COSMOS
Framework for Combining Simulation and Optimization Software
Thesis Summary

Cosmin-Gabriel Samoilă

Department of Computer Science
Universitatea Națională de Știință și Tehnologie POLITEHNICA Bucureşti

SDIALA

Table of contents

Chapter 1

Introducere

If ten years ago we had servers with peak double-precision floating point performance no more than 10 TFLOPS, we are now reaching limits that are orders of magnitude higher both in raw computing power and energy efficiency. Another area where we have spotted some major changes is the hardware complexity and we might notice that one of the main difficulties we are facing right now is developing software that can benefit those advancements while maximizing the performance.

One notable example that shows us not only software has to keep up with hardware is the implementation of INT8 arithmetic units in accelerators. This requirement came from uprising Machine Learning frameworks that use INT8 computations extensively so designers had to implement energy efficient solutions in hardware. As we have mentioned, the hardware and software complexity has risen to levels where writing applications from scratch require a lot of architectural and programming knowledge. Taking into account the costs of developing and testing new software, in many scenarios it is better to use or extend existing open or proprietary code. Since a big chunk of this simulation software is used in conjunction with optimization packages, integrating them is probably not a trivial task most of the time.

Having this problem in mind, we have developed an open source solution that it is aimed to make this coupling possible as seamless as possible, without requiring you to have an advanced programming degree.

1.1 COSMOS Framework

COSMOS is a framework that combines simulation software with optimization packages. The main goal is to provide a seamless experience when integrating existing software applications without having to modify the original source code. While this might not be always possible, we will strive to explain in this thesis how it can be done using the framework and some basic Python programming.

Another function that we have developed is optimal task scheduling on heterogeneous architectures. As we will further describe, this will be done taking into consideration task characteristics and available resources based on some user-defined strategies. Given the open structure of the framework and the design principles we have built on, the implementation of new userspace scheduling algorithms is trivial.
One of the core idea that we have built our framework on is that we have to provide an easy and straightforward method to integrate optimization packages already available for almost every class of problems. We are also providing solutions for users to build solvers in their desired programming languages.

Motivated by a continuous evolution of constraints and requirements in simulation software, this work proposes an easy-to-use solution for integrating simulation and optimization software into HPC environments and it aims to:

- Present the general context and common problems encountered by simulation and optimization software packages in HPC environments
- Describe the COSMOS framework architecture and its design trade-offs
- Outline simulation (e.g. CORSIKA, LAMMPS, GROMACS) and optimization results on HPC systems employing different computing architectures (i.e. x86-64 and arm64)

We start this thesis by analyzing CORSIKA, an air shower simulation written in Fortran and C++. At the outset of our project, our sole objective was to reduce the simulation time for a real-life scenario whose execution time was more than a week. Code profiling – as described in Section 3.1 – determined the code sections to be accelerated through parallelization. However, by theoretically analyzing the best-case scenario – according to Amhdahl's Law – only an estimated speed-up of 2.2 could be achieved. Due to this theoretical limitation of the speedup we decided to tackle the problem from another angle. We then proceeded to the analysis of the underlying models involved in the simulation, through their input parameters and output files. We determined that the execution can be broken into smaller simulations by splitting the input and spawning independent instances of the CORSIKA package. A number of Python scripts were developed to automatically submit different instances of the simulation independently computing subsets of the desired output. Due to the nature of the underlying model, the simulation time is proportional to the number of parameters of the input file. Thus we were able to reduced the overall simulation time proportional with the available cores and as long as we had enough memory to start independent simulation processes for different input subsets. In real-life air shower simulations, the number of computational intensive processes spawned by our scripts reached over a hundred.

Having these requirements in mind, we decided to build COSMOS as a universal framework that can provide an interface for combining various simulation and optimization software packages as well as a wrapper for sending tasks in a High Performance Computing environment.

1.2 Context and Related Work

The COSMOS framework aims to bring together simulation software with optimization methods. Simulation software packages are usually mathematical models describing real-world phenomena (such as chemical reactions, electronic circuits, biological processes, physical measurements, etc.) translated into computer programs. Since these type of simulations are usually computationally intensive, relaxing or tightening the initial conditions and simulation parameters before any numerical experiment is crucial. Most scientific computing computational models take a significant amount of time to be developed and are written in C++ or Fortran. Accordingly, understanding them and tuning their parameters is not a trivial task. From the optimization point of view, COSMOS can be configured to use either existent numerical optimization software or to support the addition of other algorithms from the literature. This approach offers us a significant advantage since we can be flexible in choosing among several optimization methods as well as being able to tune input and problem parameters

A similar approach, offering an interactive software environment combining numerical simulation code with optimization software packages is proposed by Rasch and Bücker in EFCOSS [2]. EFCOSS is focused on optimal experimental design and is presented as an extension of a previous implementation [6] which was mainly focused on parameter fitting. The main disadvantage of this approach that we have tried to solve in our implementation is the ability to specify dependencies between simulation and optimization tasks.

An interactive software systems providing a solution for parameter estimation and identification for mathematical models, belonging to categories such as differential algebraic equations, Laplace transformations, ordinary differential equations, steady-state systems or systems of one-dimensional time-dependent partial differential equations is offered in EASY-FIT [42]. This software offers four optimization routines and a number of differential equation solvers. In COSMOS we aim to support as many optimization routines as are required by the type of data being generated from the simulation packages being used.

A similar approach to COSMOS is proposed by the Network Enabled Optimization System [24] which offers the possibility to solve optimization problems on a remote optimization server. This solution offers a high flexibility and plays the same role that the Optimizer module has within COSMOS. If we take into consideration the type of problems we are trying to solve with our software, NEOS can be used either as a replacement or in conjunction with our Optimizer module.

Finally, software packages offering support for modeling large scale computing data centers and the evaluation of algorithmic and system solutions are given by the SimGrid [4] and CloudSim [46] systems. In COSMOS, we designed a Broker module as a simplified version of these packages, allowing users to describe tasks and which can use custom static scheduling methods.

Chapter 2

Framework Design and Implementation

In this chapter, we are presenting a detailed view of *COSMOS Framework*. This application is also available for public use and further improvements in the online repository [1].

The framework is intended to optimize applications and schedule tasks in a cluster environment. The main goal of this project is to provide a stable and easy-to-use interface to any optimization program, an abstraction to underlying hardware architectures and automatic method to reschedule tasks based on the generated output.

In the following sections the elements that compose the application are presented and detailed. Our framework is written in Python and split has modules designed that can work together or independently - Controller, Broker and Optimizer, as shown in Figure 2.1.

One of the main advantages of this framework is that the modules can be used independently and any user can write its own module that replaces a given default implementation.

2.1 Setup

In this section we will present the hardware and software settings of the HPC systems used in testing the COSMOS framework. We will present each microarchitecture and motivate our hardware choices. Given the fact that we are running computational intensive code, the two main performance metrics targeted to be improved were floating point operations per second (FLOPS) and energy efficiency – as measured using standard profiling tools.

To maximize the FLOPS metric, we had to choose from a variety of machines with Intel server processors [8]: namely Nehalem, Ivy-Bridge and Haswell. Due to the fact that, in general, we have many single-threaded independent tasks to schedule, we opted for the nodes with the Ivy Bridge microarchitecture that could offer us the highest number of available cores with a minimal trade in performance when compared to the newer microarchitectures.

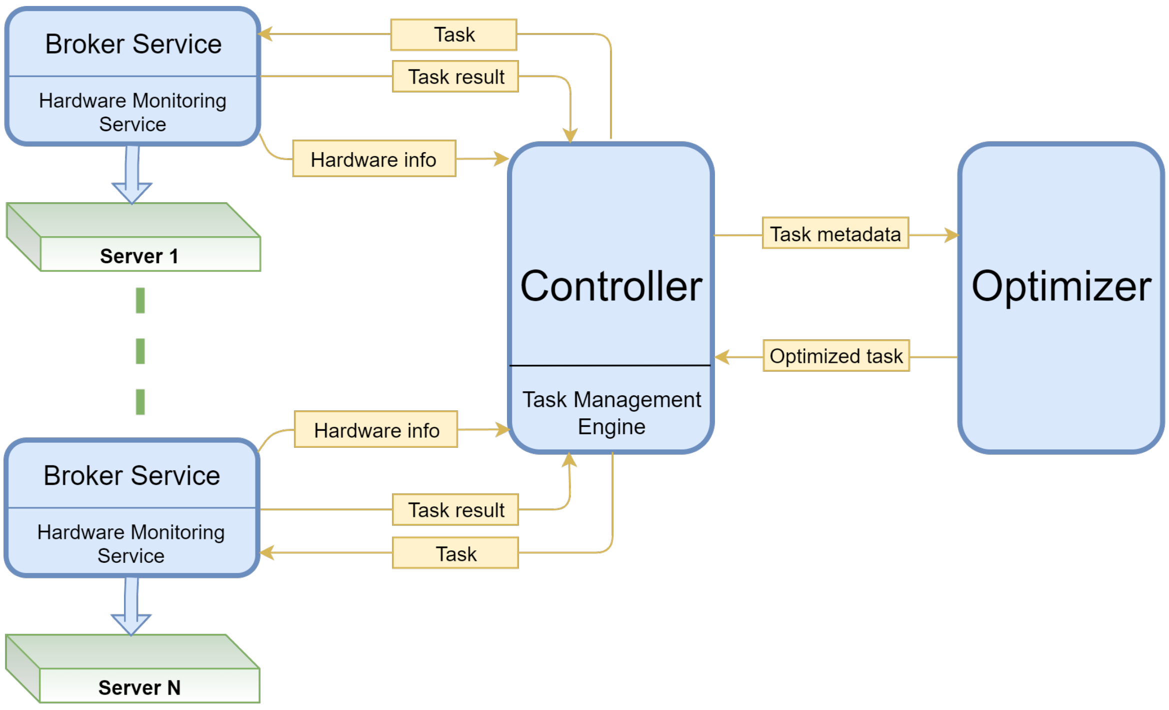

Fig. 2.1 COSMOS Framework Design

Another issue that is worth mentioning is Hyper-threading technology, which is enabled by default on all of our Intel testing machines – to increase the number of independent instructions pipelines. While having multiple logical threads for each physical core is helpful for day-by-day tasks, in high performance applications, ending with two processes/threads on the same physical core might yield worse results than a secvential execution, thus, we must take into consideration this aspect when implementing task scheduling algorithms. For our experiments, we have used Intel powered servers and Raspberry PI 4B boards with hardware specifications shown in Table 2.1

Processor	**Intel Xeon E5-2670 v2**	**ARM Cortex-A72**
Frequency	**2.5 GHz**	**1.7 GHz**
Nodes	**1**	**2**
Sockets	**2**	**1**
Cores	**20**	**4**
Threads	**40**	**4**
Microarchitecture	**Ivy Bridge**	**ARMv8-A**
Memory	**128 GB**	**8 GB**
Storage Type	**HDD**	**SSD via USB3.0 bus**

Table 2.1 Hardware Specifications

Software Configuration The operating system installed on our Intel-based computational nodes is CentOS 7.2.1511, running with a 3.10.0 kernel. The ARM-based nodes on the other hand, are running a Raspbian Stretch operating system with a 4.14 kernel. To minimize the differences among the two computing platforms we decided to use the same software packages versions, such as version 4.9 of the GNU Compiler Collection and version 2.7 of the Python runtime.

