UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC) - BARCELONATECH

FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)

MASTER IN INNOVATION AND RESEARCH IN INFORMATICS (MIRI)

HIGH PERFORMANCE COMPUTING (HPC)

Enabling Analytic and HPC Workflows with COMPSs

FINAL MASTER THESIS (FMT)

2016-2017 | AUTUMN SEMESTER

Author:
Cristián RAMÓN-CORTÉS VILARRODONA
(cristian.ramoncortes@bsc.es)

Supervisor:
Dra. Rosa M. BADIA SALA
(rosa.m.badia@bsc.es)

COMPUTER ARCHITECTURE DEPARTMENT (DAC)

"En esto, descubrieron treinta o cuarenta molinos de viento que hay en aquel campo, y así como don Quijote los vio, dijo a su escudero:

-La ventura va guiando nuestras cosas mejor de lo que acertáramos a desear; porque ves allí, amigo Sancho Panza, donde se descubren treinta, o pocos más, desaforados gigantes, con quien pienso hacer batalla y quitarles a todos las vidas, con cuyos despojos comenzaremos a enriquecer; que ésta es buena guerra, y es gran servicio de Dios quitar tan mala simiente de sobre la faz de la tierra.

-¿Qué gigantes? -dijo Sancho Panza.

-Aquéllos que allí ves -respondió su amo- de los brazos largos, que los suelen tener algunos de casi dos leguas.

-Mire vuestra merced -respondió Sancho- que aquéllos que allí se parecen no son gigantes, sino molinos de viento, y lo que en ellos parecen brazos son las aspas, que, volteadas del viento, hacen andar la piedra del molino.

-Bien parece -respondió don Quijote- que no estás cursado en esto de las aventuras: ellos son gigantes...que yo voy a entrar con ellos en fiera y desigual batalla.

Y diciendo esto, dio de espuelas a su caballo Rocinante, sin atender a las voces que su escudero Sancho le daba, advirtiéndole que, sin duda alguna, eran molinos de viento, y no gigantes..."

Miguel de Cervantes Saavedra,
Don Quijote de la Mancha

Dedication

Facing a challenging work needs self-efforts as well as the patience of the people around us, especially from our peers.

To my loving mother and father, Dolors and Joan, whose love, encouragement and gentle prodding guided me to such a success. I hope that this work will complete the dream they had for me many years ago when they chose to give me the best education they could.

Special thanks to my sweet sister, Marta, whose affection and support keeps me always up.

Last but not least, I cannot forget Laura, who did more than her share around the house as I was locked in the computer room. Without her unconditional love and constant encouragement, this would not have been possible.

Wholeheartedly,
Cristián Ramón-Cortés Vilarrodona

Declaration of Authorship

I hereby declare that, except where specific reference is made to the work of others, this Master's thesis has been composed by me and it is based on my own work. None of the contents of this dissertation have been previously published nor submitted, in whole or in part, to any other examination in this or any other university.

Signed: _____

Date: _____

Acknowledgements

I gratefully thank my supervisor Rosa M. Badia Sala for all her assistance during my career at the *Barcelona Supercomputing Center (BSC-CNS)* and for giving me the opportunity to collaborate on this project.

I would also like to thank all my colleagues, current and former members of the *Workflows and Distributed Computing* team from the *Barcelona Supercomputing Center (BSC)* for their useful comments, remarks and engagement through the learning process of this Master's thesis: Jorge Ejarque, Francesc Lordan, Francisco Javier Conejero, Raul Sirvent, Daniele Lezzi, Pol Alvarez, Ramon Amela, Sandra Corella, Albert Serven, Adrià Aguilà and Sergio Rodríguez.

Special thanks to Kim Serradell Maronda for giving me the opportunity to work with the *NMMB* application and guiding me through its internals.

UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC) - BARCELONATECH

Facultat d'Informàtica de Barcelona (FIB)

MASTER IN INNOVATION AND RESEARCH IN INFORMATICS (MIRI)

High Performance Computing (HPC)

Abstract

Enabling Analytic and HPC Workflows with COMPSs

by Cristián RAMÓN-CORTÉS VILARRODONA

In the recent joint venture between High-Performance Computing (HPC) and Big-Data (BD) Ecosystems towards the Exascale Computing, the scientific community has realized that powerful programming models and high-level abstraction tools are a must. Within this context, the Barcelona Supercomputing Center (BSC) is developing the COMP Superscalar (COMPSs) programming model, whose main objective is to develop applications in a sequential way, while the Runtime System handles the inherent parallelism of the application and abstracts the programmer from the different underlying infrastructures. The parallelism is achieved by defining an application Interface that allows COMPSs to detect methods that operate on a set of parameters (called tasks), and execute them distributedly and transparently.

This Master Thesis aims to enhance COMPSs, adapting it to the needs of the Big-Data Ecosystems, by supporting Analytic and HPC workflows. To this end, we propose a straightforward integration with the execution of binaries, and *MPI* and *OmpSs* applications. Although the COMPSs programming model is kept untouched, we extend the COMPSs Annotations and some of the COMPSs internals such as the task schedulers and the worker executors.

To support our contribution, we have ported to COMPSs two real use cases. On the one hand, *NMMB BSC-Dust*, a workflow to predict the atmospheric life cycle of the desert dust and, on the other hand, *Guidance*, an integrated solution for Genome and Phenome association analysis.

Keywords: HPC, Distributed Computing, Workflows, COMPSs, PyCOMPSs

Contents

List of Figures

List of Tables

List of Abbreviations

API	**A**pplication **P**rogramming **I**nterface
COMPSs	**COMP S**uperscalar
CPU	**C**entral **P**rocessing **U**nit
GUI	**G**raphical **U**ser **I**nterface
I/O	**I**nput / **O**utput
MPI	**M**essage **P**assing **I**nterface
NEMS	**N**OAA **E**nvironmental **M**odeling **S**ystem
NMMB	**N**onhydrostatic **M**ultiscale **M**odel on the **B**-grid
NOAA	**N**ational **O**ceanic and **A**tmospheric **A**dministration
OS	**O**perating **S**ystem
PyCOMPSs	**Py**thon binding for **COMP S**uperscalar
RAM	**R**andom **A**ccess **M**emory
SSH	**S**ecure **SH**ell
VM	**V**irtual **M**achine
WSDL	**W**eb **S**ervices **D**escription **L**anguage

Glossary

CPU
The Central Processing Unit (CPU) is the part of the computer that contains all the elements required to execute the instructions of software programs. Its main components are the main memory, the Processing Unit (PU) and the Control Unit (CU). Modern computers use multi-core processors, which are a single chip containing one or more cores.

Core
A core is an individual processor that actually executes program instructions. Current single chip CPUs contain many cores and are referred as multi-processor or multi-cores.

MareNostrum III
MareNostrum III is the most powerful supercomputer in Spain with 3056 nodes (48.896 processors), 115.5 TB of main memory and a peak performance of 1.1 Petaflops. It is hosted by the Barcelona Supercomputing Center (BSC).

Node
A compute node refers to a single system within a cluster of many systems.

Scratch Space
Supercomputers generally have what is called scratch space: disk space available for temporary use and only accessible from the compute nodes.

SSH
A protocol to securely connect to a remote computer. This connection is generally for a command line interface, but it is possible to use GUI programs through SSH.

Environment Variable
In Linux systems, each process has an execution environment. This environment can be inherited from the user session environment and can be extended with specific process variables. An Environment Variable is a value stored in the process environment that can affect its execution.

API
An Application Programming Interface (API) is a set of methods and functions that are used by another software to produce an abstraction layer.

Operating System
A system software that manages the hardware and provides services for computer software.

Framework
Framework stands for a set of standardized concepts, practices or criterias used to face a given problem. Specifically, it defines a set of programs, libraries, languages, and programming models used jointly in a project.

