E-Infrastructures
H2020-EINFRA-2016-2017

EINFRA-11-2016: Support to the next implementation phase of Pan-European High Performance Computing Infrastructure and Services (PRACE)

PRACE-5IP

PRACE Fifth Implementation Phase Project

Grant Agreement Number: EINFRA-730913

D7.6

Provision of Numerical Libraries for Heterogeneous/Hybrid Architectures
Final

Version: 1.0
Author(s): Prashanth Kanduri, Victor Holanda Rusu, Raffaele Solcà, ETHZ (Switzerland)
Date: 15.04.2019

Project and Deliverable Information Sheet

PRACE Project	**Project Ref. №: EINFRA-730913**	
	Project Title: Provision of Numerical Libraries for Heterogeneous/Hybrid Architectures	
	Project Web Site: http://www.prace-project.eu	
	Deliverable ID: < **D7.6**>	
	Deliverable Nature: <DOC_TYPE: Report / Other>	
	Dissemination Level: PU / CO / CL*	**Contractual Date of Delivery:** 30 / 04 / 2019
		Actual Date of Delivery: 30 / 04 / 2019
	EC Project Officer: Leonardo Flores Añover	

* - The dissemination level are indicated as follows: **PU** – Public, **CO** – Confidential, only for members of the consortium (including the Commission Services) **CL** – Classified, as referred to in Commission Decision 2005/444/EC.

Document Control Sheet

Document	**Title:** Provision of Numerical Libraries for Heterogeneous/Hybrid Architectures	
	ID: D7.6	
	Version: <1.0>	**Status:** Final
	Available at: http://www.prace-project.eu	
	Software Tool: Microsoft Word 2013	
	File(s): D7.6.docx.	
Authorship	**Written by:**	Prashanth Kanduri, Victor Holanda Rusu, Raffaele Solcà, ETHZ (Switzerland)
	Contributors:	Prashanth Kanduri, Victor Holanda Rusu, Raffaele Solcà, ETHZ (Switzerland)
	Reviewed by:	Florian Berberich, JUELICH
	Approved by:	MB/TB

Document Status Sheet

Version	Date	Status	Comments
0.1	01/04/2019	Draft	Internal Review
0.2	11/04/2019	Revised Draft	Address Internal Review
1.0	15/04/2019	Final version	Ready for External Review

Document Keywords

Keywords:	PRACE, HPC, Research Infrastructure, Linear Algebra, GPU Computing, Molecular Dynamics, Particle Simulations, Exascale Computing, API Design, Software Architecture

Table of Contents

List of Figures

References and Applicable Documents

[1] http://www.netlib.org/scalapack
[2] https://elpa.mpcdf.mpg.de/software
[3] https://bitbucket.org/icldistcomp/parsec
[4] http://starpu.gforge.inria.fr
[5] https://bitbucket.org/icl/slate
[6] https://gitlab.com/PRACE-4IP/CodeVault/tree/master/hpc_kernel_samples/distributed_dense_linear_algebra/dla_interface
[7] https://github.com/eth-cscs/DLA-interface
[8] https://eth-cscs.github.io/Prace-5IP/
[9] https://eth-cscs.github.io/Prace-5IP/linear-algebra/
[10] https://eth-cscs.github.io/Prace-5IP/molecular_dynamics/
[11] https://www.sciencedirect.com/science/article/pii/S2352711015000059?via%3Dihub
[12] https://eth-cscs.github.io/Prace-5IP/molecular_dynamics/dev
[13] https://eth-cscs.github.io/Prace-5IP/molecular_dynamics/user

List of Acronyms and Abbreviations

aisbl	Association International Sans But Lucratif (legal form of the PRACE-RI)
API	Application Programming Interface
CPU	Central Processing Unit
DLA	Distributed Linear Algebra
EC	European Commission
ELPA	Eigenvalue soLvers for Petaflop-Applications
FMM	Fast Multipole Method
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
MD	Molecular Dynamics
MPI	Message Passing Interface
NBI	Non-Bonded Interactions
PME	Particle Mesh Ewald
PRACE	Partnership for Advanced Computing in Europe; Project Acronym
ScaLAPACK	Scalable Linear Algebra PACKage
WP	Work Package