2.2 COSMOS

Controller Module

This module was designed to have the following minimum set of capabilities: parse the configuration files for all the modules, create an instance of optimizer and broker, send commands to optimizer and broker and take decisions based on the output of each task.

Besides the basic functionality, this module can be configured to use the machine load information sent by the Broker services to better schedule the tasks. If the machine load information cannot be retrieved in real-time, the tasks will be scheduled and decisions will be made solely on the initial machine description files written by the user. One hard requirement for all the modules is the Python virtual environment that must be present on all the machines. This virtual environment must fulfill all the required package dependencies that are listed in framework documentation at the moment of deployment.

In order to use COSMOS, one must create the configuration JSONs for each module and the commands file. In the following paragraph we will describe the required JSONs and their format.

Optimizer Module

This module plays a crucial role in our current design, being the instance which gives hints to Controller about what are the next steps in execution for each individual task.

With the idea of flexibility in mind, the Optimizer takes an input JSON and executes the **pre** method before the task is scheduled and **post** method after the task finishes. Those methods specified in the configuration JSON must exist in the Optimizer class and one might notice that each task can have a custom optimization method and pre- or post-optimization methods may not be specified.

Those task-based optimization methods can be either implemented by the user directly in the Optimizer class or use them as a wrapper for an already implemented solver for an optimization problem such as: unconstrained and constrained minimization, least-squares minimization, curve fitting algorithms, multivariate equation system solvers. As one might notice, besides the Python virtual environment, we have other hard requirements that must be solved for this module. If the optimization loop contains any method from an external optimization package (by external we mean anything that is not implemented in framework or in the simulation application), the user must solve all the dependencies. Those dependencies can contain but are not limited to optimization application, system libraries and software stack. For example, if the optimization look contains a method from cuSolver library, CUDA stack will also have to be installed on the machine where the Optimization module is running.

Broker Module

This module was created as an wrapper for an resource manager such as Terascale Open-source Resource and QUEue Manager (TORQUE) [44] . In addition to the common supported operations like submitting or killing a job, a broker instance has some extra features such as:

- specify task dependencies and create workspaces
- evaluate current machines load and schedule tasks based hardware description JSON. In the next paragraphs, we will discuss all the task scheduling heuristics that we are using or planning to implement.

Similar to Controller and Optimizer, the Broker must fulfill the Python dependencies and system libraries / software stack requirements. If the simulation is code is not implemented in the Framework or not provided via Python virtual environment, the user must install all the required packages by the task to run on a specific machine. Since the framework has a Broker instance running on each host machine added in the configuration file, the user shall do the minimum required setup for each host.

Distributed heterogeneous computing refers to a series of interconnected machines with different performance specifications. Knowing some of the capabilities for each machine, like memory, bandwidth, processing power, storage size, can be useful when you are trying to write a custom task scheduling policy for a cluster environment.
In the following paragraphs, we will try to tackle the problem presented before. Given the fact that in a real world scenario, most likely, tasks have dependencies, we are using our framework to test different task scheduling policies.
Even if the we were to simplify our problem by eliminating task dependencies, finding an optimal solution for the scheduling problem remains NP-complete. Therefore, it is necessary to have an heuristic algorithm for generating a schedule that will use all of the available resources at a minimum waste and obtain a *reasonable* result. Before designing the final solution for task scheduling in our Broker, we use CloudSim [46] and our own task generator

to simulate a real-life environment and check the performance of each task scheduling method.
Task scheduling is one of the most studied problem in computer science. Finding the optimal solution to this problem is NP-complete and a fairly good schedule can be obtained using some heuristics. Each task has an estimated length and a given machine to run. Computing priorities and finding the best suitable machine for that task depends on the algorithm heuristics.
In this section, we are presenting three static scheduling heuristics used by the Broker to minimize the make-span in a simulated heterogeneous distributed environment. In the last paragraphs of this chapter, we will present the simulation results we got in CloudSim for Min-Min, Max-Min and Work-Queue heuristics [22].
The input for our broker consists is represented by T = [T_1, T_2 ... T_n], a set of task and M = [M_1, M_2 ... M_m], a list of available machines.

In this paragraph, we are presenting the main task scheduling heuristics implemented in COSMOS:

- **Min-Min** - Min-min heuristic takes the set U of unmapped tasks and prioritizes the task with the minimum completion time (MCT). In the first phase of algorithm, the tasks are inserted in a priority queue using task length as comparison metric. In the next phase, the completion time for each available machine is calculated for the head task of the priority queue and select the machine that can solve the task fastest. Update queue head with the next task and add task completion time to estimated end time for selected machine time. The second phase of algorithm repeats until the priority queue with unmapped tasks is empty.

- **Max-Min**: Max-min heuristic also uses the MCT metric. The difference from min-min heuristic described before is that this heuristic takes the set U of unmapped tasks, prioritize the task with the maximum length and selects the machine where this task has the minimum completion time. In the first phase of algorithm, the tasks are inserted in a priority queue using task length as comparison metric. In the next phase, the completion time for each available machine is calculated for the tail task of the priority queue and select the machine that can solve the task fastest. Update queue tail with the previous task and add task completion time to estimated end time for selected machine time. The second phase of algorithm repeats until the priority queue with unmapped tasks is empty.

- **Work-Queue**: WorkQueue is one of the simplest heuristic for scheduling sets of independent tasks. In the first step, the heuristic randomly selects a task and assigns it to the machine with minimum workload(i.e. the machine where task end time is lowest).

- **Priority-Queue** : We have also implemented Priority Scheduling as a policy in our framework. As name suggests, the input tasks are scheduled based on the priority specified by the user (in the tasks JSON file) on the fastest available machines. However, we do not analyze the performance of this method since the actual scheduling is done strictly using user specifications and no decisions based on an algorithm are taken.

CloudSim Results

As we have briefly explained in the previous sections, we need some input tasks for our scheduling methods. From the CloudSim perspective, we have implemented methods that can generate cloudlets or read them from existent files. The cloudlets are generated with random attribute values in some predefined ranges and written in files.

The main reason why we have chosen such numbers for start and end computing power values is that we wanted to simulate our test architectures described in Section 2.1. The 4744 MIPS represents peak estimated value for Cortex A-72 core at full speed and the 43144 MIPS represents the estimated peak value for a Ivy-Bridge core.

If we analyze the algorithms described in the sections before, for **n** tasks and **m** machines the scheduling heuristics have the following complexity:

- Min-Min: $O(n*m+n*log(n))$
- Max-Min: $O(n*m+n*log(n))$
- WorkQueue: $O(n*m)$

Based on the average results for all of the 5000 workloads, we can rank the algorithms based on relevant metrics such as:

- average makespan: Max-Min, WorkQueue, Min-Min
- best results: Max-Min, WorkQueue, Min-Min
- complexity: WorkQueue, Max-Min & Min-Min

Chapter 3

Numerical Simulation and Optimization Software

This Chapter presents applications of high performance computing and optimization software integrated using COSMOS framework.

CORSIKA (COsmic Ray SImulations for KAscade)[15] simulates non-linear high energy hadronic interactions. Due to large execution time for real-world experiments, it represents for us a viable target for parallelization and parameter fitting using COSMOS Framework.

GROMACS (GROningen MAchine for Chemical Simulations)[13] is a molecular dynamics software and it simulates the evolution of particle systems using Newtonic equations of motions. We have chosen this application to simulate the evolution of DNA molecules when applied external forces on them. Integrating GROMACS into COSMOS allows us to provide optimal experimental design and automatically determine the best task configuration of our simulations to minimize the execution time on a given hardware setup.

LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator)[28] is also a molecular dynamics software that can simulate evolution of particle systems, especially used in material modeling. For this particular application, we are focusing on automatically generating the input command files and input atoms geometry, used for simulating the absorption of Hydrogen into Zinc Oxide nanowires.

The next sections in this chapter are focusing these three applications and we are presenting them in the scope of our experiments.

3.1 CORSIKA - Air Shower Simulation

An air shower represents the entire evolution of a high energy primary particle through the atmosphere. CORSIKA [14] is a program, written in FORTRAN and C++, that simulates the air shower using models from physics and mathematics. We are going to present solutions for parallelization and optimization of this type of simulation in a heterogenous computing environment.

Given the fact that high energy interactions are yet to be fully understood, the study of air showers remains one important *tool* that aids this area of research. An air shower represents a cascade that starts to develop when a primary particle with high energy enters the atmosphere. One example of such simulation in CORSIKA done by Fabian Schmidt at University of Leeds, can be seen in Figure 3.1a from the XOZ plane perspective while the same simulation from the XOY perspective is shown in Figure 3.1b.