Workflow
A workflow is composed of tasks and dependencies between tasks. Workflows are commonly represented as graphs, with the nodes beeing tasks and the arrows representing the dependencies. Somehow, tasks must represent an action that must be done (i.e. the execution of a binary), and the dependencies must represent the requirements that must be satisfied to be able to execute the task (i.e. the machine availability or the required data).

Binary
A file containing a list of machine code instructions to perform a list of tasks.

WSDL
Web Services Description Language (WSDL) is an Extensible Markup Language (XML) used to describe web services.

Graphical User Interface
The GUI is a software that graphically interacts with the user of a computer to ease the data manipulation.

Chapter 1

Introduction

1.1 Motivation

Several years ago the industry required the research community to shift from sequential computing to parallel computing. The extreme increase in the computational loads and the decrease of the acceptable system's response time forced the community to "think in parallel". Although writing parallel codes is not as easy as writting sequential programs, we can assume that the actual computing resources of a multi-core processor are already able to exploit the inherent instruction-level parallelism of an application in a completely transparent way for programmers.

However, the next generation of applications were requiring more high-performance computing resources than those that a single computing resource could offer. Thus, it was necessary to use many machines communicated through networks to achieve a larger amount of computing capabilities. That is what we know today as *Distributed Computing*. Distributed applications exploit the *task-level* parallelism and are even more difficult to handle for programmers. One of the major issues that araise from both parallel and distributed programming is that writing in parallel is not as easy as writting sequential programs and, more often than expected, people that can develop useful and complete end-user applications is not capable of writting efficient parallel code (and the other way arround). We can somehow consider that scientists interpreting results do not care about the computational load or distribution (how results are computed), but care of the results quality, the time spent to retrieve the results and the robustness of the system. In this line of *"writing programs that scale with increasing number of cores should be as easy as writing programs for sequential computers"* [3] several programming models have araised to help the programmer handle the underlying infrastructure.

Nowadays, the scientific community not only wants to work in parallel and distributed systems but also needs to handle a large amount of data. In this sense, *Big-Data (BD) Ecosystems* appeared some years ago; allowing the community to store, check, retrieve and transform enormous amounts of data in acceptable response times. For this purpose, several programming models have also arised but are completely different to the ones used by the *High Performance Computing* community.

In the race towards the Exascale Computing [2], the scientific community has realized that unifying *High Performance Computing (HPC)* platforms and *Big-Data (BD) Ecosystems* is a **must**. Nowadays, these two ecosystems differ significantly at hardware and software level but *"programming models and tools are perhaps the biggest point of divergence between the scientific-computing and big-data ecosystems"* [50]. In this respect, *"there is a clear desire on the part of a number of domain and computer scientists to see a convergence between the two high-level types of computing systems, software, and applications: Big Data Analytics (BDA) and High Performance*

Computing (HPC)" [22]. We can then conclude that the joint venture between HPC and BDA ecosystems opens a great oportunity for programming models that is worth to investigate.

1.2 Context

The current project is conducted as the *Final Master Thesis* in the *Master of Innovation and Research in Informatics - High Performance Computing (MIRI - HPC)* offered by the *Universitat Politecnica de Catalunya (UPC)* [16] and has been funded by the *Barcelona Supercomputing Center (BSC)* [7].

The project has been developed as a research support engineer in the *Workflows and Distributed Computing* group of the *Computer Science* department at the *BSC*. The main goal of this group is to ease the development of applications and services for distributed infrastructures (such as Clusters, Grids and Clouds) through the *COMP Superscalar (COMPSs)* [19] programming model. The group also covers Big-Data and HPC applications, energy constrained environments, GPU offloading and execution in mobile devices.

1.3 Objectives

To take profit of the abovementioned join venture between HPC and BDA ecosystems this Final Master Thesis **enhances the *COMPSs* Framework to support Analytic and HPC workflows**. In next subsection we provide detailed information about the specific objectives that have been considered to successfully achieve this contribution.

1.3.1 Detailed Objectives

The following points summarize the main goals of the project:

O1 Conduct a survey among the Analytic and HPC workflows to determine the most common used technologies, compare them to the solutions proposed by other state of the art Workflow Managers and determine the most adequate and feasible features for COMPSs.

O2 Extend the current COMPSs syntax to support the selected features in a non-invasive way; keeping the backward compatibility. More specifically, extend the COMPSs annotations to support binary, MPI and OmpSs tasks.

O3 Enhance the current COMPSs Runtime by implementing the required mechanisms to schedule and execute the new type of tasks. At scheduling level, this objective includes the treatement of multi-node tasks and the inclusion of scheduler hints for a certain type of tasks. At execution level, the COMPSs workers must transparently execute the different kind of tasks.

O4 Demonstrate that the proposal is able to execute Analytic and HPC Workflows by executing and analysing real use-cases.

1.4 Document Structure

The rest of the document is organized as follows. Chapter 2 gives an overview of the current state of the art of HPC and Analytic workflows, describing the most used technologies and comparing them to what is supported in the most common Workflow Managers. Chapter 3 introduces the COMPSs programming model, describing its main features before our

contribution. Chapter 4 introduces describes the tools and the development methodology followed during all the project. Chapter 5 accurately describes our contribution to provide the reader a closer look of the solutions that we have chosen to face each specific challenge. Chapter 6 evaluates the obtained results and the performance of our proposal. Chapter 7 provides a brief summary of the thesis and, finally, in Chapter 8 we discuss the conclusions and state the guidelines for the future work.

Chapter 2

State of the art

Scientific programs from different fields (such as bioinformatics, biomechanics, earth sciences or engineering simulations) that are either computationally intensive or require a huge amount of storage capabilities are known as *e-Science* applications. In the past decade, the computational and storage requirements of these *e-Science* applications have grown enough to do not fit in a single machine. This fact has forced *e-Science* applications to move from parallel computing to distributed computing because the computing resources within a multicore processor are limited by the architecture and the technology of the system and were already fully exploited. However, in the distributed computing paradigm, the only room for improvements is to exploit the *task-based* parallelism between nodes.

Considering that writing applications for parallel or distributed environments is much more difficult than writing sequential programs due to the concurrency issues, the IT community has developed several libraries and frameworks to ease the development of applications (and consequently, the development of *e-Science* applications). In an attempt to classify all the tools provided by the community that can be used for the design of *e-Science* applications, we have divided them into two groups: *Distributed Libraries* and *Workflow Frameworks*. Next, Section 2.1 defines and provides examples of some state of the art *Distributed Libraries* and, Section, 2.2 defines and introducing some state of the art *Workflow Frameworks*.

2.1 Distributed libraries

On the one hand, distributed computing can be achieved by programming the low-level communication between processes. Although using distributed libraries can lead to excellent performance results, it is a tedious work for the programmer to develop efficient code for a complex application using such libraries. Next subsections provide some examples of these libraries.

2.1.1 MPI

The Message-Passing Interface (*MPI*) standard *"includes point-to-point message-passing, collective communications, group and communicator concepts, process topologies, environmental management, process creation and management, one-sided communications, extended collective operations, external interfaces, I/O, some miscellaneous topics, and a profiling interface"* [38]. The MPI standard includes bindings for C and Fortran, and its goal is *"to develop a widely used standard for writing message-passing programs"*[38]. It has several different implementations but the most well-known are *OpenMPI*[54] and *IMPI*[30]).

For the end user, using MPI requires handling explicitly the spawn of the processes, the code executed by each process and the communication between them. All this management is done by using API calls, and thus, the application code must be compiled and executed with the MPI compiler of the specific MPI implementation. The main advantage is that the users have full control of all the processes and the communication between them which, for

experienced users, leads to high efficient codes. However, for inexperienced users, an efficient code can become unreadable and handle many processes can become a tedious work. Moreover, when porting a sequential application to an MPI application, the users must explicitly distribute the data between the processes and retrieve back the results (which can lead to load imbalance or inefficient communications). Additionally, another inconvenience of MPI is that, once the application's execution has started, the number of processes cannot be changed dinamically, limiting the maleability of the applications.

