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

D8.3

Interim Progress Report: Public Prototype Software Release and Development Infrastructure

Final

Version: 1.2
Author(s): Fabio Affinito, Joost VandeVondele, Alex Upton
Date: 29.04.2020

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: < **D8.3**>	
	Deliverable Nature: <Report>	
	Dissemination Level: PU	**Contractual Date of Delivery:** 30/04/2020
		Actual Date of Delivery: 30/04/2020
	EC Project Officer: Leonardo Flores Añover	

Document Control Sheet

Document	**Title: Interim Progress Report: Public Prototype Software Release and Development Infrastructure**	
	ID: D8.3	
	Version: 1.2	**Status:** ***Final***
	Available at: http://www.prace-project.eu	
	Software Tool: Microsoft Word 2016	
	File(s): D8.3	
	Written by:	Fabio Affinito, Joost VandeVondele, Alex Upton

Authorship	**Contributors:**	Fabio Affinito, Constantia Alexandrou, Momme Allalen, Simone Bacchio, Marco Bettiol, Mauro Bianco, John Biddiscombe, Fabian Boesch, Ricard Borrel, Matic Brank, David Brayford, John Brennan, Dirk Brömmel, Tomas Brzobohaty, Mark Bull, Junxian Chew, Zahra Chitgar, Laurent Chôné, Olivier Coulaud, Tilman Dannert, Edoardo Di Napoli, Jacob Finkenrath, Urs Ganse, Christophe Geuzaine, Paul Gibbon, Luc Giraud, Aleksander Grm, Kenneth Hanley, Berk Hess, Koen Hillewaert, Victor Holanda, Guillaume Houzeaux, Luigi Iapichino, Alberto Invernizzi, Ferdinand Jamitzky, Niclas Jansson, Joe Jordan, Prashanth Kanduri, Sebastian Keller, Leon Kos, Giannis Koutsou, Marcin Krotkiewski, Carlos Lopez, Martti Louhivuori, Michele Martone, Michal Merta, Niall Moran, Teodor Nikolov, Henrik Nortamo, Lee O'Riordan, Phillip Otte, Dejan Penko, Adam Peplinski, Janez Povh, Lara Querciagrossa, Philipp Schlatter, Gregor Simič, Ujjwal Sinha, Raffaele Solcà, Thomas Toulorge, Alex Upton, Joost VandeVondele, Ivona Vasileska, Xinzhe Wu, Shuhei Yamamoto, Jan Zapletal, Artem Zhmurov
	Reviewed by:	Florian Berberich, JUELICH Cédric Jourdain, CINES
	Approved by:	MB/TB

Document Status Sheet

Version	Date	Status	Comments
0.1	31/03/2020	1st Draft	Missing executive summary, introduction and conclusions
0.2	02/04/2020	2nd Draft (ready for internal review)	Added executive summary, introduction and conclusions
0.3	14/04/2020	3rd Draft (following internal review)	Changed spelling to British English, changed formatting of links to

			references, minor changes to text and overall formatting
1.0	21/04/2020		Minor formatting changes and additional references
1.1	22/04/2020		
1.2	29/04/202	Final	Updated contributors

Document Keywords

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

Table of Contents

List of Figures

List of Tables

References and Applicable Documents

[1] https://doi.org/10.1088/0741-3335/57/11/113001
[2] https://doi.org/10.1109/PDP.2010.47
[3] https://cfsa-pmw.warwick.ac.uk/EPOCH/epoch
[4] https://gitlab.version.fz-juelich.de/SLPP/epoch
[5] https://bitbucket.org/lecadpeg/simpic/src/master
[6] https://gitlab.version.fz-juelich.de/SLPP/lecad/simpic
[7] https://bitbucket.org/lecadpeg/bit1/src/master
[8] https://gitlab.version.fz-juelich.de/SLPP/epoch
[9] https://cfsa-pmw.warwick.ac.uk/EPOCH/epoch
[10] http://www.fz-juelich.de/jsc/jube
[11] https://bitbucket.org/lecadpeg/simpic/src/master/doc/
[12] https://github.com/MoPHA
[13] https://github.com/MoPHA/gene-tasks
[14] https://github.com/fmihpc/vlasiator/tree/openacc
[15] https://github.com/MoPHA/sympife-vmax
[16] https://github.com/MoPHA/strugepic
[17] https://bitbucket.org/lecadpeg/simpic

[18] https://github.com/MoPHA/sympife-vmax/
[19] https://gitlab.com/gromacs/nb-lib.git
[20] https://gitlab.com/gromacs/nb-lib/-/wikis/NB-LIB-Public-Documentation
[21] https://pm.bsc.es/ompss-2-downloads
[22] https://github.com/bsc-pm/tampi
[23] https://pm.bsc.es/ompss-2
[24] http://numbox.it4i.cz/
[25] https://github.com/It4innovations/espreso
[26] https://code.it4i.cz/
[27] https://nose.readthedocs.io/en/latest/
[28] https://github.com/It4innovations/espreso
[29] http://numbox.it4i.cz/
[30] http://www.netlib.org/scalapack
[31] https://elpa.mpcdf.mpg.de/software
[32] https://github.com/STEllAR-GROUP/hpx
[33] https://github.com/eth-cscs/DLA-Future/
[34] https://github.com/eth-cscs/DLA-Interface/
[35] https://github.com/SimLabQuantumMaterials/ChASE
[36] https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/github-flow
[37] https://user.cscs.ch/tools/continuous/
[38] https://simlabquantummaterials.github.io/ChASE/
[39] https://github.com/GridTools/GHEX
[40] github.com/sbacchio/lyncs
[41] http://gitlab.Inria.fr
[42] https://gitlab.Inria.fr/solverstack/spack-repo
[43] https://guix-hpc.bordeaux.Inria.fr
[44] https://github.com/sy3394/DDalphaAMG
[45] http://librsb.sourceforge.net/
[46] https://github.com/michelemartone/pyrsb
[47] https://octave.sourceforge.io/sparsersb/
[48] https://github.com/Nek5000
[49] https://repository.prace-ri.eu/git/UEABS/ueabs/-/tree/master/alya
[50] https://sites.uclouvain.be/madlib
[51] https://gitlab.onelab.info/gmsh/gmsh
[52] https://github.com/DmitryLyakh/TAL_SH
[53] https://github.com/ngnrsaa/qflex
[54] https://developer.nvidia.com/cutensor
[55] https://github.com/jcmgray/quimb

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

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 Juelich 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)
SURFsara	Dutch national high-performance computing and e-Science support center, part of the SURF cooperative, Netherlands
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 has successfully initiated ten projects developing forward-looking software solutions. Eight of these projects started in April 2019 immediately, whilst two projects started in January 2020, after selection in a second call for proposals. This deliverable reports on the public release of prototype software by all projects. This early release of work-in-progress software guarantees software availability to the community, and provides the community with an opportunity to inspect, test, and provide feedback. All first phase projects have provided links to accessible code repositories such as GitHub, Bitbucket and similar. Projects typically use a modern development infrastructure, including version control, automated continuous integration (CI), and standard documentation formats. Testing includes correctness as well as performance. Whereas the readiness level of the projects differs, integration in user codes has taken place, first performance results have been included in this report, and certain codes have already become part of the procurement benchmarks of the EuroHPC Joint-Undertaking Pre-Exascale systems. The first phase of this work package can thus 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. This deliverable aligns the project teams of the first call by requiring a public prototype release of the software, as well as an update on the development infrastructure used, and invites the new project to do the same. This release helps to ensure that software sustainability is taken in serious consideration, using industry standard tools, issue tracking, continuous integration, validation and verification, documentation, etc. This document is structured per project, providing a brief introduction for each of them, references to the prototype software releases (i.e. public repositories), an overview of the development infrastructure, as well description of the planned short term work.