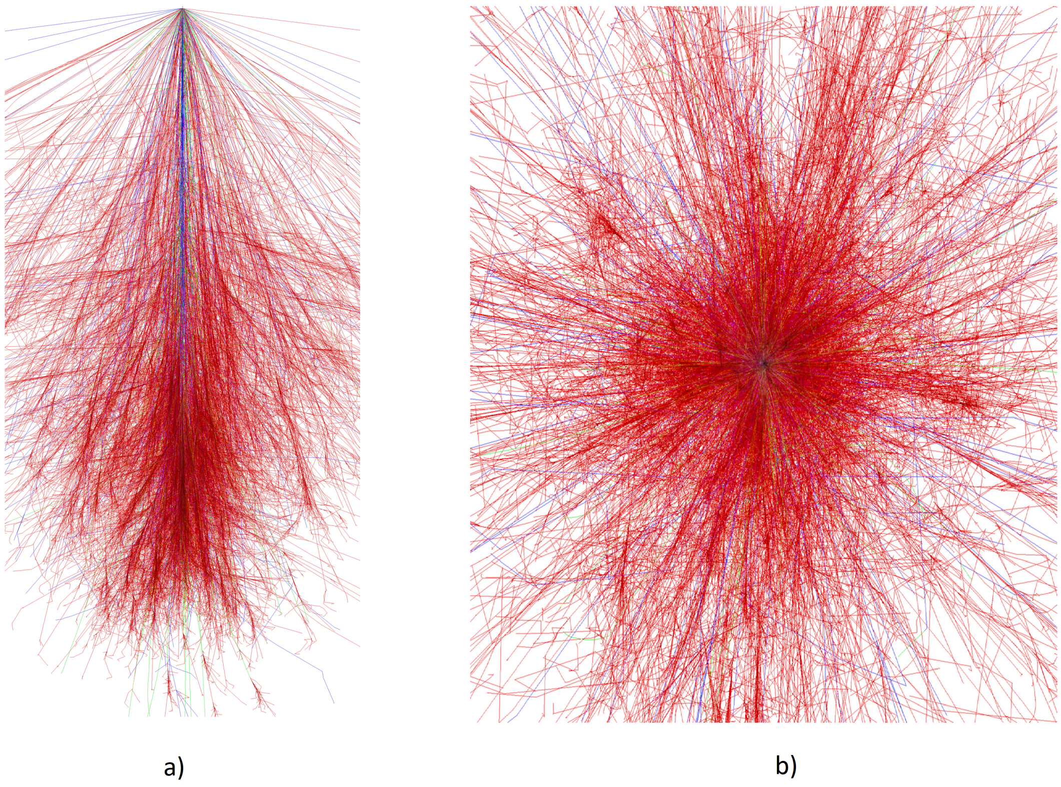

Fig. 3.1 Iron Shower for particle with Primary energy of 1 TeV [43] a) XZ-projection b) XY-projection

The primary particle can be a proton, a nucleus, a photon, an electron or a positron. When this astro-particle collides with an atom's nucleus in the atmosphere at a very high speed (i.e. close to the speed of light), energetic hadrons, as muons, protons, anti-protons, electrons etc., are created. In a real life scenario, the cascade can be observed due to the effects that are created in the atmosphere or at ground level: a flash of light and fluorescence light produced by the secondary particles and radio waves emitted by the deflection of positrons and electrons by the geomagnetic fields.

Detection at ground level is done using water tanks and it's based on the Cerenkov effect [7]: electromagnetic radiation is emitted when a charged particle passes through a dielectric medium with speeds higher than the phase velocity of light in that medium. Another method of detection, other than using telescopes used to observe the light produced in the atmosphere, is the radio detection using antennas. The later method has a great advantage over other optical techniques: it is also efficient during day time and when the sky is cloudy.

The structure of an air shower is influenced by some of the key attributes of the primary particle, such as particle type, energy, direction. Another factor that plays a key role in the evolution of a cascade is the environment: atmosphere composition and density, earth magnetic field strength and orientation etc. As we have stated before, we are interested in studying high energy interactions and one might ask what is the energy range of an astroparticle that can enter in Earth's atmosphere and create a notable air shower. Due to the fact that the flux of very high energy particles (greater than 10^{20}eV) is very small (one particle per square kilometer every 100 years), particles with lower energies (greater than 10^{16}eV), that are more frequent, are also worth tracking.

Considering that the probability of observing a very high energy particle is extremely low, a computer program for detailed simulation of extensive air showers is absolutely necessary. Motivated by the gap between software and hardware, in terms of how efficient the hardware resources are used by this physical simulation, we will provide a solution for parallelizing and optimizing the code using the COSMOS framework described before.

CORSIKA Optimization

The optimization step is carried out by the COSMOS Optimizer Module. Based on the primary particle coordinates, Azimuth and Zenith angles as well as the observers elevation specified in input files, we can compute the trajectory of the shower and some discrete projection points on XY plane, as shown in Figure 3.2. For each of the projection points we search the antennas in the proximity using the Euclidean distance and exclude those antennas that are too far away from the shower trajectory. This optimization is possible due to the fact

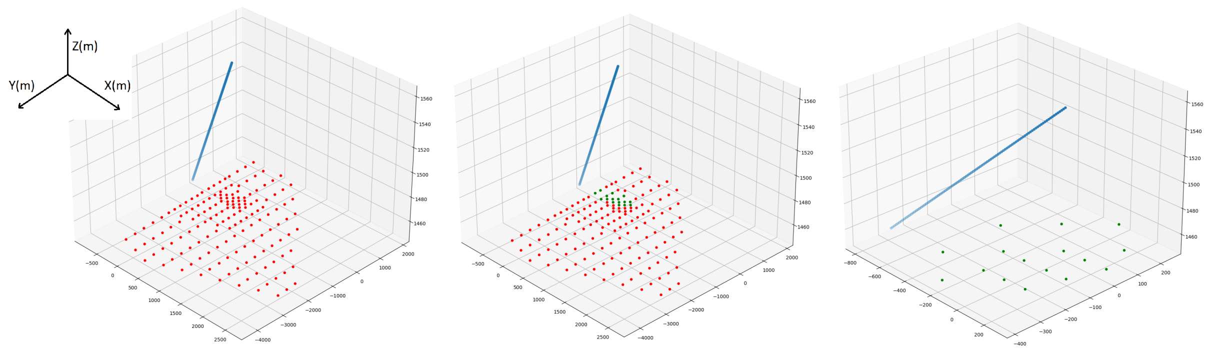

Fig. 3.2 CORSIKA Optimizer on simulation with 156 antennas

that the relevant signal in a simulation can be analyzed by physicists only for the antennas in the proximity of the air shower.

Another capability of the optimizer is to split the input files and select only a subset of antennas using either a N = N x 1 split (N smaller input files of 1 antenna) or a N = P x Q split (P smaller input files of Q antenna).

Results

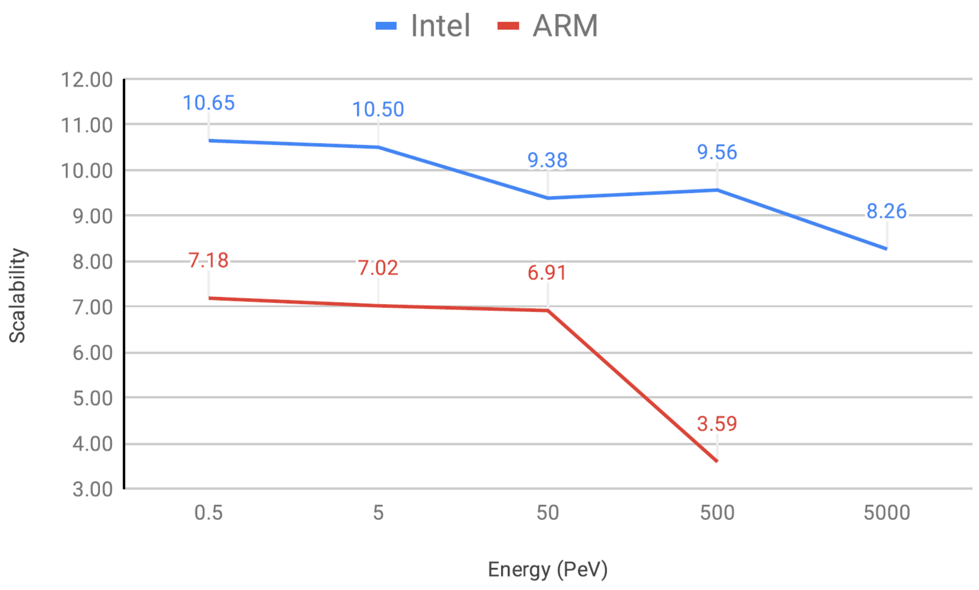

Fig. 3.3 Weak Scaling

As we have explained in Chapter 2.1, we used in our testing two different architectures, namely Intel for best FLOPS performance and ARM for best energy efficiency. As we have previously explained, we have managed to parallelize CORSIKA by spliting the input files and divide the task in smaller tasks. We have chosen the N = P x Q stragegy by automatically distributing $floor(N/P)$ antennas on P processes and then appending one antenna to first $N - floor(N/P) * Q$ processes. Using the algorithm that we've just presented, we have ended up with the following configurations:

- Intel configuration: 16 input files with 8 antennas and 4 input files with 7 antennas. In total, we have spawned 20 independent processes.
- ARM configuration: 4 input files with 20 antennas and 4 input files with 19 antennas. In total, we have spawned 8 indepenedent processes.

With this experiment setup in mind, we are presenting the scalability results in Figure 3.3. Since the simulation time grows exponentially with the primary particle energy, on ARM it was only feasible to simulate on input energies in the range of 0.5 PeV to 500 PeV. One more limitation that we have faced with simulations for high energy particles was the lack of available memory for each process. The drop in scalability was noticeable for input energies higher 300PeV when each process ran out of main memory and began to use swap.

