such as the python-linkage PyRSB.

9.4 Outlook for pre-exascale and benchmark projections

Algorithm and software development within the LyNcs project are linked to four different tasks, and the separation of concerns within our software packages and tasks gives us the opportunity to flexibly adapt to new, unforeseen, trends, such as for example enabling HIP-support within the solver library QUDA to target the pre-exascale machine LUMI at CSC Finland.

While originally intended as a linkage to solver libraries, namely fabulous, DDalphaAMG and QUDA, to provide a common interface for community simulation codes, the API now enables task parallelism via Dask and is using community simulation codes for CI/CD. This community code linkage, formerly designed to be from the community codes to the API, is providing a platform for solid software developments using cross-checking and enabling a novel, flexible python API, which will enable exascale simulations. The redefinition proved very timely, since the API will become the backend for the future community simulation code of the Extended Twisted Mass Collaboration (ETMC). Within this effort, the existing linkage to the simulation code tmLQCD and the GPU solver package QUDA is essential. The usage by ETMC will guarantee software support for the API beyond PRACE-6IP WP8. This also shows that community software support is and was needed outside of the European CoE structure and shows the impact of PRACE-6IP WP8 for the European community.

Towards D8.5 within PRACE-6IP, we will benchmark the developed software codes on various architectures and PRACE Tier 0 machines. This will include:

- Strong scaling runs of DDalphaAMG on Frontera, TACC and SuperMUC-NG, LRZ
- Performance evaluation of librsb on BEAST, LRZ and SuperMUC-NG, LRZ
- Performance and Capability runs of LyNcs-API on JUWELS Booster, JSC
- Performance evaluation of HIP ports at CSC-Frankfurt and CSC-Finland
- Performance evaluation of Block-Krylov solvers with DDalphaAMG

10 ParSec: Parallel Adaptive Refinement for Simulations on Exascale Computers

10.1 Introduction and summary

One of the key capabilities required for CFD codes to take advantage of leading-edge computing resources is the automation of the mesh generation and adaptation processes. Manual mesh generation and tuning is not conceivable in an exascale simulation workflow. Adaptive Mesh Refinement (AMR) eliminates those bottlenecks providing higher efficiency and robustness to the codes. ParSec brings together well-known CFD practitioners with the aim of sharing best practices, and collaboratively modernise the AMR implementation of three leading-edge CFD community codes. The partners involved in the project are the Barcelona Supercomputing Center, the KTH Royal Institute of Technology and together Cenaero and Université de Liège. The community codes brought to the project by these institutions are: Nek5000, the scalable high-order solver for computational fluid dynamics from KTH/UIUC, Alya, the high performance computational mechanics solver from BSC, and Argo, the high order multiphysics solver from Cenaero. These three CFD solvers cover the main approaches for the solution of PDEs using both structured and unstructured meshes: finite element, finite volume, and spectral elements. Beyond the AMR-enabled legacy codes, a number of open-source libraries providing mesh functionalities are included into the project, namely: MAdLib, that includes all mesh functionalities of Argo, Gmsh, a finite element mesh generator with a large community of users, and GeMPa including all the geometric mesh partitioning tools included into Alya.

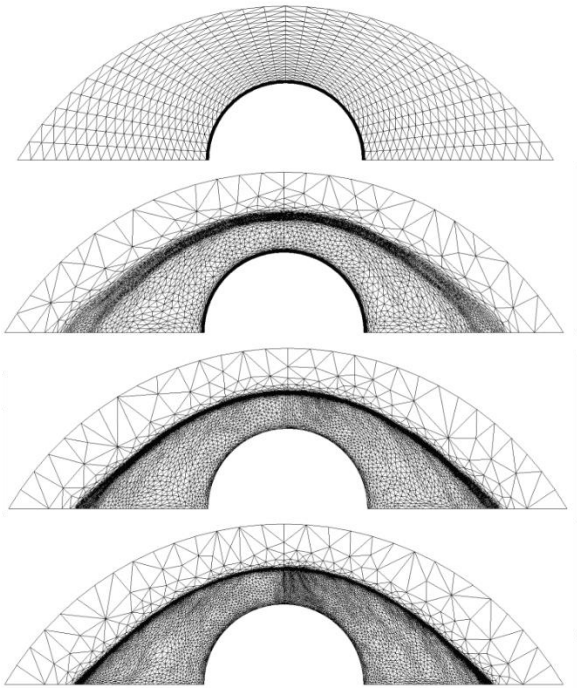

Figure 18: Iterative mesh adaptation, governed by a hypersonic flow facing a cylinder generated with Madlib.