2 PiCKeX- Particle Kinetic codes for Exascale plasma simulation

2.1 Introduction and summary

Particle-in-cell (PIC) codes have now become an essential part of the modelling toolkit for many areas of plasma physics, whether for modelling particle acceleration with high-power lasers, or to understand detailed dynamics and transport processes near the edge – or scrape-off layer (SOL) – of magnetised plasma confinement vessels. The PIC algorithm relies on a highly versatile, robust, finite-difference discretisation of the Vlasov equation for the particle distribution function in coordinate and velocity space. State of the art three-dimensional PIC simulations involve up to 1012 particles on 106 cores, which generally requires careful management of the memory access and particle book-keeping to implement efficiently. As reported in D8.2, the PicKeX project focusses on two important community codes: EPOCH [1], a fully relativistic, electromagnetic model and BIT1 [2], a sophisticated PIC/Monte-Carlo model, both of which are heavily used in the laser-plasma and magnetic fusion communities respectively, but which both need heavy refactoring work to enable them to run effectively on future PRACE Tier-0 systems. The project partners at the Jülich Supercomputing Centre (JSC) and University of Ljubljana (UL) are exploring and implementing advanced algorithmic techniques such as task-based programming models, and dynamic load-balancing based on space-filling curves, to achieve this goal. Progress on these points is reported here together with information on an initial public release of the enhanced code versions.

2.2 Prototype software release

The main production version of the EPOCH code is maintained by the lead developer group at the University of Warwick. New users can access this repository on request at the following site [3], which includes well-maintained documentation, and a comprehensive database of issue-tracking dating back to the initial public release 10 years ago. The repository has been cloned on the JSC GitLab server as described below, with access to the prototype 'EPOCH-X' available via [4], after registration on the JSC LDAP system. The prototype version has the full functionality of the main EPOCH branch, but includes verified refactoring measures such as those highlighted below, along with selected test cases designed to probe particular performance issues.

The refactoring work of the BIT1 code was carried out on the new prototype code SIMPIC. This code and also the ongoing StarPU prototype are available on the following repository [5], which is mirrored at the JSC site at [6]. The latest BIT1 code release 18 has been imported into the following portal [7], for future refactoring based on the SIMPIC prototype.

2.3 Development infrastructure

The development on EPOCH-X takes place on JSC's internal GitLab infrastructure [8], which is automatically synchronised with the official repository at Warwick University [9]. The decision to use an internal server - rather than e.g. GitHub - is based on the need for control over the collaborative system, particularly with respect to benchmarking. For example, this choice will allow us to couple the CI system from the local GitLab with job execution on JSC's supercomputers to get reliable and reproducible performance measurements and enable larger tests on target pre-Exascale architectures. All testing is implemented using JUBE [10], the Jülich Benchmarking

Environment (to perform verification, automate scalability tests and compare results with various refactoring stages, libraries, compiler options and so on).

Results of benchmarking tests, performance milestones, discussions, issue reports and strategic development decisions will be documented within the JSC GitLab. Final EPOCH code improvements, once their correctness is confirmed via internal tests, will be pushed to the official public repository at Warwick University. Intermediate prototypes can be made available via JSC's GitLab as described above. The documentation for the first release SIMPIC code is available at [11]. All developments and discussions will be done in the Bitbucket infrastructure; final enhancements fed back into the main BIT1 developer repository held at IPP Prague.

2.4 Planned work for the next six months

The work plan for enhancing the EPOCH and BIT1 code performances on near-term and next-generation Tier-0 supercomputers was outlined in the previous deliverable. Overall at least a dozen major potential hotspots were identified and slated for refactoring work.

Prioritised steps for EPOCH-X code are:

- Complete verification and benchmarking redesign of the 'moving window' mode (for laser-electron acceleration simulation) to remove/mitigate data transport overheads
- Implementation of new improved dynamic load balancer utilising OpenMP
- Enhanced data reuse to drive down the ratio of stalled CPU cycles
- Expand MPI/OpenMP hybridisation to all main code models (particle integrator, field solver, current gather)

Planned activities for the BIT1 code:

- Testing the task-based parallelisation of the prototype SIMPIC code, that was done and adapted for D.A.V.I.D.E, on other HPC clusters;
- Major refactoring on the BIT1 code, including the task-based schemes tested within SIMPIC, using different pre-Exascale architectures.

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 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 explore task-based parallelism for plasma simulations and test ways to add support for GPUs or other accelerators to plasma simulation codes. The aim is to pave the way for the plasma simulations codes to be ready for the upcoming pre-Exascale systems.

3.2 Prototype software release

In the MoPHA project we are developing a number of different codes, some of which are hosted on their own repositories, but the main site for publishing prototype mini-apps and documentation related to the project is on GitHub [12].

GENE Solution of the 2D heat equation using task-based parallelism, the main purpose of this code is to get experience with task-based parallelism and StarPU for a Eulerian scheme similar to the approach used by the GENE code. In this implementation, the 2D Laplacian in the heat equation is treated as a convolution operation which is split into tasks that can be executed by the CPUs or GPUs and are scheduled using StarPU. The computational domain of the heat equation is block-partitioned according to the number of MPI ranks available. Each domain exchanges ghost cells with its neighbours to make each sub-domain data independent. Each cell exchange is considered a task and is also scheduled with StarPU. The implementation is done in C++ using templates and the STL library and has been tested with StarPU 1.2.9. Serial implementations to test correctness of convolution and the solution of heat equation are also included [13].

VLASIATOR Experimental Vlasiator version with partial support for GPUs using OpenACC directives: Implemented a set of the main computational algorithms, namely the velocity space acceleration update, for offloading to the GPUs using a directive-based approach. Initial results are promising, but further improvements are needed to optimise data movement between host and device memory. The code is available as a separate branch in the main Vlasiator git repository [14].

SYMPIFE-VMAX / ELMFIRE Mini-app for particle-in-finite-elements of 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 [15].

STRUGEPIC 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 [16].

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 codes. Current status of the prototype describes how to refactor a MPI code into a task-based application by introducing "codelets" first for CPU and later can selectively introduce GPU codelets. Different scheduling mechanisms were also tested with the mini-app. SIMPIC is available from a separate repository on Bitbucket [17].

3.3 Development infrastructure

GENE: The main GENE code is developed on a GitLab repository (hence version-controlled via git) and makes use of the GitLab CI functionality. Compilation tests with GNU, Intel, PGI and Cray compilers are regularly done, unit tests are run and two test sets are done for each commit. On a daily basis, larger tests are run on GitLab runners on different machines. The prototype test code, that uses the heat equation as basis and helps in testing and understanding the usage of StarPU, is also versioned with git and published on MoPHA's GitHub page. Documentation of the prototype code is done inside the code and online README files in markdown format.

VLASIATOR: Vlasiator is developed using the git distributed version control software and GitHub for tracking issues and for managing contributions from the community. Vlasiator includes an integrated test package and uses CI runners for automatic compilation tests. Documentation is provided as a part of the source code and as wiki pages on GitHub.

SYMPIFE-VMAX / ELMFIRE: The prototype is hosted at [18]. Building is managed using CMake. Documentation will be expanded in the future.

SIMPIC: The development and the documentation of the prototype code makes use of the Bitbucket infrastructure that is similar to GitHub in functionality providing all means of collaborative tools. Repositories are open to the public and CI is used for documentation building and simple tests.

3.4 Status and outlook Planned work for the next six months

GENE: With the prototype code, first insights into the usage of StarPU tasks has been collected which are now used to rewrite GENE in a way to use these tasking model. An existing cache-blocking loop in the calculation of the right-hand side of the Vlasov equation is to be taskified as a first step. The different blocks are computed mainly independently and suits therefore well into the tasking approach. In a further step, the different terms of the rhs computation will be transferred into separate tasks with the interdependencies taken care of. Dependent on the progress of the taskification, we might also write codelets for running the tasks on a GPU.

VLASIATOR: Initial results with the experimental version that uses OpenACC to offload some of the solvers to GPUs were promising, but performances were sub-optimal due to overheads from data movement between CPU and GPU memory. In order to improve performance, further work is needed to optimise the data movement and/or to refine the data structures used. If possible, the support for GPUs should also be extended to cover more solvers.

SYMPIFE-VMAX / ELMFIRE: Development of a GPU version, leveraging native MFEM GPU support and the associated libCEED. Development of fusion-specific inputs, diagnostics and test cases. Implementation and testing of a guiding-centre geometric integrator for reduced electron dynamics in strong magnetic fields. Expansion of the documentation.