3.2 GROMACS - Molecular Dynamics Simulations

In this section, we will present the capabilities of COSMOS framework to provide optimal experimental design for GROMACS simulations. The targeted experiment is formed of Molecular Dynamics Simulations of DNA Adsorption on Graphene Oxide and Reduced Graphene Oxide-PEG-NH_2 in the Presence of Mg^{2+} and Cl^{-} ions [33]. Graphene and its functionalised derivatives are transforming the development of biosensors that are capable of detecting nucleic acid hybridization. Using a Molecular Dynamics (MD) approach, we explored single-stranded or double-stranded deoxyribose nucleic acid (ssDNA or dsDNA as shown in Figure 3.4) adsorption on two graphenic species: graphene oxide (GO) and reduced graphene oxide functionalized with aminated polyethylene glycol (rGO-PEG-NH2). Innovatively, we included chloride (Cl-) and magnesium (Mg2+) ions that influenced both the ssDNA and dsDNA adsorption on GO and rGO-PEG-NH2 surfaces. Unlike Cl-, divalent Mg2+ ions formed bridges between the GO surface and DNA molecules, promoting adsorption through electrostatic interactions. For rGO-PEG-NH2, the Mg2+ ions were repulsed from the graphenic surface. The subsequent ssDNA adsorption, mainly influenced by electrostatic forces and hydrogen bonds, could be supported by pi-pi stacking interactions that were absent in the case of dsDNA.

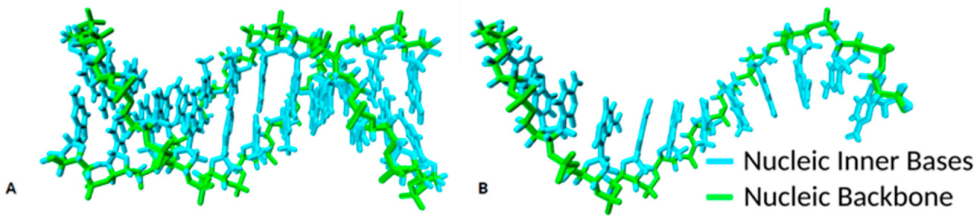

Fig. 3.4 DNA molecules used, generated with the sequence 5'-CGCGAATTCGCG-3': (A) Double-stranded deoxyribose nucleic acid (dsDNA); (B) Single-stranded DNA (ssDNA)

GROMACS Optimization

As we have previously presented, COSMOS offers a high flexibility that allows current tasks to create new jobs and rewrite the tasks dependencies (the only limitation is that a job should not be in execution or finished state at the point of adding new dependencies). In order to be able to obtain an optimal scheduling, we first had to downsize our molecular dynamics experiment such that we can estimate the performance of the setup in a limited amount of time. Using GROMACS *maxh* command line option [13], we have allowed our simulation to run for maximum of 0.33 hours per task. In this profiling experiment, we have created 72 Tier 1 tasks and 144 Tier 2 tasks with a total execution time of 72 hours. To evaluate the performance of the experiment setup, we extract the current step of simulation, compute the *nanoseconds per day* and use it as a metric of evaluation. Our goal is to automatically provide the parameters for the simulation that controls the hardware usage and scheduling on the available servers, targeting either an optimal resources utilization or minimal execution time.

For this experiment, we are using the same servers as described in Section 2.1 with two NVidia Tesla K40 linked to each machine. First step in this process is to formally describe the parameters that controls:

- NSERV: number of available servers: 1, 2 or 3
- NPROC: number MPI processes: 1, 2, 4, 6, 10, 20, 40 or 60
- NOMP: number of OpenMP threads per MPI process: 1, 2, 4, 10, 16, 20
- NGPUS: number of GPUs: 0, 2, 4, 6
- THAFF: thread affinities: on, off

Having the experiment parametrized, we wrote optimization methods that were able to provide the optimal values either for best hardware utilization or best performance. At this point, we will use the terminology of *serial execution* for an experiment that is run on 1 server, 1 MPI process, 1 OpenMP threads per MPI process, 0 GPUs and thread affinities set to off. Furthermore, the serial execution will be our baseline when we will analyse the scalability results for each setup.

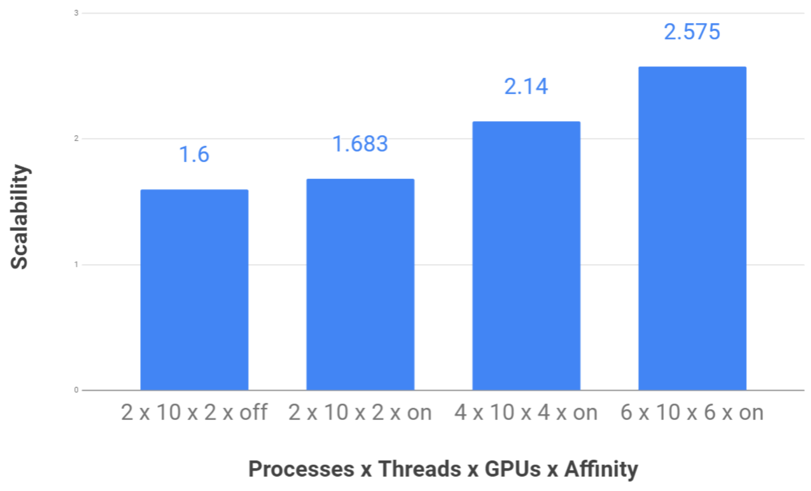

Fig. 3.5 Configuration scalability

Results

In Figure 3.5a, we can see the scalability results for a set of representative configurations. The **baseline** for this chart is represented by what we consider a serial configuration run: one MPI process, one OpenMP thread per MPI process, no accelerators and thread affinity set to off. For the serial configuration, the scalability is considered to be 1. The first and second column shows the scalability obtained using a single server with thread affinity set to on and off respectively. We have chosen to plot both to outline the impact of setting the thread affinity. Because of this setting, the scheduler doesn't spend time in attempting to reschedule threads. Also, since the threads are pinned, the CPU caches are not invalidated anymore when a thread is migrated from one core to another. Compared to baseline, a speedup of 1.683 is obtained if all the resources of one machine are used.

The last two columns in Figure 3.5a show the scalability obtained when using two and respectively all three available machines. However, the speed-up gains shown in these two configurations are overshadowed by the drop in efficiency. In order to run 53% faster, compared to *2 x 10 x 2 x off* configuration, we have to use 3 times more resources.

To obtain our final results of DNA Absorption on Graphene [33], we have chosen a configuration of 2 MPI processes, 10 OpenMP threads per MPI process, 2 GPUs and thread affinities set to *on* (2^{nd} column in Figure 3.5). Even if the scalability is better when using all

3 machines, as we have previously explained, the efficiency drops significantly and it is not cost effective to run in that configuration for multiple weeks of simulation.

3.3 LAMMPS - Molecular Dynamics Simulation

Molecular Dynamics is a technique for simulating the evolution in time of a system of particles by calculating the forces between atoms and integrating Newton's second law. The initial velocities of the particles are generally produced by random generation schemes based on Gaussian functions very close to the Maxwell-Boltzmann distribution that characterizes the equilibrium state of molecular systems. However, these initial velocities do not take into account the interaction between the particles and the system must be brought to equilibrium before collecting structural and dynamic data to characterize the system. Such molecular dynamics simulations require large computational resources in the equilibration phase, but, from this stage of the simulation, only the final positions and velocities of particles that follow the interaction forces of the system will be saved and used in the next stage of simulation. In order to reduce the calculation time for balancing the system, methods for determining the initial velocities can be developed.

LAMMPS is a package of particle and field simulation methods that allow molecular dynamics or Monte Carlo simulations for particle systems. The program contains a very large number of force fields that can be applied at the nano-, meso- or even macro-scopic scale for most existing or virtual materials. The particles can be electrons, atoms, dipoles, united-atoms and coarse-grained particles, spherical and ellipsoidal grains and their combinations. As the name of the LAMMPS program suggests - Large-scale Atomic/Molecular Massively Parallel Simulator - this is one of the most advanced programs developed for HPC systems, with a very good parallel scaling, able to be run on HPC systems with hundreds of thousands of cores [30], for input systems consisting of millions of particles.

The LAMMPS input file is of script type and allows in a flexible way the construction of the system, the allocation of particles, the initial velocities, the application of external forces, as well as the control of the evolution of the system. To analyze the results and modify the particle systems, the Tools4Lamps program [45, 26] was developed within the Summer Guest-Student Program at the Juelich Supercomputing Center, Forschungszentrum Jülich GmbH, Germany. The program is a tool that allows the characterization of system balancing, the geometric manipulation of system subcomponents and the modification of their speeds based on new algorithms.

COSMOS is used here to integrate the two programs LAMMPS and Tools4Lammps and automate their execution on HPC computing systems.

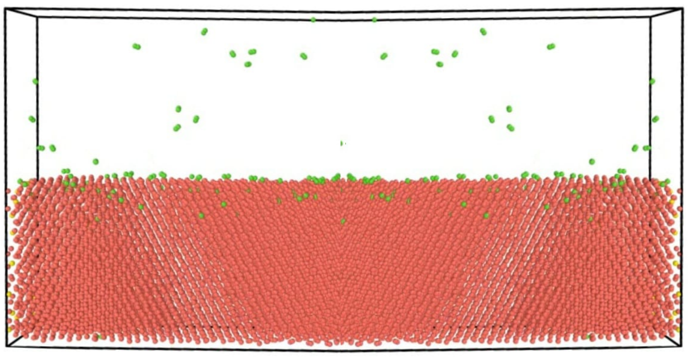

Fig. 3.6 Simulation of Hydrogen absorption Magnesium Oxide structure

In Figure 3.6, we can see a simulation box that contains a snapshot where some hydrogen atoms (green) are absorbed into the structure of the wire (red and yellow). We will consider the LAMMPS simulation as a black box and will not discuss the purpose and details of the simulation itself since it is out of the scope for this thesis. COSMOS integration and results can be found in Section 3.3 and it has two main objectives:

- provide a common interface of communication between LAMMPS and optimization package Tools4Lammps through a pseudo-scripting language
- generate the input structures for LAMMPS simulations: ZnO nanowires with specific length, width and imperfections percentage parameters

As we have previously stated, our objective for integrating LAMMPS into COSMOS is to close the optimization loop using Tools4Lammps while being able to interact with input geometry and command file. This offers us a great flexibility and the end user can automate the whole simulation process using the framework.