List of Project Partner Acronyms

CaSToRC	Computation-based Science and Technology Research Center, Cyprus
CEA	Commissariat à l'Energie Atomique et aux Energies Alternatives, France (3rd Party to GENCI)
CINECA	CINECA Consorzio Interuniversitario, Italy
EPCC	EPCC at The University of Edinburgh, UK
ETHZurich (CSCS)	Eidgenössische Technische Hochschule Zürich – CSCS, Switzerland
GENCI	Grand Equipement National de Calcul Intensif, France
INRIA	Institut National de Recherche en Informatique et Automatique, France (3rd Party to GENCI)
IT4Innovations	IT4Innovations National supercomputing centre at VŠB-Technical University of Ostrava, Czech Republic
JUELICH	Forschungszentrum Juelich GmbH, Germany
KTH	Royal Institute of Technology, Sweden (3rd Party to SNIC)
NCSA	NATIONAL CENTRE FOR SUPERCOMPUTING APPLICATIONS, Bulgaria
PRACE	Partnership for Advanced Computing in Europe aisbl, Belgium
PSNC	Poznan Supercomputing and Networking Center, Poland
STFC	Science and Technology Facilities Council, UK (3rd Party to EPSRC)
SURFsara	Dutch national high-performance computing and e-Science support center, part of the SURF cooperative, Netherlands

Executive Summary

This task aims to develop low-level numerical libraries that would provide the necessary building-blocks to compose more complex scientific applications and workflows. Looking forward, it also focusses on enabling the required modularity so that scientific codes perform optimally on heterogenous/hybrid architectures of the Exascale era.

Two specific problems are targeted initially:

1. **Linear Algebra Libraries for Distributed/Hybrid Architectures**

Here, the objective is to accomplish a usable, distributed, (GPU) accelerated dense linear algebra library that can potentially minimize code rewrite on existing code bases via a common interface powered by a selection of compute backends. Many simulation packages are written with ScaLAPACK data structures. A new Distributed Linear Algebra (DLA) interface is proposed that is fully compatible with this data layout, but allows use of more efficient libraries such as D-Plasma for better performance.

This API aims to minimize code changes while improving performance of the Linear Algebra dominated parts of the package. Preliminary results show over 2x speed-up on GPU accelerated nodes as ScaLAPACK doesn't use GPUs. On multi-core nodes, there is an improvement of 10%. Both these studies include the non-negligible computational cost of data layout conversion.

2. **Non-Bonded Interactions (NBI) in Classical Molecular Dynamics**

In any molecular dynamics (MD) simulation, computation of the NBI between various particles overwhelmingly dominate the compute loads. There are several high-performance simulation packages, but none of those performant parts are available as reusable libraries. This is mainly because there isn't a unified interface to express n-body problems.

This task proposes an API, and attempts integration of significant parts of it into the community code GROMACS. This integration involves significant refactor to separate concerns along with multi-level APIs for integration of new features for developers, as well as for composing high level workflows for end-users. Sections of the API are slated for public release in GROMACS 2020.

This project explored the various challenges involved in defining the analogue of BLAS in the particle simulations space setting the foundation for concrete plans for transitioning GROMACS from the stand-alone package to a reusable library through another undertaking (NB-LIB project, PRACE 6IP WP8).

1 Linear Algebra Libraries for Distributed/Hybrid Architectures

1.1 Introduction

The de-facto standard library for distributed linear algebra is ScaLAPACK [1], a library that has been developed in 1995, when supercomputers were based on nodes which had a single CPU core. Since then, the node architecture has evolved; nowadays, supercomputers are built upon multi-socked nodes, multi-core CPUs, and accelerators. The parallelism approach of the ScaLAPACK library does not perform well on these new architectures and the corresponding algorithms have not evolved to keep up with developments in hardware or modern solving strategies.

Different libraries have been developed to address these problems. For example, ELPA [2] provides the implementation of the two-stage diagonalization algorithm which performs better than ScaLAPACK. On the other hand, DPlasma [3] and Chameleon [4] use a different approach for parallelization using a runtime system that does not have the limitation of the fork join approach used in ScaLAPACK.

Recently a new project called SLATE [5] (which was not available when this project started) has been created at the Innovative Computing Laboratory of the University of Tennessee. The ultimate goal of the project is to provide a replacement of ScaLAPACK, aiming to extract the full performance potential and maximum scalability from modern, many-node HPC machines with large numbers of cores and multiple hardware accelerators per node.