STRUGEPIC: Investigate what kind of GPU functionality AMReX can provide.

SIMPIC: Finalisation of the SIMPIC mini-app with benchmarking on different cluster architectures and integration of the CI with the StarPU simulator.

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

4.1 Introduction and summary

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. 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 2021. 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 Prototype software release

The core functionality of NB-LIB is the ability to calculate forces and energies for multiparticle systems. For the prototype release, we have targeted the ability to compute forces for a Van Der Waals gas, such as argon, and return these forces to the user. This goal has been achieved with the prototype code at [19]. In order to complete this task, a two-pronged approach has been utilised. The first task has been to design an API specification that will allow users to programmatically specify simulation systems. Specifying an API that is flexible enough to accommodate as yet unforeseen use cases is a rather large task in the field of particle simulation because of the variety of different functional forms that intra- and intermolecular interactions can take. For the NB-LIB API we have ensured that all currently supported non-bonded interactions in GROMACS are also supported by NB-LIB. In order to ensure extensibility, we have also been in conversation with the OpenMM molecular simulation software developers as well as the developers of the Open Force Field Toolkit. The result is that the NB-LIB particle topology and system setup API functionalities are in-principle compatible with these other open source simulation codes. In addition to developing the API for setting up particle systems, we have defined API functionality for computing forces and updating coordinates on these systems. The result is the ability to write self-contained particle simulation codes in a matter of minutes using the NB-LIB API. The second aspect of development efforts has been refactoring the GROMACS code-base so that the translation layer between NB-LIB and GROMACS is minimised. This has already led to many patches which streamline GROMACS internal data flows and interfaces being merged into the master branch of GROMACS.

4.3 Development infrastructure

The public prototype release is available for inspection at [19]. The API is self-documented with Doxygen and there is also a short overview of the core functionalities at [20]. All non-trivial functions have tests and integration tests also ensure that the various parts of the API work together as well as with the GROMACS backend. Some aspects of GROMACS functionality that are not currently possible to test within GROMACS but that the API depends upon are also tested. Once NB-LIB is moved to the main GROMACS repo these tests will move out of the NB-LIB testing infrastructure, which, like GROMACS utilises the Google test framework. All pull requests require approval of two reviewers to be merged and go through the same testing pipeline as used by GROMACS. To communicate between the various development locations (Lugano, Stockholm, and Zürich), a number of strategies are used. Besides comments in code review, there is also a Slack channel used for daily stand-up and design discussion threads. In addition, discussions about design and distribution of tasks takes place in a weekly video conference. Perhaps the most crucial aid in the development of NB-LIB has been regular week long hackathons to rapidly iterate on API designs and implementations. These have taken place in Zürich in November, in Stockholm in January jointly with the developer of the GROMACS modular simulator, and in Lugano in February. The planned April hackathon in Stockholm, in collaboration with the GMXAPI developer, has been cancelled due to the Coronavirus outbreak.

4.4 Planned work for the next six months

The development efforts of NB-LIB for the next six months will continue to pursue a two-pronged approach of working on API specification and implementation while also refactoring GROMACS to minimise translation layers between the NB-LIB API and the GROMACS back-end. On the API specification and implementation front, work is planned on adding other types of particle-particle interactions, such as bonds and angles, to the system setup functionality. It is also planned to add the ability to return energies to the user, in addition to the currently available forces. Most work over the coming period will be focused on GROMACS refactoring. The largest share of effort will be spent on reworking data flow models related to non-bonded force calculation within GROMACS. This will in turn require some effort to be spent refactoring simulation system initialisation in the main MD loop within GROMACS. This simulation system initialisation work will proceed in parallel to, and in conversation with, similar ongoing efforts by the GMXAPI developers, as well as work in the core GROMACS team. This refactoring will also be needed to allow NB-LIB to utilise the parallelised, heterogeneous compute capabilities of GROMACS. One final target for the next period is to migrate NB-LIB into the main GROMACS code base. This means that NB-LIB will be available on HPC systems all over the world as well as Linux repositories for Ubuntu and Debian. Finally, we plan to step up dissemination efforts over the coming period. It had been planned to present NB-LIB at relevant conferences in May and July, but these have been cancelled due to the Coronavirus outbreak. Efforts will be made to find suitable opportunities to broadcast the existence of NB-LIB once scientific meetings resume.

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. These techniques include:

- Using OpenMP/OmpSs-2 tasks with data dependency clauses. This includes not only expressing computation as tasks, but also communication, by wrapping MPI or GASPI library calls inside tasks. We utilise the Task-Aware MPI (TAMPI) and Task-Aware GASPI (TAGASPI) interoperability libraries developed by Barcelona Supercomputing Center to make this as efficient as possible.
- MPI single-sided communication. Recent developments in MPI libraries have significantly improved the performance of single-sided communication to the point where its benefits can be realised in real applications.
- GASPI single-sided (put-notify) communication. This is a lightweight alternative to MPI single-sided communication which interoperates well with MPI, and offers different synchronisation semantics which can help remove serialisation constraints.

The project is initially implementing these techniques in small kernels and mini-apps, with the aim of moving to key kernels of larger applications later in the project. In addition, development is being carried out on runtime library implementation to support this work. This includes:

- Continuing the development of the TAMPI and TAGASPI interoperability libraries to support interaction of MPI and GASPI with OpenMP/OmpSs tasks with dependencies.
- Exploring extensions to the OpenMP tasking model to support task dependencies on external events, task-nesting, fine-grained dependencies, weak dependencies and early release of dependencies, to avoid artificial synchronisation and serialisation effects.
- Making use of performance analysis tools and techniques to identify optimisation targets in real applications where these techniques can be most beneficially applied.

5.2 Project Prototype software release

The prototype software currently available consists of the OmpSs-2 programming environment and the TAMPI library.

The OmpSs-2 programming environment is comprised of the Nanos6 runtime library and Mercurium compiler, which are available for public download from [21], either in the form of packaged versions of stable releases or as Git repositories of the current development versions. Recent versions of OmpSs-2 contain the required support for the TAMPI library, for example the ability to pause tasks which are blocked in MPI calls, poll for completion of the relevant MPI operations, and resume the task when completion occurs.

The TAMPI library uses the PMPI interface to intercept MPI calls, interact with the Nanos6 runtime, and call the required actual MPI routines. It also contains a small number of extension to the standard MPI interface which allow non-blocking MPI calls inside tasks to be handled correctly

integrates their completion with the task dependency system. The TAMPI library is available from a Git repository at [22].

5.3 Development infrastructure

Mercurium, Nanos6 and TAMPI use Git as a version control system. There is an internal GitLab server that hosts the master and development branches for all three projects. The master branch of each project on the internal GitLab server is also mirrored on a public GitHub repository, that is updated twice a year with a new stable release or for hot-fixes.

The three projects have their own set of correctness tests that are periodically executed to validate any new development. All internal merge requests are automatically tested with Jenkins on four production machines (Marenostrum4, CTE-Power9, CTK-KNL and Nord3) before they can be merged on the master branch. Moreover, all merge requests are also reviewed at least by one additional developer. Once the internal review is completed the merge request is integrated into the master branch and it is automatically tested by Jenkins on four production machines.

There is also a framework developed in-house to continuously measure the performance of Mercurium and Nanos6 runtime. This framework runs a set of predefined benchmarks on the four production machines at night if the code of the master branch has been modified. The results of the benchmarks are automatically saved to a MySQL database and they can be visualised using Grafana. The project home page contains links to the online documentation, OmpSs-2 specification and OmpSs-2 examples [23].

5.4 Planned work for the next six months

The project is currently working on porting a number of kernels and mini-apps to the “taskified communication” hybrid programming supported by OmpSs-2 and TAMPI. Several of these (Gauss-Seidel, IFSKer, HPCG, HPCCG) are substantially complete, and undergoing performance analysis and testing. Others (Co-MD, miniMD, miniAMR, LULESH) are still in progress. Porting work on this latter set will be completed, and the performance analysed. Further target mini-apps and larger codes will be identified. We will synthesise our porting experiences into a best-practice guide to assist other developers who wish to test out this model.