Figure 2.1 shows an example of an MPI application written in C. The code spawns a given number of processes, sets up one process as coordinator and the rest as slaves that send back a message to the coordinator saying that they are ready to work. As seen in the figure, each process has a unique identifier, and the communication between them is done by using the MPI_Send or MPI_Receive API calls (obviously, more complex programs will require more complex API calls).

```c
#include <assert.h>
#include <stdio.h>
#include <string.h>
#include <mpi.h>

int main(int argc, char **argv) {
    char buf[256];
    int my_rank, num_procs;
    /* Initialize the infrastructure necessary for communication */
    MPI_Init(&argc, &argv);
    /* Identify this process */
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
    /* Find out how many total processes are active */
    MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
    /* Until this point, all programs have been doing exactly the same.
       Here, we check the rank to distinguish the roles of the programs */
    if (my_rank == 0) {
        int other_rank;
        printf("We have %i processes.\n", num_procs);
        /* Send messages to all other processes */
        for (other_rank = 1; other_rank < num_procs; other_rank++) {
            sprintf(buf, "Hello %i!", other_rank);
            MPI_Send(buf, sizeof(buf), MPI_CHAR, other_rank,
                     0, MPI_COMM_WORLD);
        }
        /* Receive messages from all other process */
        for (other_rank = 1; other_rank < num_procs; other_rank++) {
            MPI_Recv(buf, sizeof(buf), MPI_CHAR, other_rank,
                     0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            printf("%s\n", buf);
        }
    } else {
        /* Receive message from process #0 */
        MPI_Recv(buf, sizeof(buf), MPI_CHAR, 0,
                 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        assert(memcmp(buf, "Hello ", 6) == 0),

        /* Send message to process #0 */
        sprintf(buf, "Process %i reporting for duty.", my_rank);
        MPI_Send(buf, sizeof(buf), MPI_CHAR, 0,
                 0, MPI_COMM_WORLD);
    }
    /* Tear down the communication infrastructure */
    MPI_Finalize();
    return 0;
}
```

FIGURE 2.1: MPI Hello example in C.
Source: Wikipedia, Message Passing Interface

Figure 2.2 shows the command line result and Figure 2.3 shows a diagram of executing the aforementioned code with 4 processes. Notice that the spawn time of the processes and the communication between them is not always done at the same time and thus, the diagram is only one of the possible execution diagrams of the same code. For instance, Process 0 will always send messages to Processes 1, 2 and 3 in the same order and will receive the messages back in the same order, but Processes 1, 2 and 3 can receive the message in different orders and send the reply in different orders. This issue is one of the hardest things to overcome when developing applications with MPI because blocking processes in a receive call can lead to significant overheads. For instance, in our diagram, Processes 2 and 3 have sent all their data, but Process 0 does not receive the message until the data from Process 1 is received.

```
$ mpicc example.c
$ mpirun -n 4 ./a.out
We have 4 processes.
Process 1 reporting for duty.
Process 2 reporting for duty.
Process 3 reporting for duty.
```

FIGURE 2.2: MPI Hello execution example

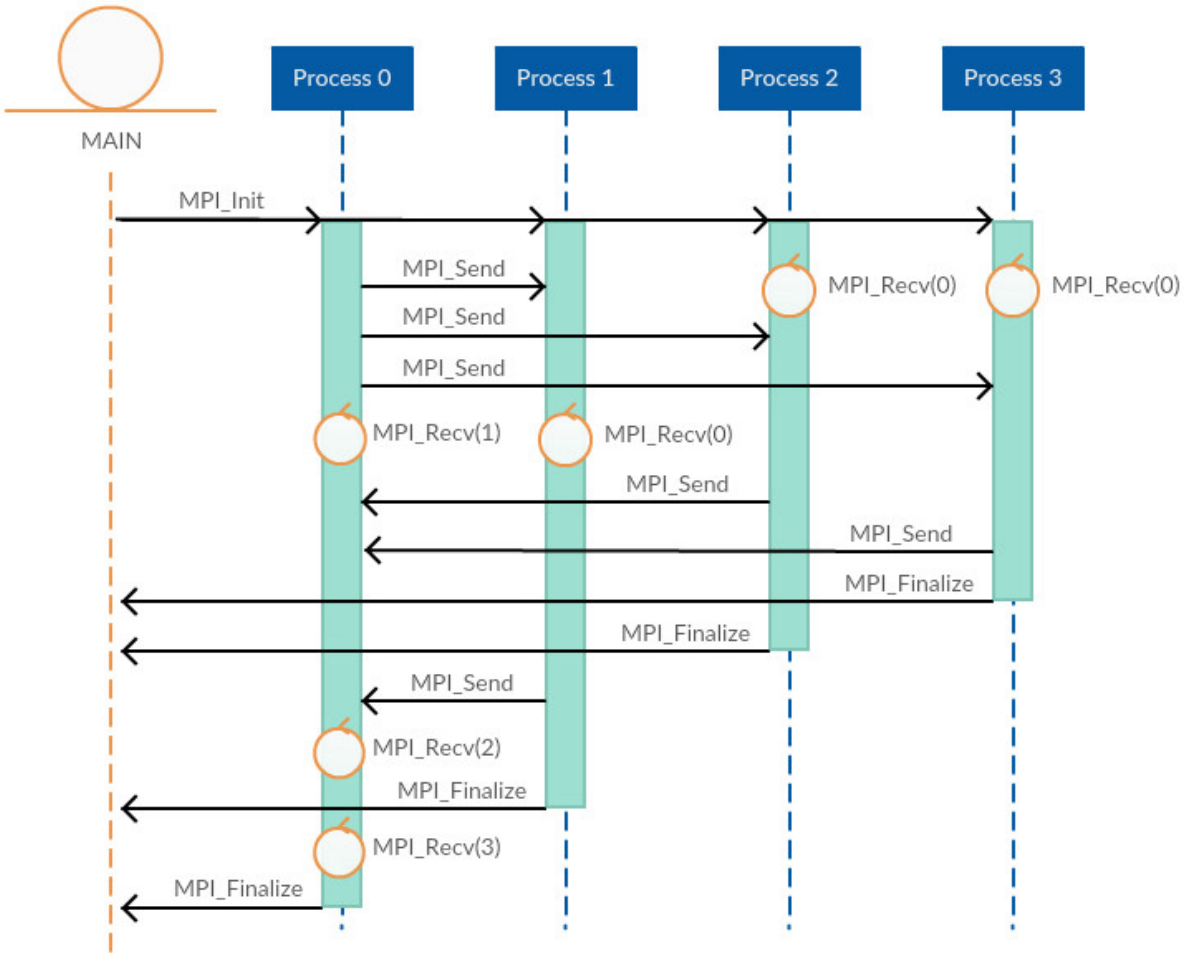

FIGURE 2.3: MPI Hello diagram of execution

2.1.2 Sockets

A Socket[42] is an endpoint on a machine to send and receive data and can be used either to communicate processes within the same machine or through the network. Each process refers to a socket by means of a *Socket Descriptor* and then establishes a connection through the socket creating a *Channel*. The underlying Operating System of the machine provides a socket API, but applications normally use a higher-level API implementation that depends on its language (for example, sys/socket.h for C [26], socket library for Python [49] or java.net.Socket for Java [40]).

Like the previous case, for the end user, using Sockets only enables the communication between different processes. Thus, it requires handling explicitly the spawn of the processes, the code executed by each process and the communication between them. Since the application will also use an API, the source code must be developed using this underlying technology and compiled accordingly. However, contrary to MPI, it does not require a specific compiler.