Every library has his own interface which in most of the cases is not compatible with the others. In some cases, the matrices have not even the same layout and have to be converted to be able to use a different library. The main goal of the DLA interface [6][7] is to provide a single interface compatible with ScaLAPACK that allows any supported library to be used.

1.2 Distributed linear algebra (DLA) interface

The DLA interface [7] is the library developed in this project that provide wrappers to different distributed dense linear algebra libraries (ScaLAPACK, DPlasma, ELPA) which has been developed to fulfil the following goals:

1. Possibility to choose at runtime which computation library to use.
2. C++ objects to simplify the development of new applications which provide distributed matrix functionalities.
3. Interoperability with ScaLAPACK. Since the DLA interface will support only a small subset of the routines implemented in ScaLAPACK (the most relevant for scientific applications) the other routines have to be accessed using ScaLAPACK directly.
4. Minimal changes to existing applications: Replacing ScaLAPACK with the DLA interface should require minimal changes to the source code of existing applications written in Fortran, C or C++.
5. Possibility for adding support for new libraries without API changes.

1.2.1 Library Structure

To fulfill the goals defined in this section the library has been designed in the following way.

A distributed matrix class has been provided which provides the basic functionality as accessing the (i, j) element, global to local index conversion, layout conversion. It offers the possibility to manage the memory (it allocates the memory during the object construction and deallocate it during the destruction) which is useful for new application, or the possibility to use an already allocated matrix (such as the memory of a ScaLAPACK matrix) which is needed for existing applications. The class also provides a method to create a descriptor such that its memory can be used to call ScaLAPACK routines that are not supported directly by the DLA interface.

DLA interface provides an API for C++, C and Fortran supporting the following linear algebra libraries:

- ScaLAPACK
- DPlasma
- ELPA

The supported distributed linear algebra routines are listed in Section 1.2.2.

In the C++ interface the distributed matrix class object is used for matrix arguments, while in the C or Fortran the interface the matrix arguments use the ScaLAPACK syntax. An extra parameter is used to specify which computation library has to be used to give the user full control on the solvers used.

In [9] the interface API is described and some example of usage are presented.

1.2.2 Libraries and Routines Supported

Thanks to the interoperability with ScaLAPACK, DLA interface does not have to support all the routines, therefore only the most used routines in material science application has been selected. The list of supported routines includes the Cholesky decomposition, matrix-matrix multiplication, inversion of a positive definite matrix given its Cholesky factor, inversion of a tridiagonal matrix, symmetric/Hermitian eigenvalue problem and LU decomposition.

The limited resources available for this project imposed to choose three libraries to be supported: ScaLAPACK, DPlasma which is a representative for tile layout matrix distribution and novel parallel programming paradigms, and ELPA which provides very good eigensolvers.