Work on a release of the TAGASPI library has been delayed by a short time: this is now ready for testing (prior to a prototype release) which will begin very soon. We also plan to release an OpenMP runtime library (based on LLVM) which supports the non-blocking mode of TAMPI.

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 [24], to support highly scalable solution of sound scattering and harmonic analysis problems. Distributed parallelisation of the library is based on the FETI (Finite Element Tearing and Interconnecting) domain decomposition method and the implementation for problems in complex domain will be based on the FETI-H (FETI-Helmholtz) method and its variants where the regularisation is done using the complex interface mass matrix and the preconditioning is based on the plane wave deflation. Refactoring and optimisation of the existing ESPRESO code is also an important part of the project.

Within the first 12 months, we focused on several problems based on the original project schedule:

- Refactoring and optimisation of ESPRESO: this includes refactoring of the global matrix operations, new interface for external solvers and mesh partitioners (such as Pardiso, SuperLU, Watson Sparse matrix Package, HYPRE, ParMetis, or PT-Scotch), optimisation of the parallel input workflow, or redesign of the ESPRESO configuration file.
- Optimisation of the system matrix assembler: replacing BLAS routines by manually tuned code for small matrices provided significant speedup of the assembler of the heat transfer matrices in shared memory.
- Development of the MATLAB prototyping application: we have successfully implemented and tested the FETI-H within our in-house fast-prototyping code (this includes matrix regularisation and preconditioning by the artificial coarse space).
- Distributed memory parallelisation of the harmonic analysis solver within ESPRESO: extended to support parallelisation in both frequency and spatial domain (limited scalability due to lack of preconditioner).
- Testing and documentation: project's private and public repository have been established and continuous integration is in progress.

In addition, a development of the "Solver-as-a-Service" platform has continued. This platform will enable users with limited or no experience with HPC to use ESPRESO to access supercomputer resources via an online platform. The backend has been developed and the implementation of frontend is in progress. We have also started with the GPU acceleration of the system matrix assembler and solver.

6.2 Prototype software release

The developed code is publicly available within the master branch of the ESPRESO library in its official repository at GitHub [25]. Currently, the code supports solution of the harmonic analysis problems in real domain parallelised across both frequency and spatial domains. However, the parallel scalability is limited due to the lack of a suitable preconditioner (implementation of the preconditioner based on an artificial coarse space is in progress). Most of the above-mentioned code developed in the first year is available within the repository. The public repository also contains the ESPRESO installation manual. The solver-specific documentation and tutorials will be provided in the next phase of the project.

6.3 Development infrastructure

ESPRESO uses git as a version control system and an internal GitLab infrastructure is dedicated for the development of the library at IT4Innovations National Supercomputing Center, available at [26]. The repository contains the C++ code, benchmarks and tests, and short installation documentation. Waf is used as a build automation tool and for testing, we use the Nose framework [27]. The git repository consists of two major branches – dev and master; the main development takes place in the dev branch and implemented features are merged into the master branch. The GitLab continuous integration (CI) pipelines then ensures the code is properly tested and eventually pushed to the public repository.

The public repository is located at [28], and contains the latest stable version of the code and documentation. The documentation is written using the MarkDown language and currently describes mainly the installation procedure and description of API for calling from external software. Additional information is available at the official product webpage [29].

6.4 Planned work for the next six months

Several topics have to be tackled in the following six months:

- Implementation of a preconditioner based on the artificial coarse space projection within the ESPRESO library. This will further improve the parallel scalability of the harmonic analysis code and enable solution of spatially large problems.
- GPU acceleration – the acceleration will rely on assembling the so-called local Schur complement matrix on GPU and replacing the sparse direct solver by iterative solution with smaller dense matrix.
- Documentation will be extended to provide additional solver-specific information.
- Development of the frontend of the “Solver-as-a-Service” platform will continue.

Development of the boundary element interface has been postponed in order to focus on the currently more important finite element code.

7 Performance portable linear algebra

7.1 Introduction and summary

In general, linear algebra algorithms have 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. Dense eigenvalue problems are currently solved mainly using the ScaLAPACK [30] or ELPA [31] libraries. ScaLAPACK has been developed in 1992 and 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 [32], that can replace ScaLAPACK in scientific applications.

An alternative strategy in the development of an eigensolver is to leverage on well-known and well-established iterative algorithms such as subspace iteration. A modern example of such algorithm has recently been implemented in the Chebyshev Accelerated Subspace iteration Eigensolver (ChASE) library. 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

The DLA-Future [33] and DLA-Interface [34] projects are both available at GitHub. Currently DLA-Future functionalities include the Cholesky decomposition and the solution of the triangular system of equations for distributed multi-core systems. Performance tests (Figure 1) show that our implementation performance is in-line with the performance of our initial prototype and the state-of-the-art libraries. Moreover, it performs better than two highly optimised ScaLAPACK implementations.

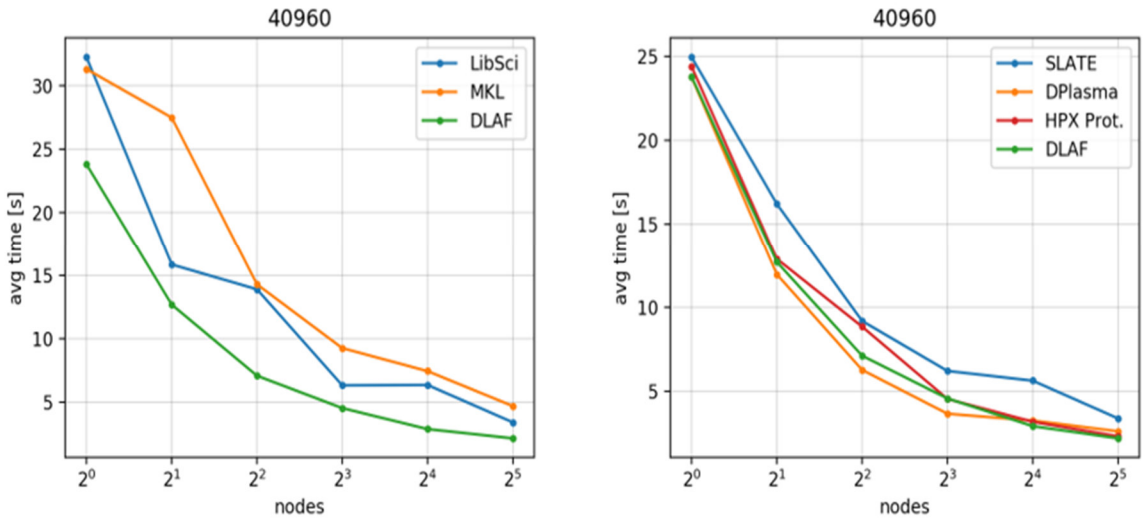

Figure 1: Comparison of the performance of DLA-Future Cholesky decomposition compared with other libraries (left: ScaLAPACK implementations, right: other task-based approaches).

7.2.2 Chase library

The Chebyshev Accelerated Subspace Eigensolver (ChASE) library is a modern and parallel library to solve dense Hermitian (Symmetric) algebraic eigenvalue problems. ChASE is templated for real and complex numbers and can be used to solve real symmetric eigenproblems as well as complex Hermitian ones. ChASE algorithm is designed to solve for the extremal portion of the eigenspectrum. By default, it computes the lowest portion of the spectrum but it can compute as well the largest portion. The library is particularly efficient when no more than 20% of the extremal portion of the eigenspectrum is sought-after. ChASE is particularly efficient when dealing with sequences of eigenvalue problems, where the eigenvectors solving for one problem can be used as input to accelerate the solution of the next one. The library can be used both in single and double precision. The distributed version of ChASE is parallelised over MPI and can be effectively used on multi- and many-cores. The latest release of the library can be executed on multi-GPU devices per computing node. ChASE is available in Github at the following link [35].

7.3 Development infrastructure

7.3.1 DLA-Future and DLA-Interface

To manage the development and the versions of the code we use a git repository which includes a master branch containing the latest version of the code that has been reviewed and tested, and other branches which contain changes that have not been reviewed (or for which the review is in progress).