LAMMPS Optimization

The LAMMPS simulation is characterized by two types of meta-parameters, the input command file which sets the parameters of the simulation and the types of operations and the data file that contains the atoms geometry. The input command file is used by us to extract specific information about the atomic structure and the optimization of the simulation process is done using external package Tool4LAMMPS. For the integration with LAMMPS, we are focusing on operations that handle the data file structure. In order to read and modify the input geometry, we wrote an internal parsing module in COSMOS framework for this type of input file [31].

```

read_data ZnO.data
variable radius equal 20
variable dx equal 50
variable length equal 100
replicate dx dx ${length}
region DEL cylinder z  ${radius} ${radius} ${length}  \
                    INF INF side out units box
delete_atoms region DEL
```

Listing 1 Command file that generates nanowire from LAMMPS

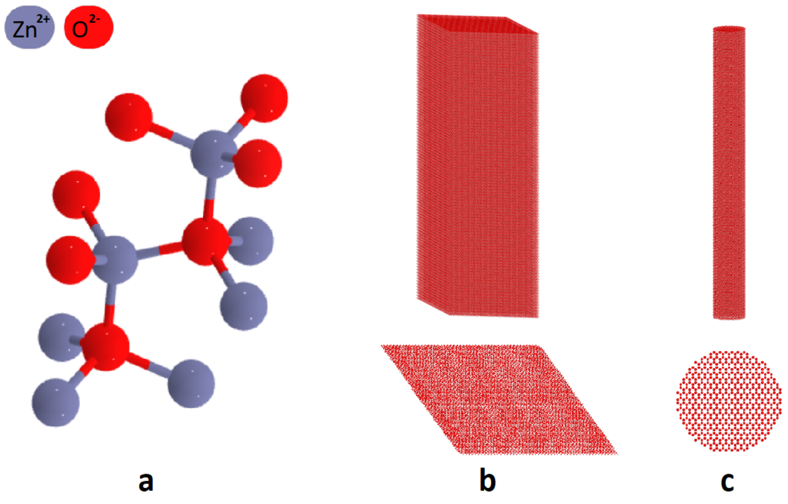

Fig. 3.7 Structure of ZnO

Results

Using LAMMPS to generate ZnO nanowires

One method to generate the nanowire shown in Figure 3.7c is to write an input command file for LAMMPS where we can describe the parameters of the structure and the operations that can be done to achieve the desired result. The sample input command file listed below achieves this by doing the following actions:

- read the initial simple symetrized ZnO cell consisting of 4 atoms as shown in Figure 3.7a
- define the nanowire radius and length using the *variable* command (in this particular example, we are generating a nanowire with a diameter of 40 nm and length of 100nm)

- generate a box starting from initial cell having the following dimensions: dx = dy = 50nm and dz = 100nm. To do this we use the *replicate* command. The output structure after this command can be seen in Figure 3.7b
- now that we have a box of atoms, we must define the bounds of the desired structure structure. For this we are using the *region* command with the following parameters:

  ID: DEL

  style: cylinder

  dim: z - axis of cylinder

  cylinder radius (20nm) and height (100nm)

  side out: the region is outside of the specified geometry

  units box: the geometry is defined in simulation box units
- finaly, we are using the *delete_atoms* command to remove all the atoms that are outside of the nanowire (defined by the region DEL described earlier)

One might notice that it is quite easy to generate such a cylinder structure from the LAMMPS itself and, for some scenarios, it is desired to follow this procedure rather than writing your own structure generator. However, the main disadvantages of this approach are:

- lack flexibility in defining complex structures
- hard to generate imperfections in the structure layout, as it would happen when the nanowires are *grown* in a laboratory
- given the fact that we are controlling the simulation from COSMOS Framework, is quite hard to generate the input file for multiple tasks from Python and it is also prone to error
- this method is good for generating simple structures but it lacks the ability to order the atoms list such that the memory communication will be limited. More details on this feature are provided in the next section.

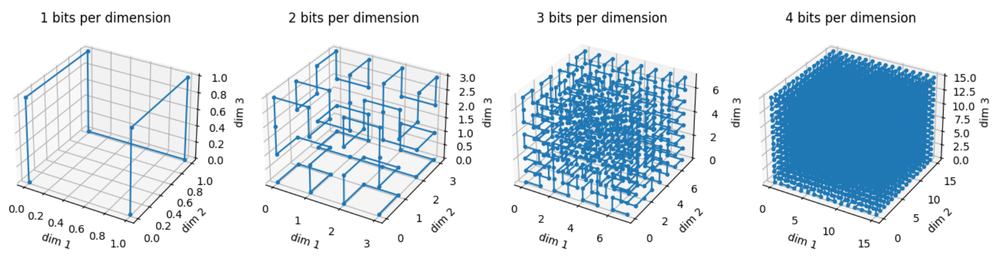

Fig. 3.8 Hilbert curve in 3D [34]

Using COSMOS to generate ZnO nanowires

This section presents a method of generating ZnO nanowires from COSMOS which obtains the same result as the LAMMPS equivalent method described in the previous section. We are generating the ZnO nanowire starting from an input structure as shown in Figure 3.7a. We will then replicate the initial box to generate a supercell as shown in Figure 3.7b and we will start to remove atoms that are outside of the nanowire. For each atom that we delete from the final structure, we also remove every bond **from** or **to** it.

In the next step, we will also generate some imperfections in the layout: the atom is removed with a given probability, the atom will be shifted in one of its directions or one of the atom bonds will be removed. Before doing the final write-back in LAMMPS data file, we can rearrange the atoms list using an algorithm for minimising the communication between processes when LAMMPS uses MPI for parallelization. One such algorithm for shuffling the LAMMPS input data is the Hilbert space filling curve. There are multiple implementations of this algoriuthm in most of the usual programming languages but, for the approach we've taken in COSMOS, we can use the NumPy version [34]. As shown in Figure 3.8, we can encode the positions of 16.777.216 atoms using only one byte per dimension. One of the main reasons why we would want to rearrange the list by following the Hilbert curve for a 3D space is because it improves the of locality of our data, hence reducing communication in a distributed system. A similar approach, but in the rendering domain, is used by Keller et al [25] to render along the the Hilbert curve.

Finally, the obtained output is a nanowire with a predetermined height and radius, as shown in Figure 3.7c. One might notice that the resulted structure is identical to the one generated in LAMMPS but we now have full control of it from COSMOS.

Chapter 4

Framework Integration

The previous chapters discussed the integration with various simulation engines and optimization packages. As we have previously stated in Chapter 1, an exhaustive list of this type of software cannot be covered by this thesis and we provided relevant examples of integration with COSMOS for a relevant subset of applications and optimization packages. In this chapter, we will present other work that we've done which is either suited to be integrated with COSMOS or brings a novel perspective to High Performance Computing. Contributions that are worth noted will cover:

- numerical simulation: topological calorimeter clustering on GP-GPU at ATLAS using CUDA
- numerical optimization: optimization of 3D search using COSMOS Framework
- task scheduling: high throughput computing on HPC: a case study of medical image processing applications
- academia: integrating parallel computing in the curriculum of the University Politehnica of Bucharest

At the end of this Chapter, we will discuss the contributions that we had for the topics presented here.

4.1 Topological Calorimeter Clustering on GP-GPU at ATLAS using CUDA

In this section, we are presenting the work that we have done in collaboration with CERN on processing large amounts of data obtained from the Large Hadron Collider (LHC). As mentioned in The ATLAS Experiment at the CERN Large Hadron Collider [9], huge amount of data will be collected from the $4*10^7$ collisions per second of 10^{11} protons (p) and heavy ions (A), in particular lead nuclei. The energy of proton-proton (p-p) collision is 14TeV and 5.5TeV per nucleon pair (A-A). Two general purpose detectors, ATLAS (A Toroidal LHC ApparatuS) and CMS (Compact Muon Solenoid) have been built for probing p-p and A-A collisions. The structure of the ATLAS detector can be seen in Figure 4.1.

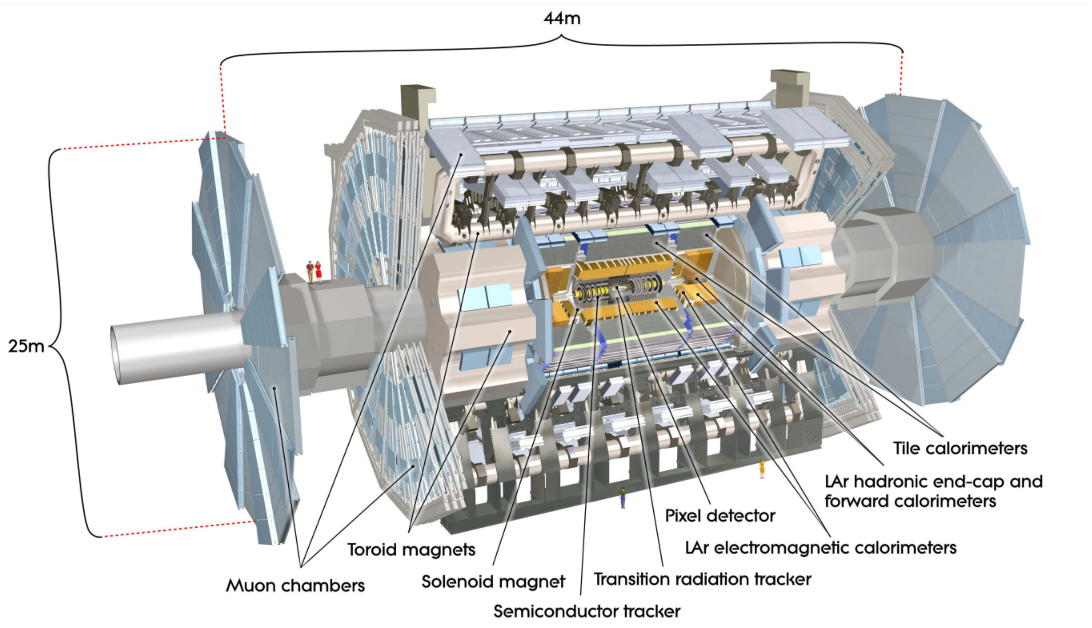
44m
25m
Tile calorimeters
LAr hadronic end-cap and forward calorimeters
Pixel detector
LAr electromagnetic calorimeters
Toroid magnets
Muon chambers
Solenoid magnet
Transition radiation tracker
Semiconductor tracker

Fig. 4.1 Cut-away view of the ATLAS detector [10]

From the ATLAS detector, we are interested in the Calorimeter subsystem, shown in Figure 4.2, and the algorithms that process the data obtained from it. ATLAS Calorimeter is formed by individual detector cells placed in a cylindrical geometry. Each individual cell will record information when particle collisions happen and this data is stored to be later processed by algorithms provided by ATHENA Framework [5]. One of the main goals of those algorithms is to analyze the collision events happening inside the Calorimeter and build topological clusters based on the energy recorded by cells. Topological clustering [10] has a growing stage when neighbour cells are grouped together into clusters based on their recorded energy. The second stage splits the clusters generated in the growing part only if they contain more than one cell having a local maximum recorded energy.