1.2.3 Library Limitations

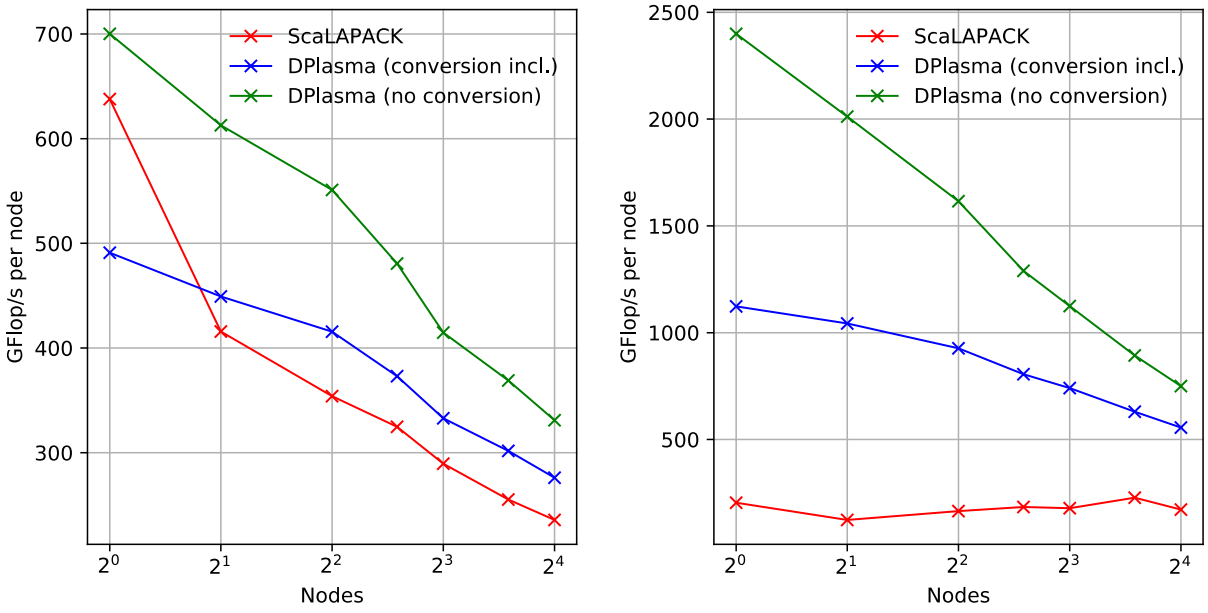
ELPA and DPlasma have stricter limitations compared to ScaLAPACK on the distribution of the matrices. For example, ELPA does not support submatrices which first element is not the first element of a block owned by the first rank and DPlasma matrix-matrix multiplication requires the block sizes and indices of the first element to be compatible.

The DLA interface checks if the extra limitations are fulfilled, otherwise it will use ScaLAPACK to avoid application crashes. At the end of the run the user is notified about the number of times this has been occurred.

1.3 Results

We present the results of the Cholesky factorization on two systems with different architectures. This algorithm has the lowest complexity among the routines supported by the DLA interface, therefore the matrix conversions (ScaLAPACK to tile layout for the input matrix and tile to ScaLAPACK layout for the output) have a greater impact on the time to solution and performances compared to other algorithms.

The benchmark has been executed on the Piz Daint supercomputer located at CSCS. Figure 1(a) shows the results using multicore nodes, featuring 2 Intel Xeon E5-2695 v4 (with 2 x 18 cores @ 2.1GHz each). Figure 1(b) presents the results using hybrid nodes, which are equipped with one Intel Xeon E5-2690 v3 (featuring 12 cores @ 2.6 GHz) and one Nvidia P100 GPU. The benchmark has been executed using the LAPACK and ScaLAPACK implementation provided by Intel in the MKL library.



(a): multicore nodes **(b): hybrid nodes**

Figure 1: Performance measurements of the Cholesky decomposition of a (40960x40960) using the DLA interface (higher is better).

From the results it is clear that DPlasma (green line) performs better than ScaLAPACK (red line). On hybrid nodes the difference is more evident, since ScaLAPACK does not utilize the GPU. It can also be noticed that the matrix conversions have a non-negligible impact on the performance, however DPlasma shows a performance gain (10% on multicore nodes, even if the layout conversions are considered (blue line)).

1.4 Future Directions

As a possible extension of the activity, we could envisage the adoption of the DLA interface in the release of mini-apps performing scientifically relevant linear algebra operations, as already started at CINECA. CaSToRC showed interest to adapt the LU-decomposition in their Lattice QCD code, therefore a possible implementation of the interface for eigenvalue solvers could be discussed.

2 Non-Bonded Interactions (NBI) in Classical Molecular Dynamics

2.1 Introduction

In the field of Molecular Dynamics (MD) several large, complex codes are available. Some of these codes are highly performant, but relatively complex, and not in the form of reusable libraries. These codes consume the majority of the computing cycles in very well identified kernels, as, for instance, those calculating NBI using various different algorithms (PME, FMM, etc), but have a complex workflow to optimize performance and to provide functionality. Kernels, once isolated, can be refactored and optimized for different software and/or hardware architectures and employed in different programming models.

In this PRACE work package we have developed an API that can be efficiently implemented with high performance components that can be glued together using either static pipeline of operations, or with a dynamic runtime. The API will make the access to the different backend algorithms seamless and transparent to the end user, thereby allowing an optimal and efficient exploitation of the available computing environment. At the same time, this makes implementation of new functionality, both at the backend, but especially of higher level (sampling) algorithms easier. These are important especially as this field is aiming at Exascale through high throughput type calculations based on the exploration of phase space.