The development follows the GitHub workflow [36], and the possibility to merge is blocked until the pull request is approved by at least another member. The GitHub workflow method also simplifies the integration with a CI method. A Jenkins instance is running at CSCS [37], and is setup to automatically build and test each pull request open in GitHub. Currently the results of the CI tests are not available to the general public.

The code is documented in two ways. The public API has an inline documentation formatted according to the specifications given by Doxygen. The doxygen tool can be used to extract this inlined documentation and produce a browsable version of the API documentation.

The documentation of other aspects of the library (e.g. installation procedure) is available through markdown documents.

7.3.2 Chase library

The ChASE library development is organised around a double repository level. The first level is a Gitlab repository hosted on a GitLab server located at the Juelich Supercomputing Centre (JSC), and a second level hosted by GitHub where the library is publicly available. The GitLab repository is used exclusively by the developers of the code. It includes a simple Continuous Integration (CI) system and two main branches, devel and master. When a major development is completed on the devel branch it is merged to the master branch and undergoes a series of automatic testing by the CI on a server locally based at the JSC. Once a development, merged to the master branch, is stable and thoroughly tested, the GitLab master branch is synchronised with the GitHub master branch and it is released to the public.

ChASE online documentation is developed using the Sphinx platform and hosted at the web site [38]. The documentation is split in two sections: User and Source. Currently only the User section is complete, while the Source section is under development.

7.4 Planned work for the next six months

7.4.1 DLA-Future and DLA-Interface

- Complete the inter-node communication optimisations.
- Re-implement the GPU accelerated Cholesky prototype using the DLA-Future API.
- Continue with the implementation of the routines needed by the Hermitian eigensolver.

7.4.2 Chase library

One of the most important kernels in ChASE is the distributed **Ge**neral **M**atrix-**M**atrix multiplication (GEMM) operation. A benchmark of GEMM based on HPX has been developed, including the task-based implementation in the node, and distributed implementation of SUMMA (**S**calable **U**niversal **M**atrix **M**ultiplication **A**lgorithm) with the overlap of communication. In the next six months, the task with the highest priority is to finish and submit a conference paper related to this topic. Compared to ScaLAPACK, the Elemental package provides a different layout for the distribution of data across the nodes, which can separate the algorithmic block size and the distribution block size of matrix. We have interests on Elemental data layout since it makes the tuning of algorithmic block size possible without redistribution of matrix. The second task for us is to investigate the possibilities of a combination of Elemental and HPX, especially for GEMM. ChASE provides a class interface that abstracts the related numerical kernels. With the help of this interface, the third task is to integrate the SUMMA-HPX kernel into ChASE as a new backend.

8 GHEX: Generic Halo-Exchange for Exascale

8.1 Introduction and summary

GHEX provides functionalities to perform halo-exchange operations in domain decomposed applications. While this operation is pervasive in HPC, its implementation is usually custom written by developers, who best know the logic of the domain decomposition. Typically, the implementation is done using MPI, but this approach has some drawbacks as the architectures become more diverse with different address spaces and synchronisation behaviours.

GHEX provides higher level interfaces to express halo-update operations to abstract the detail of the architectures and, at the same time, adapt to the logic present in the applications. An overview of GHEX can be found in Figure 2 below.

The main components of GHEX are pattern, communication object and transport layer. The user passes to the pattern the information about domain, the information about the halos and data wrappers to the data to be exchanged. This happens through user provided functions to gather the required information that is already available in the application. After this, the user associates sub-domains to communication objects that access the transport layer to perform the exchanges. This allows us to abstract different address spaces, such as GPUs, and transport mechanisms. To allow modern programming paradigms, like coarse grained multithreading (each thread manages a number of sub-domains), and multi-tasking, GHEX provides basic functionalities to interface with the most common threading mechanisms, such as OpenMP, pthreads and C++ threads. More can be added as necessary.

Key features of GHEX is the future-based halo-exchange mechanism, and the call-back message exchanges. These features enable overlapping communication and communication at API level, so that, if the platform allows for it, it can be automatically used. The low level transport layer can also be accessed directly to perform point to point communications. This feature will be used by DISPATCH, an application to simulate solar atmosphere.

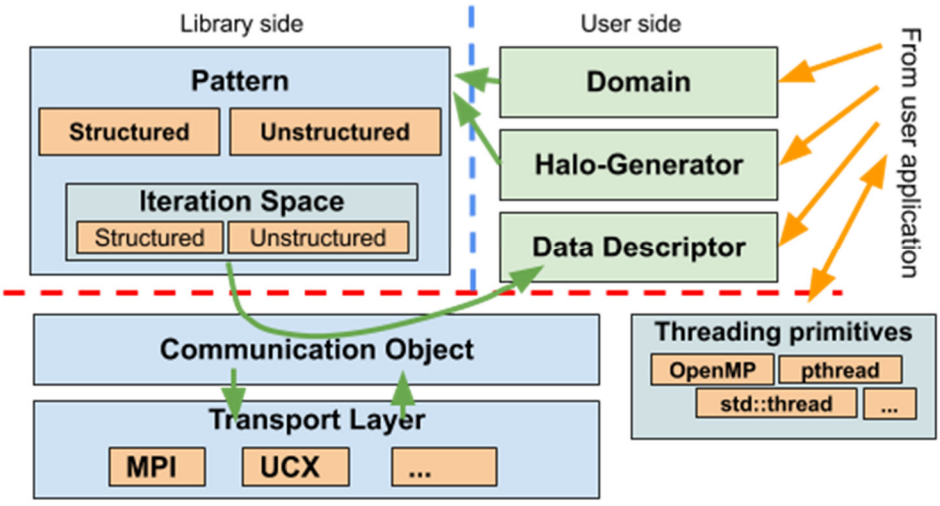

Figure 2: Overview of GHEX

8.2 Prototype software release

GHEX is available on GitHub at [39]. The license under which the software is provided is a BSD-3-Clauses. The current releases are not tagged, since the software has not reached the level of maturity for a version number (see the next Section for status of CI and testing). The master branch is anyway tested and should always be working. The repository also offers Fortran bindings to the C++ function to ease the use of the communication operations from Fortran applications, called FHEX.

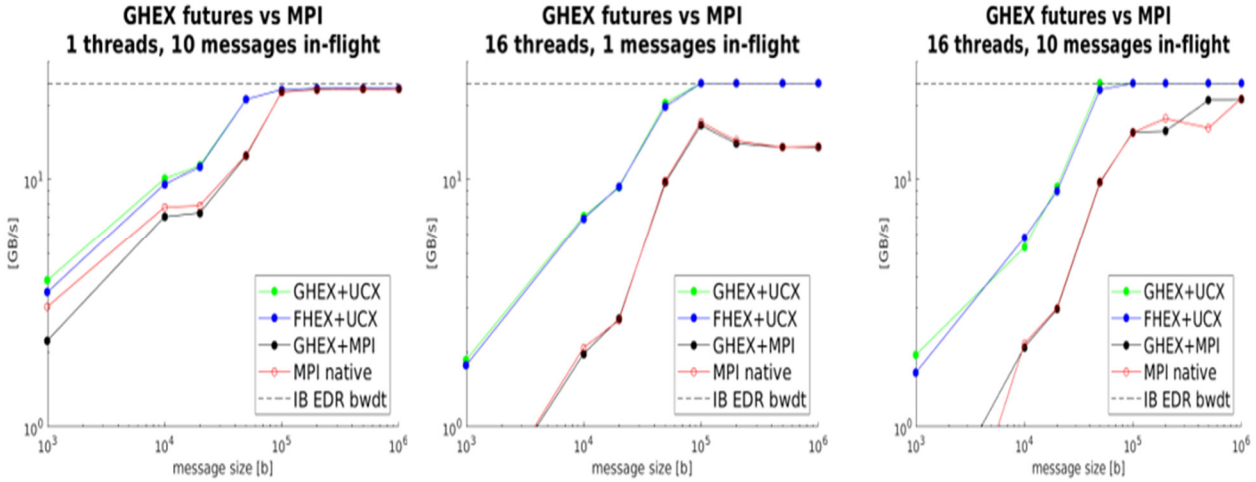

Figure 3: Comparison of GHEX and MPI transport layers for different sizes of messages on Infiniband network.