10.2 Production software release

Nek5000 AMR and GPU implementations are developed in a private git repository. However, KTH is in contact with Nek5000 developers at Argonne National Laboratory and coordinate their private repository with the main repository. In this case their OpenACC GPU code is an extension of the existing branch at [78]. KTH plans as well to add their AMR branch to the main code v21 release, which should appear this year.

The mesh adaptation library **MAdLib** now includes high-order anisotropic solution-based mesh adaptation features, as well as a new C-compatible interface. The latter will be present in the official release (2.2.0) of MAdLib, whereas the anisotropic mesh adaptation will come with the next release (2.3.0). Both functionalities are included in the development version of MAdLib, which can be downloaded from [79]. The latest stable release of the mesh generation library Gmsh (4.8.1) [80] now includes a new 3D multi-threaded meshing kernel that can handle non-manifold geometries and arbitrary sized isotropic fields, available either as a standalone application or through its C, C++, Python and Julia API.

Alya [81] now includes AMR capability for general unstructured finite element meshes. It has required major restructuring of the code to support dynamic meshes and the parallelisation has been based on an interface freezing approach. The remeshing part is supported by Gmsh which has been integrated into Alya. Moreover, the mesh partitioning functionalities have been released into the stand-alone library GeMPa [82]. This library supports parallel mesh partitioning based on space-filling curves.

10.3 Community outreach and integration

For Nek5000 both the AMR and GPU implementations are in the development phase, however, they have already been shared with external users, e.g. within the framework of the EXCELLERAT EU project. Another example is a collaboration with CINECA (Italy), where both implementations are tested and developed. The result of this is a simulation of a simplified rotor, shown in Figure 19 below, which is a test run before a real drone rotor would be modelled.