2.1.1 Key Priorities

- Developer community should have full control over APIs external dependencies and their performance portability.
- Separation of concerns, from a developer perspective, between NBI and domain/problem specific calculations (such as bonded interactions, constraints in biochemistry, model description, etc).
- User adoption, via integration into the development cycle of an established community codebase.
- Establish a foundation to integrate new classes of methods, and alternative pair listing strategies that would be useful for particle simulations in other contexts. Modular software design will encourage greater participation to build missing features.
- The API should provide hooks to implement most, if not all, pairwise NBI methods available in the most popular simulation packages.

2.2 Integration in Community Codes

User Adoption is the most important milestone for any library/API development project. When users participate in the feedback loop or the development process, the API evolves into a sustainable codebase. Developing a new code base for the API is not an option, because the feature set would not be able to match the well-established codes. The latter would limit the scope of the API, which in turn would discourage adoption.

2.2.1 GROMACS as an API Testing Ground

Popular community codes include GROMACS, NAMD, LAMMPS and AMBER. Each one of them took several hundred man-years to develop and comes with a very sophisticated set of features and a loyal user base. GROMACS was picked as the first candidate to test the API design. This is chiefly because:

- GROMACS supports a large number of features and force fields, and is highly performant across a variety of hardware architectures and configurations [11]
- Its NBI implementation does not depend on external libraries, such as Boost, which increases the porting potential to other architectures. Already supports different programming models, such as MPI and pthreads through a unified interface and is implemented in a modern high performant language (C++) which allows an easier adoption of different programming models through external libraries (such as HPX, quicksched, argobots, cpp-taskflow, etc).
- GROMACS wants to transition from a standalone simulation package to a reusable library, which converges with the effort of this work package, thus aligning with its long-term development strategy.
- GROMACS benefits from feedback from an experienced developer and a large user community

2.2.2 Key Challenges

Integrating the API design inside a community code from the start brings its own benefits. There is a possibility to develop and test the API in a production code and measure the direct impact on performance and on productivity. However, it comes with the following challenges:

- Large number of incremental modifications due to the complexity of the GROMACS code base and the overheads in reviewing patches
- Additional refactoring is required to establish a clear separation of concerns, allot specific responsibilities to objects, and a modular redesign for sustainable development.
- Tackling the data flow inside the code. Some of the discussions pertaining the API were of interest across the GROMACS community such as the definition of the (non)state-dependent simulation data.
- Design decisions must complement other ongoing projects on the GROMACS roadmap.

2.3 API Scope

The API has two faces: namely the developer side and the user side. The developer side is meant to be used by the community code developers or by any programmer planning to implement a library using this API. While, the user facing API is meant to be used by domain scientists to prototype simulation workflows. This aspect of the API hides the complexities of implementing a highly performant n-body solver. Such complexities encompass the parallel programming model, the communication infrastructure and the underlying hardware.

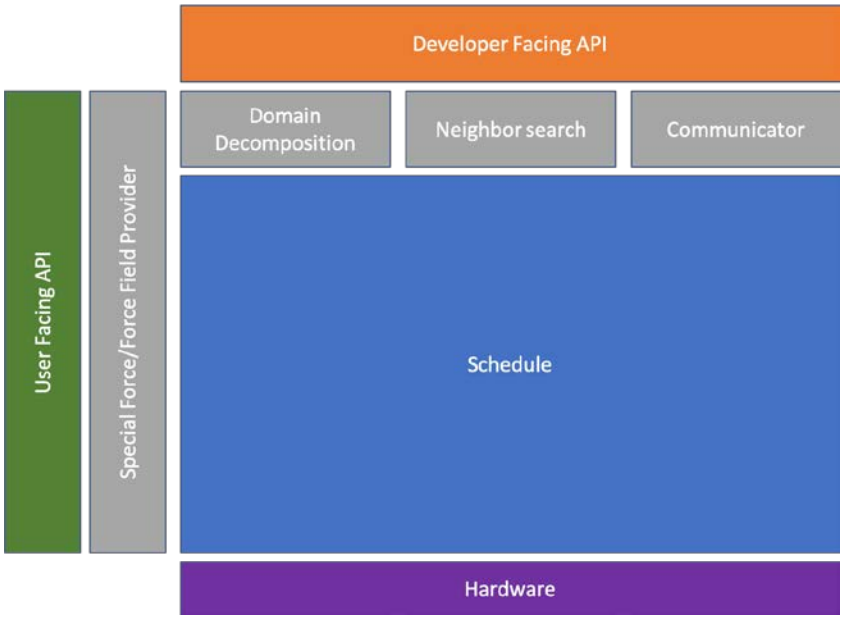


Figure 2: User and Developer-facing APIs and aspects of the library it interacts with