GHEX includes the Atlas library (developed at ECMWF, which provides domain decomposition and communication for unstructured grids) allowing the definition of adjacency relations to define unstructured meshes. Early results, shown in Figure 3, show that GHEX interfaces allow it to outperform the native MPI exchanges provided in Atlas. Figure 4 below shows the execution times for meshes with 1M nodes and up to 64 compute nodes. The relatively small sizes of the tests are due to limitations of recently added support for GPUs to Atlas. GHEX can handle bigger sizes correctly, but cannot be currently compared for performance against the main implementation. GHEX APIs allow for the transport layers to better manage communications in comparison with traditional MPI-only solutions.

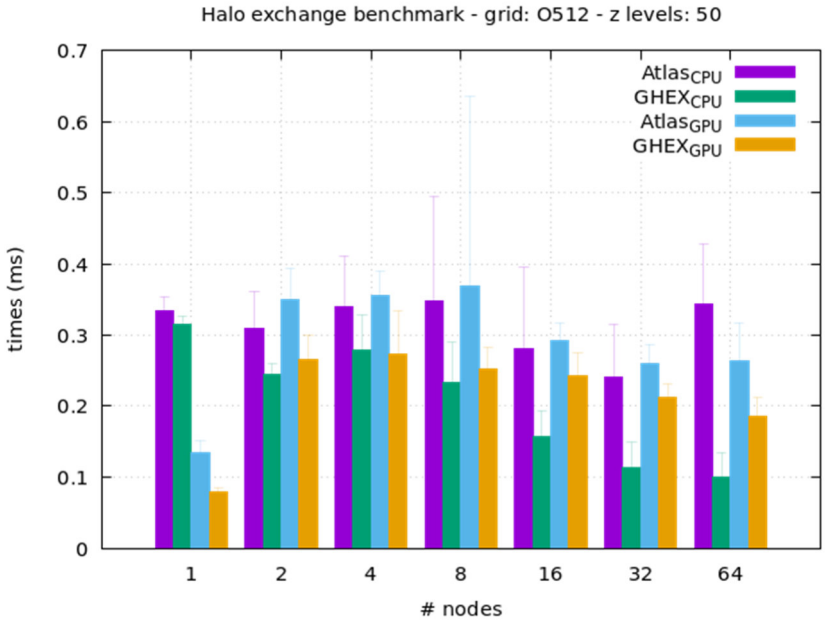

Figure 4: Execution times for meshes with 1M nodes and up to 64 compute nodes.

Even though GHEX has not been released officially, it is already employed in proof of concepts implementations of user applications such as the COSMO model and ECMWF's Atlas library mini-apps. More importantly, GHEX is used in official benchmarking suites for the next generation pre-Exascale system as the communication backed of the GridTools Benchmark (GTBench), which serves as a representative mini-app for weather and climate simulations.

The GHEX repository also provides benchmarks that can be used for comparing performance. In Figure 3 you can see low-level transport layer experiments showing the performance of different GHEX transport layers compared to native MPI. As it can be seen, the UCX backend is superior to MPI especially when the message sizes are not very big (which is a common use case for our target users), and when many threads are used. Also visible is the absence of overhead for the Fortran bindings (FHEX).

8.3 Development infrastructure

Before merging into the master branch, each contribution is validated against the tests, and merged only if the tests passes. Most of the testing is done manually, at the moment, but a basic CI infrastructure using GitHub-actions is in place for functionality tests. The reason why we consider this minimal at the moment, is that GHEX is designed to run on different architectures and different transport layers, and not all transport layers are supported equally on different architectures (for instance UCX is currently not optimised on Cray machines). For this reason, we are currently running manually the tests on different machines, but we plan to select the suitable CI solution that allows us to test GHEX automatically on multiple platforms.

GHEX is copyrighted to ETH Zurich. Contributions to GHEX are welcome, through the mechanism provided by GitHub issues and pull-requests. Contributors are required to sign a Contributor License Agreement to release the copyrights of the changes to ETH. This guarantees the stability of future releases of the software and a clear point of contact for support.

8.4 Planned work for the next six months

In the next six months the GHEX team will be engaged on multiple fronts:

- Develop a transport layer based on libfabric, a low-level network level natively supported by several vendors, including CRAY.
- Develop the facilities to perform halo-updates on inflated-cube spherical grids.
- Finalise the Atlas implementation and provide facilities for domain decomposition based on commonly used libraries, such as ParMETIS.
- Use GHEX in applications such as BIFROST solar atmospheric simulation, and others.
- Improve on continuous integration testing to increase automation for functionality and performance tests.
- Perform more through benchmarking on different platforms and transport layers.
- Expand the benchmarks.

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

9.1 Introduction and summary

The project, Linear Algebra, Krylov methods, and multi-grid API and library support for the discovery of New Physics (LyNcs), is addressing challenges which are arising on modern and upcoming architectures due to massive parallelisation. These challenges can only be met with disruptive approaches to parallelism because traditional parallelisation strategies for solving partial differential equations are no longer effective. LyNCs is targeting efficient solutions for linear systems based on large sparse matrices by pooling together software development efforts across Europe. This will provide the European communities with the next generation of parallel libraries for solving sparse linear systems at the Exascale. LyNCs 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 Center (LRZ). Part of LyNCs is the development of an API that is targeting massive parallel machines to perform simple task management with shared memory among huge parallel partitions. This API together with the implementing of 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 utilise the upcoming pre-Exascale 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 Prototype software release

LyNCs is targeting software which spans all levels of the scientific software stack to ensure readiness for the upcoming massively parallel pre-Exascale and Exascale systems. Part of the developed software are:

LyNCs: The first prototype version of the API LyNCs is available at GitHub [40], and published under a BSD 3-Clause license. The prototype version is targeting lattice QCD applications utilising DDalphaAMG as a solver library. The API is written in Python and based on the package dask, which enables simple memory shared task management especially designed for large allocations for the next generation of High Performance Computing Systems. The functionality of the prototype is currently limited to benchmark applications that target data exchange between smaller partitions and checks enabling the call of library functions written in C or C++ using python environment.

Fabulous: Fabulous implements Block Krylov solver methods, is written in C++ and its development is ongoing. During the first year of LyNCs new capabilities enabling flexible preconditioner were enabled through the implementation of the block GRC with inexact breakdown detection. The study of a novel Block Krylov solver has been initiated that would allow to recycle spectral information between sequence of multiple right-hand-sides that would fit very well the context of the coarse grid solve in multigrid; the new method is a flexible Block GCRO technique with deflation at start and restart. Fabulous is distributed under CeCill license and available at the

INRIA GitLab [41]. The GitLab page for INRIA linear algebra software packages (Chameleon, Fabulous, Maphys), using continuous integration, issue tracking, unitary testing, and complex scenarios testing. Integration is enabled through two high performance software distributions: spack [42] and guix-hpc [43].

DDalphaAMG with multiple rhs: In the first year of LyNCs a new version of DDalphaAMG is developed, which enables multiple right hand side, where the vectors are ordered with row major ordering. This enables vectorisation during compilation without explicitly using vector instructions. This guaranties portability without major performance lost to different CPU architectures, like ARM, Intel or AMD CPUs. The first version is available at GitHub [44], and is currently tested on various systems, such as the SuperMUC-NG, the Mont Blanc 3 system Dibona and the up-coming HAWK system. As shown in Figure 5 below, the multiple right hand sides is extending also the scalability which makes it additional suitable for massive parallel machines.