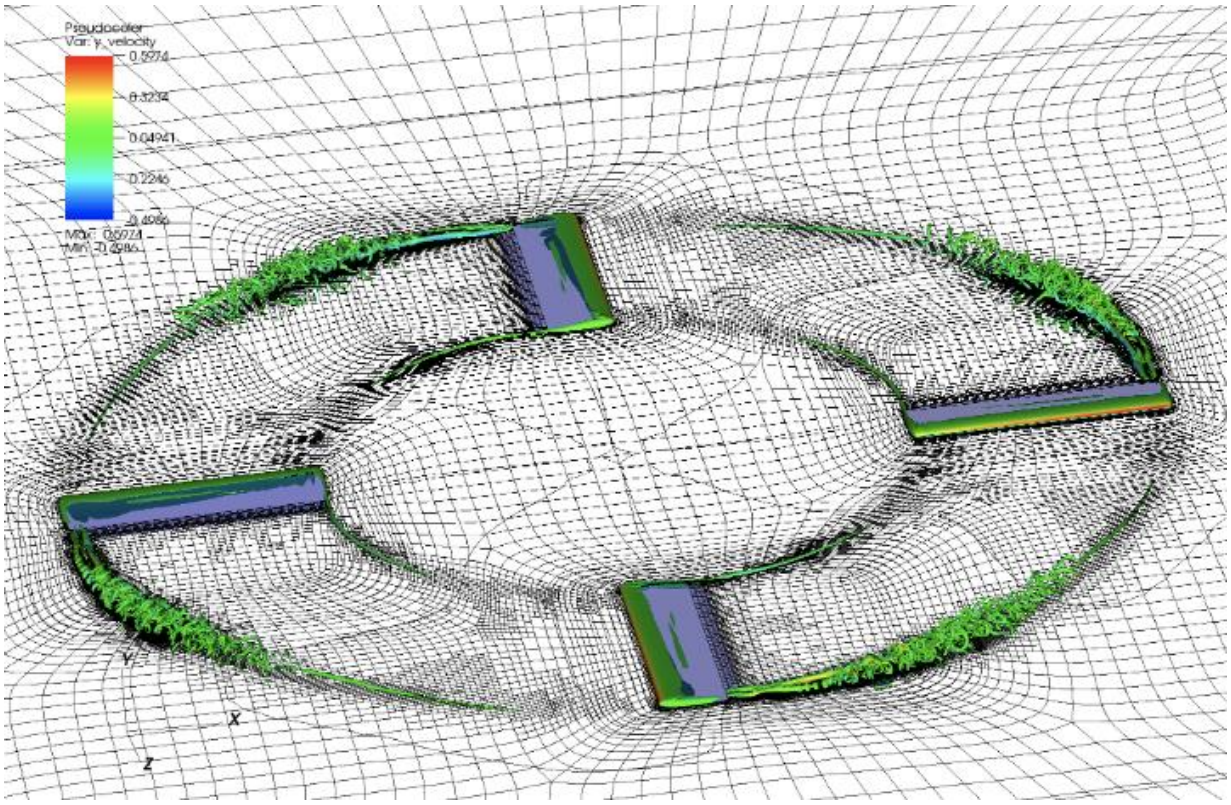

Figure 19: Vortical structure of the flow around a simplified rotor obtained with the AMR branch of Nek5000

A new pure C language API has been added into MAdLib so that it can be easily interfaced to other software apart from Argo. It is now about to be linked with Alya, which would take advantage of its sequential anisotropic mesh adaptation features. The partitioning work would be let to Alya itself. The integration of the Gmsh C API in Alya is completed and is under testing. An initial integration of the Gmsh C++ API in MAdLib is under way. Moreover, Gmsh has an outstanding community of users (~5000) that benefit from the developments carried out in ParSec.

Finally, Alya is part of the software hub of various Centres of Excellence (EXCELLERAT, CoEC, EoCoE, CombioMed, Raise), the use cases of those CoEs are directly benefiting from the AMR capabilities developed within ParSec. GeMPa will be integrated in the community codes of ParSec in the last phase of the project.

10.4 Outlook for pre-exascale and benchmark projections

There are two main aspects of the Nek5000 development within ParSec: enhancement of the Adaptive Mesh Refinement algorithm and improved parallel performance on the heterogeneous architectures. In the second case we focus on the OpenACC/OpenMP based GPU implementation, which was currently tested on 1500 GPUs. Although the mortar elements or p-refinement strategies have not been considered yet, the AMR implementation has been significantly enhanced making it a robust tool for performing simulations relevant for industry. This has been achieved by reimplementation and modularisation of the code, exploring different partitioning libraries, improving pressure preconditioning and investigating various strategies for generating high-order hex-based meshes. Regarding heterogeneous architectures, the work has focused on the GPU-

version of Nek5000 relying on an OpenACC/CUDA framework, which is currently being upgraded to an OpenMP/HIP framework. Its performance on JUWELS Booster system for the turbulent pipe case (conformal mesh) on NVIDIA /AMD architecture is shown in Figure 20 below, indicating scaling up to 512 GPUs with good efficiency.

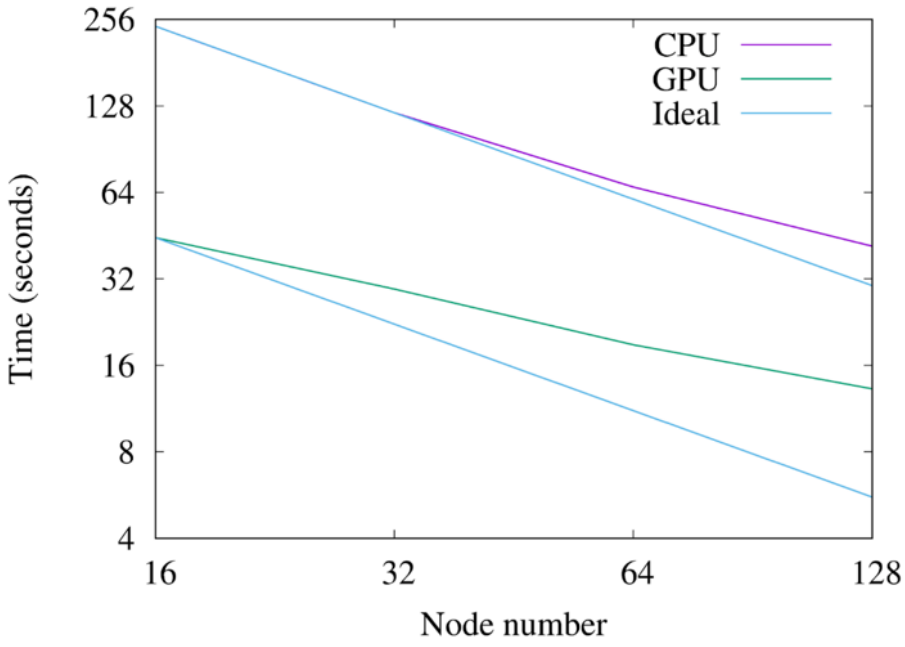

Figure 20: Strong scaling of a turbulent pipe case on JUWELS Booster system. Results for CPU and GPU are presented. A single node corresponds to 4 A100 GPUs or to 48 EPYC CPU cores. A mesh consists of 823632 elements with polynomial order 10.