The long-term goal of this API is a software redesign which on one hand allows users to compose high-level workflows using a NBI calculation library, and allows developers to implement new features and alternative backends with great modularity. The design cleaves seams through the MD codebase at various levels, which can be better explained using the figure below.

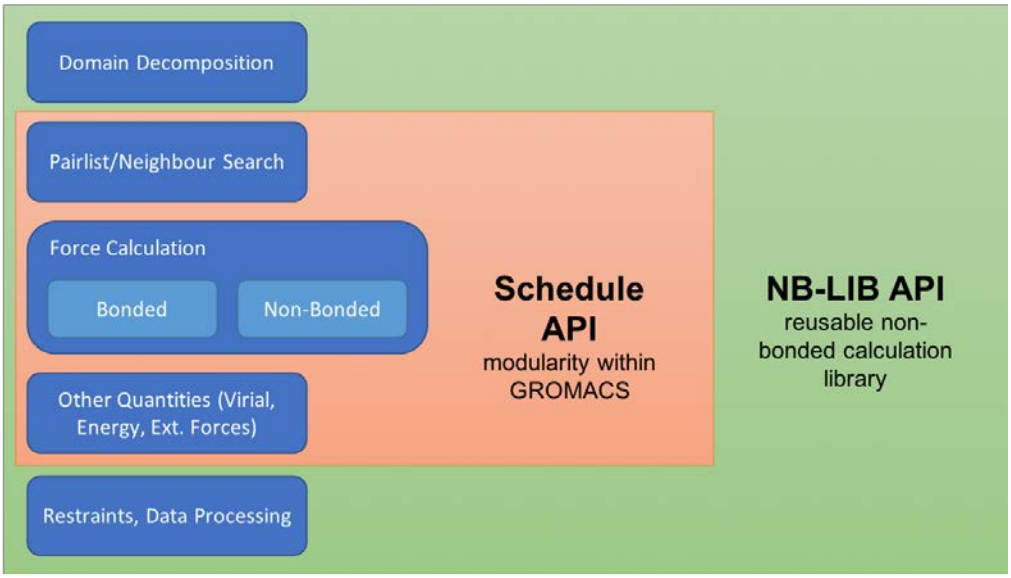


Figure 3: Seams in the software package introduced via the proposed API

The API input data format is planned to be extended to many different simulation packages. The current supported data input format is GROMACS version 2020. The internal API data format is not published, so that it allows the library implementers to choose the most relevant data format for the hardware that their library supports. The output data format is also extensible and is currently GROMACS version 2020. The final output of API should be energy and optionally forces and virial.

Therefore, to use the API, the domain scientists must express the simulation system (particles, topologies, interaction details, etc) in data structures that are supported by the API.

2.4 API for Composing Static Schedules

2.4.1 Motivation for Static Schedules

While developing simulation software for diverse, heterogeneous hardware, there exist two major strategies. One direction is to dynamically assess loads on each of the available hardware resource and assign “computation tasks”. This approach of dynamic dispatch of workloads was discarded to favor another approach involving static schedules.

A typical MD step takes place within a few milliseconds for typical problem sizes. During each iteration, the key operations for computing interactions include: distributing data among nodes, splitting the compute tasks of long and short-range interactions, copy memory blocks to accelerators, invoke computations and then perform two reduction operations at both node-local and non-local levels.

It is possible to order the various steps in each of these tasks in such a way that **overlap of computation, communication and reduction duties is maximized**. Within this span of a few milliseconds, it is hard to incorporate decision and assignment logic. Additionally, an optimal

choice for the sequence is less likely to change more than a few times during the lifetime of the simulation. This makes most of this decision-making for sequencing operations redundant.

For an n-body solver**, library of static schedules is envisioned to ensure optimal resource usage for given methods, physics and hardware**. One of the key goals of this API is to provide hooks to do so.