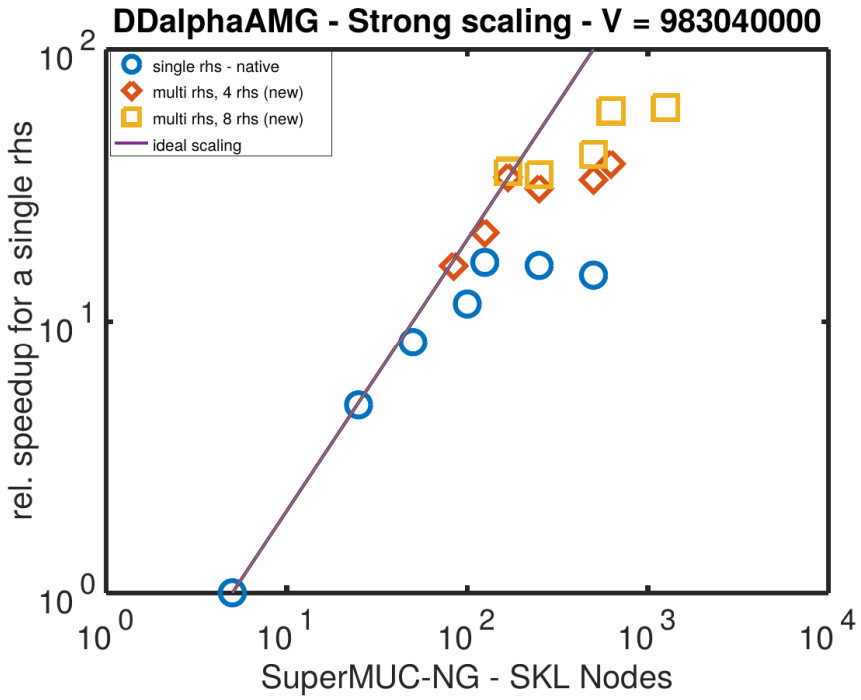

Figure 5: Comparison of scalability in the range from 5 to 1200 nodes of DDalpaAMG on SuperMUC-NG using a physical point lattice of lattice size V=80x80x80x160 using 1 (blue circles), 4 (red, diamonds) and 8 (yellow, squares) right hand sides.

Librsb: The librsb library is written in C99 and OpenMP, and is distributed under a GPLv3 license at [45]. Since the start of the LyNCs project, librsb is being further developed. This resulted into a new release with minor bug fixes, and considerable bug fix/refactoring/documentation. Namely, the internal test suite has been expanded very notably, and fixes have been found necessary to adjust inconsistencies. Tightly related to these consolidation activities are developments to the Python [46], and GNU Octave [47] access layers, which are of interest to diverse user communities. Interaction with them has led to many improvements not only in connection to the aforementioned 'pyrsb' and 'sparsersb', but to librsb itself. 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 January, librsb is being included in the "Spack" HPC software distribution.

9.3 Development infrastructure

As previously mentioned, LyNCs is working on a diverse software stack targeting all different levels. This diversity splits the effort within the LyNCs project into various tasks, e.g. like a high-level API for which we are employing a novel framework or the low level sparse BLAS library. This diverse effort is reflected in the development infrastructure that is complement by the strength of each partner. Namely, the tasks which target the community library DDalphaAMG is led by CaSToRC while INRIA acts as an advisor in order to exploits most efficient block-Krylov solver solutions. Additional tasks, which are developed in cooperation with projects partners are: i) testing of librsb for QCD kernels (LRZ, CaSToRC), ii) LyNCs API software development for simplified memory shared task management (CaSToRC, LRZ, INRIA), and iii) pipeline version of Block-Krylov solvers (INRIA, CaSToRC).

LyNCs has established effective communication channels via regular telcons, usually once per month, and an email-list, which is used to coordinate actions among all involved members. For every single task, each partner is connected via mail, phone calls and telcons. For all software developments git is used as a version control and software is published via GitHub, INRIA GitLab and sourceforge. For software development in which two or more partners are involved, we established a GitLab-side.

Most of LyNCs personnel positions had to be filled via recruiting, which introduced a delay of several tasks. Although this delay is mitigated by shifting priorities, we still expect some further delays of subtasks. Unfortunately, this is made worse due to the current situation connected to the pandemic of COVID-19. In particular, we are currently re-considering the planed face-to-face meetings.

9.4 Planned work for the next six months

The next step of LyNCs is connected to its second milestone MS2, which is the evaluation of prototype methods for Exascale. This will be met for block-Krylov solvers in multi-grid preconditioners and librsb used for QCD kernels. In summary, up-coming developments are as follows:

- Extending and enabling further features in the LyNCs API, which includes testing on PRACE Tier-0 systems and modular architectures, such as the DEEP-EST cluster to which we have secured access.
- Investigating numerical schemes based on block pipelined Krylov subspace methods, like pipelined GMRES and possibly implement the flexible block GCRO-DR method.
- Integration and testing the Fabulous block Krylov solver methods in DDalphaAMG, both for the solution of the coarse grid as well as Krylov solvers preconditioned by multigrid at the finest level.
- Performance of preliminary scalability experiments to assess the algorithmic choices.
- Enhancing librsb's test suite further towards the next release, so that it may be used by e.g. Travis CI on GITHUB
- More far-reaching optimisations (foremost on SpMM), preferably after the aforementioned librsb consolidation step.

10 ParSec: Parallel Adaptive Refinement for Simulations on Exascale Computers

10.1 Introduction and summary

ParSec brings together well-known HPC CFD practitioners with the aim of sharing best practices, and collaboratively modernise the AMR implementation of three leading-edge CFD community codes for the exploitation of future (pre) Exascale machines. The partners involved in the project are Barcelona Supercomputing Center (BSC), the KTH Royal Institute of Technology and Cenaero - Université de Liège. The community codes brought by these institutions are:

1. **Nek5000**, the scalable high-order solver for computational fluid dynamics from KTH/UIUC [48],
2. **Alya**, the high performance computational mechanics solver from BSC [49],
3. **Argo**, the high order multiphysics solver from Cenaero. In particular the mesh functionalities of Argo are supported by the OS library MAdLib [50], which will be further developed during the project, including a tight interface with the finite mesh generator Gmsh [51].

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. From this broad perspective, ParSec aims at reaching robust HPC solutions that can be useful for the entire community. The objectives of the project are:

1. Analysis of the various separation of concerns (SoC) used for the AMR implementation - that will allow consistent performance comparisons and software sharing amongst codes,
2. Cross-verification and analysis of the performance of the codes on (pre) Exascale architectures,
3. The modernisation of codes based on disruptive solutions motivated by the previous analysis,
4. Deliver self-contained OS software components solving different steps of the AMR process
5. Deliver three AMR-enabled CFD legacy codes to exploit (pre) Exascale systems.

Figure 6 below illustrates the work plan for ParSec. Since the project started in January 2020, it is on its very first steps. The logistics have been resolved, in particular, a mailing list and a common repository have been created and a monthly teleconference has been established. Apart from sorting out the logistics, the first three teleconferences have been focused on presenting the base-line of each code in terms of AMR capabilities, comparing the SoC implemented, and identifying components and know-how sharing opportunities that will take place in the course of the project. The developments have focused on the refactoring of codes to increase modularity and on enabling the interfacing between the already shareable components. Next steps will be focused on performance analysis, optimisation and implementation of new features.

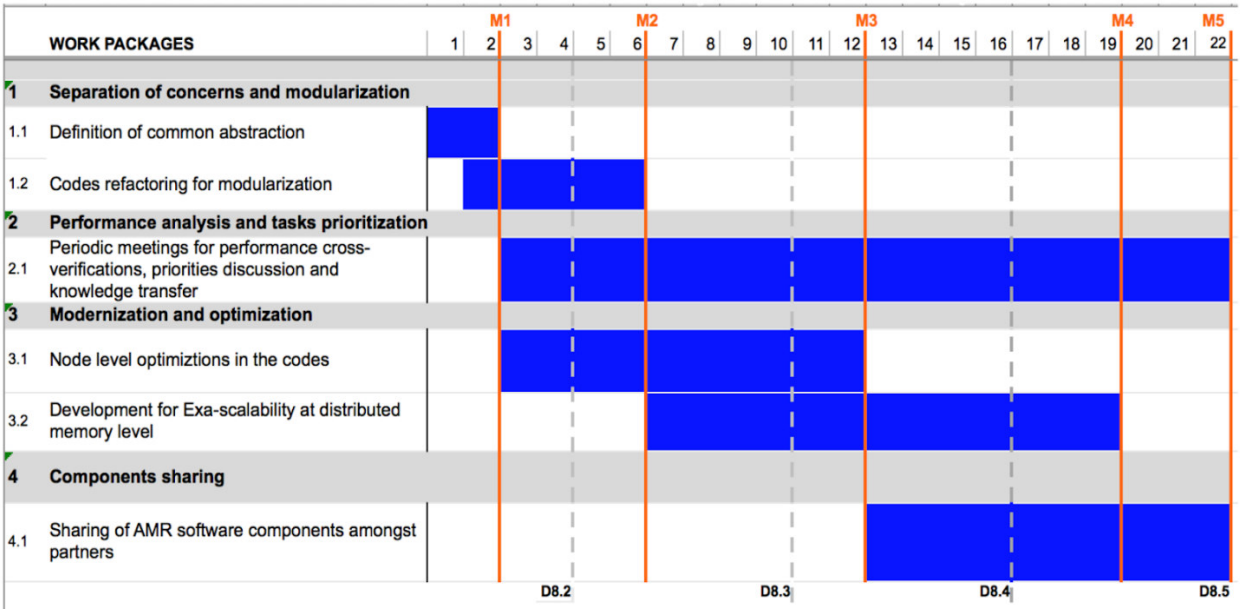

Figure 6: ParSec work plan.