The next figures show the same example than the MPI case but written in Java and using Java Sockets. Figure 2.4 contains the main code to initialize and spawn all the processes.

```
package helloSocket;

public class HelloSocket {

    private static final int BASE_SOCKET_PORT = 47_000;
    private static int numThreads;
    private static int[] socketPorts;

    public static void main(String args[]) {
        // Get arguments
        if (args.length < 1) {
            System.err.print("[ERROR] Invalid usage");
            System.err.print("      Usage: ./main <numThreads>");
            System.exit(1);
        }
        numThreads = Integer.valueOf(args[0]);
        // Create the array of usable ports
        socketPorts = new int[numThreads];
        for (int i = 0; i < numThreads; ++i) {
            socketPorts[i] = BASE_SOCKET_PORT + i;
        }

        Thread[] procs = new Thread[numThreads];
        // Spawn the master process
        procs[0] = new MasterProcess(numThreads, socketPorts);
        procs[0].start();
        // Spawn the slave processes
        for (int i = 1; i < numThreads; ++i) {
            procs[i] = new SlaveProcess(i, socketPorts[i]);
            procs[i].start();
        }

        // Join the processes
        for (int i = 0; i < numThreads; ++i) {
            try {
                procs[i].join();
            } catch (InterruptedException ie) {
                Thread.currentThread().interrupt();
            }
        }
    }
}
```

FIGURE 2.4: Main code of Java Hello example with Sockets

Figure 2.5 contains the code for the main process (the coordinator).

```java
package helloSocket;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.Socket;


public class MasterProcess extends Thread {

    private static final String SERVER_MACHINE = null; // hostname

    private final int numThreads;
    private final int[] socketPorts;


    public MasterProcess(int numThreads, int[] socketPorts) {
        super();
        this.numThreads = numThreads;
        this.socketPorts = socketPorts;
    }

    @Override
    public void run() {
        System.out.println("We have " + this.numThreads + " processes");

        /* Send messages to all other processes */
        for (int i = 1; i < this.numThreads; ++i) {
            try (Socket socketClient = new Socket(SERVER_MACHINE, this.socketPorts[i])) {
                BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(
                                            socketClient.getOutputStream()));
                writer.write("Hello " + i);
                writer.flush();
            } catch (IOException e) {
                System.err.println("[ERROR] Cannot send message to client "
                                   + this.socketPorts[i]);
                e.printStackTrace();
            }
        }

        /* Receive messages from all other process */
        for (int i = 1; i < this.numThreads; ++i) {
            try (Socket socketClient = new Socket(SERVER_MACHINE, this.socketPorts[i])) {
                BufferedReader reader = new BufferedReader(new InputStreamReader(
                                            socketClient.getInputStream()));
                String msg = null;
                while ((msg = reader.readLine()) != null) {
                    System.out.println(msg);
                }
            } catch (IOException e) {
                System.err.println("[ERROR] Cannot receive message from client "
                                   + this.socketPorts[i]);
                e.printStackTrace();
            }
        }
    }

}
```

FIGURE 2.5: Master process code of Java Hello example with Sockets

Figure 2.6 contains the code for the slave processes.

```
package helloSocket;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.ServerSocket;
import java.net.Socket;


public class SlaveProcess extends Thread {

    private final int id;
    private final int socketPort;


    public SlaveProcess(int id, int socketPort) {
        super();
        this.id = id;
        this.socketPort = socketPort;
    }

    @Override
    public void run() {
        try (ServerSocket mySocket = new ServerSocket(this.socketPort)) {
            /* Receive message from process #0 */
            try (Socket connectionSocket = mySocket.accept()) {
                BufferedReader reader = new BufferedReader(new InputStreamReader(
                                        connectionSocket.getInputStream()));
                String msg;
                while ((msg = reader.readLine()) != null) {
                    assert (msg.equals("Hello" + this.id));
                }
            } catch (IOException e) {
                System.err.println("[ERROR] Thread " + this.id
                                   + " cannot read from process 0 socket");
                throw e;
            }

            /* Send message to process #0 */
            try (Socket connectionSocket = mySocket.accept()) {
                BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(
                                        connectionSocket.getOutputStream()));
                writer.write("Process " + this.id + " reporting for duty.");
                writer.flush();
            } catch (IOException e) {
                System.err.println("[ERROR] Thread " + this.id
                                   + " cannot write to process 0 socket");
                throw e;
            }
        } catch (IOException e) {
            System.err.println("[ERROR] Thread " + this.id + " cannot open socket");
            e.printStackTrace();
        }
    }

}
```

FIGURE 2.6: Slave process code of Java Hello example with Sockets

Notice that, like in the previous case, since the master process reads the messages in an ordered manner, the execution output is always the same (see Figure 2.7) even if the process spawn order and the messages' arrival can vary between executions.

```
$ javac helloSocket/*
$ jar cf hello.jar helloSocket/
$ java -cp helloSocket.jar helloSocket.HelloSocket 4
We have 4 processes
Process 1 reporting for duty.
Process 2 reporting for duty.
Process 3 reporting for duty.
```

FIGURE 2.7: Execution example of Hello with Sockets

2.2 Workflow Frameworks

On the other hand, distributed computing can also be achieved by using a workflow framework. Frameworks are designed to encapsulate a programming model, a runtime or even a programming language that eases the development of distributed applications. Although all the frameworks hide the communication mechanisms between the different processes, they can be classified regarding its task definition. Thus, subsection 2.2.2 provides several examples of *frameworks with implicit workflows' definition* and subsection 2.2.1 provides several other examples of *frameworks with explicit workflows' definition*.

2.2.1 Frameworks with explicit workflows' definition

Frameworks with explicit workflows' definition allow the users to design the full application workflow using a receipt file or a visualization tool. The main advantage is that the users can specifically control the dependencies between the different stages and, thus, they have a clear overview of how their application is executed by the framework. However, its main advantage is also its main disadvantage, because designing complex workflows can result in a tedious work.

2.2.1.1 Taverna

Taverna [44] is a Workflow Management System that includes a set of tools to design and execute scientific workflows. The Taverna Suite consists of three components: Taverna Engine (used for enacting workflows), Taverna Workbench (the desktop client application) and the Taverna Server (to execute the remote workflows). It has been recently moved to the Apache Incubator project, and it is beeing used in a wide variety of domains such as bioinformatics, biodiversity, chemistry, astronomy, data mining, digitalisation and image analysis.

Although Taverna has a command line interface, its most powerful tool is the visual workflow design. This tool allows the users to graphically define workflows by constructing a diagram with inputs, outputs, actions and dependencies between actions. Each action defines its input and output ports so that users can incrementally build a diagram by linking the input and output ports of the different actions or by adding static content values to the input ports. Notice that the actions can be of any type of service since Taverna provides a set of internally configured services but allows the users to add any service by providing its WSDL address. Moreover, when the workflow is finished, Taverna can validate and run it; allowing the users to check the execution status and the partial results.

Figure 2.8 shows a Taverna diagram example of a BLAST application. The workflow has four outputs generated by different actions and does not have any input since the values are obtained from static content values (i.e. program value or database value). The light blue,

green and magenta actions are services; beeing the green one the invocation to BLAST with program, database and query sequence.