2.4.2 Key Features

- The schedules would be swappable during runtime
- Enable hand-crafted overlap of tasks for a variety of force-calculation algorithms (PME, FMM, Poisson Solvers, etc)
- Customized schedules for important contemporary hardware architectures. They simplify coding the performance intensive parts by removing a lot of architecture-specific branching. Eg: multi-GPU-multi-node clusters, single multicore workstation with GPU, etc.
- Quantity-specific schedules for dealing with forces, energies and/or virial calculations can allow speed-ups by dynamically swapping a schedule for a time-step that may not require certain quantities.
- Schedule selection to be done at the start of the simulation and periodically during specific times in the simulation

An illustration of the sequence of operations handled by the current runtime is shown below. It is possible envision scenarios with different compute load distributions and very different communication requirements. This abstraction allows developers to express hand crafted schedules with minimal conditional branching, resulting in a more organized codebase.

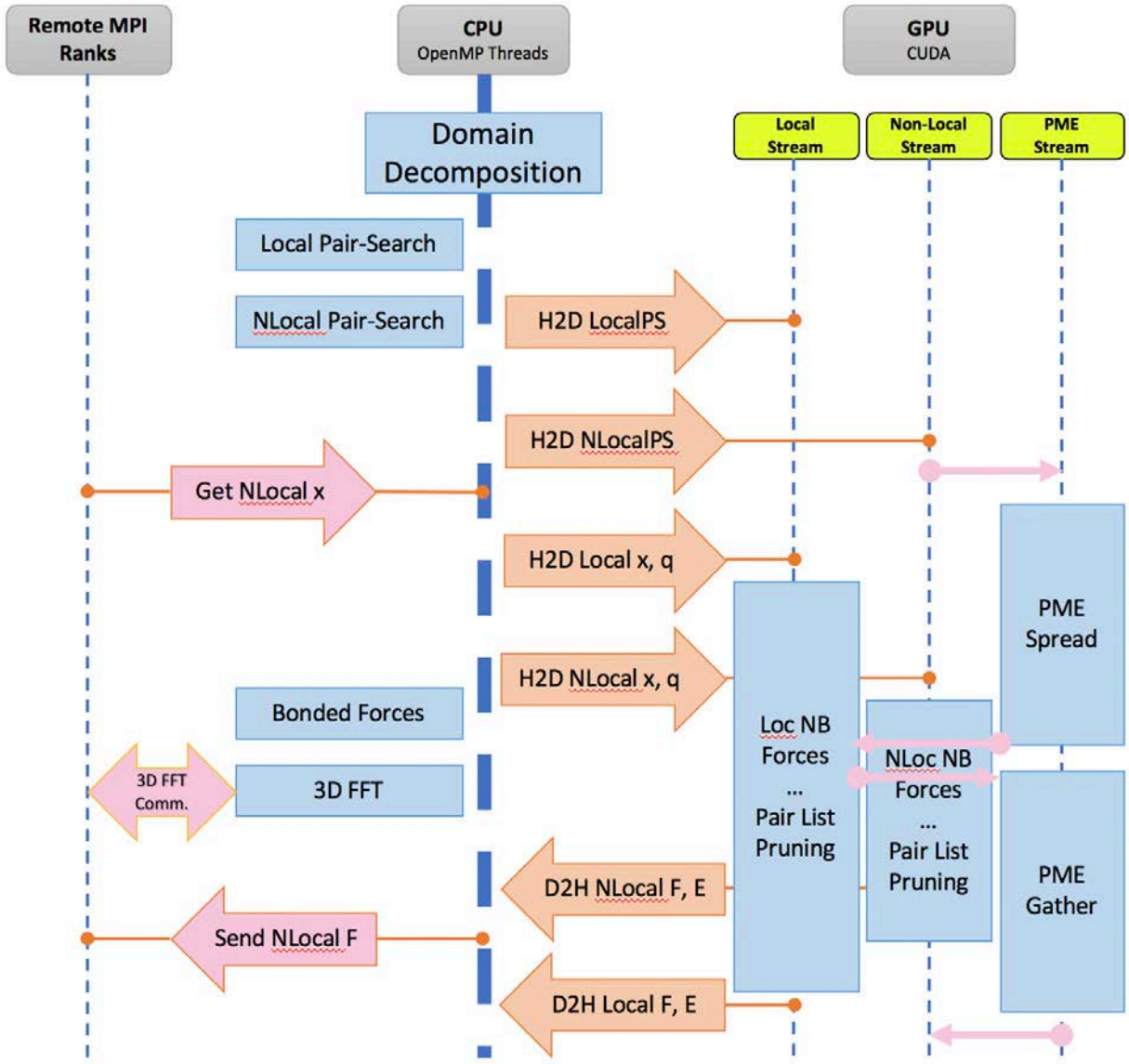


Figure 4: Schedule illustration showing overlap between inter-node (network) communication (pink arrows), intra-node (accelerator) communication (in orange arrows) and compute tasks (blue blocks) for an MD process with PME-based long-range calculations.

2.5 API Structure and Design

As mentioned before, API consists of two levels: the developer level and an end-user one. These are structured in the following way:

2.5.1 Developer Level (Schedule Abstraction)

An interface for expressing Schedules is implemented. It contains protected objects that would be needed to implement a force calculation routine, such as GROMACS-specific communicators, domain decomposition utilities and its own description of the state and system variables along with lower level routines for obtaining the execution context (hardware details).

This is accomplished via inheritance. More concrete types can be built on the interface. To avoid code-duplication, even decorated schedules are an option for adding neighbour search and pair-list construction to already composed schedules. Further details documented in [12].

To assist composing or assembling schedules, a collection of reusable (and unit-testable) functions shall be provided. These functions shall accomplish common tasks like host-device data transfer, or putting atoms on grid and so on. The concrete implementation of these functions would be specific to a certain application (GROMACS in this case).

The schedule is selected based on the execution context and problem details using a schedule builder, which plays the role of a factory in this case to initialize the required dynamic type.

2.5.2 User Level (NB-LIB)