Our work in this project was to analyze the current cluster splitter algorithms written in C++ and port them to GPU using CUDA. In the process of developing the code, we also had to do the conversions required for data structures to be *GPU-friendly*.

We have chosen to present this as a contribution to this thesis even if one might notice that, at a first glance, the parallelization of Athena algorithms has little to do with the COSMOS

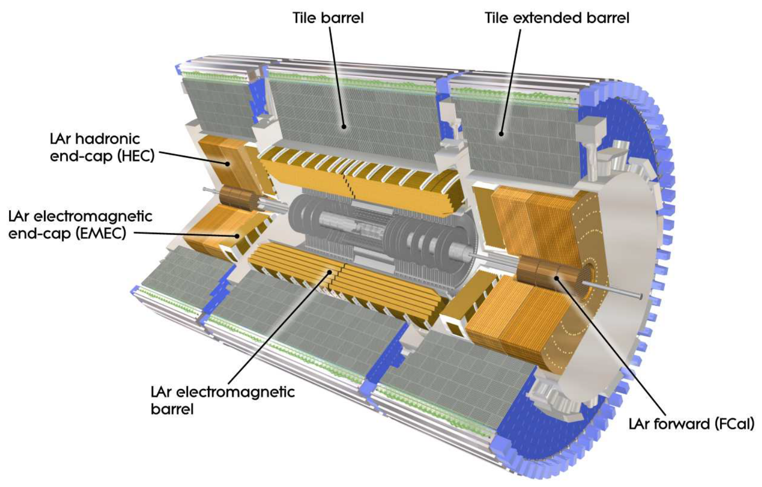
Tile barrel
Tile extended barrel
LAr hadronic end-cap (HEC)
LAr electromagnetic end-cap (EMEC)
LAr electromagnetic barrel
LAr forward (FCal)

Fig. 4.2 Cutaway view on the ATLAS calorimeter system [10]

framework. In reality, both frameworks are integrating data processing and optimization algorithms, thus at a different scale. While Athena is mainly focused on algorithms crafted for processing data obtained in the LHC, COSMOS is a general type of framework that can pipeline a broader spectrum of optimization and simulation software packages.

One of the main reasons why we have chosen to not integrate Athena in COSMOS is because we want to keep a general scope for our framework and going through the path of integration will require a lot of hardware and software resources.

4.2 Optimization of 3D Search using COSMOS Framework

This section presents an extension of the COSMOS framework through the addition of numerical simulations and optimizations. Consequently, it is necessary to introduce new modules that interact with the Controller module and that focus on the post-optimization process in order to determine the optimal steps of the numerical simulation. The structure is based on three main stages, namely: reconstructing of a 3D continuous space with a mobile camera, gathering information from the scene and processing it; and the optimization process that will determine the next step of the numerical simulation, based on the previously collected data.

COSMOS main modules can work individually, opening the possibility of bringing separate implementations and integrating various other external simulation modules. The main goal is to extend the coverage area of simulations, by introducing a new problem, an informative search in a continuous 3D space. Being given the initial view of the scene as input, it is wanted to start the search from a random position in order to determine the coordinates of the camera from the input perspective, with data collected across the search. In order to achieve the goal, the simulation should be optimized by generating the best next steps of the search. All these will be synchronized by the main COSMOS modules. Furthermore, it will be possible for both simulation and optimization processes to be used in solving real problems for unmanned aerial vehicles. The scope can include tracking endangered species [32], monitoring vegetation in precision agriculture [37] or enhancing the capabilities of search for CCTV systems. This domain is vast and rapidly growing, due to the continuous, significant improvements in technology. As previously stated, projects can be easily divided into three main components that will require integration:

- Reconstruction of the scene, where a physically based render Mitsuba2.0 is used. This render permits the modification of the scene layout, camera specification, position, and orientation, in order to simulate the continuous 3D space;
- The extraction of the information from the scene, where the main module introduced is You Only Look Once: Unified, Real-Time Object Detection, which, according to R. Girshick et al. [39], will detect and provide data for the object mapping, in order to estimate the coordinates of the object;
- The optimization, in which case, based on the results given from the second component, the next best step in the simulation is going to be computed by analyzing the objects and their position in the scene.

After implementing the proposed solution, a complete package is expected to be introduced in COSMOS Framework, enhancing the capabilities of this framework. It will be able to render various 3D scenes only by modifying the input file, to recognize different object in the scene and to estimate their position. Furthermore, being given an input file (initial image) and a starting position, by combining the features described above, the initial position of the camera, from where the given picture was taken, will be found in an optimized manner.

4.3 High Throughput Computing on HPC: a Case Study of Medical Image Processing Applications

The last few years have shown a unification of the traditionally separated heterogeneous high-performance computing and big data analytics environments. This has provided HPC computing power to a variety of image processing applications in many fields, including astronomy (e.g. for analyzing dark energy [27] or for classifying galaxies [17].); bioinformatics (e.g. for genome-scale prediction of protein structure and function [20]); fusion (e.g. for reconstructing the 2D plasma profile [18]), etc. Similarly, in the medical field HPC has been widely utilized for various imaging tasks, including reconstruction, restoration, enhancement, classification, segmentation or detection. However, medical imaging presents unique challenges that confront large-scale executions. Medical image processing typically relies on machine learning methods in their applications. Due to the high variety in the samples used for training and in the modalities for inference, the applications are build using a high variety of types of AI/ML methods (from deep learning models and random forests to ensemble learning containing multiple types of models within one single application). In addition, the fact that the studies used are still experimental having a exploratory nature further adds to the dynamic and complex nature of the codes and optimizations. For example, the neuroscience department at Vanderbilt uses over 50 codes that are in continue development and combines them in different ways based on what each study needs to test [23]. Optimizations for AI methods of GPUs [35, 19] or for data movement [49, 12] can be successfully applied to some studies. However, due to the diverse range of behavior patterns in medical image processing, there is no optimization that can fit all scenarios.

We focus in this paper on analyzing the performance of a high-throughput image processing applications that uses a data intensive execution model, but not in the traditional sense of dealing with large images. High-throughput applications shift the focus from individual runs to ensemble by analyzing a huge dataset of small size images that need to go through a pipeline and whose throughput needs to be optimized. For this purpose, we looked at `SLANTbrainSeg` - Deep Whole Brain High-Resolution Segmentation - a whole brain segmentation pipeline on structural magnetic resonance images which is essential for understanding neuroanatomical-functional relationships [48]. Our main objective is to find out the best way to run this application on large-scale systems in order to increase its throughput, beginning with a comparison between existing versions of the program, and continuing with bringing changes to the code and the execution model so that it utilizes the HPC system's heterogeneous architecture to the fullest. Our contributions in this domain are the following:

- A **performance analysis** of a neuroscience image processing application whose focus is on achieving high-throughput going though a large dataset of small MRIs. We identify bottlenecks when deploying such a workflow on large-scale systems.
- Based on the results we propose **a new execution model for high-throughput** that applies to image processing application that have a high data volume but do not generate or read images of large size.
- We **compare the results for the new model on different hybrid architectures** (including Summit [29], ORNL's leadership facility supercomputer) showing speed-ups of up to 4x and we discuss limitations and future open issues.

4.4 Scientific Computing in Higher Education

This Chapter is aimed to give us a better understanding on why we have chosen a particular subset of technologies and programming languages like Python for development of both COSMOS Framework and optimization techniques presented before. This Chapter presents an approach for integrating parallel computing in the curriculum of University Politehnica of Bucharest [3]. One of the main reasons why we have chosen Python for the development of COSMOS is because it has a relatively shallow learning curve and it is accessible to students all over the world at an early stage in higher education programs. Another topic of interest for high performance computing is the optimization techniques, which represents one of the core subjects tackled by the graduate lectures, as described in the following sections. As one might notice, all the information by Mihai Carabas et al [3] is used or desired to have when developing a complex project like COSMOS Framework.

Given the current evolution of the IT industry, Parallel and Distributed Computing is seen as an essential topic for any IT professional. University “Politehnica” of Bucharest is one of the oldest and most prestigious engineering school in Romania. Over the last 20 years, the Computer Science and Engineering Department has conferred a special importance to the PDC curricula. The importance of parallel and distributed systems as well as a distinction between parallel versus distributed systems is discussed in [36] [38], and the approach offered by the UPB is consistent with the view presented therein, since our curriculum already contains different courses for distributed and parallel systems. Most courses containing PDC issues are taught in the first three years of CS and touch a wide audience of around 400–500 students per year. Similar lectures are being offered around the world by various CS groups in Tennessee [21], Cadiz [47], or Cluj-Napoca [47].

In the bachelor program of the UPB, we can find three main lectures where Parallel and Distributed Computing issues are presented to the students, namely the Parallel and Distributed Algorithms (PDA), Computer Systems Architecture (CSA), and Parallel Processing Architectures (PPA). The number of students taking the PDA and CSA lectures ranges from 350 to 450 each year – these two lectures being compulsory for all students enrolled at the Computer Science and Engineering Department. The PPA lecture gathers from 130 to 150 students, in the Advanced Computer Architectures specialization of our bachelor Computer Science Program.