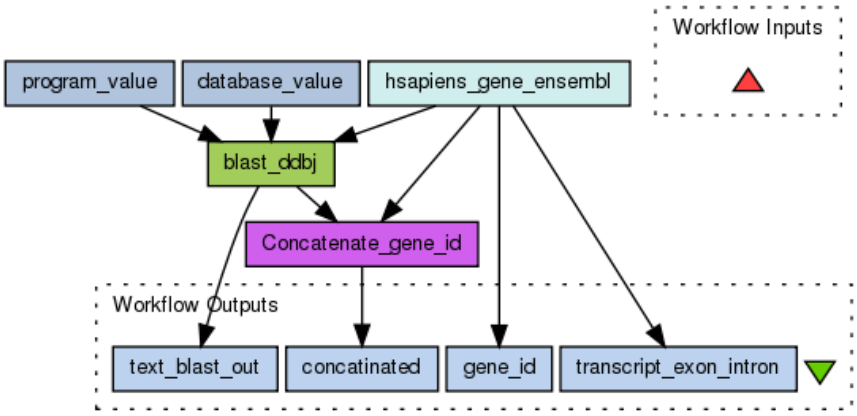

FIGURE 2.8: BLAST design example using Taverna
Source: myExperiment[51]. Taverna workflows

As previously demonstrated, Taverna is a powerful tool to design workflows for inexperienced users since they do not have to deal directly with parallelism issues. However, this framework is only oriented to applications that can be defined as a pipeline of services and thus, its execution relies on the service availability. In this sense, the community has done a hard work in making available several services and predefined workflows that can be used as nested workflows at the end user applications.

2.2.1.2 Fireworks

FireWorks [32] is an open source Framework to define, manage and execute workflows. Workflows are defined using Python, JSON [46] or YAML [47], stored using MongoDB [37] and can be monitored through a web interface. The workflows' execution can be automated over arbitrary computing resources, and the framework provides fault-tolerant mechanisms and multiple execution modes (to run on different underlying infrastructures such as multi-core machines or clusters managed by queues). FireWorks includes two components (see Figure 2.9): LaunchPad (a server that manages the workflows) and one or more FireWorkers (a worker to run the jobs).

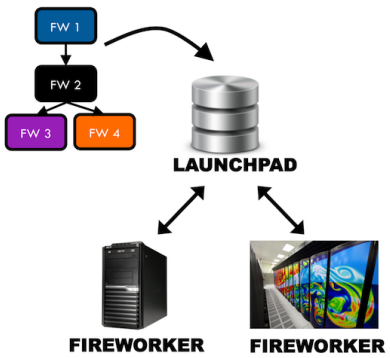

FIGURE 2.9: FireWorks components. Source: FireWorks[32]

Workflows in FireWorks have three main components (see Figure 2.10). Firstly, the users can define a FireTask which represents an atomic computing job. FireTasks must execute a single shell script or a Python function. Secondly, users can define a FireWork, which contains all the information needed to bootstrap the job execution. For example, a FireWork may contain a list of FireTasks to execute sequentially and its parameters. Finally, a Workflow is a set of Fireworks with dependencies between them.

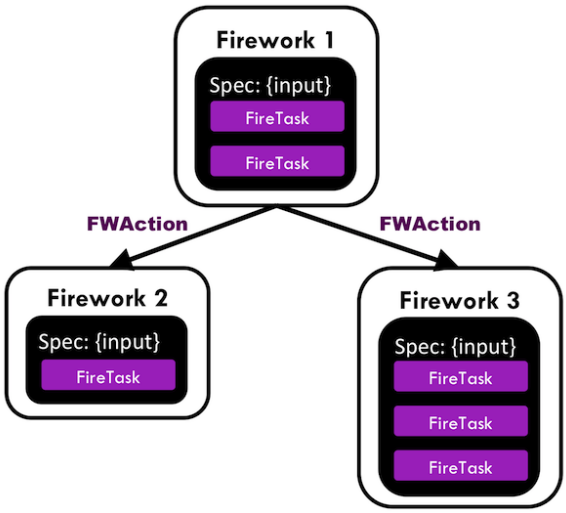

FIGURE 2.10: Workflows' components in FireWorks
Source: FireWorks[32]

Next, Figure 2.11 shows a Python example using FireWorks. At left, we provide the code to create the workflow and, at right, a diagram of the obtained workflow. Notice that the FireTasks and FireWorks are easily created but that the Workflow must be explicitly constructed which, for complex workflows, can be difficult to understand. Thus, although the users do not need to lead with parallelism directly, they must define the workflow explicitly.

```
from fireworks import Firework, Workflow, FWorker, LaunchPad, ScriptTask
from fireworks.core.rocket_launcher import rapidfire

# set up the LaunchPad and reset it
launchpad = LaunchPad()
launchpad.reset('', require_password=False)

# define four individual FireWorks used in the Workflow
task1 = ScriptTask.from_str('echo "Ingrid is the CEO."')
task2 = ScriptTask.from_str('echo "Jill is a manager."')
task3 = ScriptTask.from_str('echo "Jack is a manager."')
task4 = ScriptTask.from_str('echo "Kip is an intern."')

fw1 = Firework(task1)
fw2 = Firework(task2)
fw3 = Firework(task3)
fw4 = Firework(task4)

# assemble Workflow from FireWorks and their connections by id
workflow = Workflow([fw1, fw2, fw3, fw4], {fw1: [fw2, fw3], fw2: [fw4], fw3: [fw4]})

# store workflow and launch it locally
launchpad.add_wf(workflow)
rapidfire(launchpad, FWorker())
```

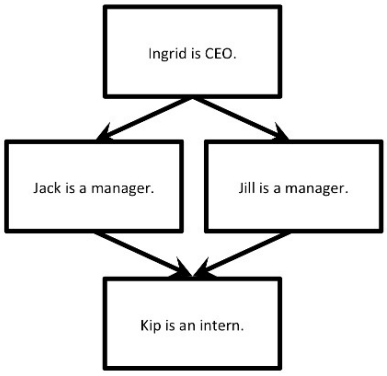

FIGURE 2.11: Example of Workflow using FireWorks
Source: FireWorks[32]

2.2.1.3 Kepler

Kepler [5] is designed to create, execute and share models and analyses of scientific and engineering workflow applications. It can easily merge R [24] scripts, compiled C code or facilitate the execution of models remotely. The users use a Graphical User Interface to select and connect analytical components and data sources to define a scientific workflow.

Scientific Workflows in Kepler consist of customizable components, relations, and ports. On the one hand, the components can be either a director (to control the execution of a workflow), an actor (to execute the instructions given by the director) or a parameter (to add a configurable value). On the other hand, relations and ports are used to facilitate communication between the different components. Figure 2.12 shows a Lotka-Volterra workflow defined in the Kepler Interface where the components, the relations, and the ports are identified with callouts.