High level execution wrappers are proposed in the API that would encapsulate application-specific details for simplified usage for domain scientists. Using standardized interfaces with getters and setters, application-specific data structures can also be handled. This would allow domain scientists to use a performant n-body solver for a variety of use cases described in [13].

Some of the possible use cases envisioned using the API include:

1. Run a vanilla MD simulation using a program to setup the simulation and analyse the data
2. Add a custom constraints method in the MD simulation
3. Add an external force without modifying the GROMACS package

Rapid prototyping MD programs with custom routines for a variety of compute and analysis goals would be simplified.

2.6 Future Directions

2.6.1 Static Schedule Library

A small collection of schedules supporting a variety of architectures and configurations is envisioned. Early implementations of new force calculation methods can be integrated, benchmarked and profiled to understand ways enhance overlap between concurrent operations.

2.6.2 Modular Descriptions of System and State

The API shall evolve to have cleaner, well-separated descriptions for the simulation state that are program independent. There needs to be distinction between objects relevant to an end user: atom data, charges, masses, etc, and the objects specific to the simulation package: Communicators, ForceProviders, etc.

Proposals for such abstractions evolve in tandem with that of the API and the modernization of the simulation package itself.

2.6.3 Quantity-Specific Verlet Schedules

Earliest benefits of the current abstractions would be for custom-schedules that depend on the quantity of interest. During the simulation, some steps require computing the virial, and others don't. Likewise, some steps don’t require energy computations and there are use cases that only require energies, such as those in monte-carlo sampling.

Schedules can save on significant amounts of compute time if one is able to limit the quantities that they compute based on what is required for the problem.

2.6.4 NB-LIB Project (PRACE 6IP WP8)

This undertaking sets the foundation for the NB-LIB project that aims to transform GROMACS from a stand-alone simulation package to a reusable library for force calculations. These future goals, and full integration of the proposed API is among the goals of this project.

3 Conclusions

The activity of the task was focused on the development of a standardised API allowing the use of different implementations of underlying numerical libraries, optimised on different computing architectures. The standardised API will enable developers of scientific applications to adopt the best implementation of a library in a seamless way, without time consuming changes in the source code. The application areas that will benefit from the API are Linear Algebra and Molecular Dynamics. Linear algebra libraries supported are Scalapack, DPlasma and ELPA, while non-bonded interactions have been addressed in the standardised API for Molecular Dynamics.