MAdLib has two distinct features: i) its data structures for distributed mesh management that have been implemented and demonstrated through their use in large-scale, fixed-mesh computations with Argo, and ii) its mesh adaptation algorithms that currently work only sequentially. Our current effort consists in implementing a parallel mesh adaptation strategy that combines these two ingredients, including a load-balancing step between partitions that have been independently adapted, in order to obtain a scalable adaptation procedure.

Concerning Gmsh, most of the efforts in the next months will be focused on performance testing and on the continuing integration with MAdLib. Finally, regarding Alya, the first goal of ParSec has been achieved by completing and validating the overall AMR workflow. At this point the focus is set on the optimisation of the code to run on (pre)exascale supercomputers. The initial focus of the optimisation has been the 3D interpolation models for the migration of the solution between consecutive meshes. Regarding the mesh partitioning tools of Alya, those have been used on production runs for meshes of several billions of elements, those results will be now reproduced with the recently released stand-alone library GeMPa.

11 QuantEx: Efficient Quantum Circuit Simulation on Exascale Systems

11.1 Introduction and summary

The QuantEx project aims to develop a scalable, extensible framework for quantum circuit simulation on large distributed clusters using tensor network methods. These methods provide a complementary approach to quantum circuit simulation which can enable the simulation of circuits with more qubits than is possible with full wave-function simulators. This approach is particularly effective for low depth quantum circuits which makes them very relevant for the simulation of NISQ inspired circuits.

Figure 21 below shows a cartoon of their expected area of applicability compared to full wave-function simulation methods. Depth in the figure below refers to circuit depth (related to entanglement in the circuit). It should be noted that there is a trade-off that for fewer qubits full wave-functions simulations will be more performant. For larger numbers of qubits, full wave-functions simulation is no longer possible due to memory requirements, and tensor network simulators can work well in this regime but only up to a certain circuit depth

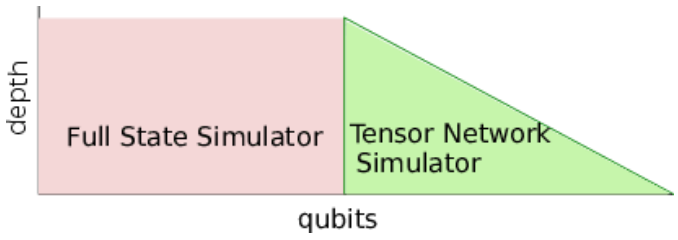

Figure 21: Expected area of applicability of simulation methods

Julialang has been chosen as the primary language for this project. This is because it provides a good balance between productivity and performance, has many useful packages in its ecosystem to aid development and offers the ability to easily tie together packages and tools written in other languages.

11.2 Production software release

The public software release consists of five Julia packages which together provide an extensible framework for performing quantum circuit simulations on large distributed memory clusters. These packages can be found on GitHub under the JuliaQX organisation at [83] and are also registered as Julia packages making them easily accessible. Each package features comprehensive unit tests and automatically generated documentation which make use of CI features of GitHub. The container diagram in Figure 22 shows how the packages work together to provide a simulation workflow for quantum circuits. A short description of each of the packages follows below.