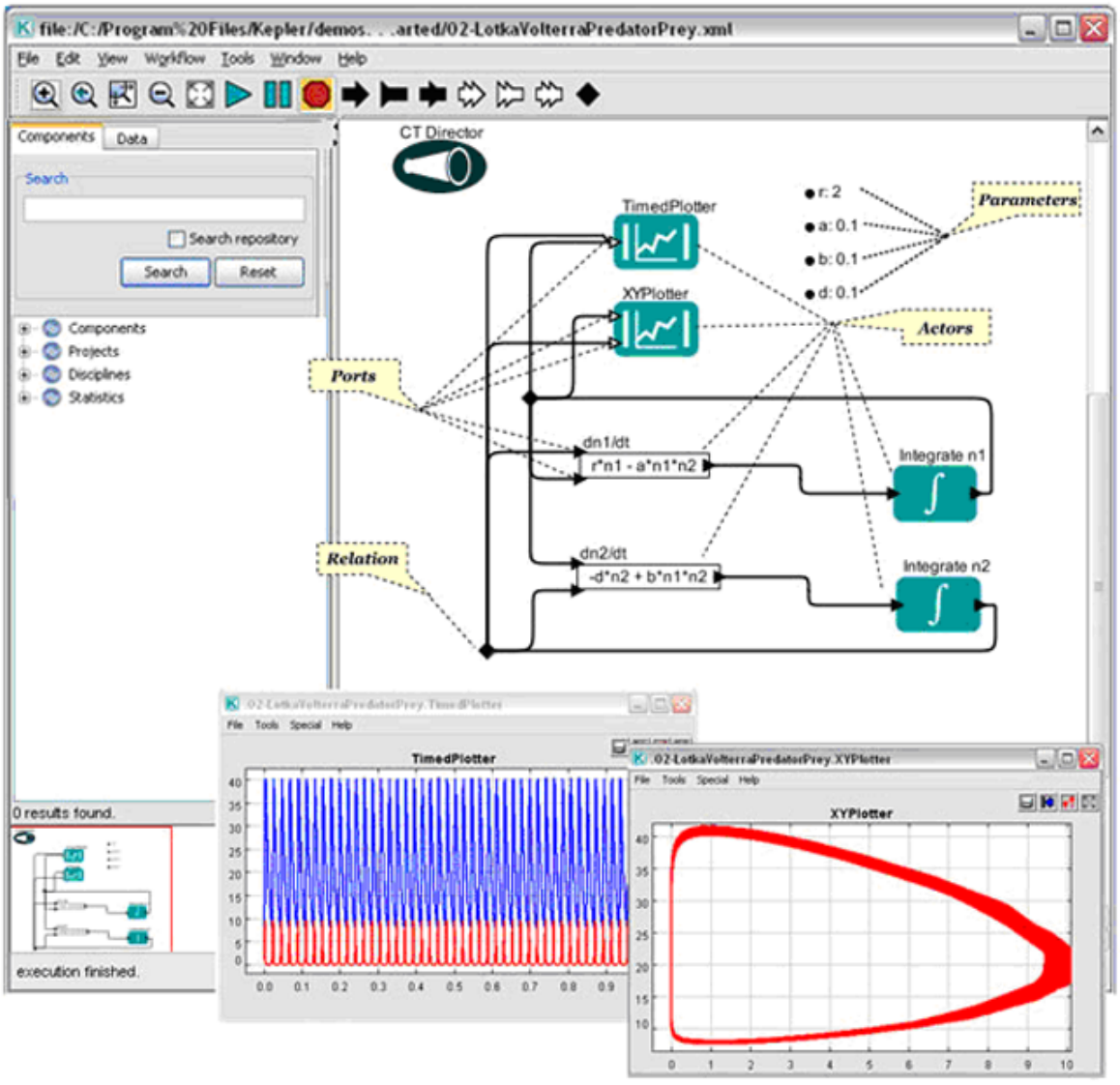

FIGURE 2.12: Lotka-Volterra workflow example using Kepler
Source: Kepler examples

Although the Kepler Interface remembers to Taverna, this Framework does not restrict the users to execute services but rather allows them to orchestrate any type of execution.

However, like Taverna, Kepler is designed for users with little experience in parallel computing and all its efforts are intended to make the workflow design easier.

2.2.1.4 Galaxy

Galaxy [28] is a web-based platform for biomedical research. It is designed for users without programming experience, providing a graphical interface to easily specify parameters and run tools and workflows. One of the greatest advantages of Galaxy is to share, publish, access and reproduce any analysis of the other users in an interactive web-based framework.

Galaxy workflows are graphically defined through its web-based platform (see Figure 2.13) by combining the execution of different tools. Several tools are provided for every user (such as retrieving data, calculating statistics or performing complex genome operations) but the users can also define custom tools. To build a workflow, the users define data dependencies between the tools' execution.

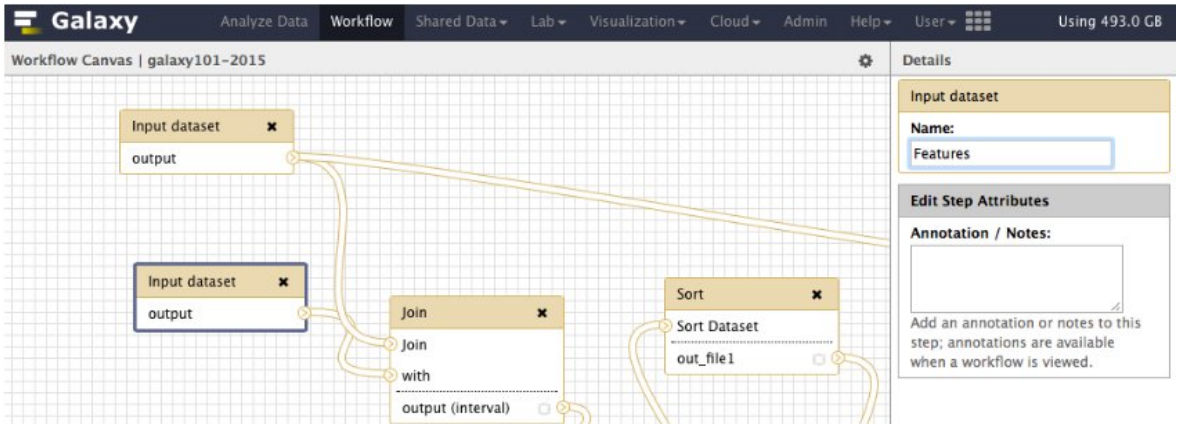

FIGURE 2.13: Galaxy graphical web-based platform to define Workflows
Source: Galaxy Project tutorials

Galaxy also provides a section to execute the workflow. On execution time, the platform retrieves information of the execution of each tool, providing a live monitoring of the whole workflow (see Figure 2.14). Once the application has finished, the users can also analyze the output data.

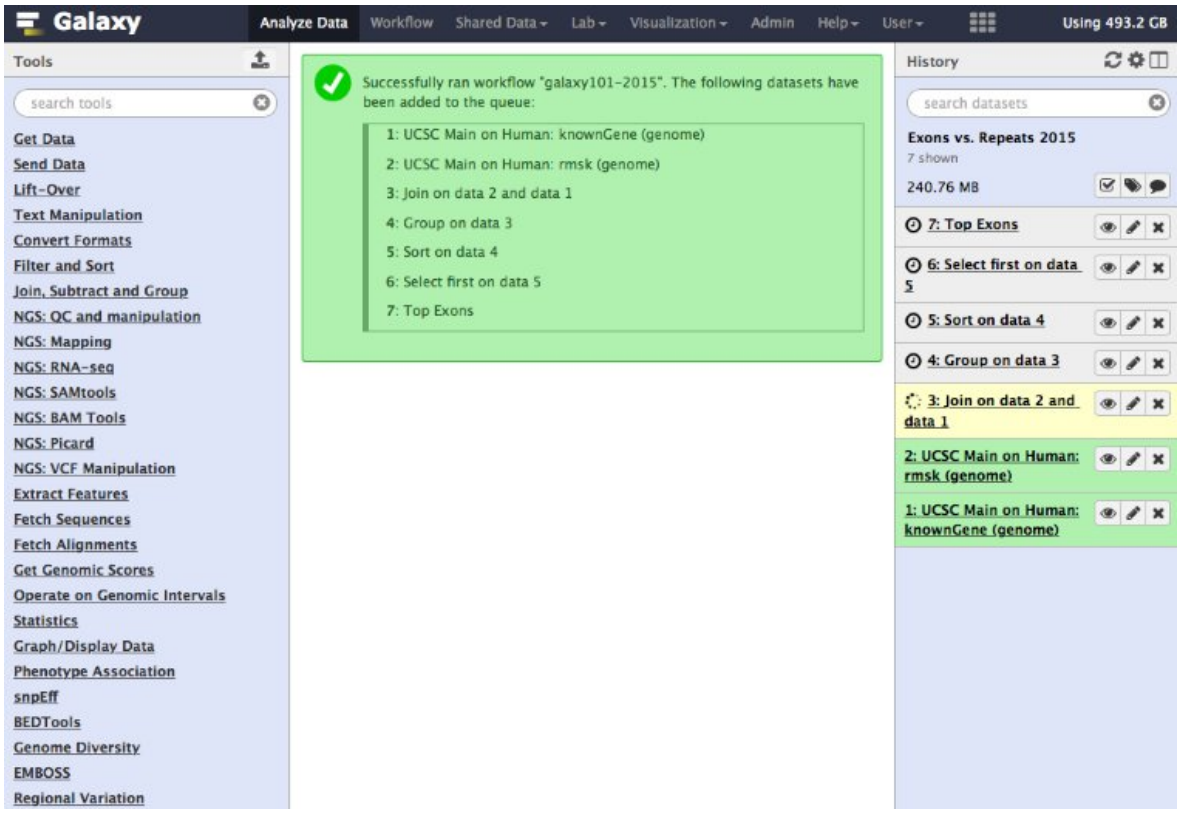

FIGURE 2.14: Galaxy graphical web-based platform to execute Workflows
Source: Galaxy Project tutorials

Notice that Galaxy is built for specific biomedical research tools, rather than a general purpose platform. Although it provides a very clean, easy and fast way to build, execute and analyse workflows, it is only designed for biomedical applications.