The team at the Computer Science and Engineering Department of the UPB is striving to improve the presentation of PDC concepts in its undergraduate curricula. In the lectures, students are taught general architecture and design aspects of PDC, while in the practical activities they explore various software approaches best suited to illustrate those general concepts. Assignments and homeworks are then meant to check that the relevant desired skills have been learned by our students. In this article, we outline the content of the lectures, the student evaluation process, as well as the lessons learned over time, and the improvements we introduced in our content and approach. The IT industry is exhibiting a particular interest in our graduates, and their PDC skills are highly appreciated. This is also due to the fact that we have continued to evolve our Materials and Methods constantly, as new technologies emerge. At the same time however, we aim for our students to have a fundamental understanding of how parallel and distributed processing architectures work, from both the hardware and software perspective. As new architectures emerge continuously, driven now by emerging domains such as AI or IoT, the essential building blocks remain the same – and PDC is one of those blocks.

We are constantly adapting our curriculum as the industry evolves. One interesting direction are cross-API intermediate languages such as SPIR which provide the underlying runtime for several APIs, such as OpenCL, Vulkan, SyCL, OpenMP, or OpenACC. Moreover, we are considering the addition of a section exemplifying the interaction of OpenCL with popular data analytics and machine learning frameworks such as Anaconda, by using the PyOpenCL [16] wrapper, thus linking together the CSA labs on Python and OpenCL.

Chapter 5

Conclusions

High Performance Computing (HPC) is the one of the domains where we have seen the most advancements in the last couple of years, both in terms of hardware and software. Every field of research that requires numerical simulations, software modeling of systems, numerical optimizations or computing capabilities of more than a machine, is most likely bound to the HPC and it will highly benefit from the research and money invested here. Important results and contributions were obtained in the following areas of software engineering and computational scrience fields:

- Numerical optimization (publications [40], [33], [41])
- Simulation engines (publication [33], contribution to open-source project CORSIKA [11]))
- Task scheduling (publications [40], publication [33])
- Parallelization techniques (publication [40], contribution to open-source project CERN Athena Framework [5])
- Academic curriculum (publication [3])

This section presents the results and contributions that we have made in this paper. Another topic that we are covering in this section is the future work in COSMOS context.

5.1 Research contributions

COSMOS Framework was developed to fill a gap between simulation and optimization fields and computer science. As we have presented in our publication **COSMOS - Framework for Combining Optimization and Simulation Software**, we are proposing a general framework for integrating simulation with optimization software into HPC environments. We have designed our solution to provide a common interface to existing applications (both numerical simulations and optimization packages) such that the user can create tasks pipelines and optimization loops only by describing the application chain in a JSON file.

One important contribution we have is in the domain of task scheduling: implemented algorithms such as Min-Min, Max-Min and WorkQueue [22] that are aware of underlying machine load and characteristics of tasks that are running. We have achieved this by implementing an independent module in the framework that is monitoring live resource usage for each machine and also provides hints on estimated time of completion for tasks that are running. We also show the impact of scheduling in both simulated environments and real world applications general, present the architecture of the framework and provide support on how COSMOS can be extended to integrate packages that are not supported yet.

From the simulation point of view, we have integraded some widely used software such as LAMMPS, GROMACS and CORSIKA. For each of this numerical simulation applications, we have wrote the interfaces of communication with COSMOS and the integration with optimization methods and external packages.

CORSIKA optimization package combines numerical optimization methods for extracting the input parameters and tune them accordingly. Using application specific metadata, we have divided the simulation task into multiple smaller ones and eliminated redundant calculations. By doing this, we managed to obtain a speed-up of 10.6 times (on x86-64 servers) and 7.2 times (on arm64 single board computer) compared to the serial run. We also emphasize on the importance of energy efficiency in our computation and the ability of executing on heterogeneous architectures without having to change anything in the application code. Because the framework is written in Python, we can use some of the language features to easily execute optimization methods or do data post/pre processing on the host machines by name. This allows us to distribute the load on broker agents instead of relying on a single controller node.

The support that we have added into COSMOS for LAMMPS simulation engine allows us to have *atom-level control* over the simulation input data. We can both generate complex input atomic structures from scratch or modify existing ones to fulfil our needs or respect specific patterns (for example, reordering the atoms list to follow Hilbert curve). We have used the framework to generate Zinc Oxide nanowires of various sizes for simulating the absorption of Hydrogen into this type of structures. Another contribution we had was the integration of third party optimizer called Tool4LAMMPS, used mostly for controlling the simulation parameters and closing the optimization loop.

We are also providing a solution to one of the most tackled problems when it comes to designing a simulation: what is the optimal hardware setup that one can use to either maximize the efficiency or the speed-up. This time, we are not interested in controlling the simulation parameters but the underlying hardware configuration: number of threads and cores, number of GPUs and the total numbers of servers. We have used COSMOS to schedule GROMACS simulation tasks and automatically analyze the performance of each task. After we have the results for a specific amount of representative configurations, we automatically provide the best candidates for both optimal resource usage and fastest execution time. Using the data gathered from simulations on our servers, we also provide some hints about what setup one could also chose from public cloud services providers.

Another fruitful collaboration we had was in Medical Image Processing. We focused on analyzing the performance of a high-throughput image processing applications that uses a data intensive execution model, but not in the traditional sense of dealing with large images. High-throughput applications shift the focus from individual runs to ensemble by analyzing a huge dataset of small size images that need to go through a pipeline and whose throughput needs to be optimized. For this purpose, we looked at SLANTbrainSeg - Deep Whole Brain High-Resolution Segmentation - a whole brain segmentation pipeline on structural magnetic resonance images which is essential for understanding neuroanatomical-functional relationships. By analyzing profiling information to understand what are the bottlenecks and using scheduling techniques tailored for this type of application, we have managed to maximize the throughput for processing batches of MRIs. By using our approach, we horizontally scale our workload to whatever servers hardware configuration we have.

In our collaboration with CERN, we have worked on processing large amounts of data obtained from the Large Hadron Collider (LHC). This data is obtained from various experiments, collected by a large number of LHC subsystems and our job was strictly limited to analyzing the events happening in the Calorimeter system. We have achieved good performance results, managing to offload workloads that were CPU specific to NVidia GPUs using CUDA.

As part of our contribution, we have demonstrated that COSMOS framework can be used to optimize 3D search algorithms. This work represents a big step forward for tracking endangered species, monitoring vegetation in precision agriculture or enhancing the capabilities of search for CCTV systems. By using our 3D space search algorithms, one can vastly reduce both the search time and the computational power required for this kind of task. By having this type of workflow integrated into our framework, we are demonstrating a high degree of flexibility for COSMOS.

One of our main topics of interest is keeping up-to-date the parallel and distributed computing curriculum. In the bachelor program of the UPB, we can find three main lectures where Parallel and Distributed Computing issues are presented to the students, namely the Parallel and Distributed Algorithms (PDA), Computer Systems Architecture (CSA), and Parallel Processing Architectures (PPA). In the lectures, students are taught general architecture and design aspects of PDC, while in the practical activities they explore various software approaches best suited to illustrate those general concepts. Assignments and homeworks are then meant to check that the relevant desired skills have been learned by our students. We outline the content of the lectures, the student evaluation process, as well as the lessons learned over time, and the improvements we introduced in our content and approach. The IT industry is exhibiting a particular interest in our graduates, and their PDC skills are highly appreciated. This is also due to the fact that we have continued to evolve our Materials and Methods constantly, as new technologies emerge. At the same time however, we aim for our students to have a fundamental understanding of how parallel and distributed processing architectures work, from both the hardware and software perspective. As new architectures emerge continuously, driven now by emerging domains such as AI or IoT, the essential building blocks remain the same - and PDC is one of those blocks.

Publications

- **Cosmin-Gabriel Samoilă**, Emil-Ioan Sluşanschi. COSMOS - Framework for Combining Optimization and Simulation Software. 2019 22nd International Conference on Control Systems and Computer Science (CSCS), Bucharest, Romania, Publisher: IEEE, ISBN: 978-1-7281-2332-5

- Sebastian Muraru, **Cosmin-Gabriel Samoilă**, Emil-Ioan Sluşanschi, Jorge S. Burns, Mariana Ionita. Molecular Dynamics Simulations of DNA Adsorption on Graphene Oxide and Reduced Graphene Oxide-PEG-NH_2 in the Presence of Mg^{2+} and Cl^- ions. Coatings 2020, Publisher: MDPI, ISSN: 2079-6412, DOI: 10.3390/coatings10030289

- Carabaş Mihai, Draghici Adriana, Lupescu Grigore, **Cosmin-Gabriel Samoilă**, Emil-Ioan Sluşanschi. Integrating Parallel Computing in the Curriculum of the University Politehnica of Bucharest: Euro-Par 2018 International Workshops, Turin, Italy, August 27-28, 2018, Revised Selected Papers

- **Cosmin-Gabriel Samoilă**, Damian Dinoiu, Emil-Ioan Sluşanschi. COSMOS Framework: Simulation and Optimization of 3D Search, U.P.B Scientific Bulletin - Series C, Volume 85, Issue 1, 2023, Bucharest, Romania, ISSN:2286-3540

Unpublished

- Maria Predescu, **Cosmin-Gabriel Samoilă**, Ana Găinaru, Emil-Ioan Sluşanschi, High-Throughput Computing on HPC: a Case Study of Medical Image Processing Applications

- **Cosmin-Gabriel Samoilă**, Nuno dos Santos Fernandes, Patricia Conde Muíño, Emil-Ioan Sluşanschi. Topological Calorimeter Clustering on GP-GPU at ATLAS using CUDA. 23rd IEEE Real Time Conference - Trigger Systems, CERN

5.2 Future Work

From te COSMOS perspective, we aim to provide more tightly couple integration with other external simulation packages and optimization software. We realise that providing support for an exhaustive list of applications will not be possible but we are striving to provide guidance for other people that are willing to collaborate to our open source project. Besides this, we will provide integration with SLANT in the near future so we can benefit from all the features the framework is providing in terms of task scheduling and load balancing. From the academic perspecitve, we are constantly adapting our curriculum as the industry evolves. One interesting direction are cross-API intermediate languages such as SPIR which provide the underlying runtime for several APIs, such as OpenCL, Vulkan, SyCL, OpenMP, or OpenACC. Moreover, we are considering the addition of a section exemplifying the interaction of OpenCL with popular data analytics and machine learning frameworks such as Anaconda, by using the PyOpenCL [24] wrapper, thus linking together the CSA labs on Python and OpenCL.