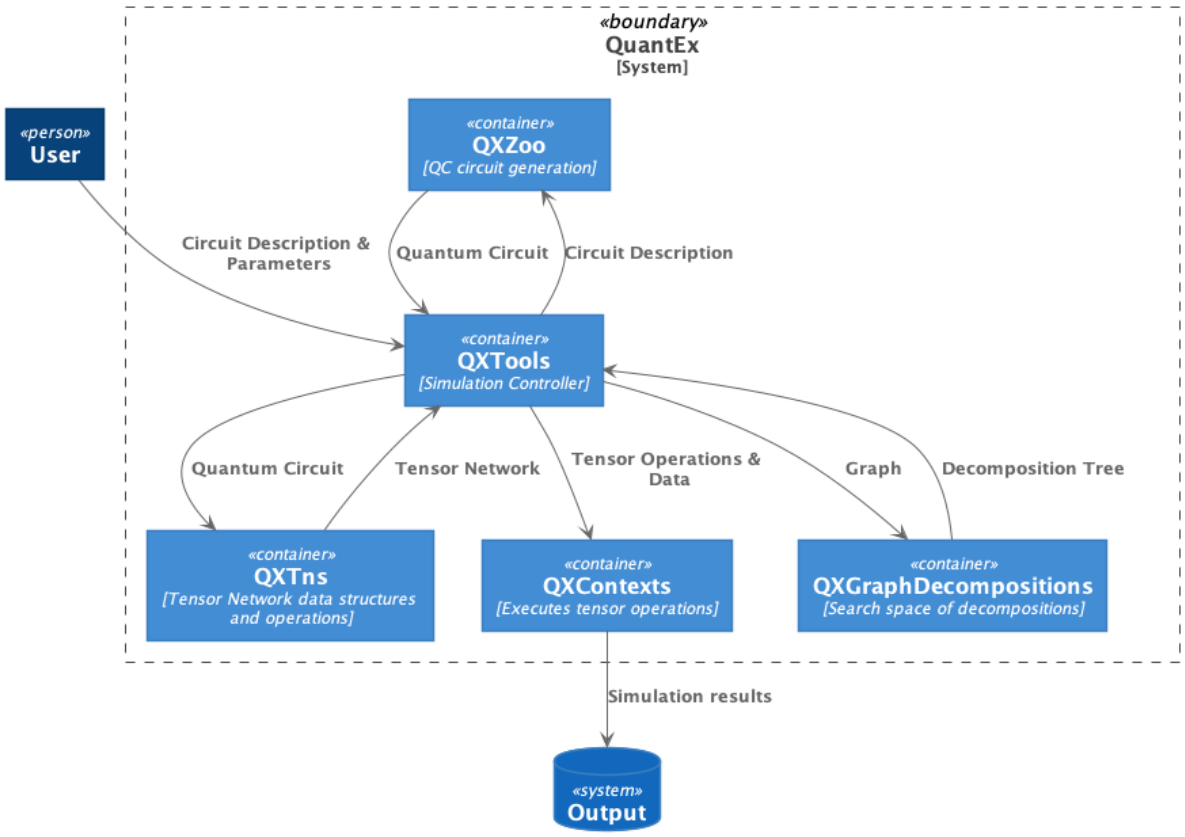

Figure 22: Overview of how the packages work together to provide a simulation workflow for quantum circuits

QXZoo: Provides data structures and functions for representing and generating quantum circuits.

QXTns: This package provides data structures and utilities for manipulating tensor networks with particular features for tensor networks derived from quantum circuits. It includes the ability to automatically identify and track hyper-indices of tensors which can lead to significant performance improvements.

QXGraphDecompositions: This package provides data structures and functions for analysing and manipulating graph representations of tensor networks. In particular, it provides functions for finding efficient tree decompositions and for identifying sets of indices which when sliced can reduce the tree width of the selected tree decomposition. This makes it possible to distribute computations across multiple processes/nodes.

QXContexts: This package implements runtime contexts for performing the tensor contraction computations. The MPI wrapper MPI.jl is used to distribute computations and GPU support is provided by JuliaGPU packages (yet to be integrated into final software release).

QXSim: This package ties together the other packages to provide a consistent quantum circuit simulation workflow which consists of the following steps:

1. Circuits are built and represented as QXZoo circuits.
2. The QXZoo circuit is converted to a QXTns tensor network.

3. This network is converted to a graph data structure provided by QXGraphDecompositions and a suitable tree decomposition and set of edges to slice are identified.
4. Using the tree decomposition and set of edges to slice a DSL representation of the computation is generated. This is then used as input to QXContexts to perform the computation using the context and settings that make the best use of the available resources.

11.3 Community outreach and integration

During the initial project planning phase and at various stages throughout the project there has been engagement with project stakeholders and potential users. This has been in the form of online meetings and workshops where design plans and prototypes were presented and stakeholders gave their feedback and recommendations.