2.2.2 Frameworks with implicit workflows' definition

Frameworks with implicit workflows' definition use different technologies to define the parallelism automatically. Users develop applications in an almost sequential manner and they do not need to handle explicitly the tasks spawned by a method call (that are distributed among the available resources). Consequently, the different frameworks differ on this "almost sequential manner" to develop the application. The main advantage of these frameworks is that the complexity of distributed programming is almost reduced to zero. However, the main disadvantage is that the users do not know beforehand how the framework will execute their application (for example, how many tasks will be created in a specific call).

2.2.2.1 MapReduce

MapReduce [21] is a programming model (and an underlying runtime) for processing and generating large datasets. The underlying runtime can automatically handle the parallelism; managing the inter-machine communication, taking advantage of the data locality and providing fault-tolerant mechanisms (recovering from partial failures of servers and storages by rescheduling the task jobs). In this sense, the users are unaware of the parallelism.

However, although it is a powerful and simple programming model, it is only suitable for applications that can be expressed in terms of *mappers* and *reducers*. This means that the applications must be designed with a *map* and a *reduce* function so that the MapReduce framework can only exploit the inherent parallelism.

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;


public class WordCount {

    public static class TokenizerMapper extends Mapper<Object,Text,Text,IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(Object key, Text value, Context context)
                throws IOException, InterruptedException {

            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());
                context.write(word, one);
            }
        }
    }

    public static class IntSumReducer extends Reducer<Text,IntWritable,Text,IntWritable> {
        private IntWritable result = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values, Context context)
                throws IOException, InterruptedException {

            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

FIGURE 2.15: Wordcount example on top of *Hadoop*
Source: Hadoop Map Reduce tutorial

Figure 2.15 shows the implementation of a Wordcount application using the *Hadoop* [43] implementation of the MapReduce programming model. The Wordcount application counts the number of occurrences of each word in a given text, and it is a commonly used example for MapReduce because it is conceptually easy to define the map and the reduce functions. Notice that, in opposite to the previous frameworks, the user defines the *map*, *reduce* and the main code in a sequential-like way; without directly dealing with parallelism and without specifying the execution workflow.

When executing the previous MapReduce application (see Figure 2.16), the runtime processes the input, divides it into smaller sub-problems and distributes each subproblem to one of the available worker nodes. A worker can then repeat this step, leading to a multi-level tree structure where the intermediate nodes merge their worker results and send them back to their master node, or simply compute the result of the subproblem. In any case, the last master node merges all the intermediate values, collecting and combining the answers of all the sub-problems to obtain the final output.

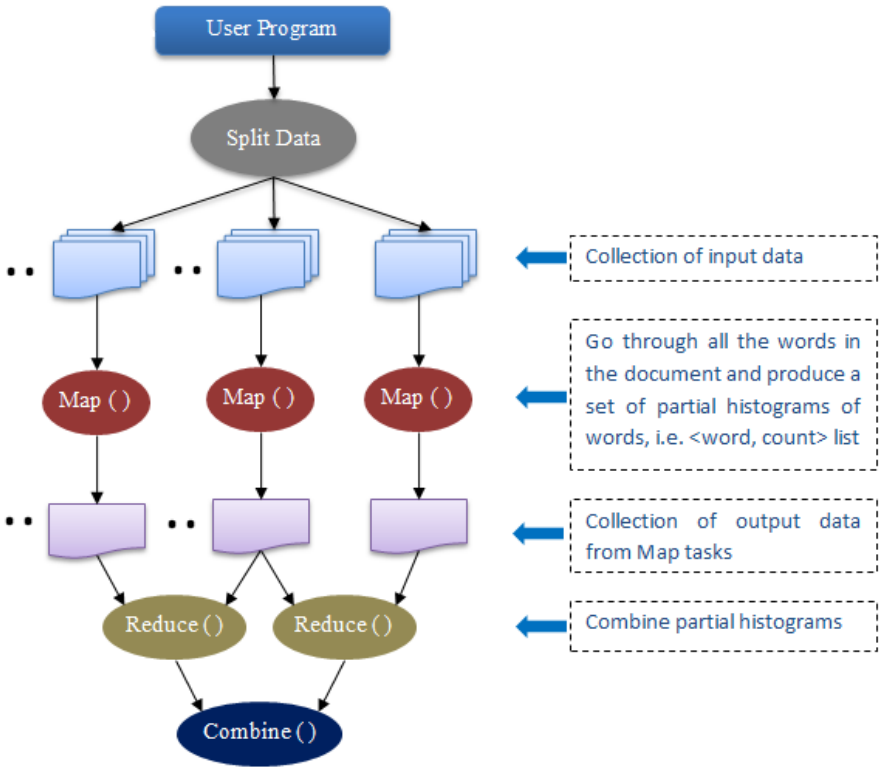

FIGURE 2.16: Execution example of Wordcount using MapReduce
Source: SALSA Group, PTI Indiana University, Wordcount User Guide

As a final note, we must highlight that the MapReduce programming model is very popular since it has been implemented in several programming languages (including C++, Python, and Java) and implementations like *Hadoop* are widely used.

2.2.2.2 Spark

Apache Spark [52] is a fast and general-purpose large-scale data processing engine that supports general execution graphs, and that can run on top of Hadoop, Mesos, standalone

or in the cloud (see Figure 2.17). It extends the MapReduce programming model to efficiently support other types of computations (such as batch applications, interactive algorithms, SQL queries or streaming) and by allowing in-memory processing. These features make easy and inexpensive for users to combine different workloads and define complex pipelines. Moreover, it has been designed to be highly accessible, providing a simple API in Java, Scala [17], Python, R and SQL, and supporting a rich set of high-level tools including Spark SQL for SQL and structured data processing, MLlib for machine learning, GraphX for graph processing, and Spark Streaming.

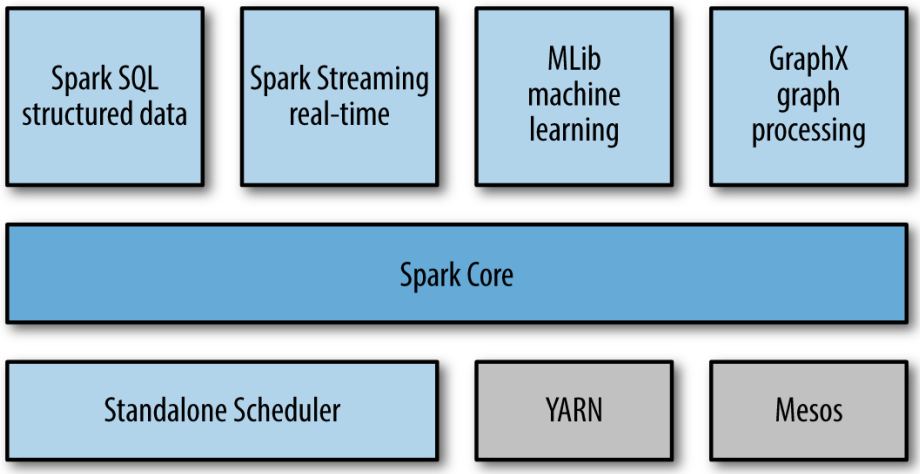

FIGURE 2.17: Spark's Components
Source: Learning Spark [35]

"At a high level, every Spark application consists of a driver program that runs the user's main function and executes various parallel operations on a cluster. The main abstraction Spark provides is a Resilient Distributed Dataset (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel"[52]. For the RDD creation, Spark relies on its underlying file system.