References

[1] Cosmin-Gabriel Samoilă. COSMOS Framework. https://github.com/gabrielcsmo/COSMOS-Framework.git. Accessed 26 Feb 2023.

[2] H.M. Bücker A. Rasch. Efcoss: An interactive environment facilitating optimal experimental design. http://doi.acm.org/10.1145/1731022.1731023, April 2010. ACM Trans. Math. Softw. 37, 2, Article 13 (April 2010), 37 pages.

[3] Mihai Carabaş, Adriana Drăghici, Grigore Lupescu, Cosmin-Gabriel Samoilă, and Emil-Ioan Sluşanschi. Integrating parallel computing in the curriculum of the university politehnica of bucharest. In *Euro-Par 2018: Parallel Processing Workshops: Euro-Par 2018 International Workshops, Turin, Italy, August 27-28, 2018, Revised Selected Papers 24*, pages 222–234. Springer, 2019.

[4] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing*, 74(10):2899–2917, June 2014.

[5] CERN. Athena framework. https://gitlab.cern.ch/atlas/athena.

[6] B. Lang C.H. Bischof, H.M. Bücker. An interactive environment for supporting the transition from simulation to optimization. Sci. Prog. 11, 4, 263–272, 2003.

[7] P.A. Cherenkov. Visible emission of clean liquids by action of γ radiation. http://iopscience.iop.org/article/10.1070/PU2007v050n04ABEH006238/pdf, 2007.

[8] Intel Co. Intel 64 and IA-32 architectures optimization reference manual. https://software.intel.com/sites/default/files/managed/9e/bc/64-ia-32-architectures-optimization-manual.pdf.

[9] The ATLAS Collaboration. The atlas experiment at the cern large hadron collider. *Journal of Instrumentation*, 3(08):S08003, aug 2008.

[10] The ATLAS Collaboration. Topological cell clustering in the ATLAS calorimeters and its performance in LHC run 1. *The European Physical Journal C*, 77(7), jul 2017.

[11] Air shower physics - corsika. https://gitlab.iap.kit.edu/AirShowerPhysics/corsika. Accessed 05 Mar 2023.

[12] Marco Dantas et al. Monarch: Hierarchical storage management for deep learning frameworks. In *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 657–663, 2021.

[13] GROMACS development team. GROMACS User Guide. https://manual.gromacs.org/current/user-guide/index.html. Accessed 26 Feb 2023.

[14] J.N. Capdevielle D.Heck, J. Knapp. Corsika: A monte carlo code to simulate extensive air showers. https://web.ikp.kit.edu/corsika/physics_description/corsika_phys.pdf, 1998.

[15] T. Pierog D.Heck. Extensive air shower simulation with corsika: A user's guide. https://web.ikp.kit.edu/corsika/usersguide/usersguide.pdf, August 2016. Institut fur Kernphysik.

[16] Massimo Di Pierro. Portable parallel programs with python and opencl. *Computing in Science & Engineering*, 16(1):34–40, 2013.

[17] Sander Dieleman, Kyle W. Willett, and Joni Dambre. Rotation-invariant convolutional neural networks for galaxy morphology prediction. *Monthly Notices of the Royal Astronomical Society*, 450(2):1441–1459, Apr 2015.

[18] Diogo R. Ferreira. Applications of deep learning to nuclear fusion research, 2018.

[19] Ana Gainaru et al. Understanding the impact of data staging for coupled scientific workflows. *IEEE Transactions on Parallel and Distributed Systems*, 33(12):4134–4147, 2022.

[20] Mu Gao et al. High-performance deep learning toolbox for genome-scale prediction of protein structure and function. In *2021 IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments (MLHPC)*, pages 46–57, 2021.

[21] Sheikh Ghafoor, David W Brown, and Mike Rogers. Integrating parallel computing in introductory programming classes: an experience and lessons learned. In *Euro-Par 2017: Parallel Processing Workshops: Euro-Par 2017 International Workshops, Santiago de Compostela, Spain, August 28-29, 2017, Revised Selected Papers 23*, pages 216–226. Springer, 2018.

[22] V. Sn*á*šel H. Izakian, A. Abraham. Performance comparison of six efficient pure heuristics for scheduling meta-tasks on heterogeneous distributed environments. Neural Network World 6/09, 695-710, 2009.

[23] Yuankai Huo et al. Towards portable large-scale image processing with high-performance computing. *Journal of Digital Imaging*, 31(3):304–314, Jun 2018.

[24] J.J. Moore J. Czyzyk, M.P. Mensier. The neos server. IEEE Computational Science & Engineering 5, 3, 68–75, 1998.

[25] Alexander Keller, Carsten Wächter, and Nikolaus Binder. Rendering along the hilbert curve. In *Advances in Modeling and Simulation: Festschrift for Pierre L'Ecuyer*, pages 319–332. Springer, 2022.

[26] Julian Keupp. Methods for fast thermal equilibration in molecular dynamics simulations, 2015. JSC Guest Student Program.

[27] Anthony Kremin et al. Rapid processing of astronomical data for the dark energy spectroscopic instrument. In *2020 IEEE/ACM HPC for Urgent Decision Making (UrgentHPC)*, pages 1–9, 2020.

[28] Sandia National Laboratories. LAMMPS Benchmarks. http://lammps.sandia.gov/bench.html.

[29] Oak Ridge National Laboratory. Summit. https://docs.olcf.ornl.gov/systems/summit_user_guide.html. [Online; accessed September-2022].

[30] Lammps benchmarks. http://lammps.sandia.gov/bench.html. Accessed 20 Mar 2023.

[31] Lammps data format. https://docs.lammps.org/99/data_format.html. Accessed 20 Mar 2023.

[32] Julie Linchant, Jonathan Lisein, Jean Semeki, Philippe Lejeune, and Cédric Vermeulen. Are unmanned aircraft systems (uas s) the future of wildlife monitoring? a review of accomplishments and challenges. *Mammal Review*, 45(4):239–252, 2015.

[33] Sebastian Muraru, Cosmin Gabriel Samoila, Emil I. Slusanschi, Jorge S. Burns, and Mariana Ionita. Molecular dynamics simulations of dna adsorption on graphene oxide and reduced graphene oxide-peg-nh2 in the presence of mg2+ and cl- ions. *Coatings*, 10(3), 2020.

[34] Numpy hilbert curve. https://pypi.org/project/numpy-hilbert-curve/. Accessed 20 Mar 2023.

[35] Yosuke Oyama et al. The case for strong scaling in deep learning: Training large 3d cnns with hybrid parallelism. *CoRR*, abs/2007.12856, 2020.

[36] Marcin Paprzycki, Ryszard Wasniowski, and Janusz Zalewski. Parallel and distributed computing education: A software engineering approach. In Rosalind L. Ibrahim, editor, *Software Engineering Education*, pages 187–204, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[37] Marija Popović, Teresa Vidal-Calleja, Gregory Hitz, Inkyu Sa, Roland Siegwart, and Juan Nieto. Multiresolution mapping and informative path planning for uav-based terrain monitoring. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1382–1388. IEEE, 2017.

[38] Michel Raynal. Parallel computing vs. distributed computing: a great confusion?(position paper). In *Euro-Par 2015: Parallel Processing Workshops: Euro-Par 2015 International Workshops, Vienna, Austria, August 24-25, 2015, Revised Selected Papers 21*, pages 41–53. Springer, 2015.

[39] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[40] Cosmin-Gabriel Samoila and Emil-Ioan Slusanschi. Cosmos-framework for combining optimization and simulation software. In *2019 22nd International Conference on Control Systems and Computer Science (CSCS)*, pages 162–169. IEEE, 2019.

[41] Cosmin-Gabriel Samoil˘ a, Damian Dinoiu, and Emil-Ioan Slu¸sanschi. Cosmos framework: Simulation and optimization of 3D search. *Scientific Buletin of Universitatea Politehnica of Bucharest, C Series*, 85, 2023.

[42] Klaus Schittkowski. *EASY-FIT: A software system for data fitting in dynamical systems*, volume 23. Springer, 03 2002.

[43] Fabian Schmidt. Corsika shower images: Iron showers. https://www-zeuthen.desy.de/~jknapp/fs/iron-showers.html. University of Leeds, UK.

[44] Garrick Staples. Torque resource manager. SC ’06 Proceedings of the 2006 ACM/IEEE conference on Supercomputing, Article No. 8, ISBN:0-7695-2700-0, November 2006. Tampa, Florida.

[45] Georgi Stoychev. Tools for preparation and analysis of molecular dynamics simulations, 2014. JSC Guest Student Program.

[46] A. Agrawal T. Goyal, A. Singh. Cloudsim: simulatior for cloud computing infrastructure and modeling. International Conference on modelling, optimisation and computing (ICMOC-2012), 2012.

[47] Antonio J Tomeu-Hardasmal, Alberto G Salguero, and Manuel I Capel. Integration of ict in concurrent and parallel programming lectures. In *Euro-Par 2015: Parallel Processing Workshops: Euro-Par 2015 International Workshops, Vienna, Austria, August 24-25, 2015, Revised Selected Papers 21*, pages 114–124. Springer, 2015.

[48] Yunxi Xiong et al. Reproducibility evaluation of SLANT whole brain segmentation across clinical magnetic resonance imaging protocols. In Elsa D. Angelini and Bennett A. Landman, editors, *Medical Imaging 2019: Image Processing*, volume 10949, pages 729 – 736. International Society for Optics and Photonics, SPIE, 2019.

[49] Yue Zhu et al. Entropy-aware i/o pipelining for large-scale deep learning on hpc systems. In *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 145–156, 2018.