To coincide with the full public software release, a number of outreach activities are planned to promote the project and reach out to potential users. These plans include stories on the ICHEC, LRZ and PRACE websites and social media channels as well as a series of workshops to introduce interested users to the tools that have been developed. In addition to this we have submitted proposals to give presentations and posters at Juliacon and ISC.

A containerised version of the software workflow was demonstrated in a cloud based HPC cluster at the Supercomputing 2020 (SC'20) tutorial "Practical OpenHPC: Cluster Management, HPC Applications, Containers and Cloud" [84]. At FOSDEM 2021 HPC container presentation, "Deploying Containerized Applications on Secure Large Scale HPC Production Systems" [85], the QuantEx software workflow was presented as an important use case for containerised workflows on traditional HPC systems.

As well as engaging with potential users, efforts are ongoing to identify suitable opportunities to integrate the developed tools into commonly used quantum circuit simulation frameworks. Two particular directions the QuantEx team are exploring are the possibility of integrating QuantEx as a backend for the Yao.jl [86] and tequila [87] ecosystems with the hope of enabling these frameworks to be used with pre-exascale and exascale HPC clusters.

11.4 Outlook for pre-exascale and benchmark projections

To successfully scale to use large exascale clusters requires the ability to decompose the problem at multiple levels. For contracting tensor networks derived from quantum circuits there are multiple levels of parallelism that can be used. This means that in theory these methods should be capable of scaling to match the available resources. In reality it is difficult to tune each level to achieve good overall performance. The levels that will be used by the QuantEx framework are:

1. At the highest level it is possible to compute the amplitudes for different sets of bit-strings independently on different subsets of nodes
2. The next level is to decompose over the partitions corresponding to different combinations of values on sliced bonds. These can be performed independently and only require a summation over the resulting scalars
3. For the individual contraction operations, it is possible to make use of CPU and GPU threads as well as vectorised operations to extract the maximum performance from the

underlying hardware. Optimising for memory layout and minimising memory allocation are key considerations here

While these levels of parallelism are used in the current implementation, significant effort is still required to profile and optimise the current implementation and improve performance on the target architectures. Work that is ongoing to characterise and improve performance include:

- Testing and performance characterisation across target architectures. This has included Intel Xeon, AMD and ARM CPUs and NVIDIA V100 GPUs to date
- Instrumentation of developed tools with the LIKWID [86] profiling tool to enable the collection of performance diagnostics
- Investigation and comparison of threading performance and use of vectorisation when using Intel-MKL and OpenBLAS libraries
- Containerisation of workflows and its effects on performance

Pilot access to additional HPC systems with more nodes and including newer NVIDIA and AMD GPUs is planned in future.

One of the potential issues faced by the developers of modern non-traditional HPC workflows developed using “high productivity” languages such as Julia is the deployment of their workflows on much more restrictive HPC systems. The difference in these development philosophies often results in the transition from the developer environment to the HPC system being extremely complicated and requiring a lot of time and effort by the developers and HPC centres to build and maintain the software workflows.

To enable scientists to take advantage of the massive amount of compute resources available on large HPC systems we need to find a mechanism that enables them to deploy their workflows and software in a way that is simple and does not require a lot of modification to their code and workflow, but also respects the existing HPC system environment, workflows and security policies. To achieve this, we employ the use of HPC containers.

12 Conclusions

In this deliverable, we report on the status of the ten projects running under the WP8 of PRACE-6IP. Eight of these ten projects started their work from the start of the PRACE-6IP, while the two remaining started only in January 2020, after a second call. These projects have all reached the stage in which quality software could be made available for users, either as standalone apps and libraries, or merged upstream. The projects have a public repository and a development infrastructure that is at a high level (i.e. continuous integration, issue tracking, integrated documentation, etc.). The projects have documented their outreach to their user communities and stakeholders, including several actors in the European landscape such as CoEs and EuroCC. No critical issues in the management of these projects emerged so far, as it also appears from the results reported here. Only a few projects had some staffing issues, with staff joining later than expected or leaving. The COVID-19 pandemic did have some impact, for example hiring people from abroad, or having in person meetings and hackathons, but generally, this impact has been well mitigated. One external factor that will impact the final phase of these projects is the later than expected installation and operation of the EuroHPC pre-exascale infrastructure. Whereas several codes have made efforts to support the planned architectures, for example by providing both NVIDIA and AMD GPU support, information is missing at this point on the 3rd system, and actual benchmark results need to be delayed till after the originally planned end of the projects of WP8 (Oct 2021).