11 QuantEx: Efficient Quantum Circuit Simulation on Exascale Systems

11.1 Introduction and summary

QuantEx is a platform combining modular quantum circuit simulation tools with the aim of running efficiently on heterogeneous compute platforms and scaling to exploit pre-Exascale and Exascale compute resources. Modern development practices and software design methodologies are used along with hierarchical layers of abstraction to encapsulate complexity and enable tools to be easily extended and integrated into users' circuit simulation workflows. The project started in January 2020 and the team consists of researchers at ICHEC and LRZ.

11.2 Evaluations

QuantEx will integrate and build on existing software components. To determine the most suitable to a number of evaluations of key software components have been completed and others are ongoing. Evaluations are concerned with assessing each software component for suitability of integration into QuantEx. The evaluation takes into account methods and technologies used, performance profile and release license. An overview of evaluations completed to date follows.

TAL-SH

TAL-SH [52] is a tensor algebra library for shared memory nodes which has been proven in production with its integration in qFlex [53], the simulator used for Google's quantum supremacy experiment. It is designed to be very flexible with support for multiple hardware back ends including multicore CPUs and NVIDIA GPUs through the use of third party libraries which include OpenBLAS and cuTENSOR [54]. Due to its asynchronous execution policy, it is able to efficiently utilise all available hardware resources within an individual compute node. Offering C, C++, and Fortran bindings, TAL-SH facilitates easy integration within QuantEx.

cuTENSOR

cuTENSOR is an optimised tensor algebra library for Nvidia GPUs of compute capability of 7.0 or higher, this includes Tesla V100s which are installed in HPC systems such as Summit and Kay. CuTENSOR has been shown to offer significant performance improvements over optimised CPU-based tensor algebra libraries such as TBLIS and is therefore an attractive option as a low level driver for the required tensor contraction primitives. Its integration within TAL-SH has also shown that the library is mature and production-ready. The library is very flexible allowing contractions to be executed using efficient algorithms, such as GEMM-like Tensor-Tensor multiplication and Transpose-Transpose-GEMM-Transpose, which may be selected at runtime either manually or automatically using integrated heuristics. It is worth noting that cuTENSOR is freely available however it currently lies behind a license agreement which requires an NVIDIA developer account.

qFlex

Flexible Quantum Circuit Simulator (qFlex) implements an efficient tensor network, CPU-based simulator of large quantum circuits. qFlex computes exact probability amplitudes, a task that proves essential for the verification of quantum hardware, as well as mimics quantum machines by computing amplitudes with low fidelity. qFlex targets quantum circuits in the range of sizes expected for supremacy experiments based on random quantum circuits, in order to verify and

benchmark such experiments. qFlex is enabled to run efficiently on GPUs with extensive optimisation. It is parallelised using MPI and CUDA. TAL-SH is used as a library within qFlex, but with improvements with respect to the standalone version.

Quimb

Quimb [55] is an open source pure python library designed for quantum information and many-body calculations. It has a submodule, called tensor, which has tools for creating and manipulating tensor networks, including a specialised class for quantum circuits. It uses the library opt_einsum to perform tensor contractions which is agnostic to the backend and can handle NumPy, Dask, PyTorch, Tensorflow, CuPy, Sparse, Theano, JAX, and Autograd arrays. It also has a slepc4py interface for easy distributed linear algebra. Integration of Quimb into the QuantEx project can be used to help find suitable contraction strategies for arbitrary tensor network graphs.

11.3 Design and use case development

Design for initial software release with exchange formats between layers has been defined. A prototype in development and expected to be completed by end of April.

Use cases are central to development and benchmarking of QuantEx. As such, sample workflows have been implemented for the quantum Fourier transform (QFT) and the variational quantum eigensolver (VQE). Both methods are written to leverage OpenQASM as the intermediate circuit description language allowing the use of IBM Qiskit to verify their functionality. A Julia package titled “QuantExQASM” was developed for generating circuits to realise these given use cases.

Additional use cases and algorithms were identified for potential implementation, including Grover’s search algorithm, superdense coding, and quantum phase estimation. All three can be used as building blocks for other methods, and are easily verified with existing frameworks. Bespoke solutions and use-cases in collaboration with stakeholder research groups are expected to follow in subsequent deliverables.

A stakeholder meeting was held on 6 March 2020 where project status was communicated and input was sought. Emerging from this were steps to engage more closely with stakeholders and get them involved in testing and evaluation of early prototypes.

11.4 Development infrastructure

For keeping track of project tasks and source code a GitLab repository is used. Compute resources at ICHEC are available to the project through ICHEC’s National Service and to those at LRZ through their involvement in the project. Docker images which can be deployed as CharlieCloud containers on SuperMUC-NG will be provided by a dedicated Docker-Hub with Jupyter Hub Interface for access to SuperMUC-NG. A development node with two high end GPUs (Nvidia Tesla V100) and latest Intel cores connected to SuperMUC-NG for interactive submission of jobs and access to the parallel file system.

11.5 Team and collaboration

The QuantEx team is made up of researchers from ICHEC and LRZ. A GitLab repository hosted at ICHEC is used for development code and the issue tracking features are used for tracking tasks

and project progress. Slack is used for ad-hoc communication with sync meetings held every two meetings and topic specific meetings organised as required. The staff involved with a rough estimate of their time commitment is listed in Table 1 below.

Name	Institution	Project role
Niall Moran (60%)	ICHEC (NUIG)	PI
Lee O'Riordan (30%)	ICHEC (NUIG)	contributor
Kenneth Hanley (30%)	ICHEC (NUIG)	contributor
John Brennan (100%)	ICHEC (NUIG)	contributor
Luigi Iapichino (50%)	LRZ	contributor
Ferdinand Jamitzky (50%) in-kind contribution	LRZ	contributor

Table 1: Staff involved in QuantEx project.

11.6 Status and outlook

The project is going well but a little behind the planned schedule due to delays in getting staff in place which will be made up over the next 1-2 months. It is planned that the initial design will be finalised and an early prototype in place by May. Iterations of profiling, benchmarking and improvements will follow this.

12 Conclusions

In this deliverable, we report on the status of the ten projects running under the WP 8 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. For each of the projects, we provided a brief summary of the project, the description of a prototype release, the development infrastructure and, in conclusion, the plans of the project on a short-medium term. The eight projects that started in April 2019 reported good progress and, in particular, all of them released a prototype in a public repository (GitHub, GitLab, etc.). Also, all projects implemented their work within a development infrastructure able to sustain the quality and maintainability of the software at a high level (i.e. continuous integration, issue tracking, integrated documentation, etc.). No critical issues in the management of these projects emerged so far, as it also appears from the results reported here. Only some minor staffing issues (i.e. recruitment) have slightly delayed the schedule of some of the projects, without leading to significant consequences. All the projects' plans for the short-medium term are in line with the objectives stated in their work plans. In some cases, it is noteworthy that work has already been integrated in existing scientific applications or have been used as benchmarks for the EuroHPC Pre-Exascale systems. The two "new" projects, which started in January 2020, reported in this document their internal structure and plans about their upcoming work, which we expect to be synchronised with the projects of the first phase in time for the next deliverables.