Figure 2.18 provides the Wordcount implementation in Java using the Spark API. Although the code can be written in a more compact way, we have kept it unrolled for the sake of clarity and to be easily mapped to the functions defined by the MapReduce programming model. Thus, as in the MapReduce example shown before, the functions provided are the Map and Reduce functions, and the main code is written in a sequential-like way. Notice that the code becomes significantly more readable than in the previous example and, as expected, the user does not explicitly define the workflow. In fact, when using Spark to develop an application, the user is unaware of the underlying infrastructure and the workflow diagram.

We must highlight that Apache Spark is widely used nowadays and it is becoming one of the most powerful Big Data engines. Although it still relies on the MapReduce programming model (making difficult the implementation of sophisticated algorithms which depend on the state of the processes running on other nodes), Spark provides an efficient and complete framework that separates the user from the common distributed programming issues such as data dependency analysis, task scheduling, fault-tolerant mechanisms and workers communication.

```java
import java.util.Arrays;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.api.java.function.FlatMapFunction;
import org.apache.spark.api.java.function.Function2;
import org.apache.spark.api.java.function.PairFunction;
import scala.Tuple2;

public class WordCount {

    private static final FlatMapFunction<String, String> SPLIT_WORDS
        = new FlatMapFunction<>() {
            public Iterable<String> call(String s) {
                return Arrays.asList(s.split(" "));
            }
        };

    private static final PairFunction<String, String, Integer> MAP_WORDS
        = new PairFunction<>() {
            public Tuple2<String, Integer> call(String s) {
                return new Tuple2<String, Integer>(s, 1);
            }
        };

    private static final Function2<Integer, Integer, Integer> REDUCE_WORDS
        = new Function2<>() {
            public Integer call(Integer a, Integer b) {
                return a + b;
            }
        };

    public static void main(String[] args) throws Exception {
        String inputFile = args[0];
        String outputFile = args[1];
        // Create a Java Spark Context
        SparkConf conf = new SparkConf().setAppName("wordCount");
        JavaSparkContext sc = new JavaSparkContext(conf);
        // Load input data
        JavaRDD<String> input = sc.textFile(inputFile);
        // Split
        JavaRDD<String> words = input.flatMap(SPLIT_WORDS);
        // Transform into words and count
        JavaPairRDD<String, Integer> pairs = words.mapToPair(MAP_WORDS);
        JavaPairRDD<String, Integer> counter = pairs.reduceByKey(REDUCE_WORDS);
        // Save the word count
        counter.saveAsTextFile(outputFile);
    }
}
```

FIGURE 2.18: Wordcount example in Java using Spark

2.2.2.3 Swift

Swift [53] is a fast and easy scripting language for executing distributed and parallel applications. The programming language is structured like a Shell script and builds a data-flow oriented workflow. The Swift programs run many copies of ordinary applications as soon as their inputs are available. The main advantage is that its workflows can be fast and easily defined, and can run over any underlying architecture (multicore computers, clusters, clouds or grids). The users must consider that Swift it is not designed to process large collections of data but to orchestrate programs that do that processing. In this sense, it is intended to be a pure Workflow Manager: handling the execution of such programs on remote sites, staging input and output files from these sites and choosing these sites.

In opposition to the previous cases, Swift is a programming language itself built on top of Java. Consequently, it provides data types, mapped data types, mapped functions, conventional expressions, structured data, loops and data flow instructions so that users can define their workflows (see Figure 2.19). Additionally, Swift uses a configuration file to specify the underlying infrastructure.

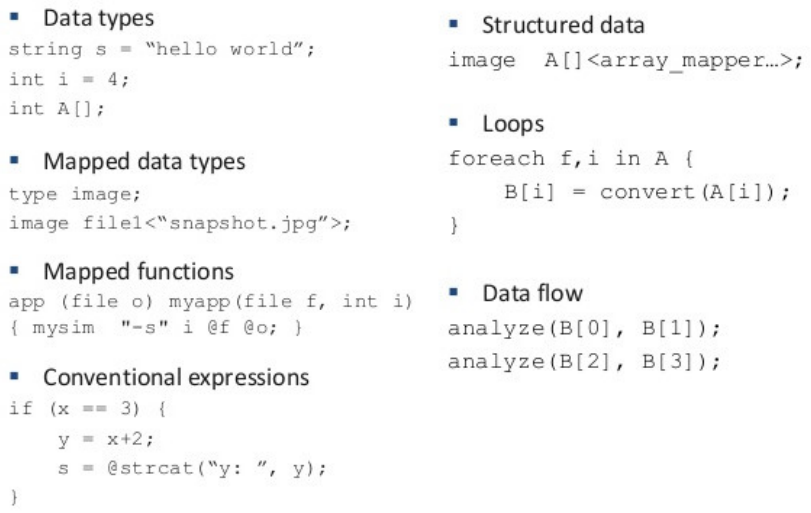

FIGURE 2.19: Swift programming language. Source: Swift [53]

To exemplify how Swift works, Figure 2.20 shows the simulation workflow of the Swift code shown in Figure 2.21. Notice that the code first defines the two applications that the workflow will invoke: the simulation and the analysis. For both applications, it also defines their input and output parameters, and their execution instructions (in both cases, the execution is an invocation of a bash script to launch the simulation or the analysis). Next, it retrieves the workflow arguments and define the intermediate variables needed to execute the workflow. In a third step, it spawns *nsim* simulations; storing its result into separated files. Finally, it launches an *analyze* application that uses the output files of all the previously launched simulations.

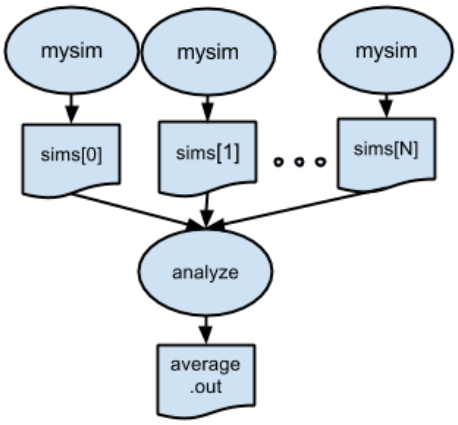

FIGURE 2.20: Swift simulation workflow example. Source: Swift tutorial[53]

```
type file;

app (file out, file log)

simulation (int sim_steps, int sim_range, int sim_values, file sim_script)
{
  bash @sim_script "--timesteps" sim_steps "--range" sim_range
                   "--nvalues" sim_values stdout=@out stderr=@log;
}

app (file out, file log) analyze (file s[], file stat_script)
{
  bash @stat_script filenames(s) stdout=@out stderr=@log;
}

int nsim   = toInt(arg("nsim",   "10"));
int steps  = toInt(arg("steps",  "1"));
int range  = toInt(arg("range",  "100"));
int values = toInt(arg("values", "5"));

file sims[];
file simulate_script <"simulate.sh">;
file stats_script <"stats.sh">;

foreach i in [0:nsim-1] {
  file simout <single_file_mapper; file=strcat("output/sim_",i,".out")>;
  file simlog <single_file_mapper; file=strcat("output/sim_",i,".log")>;
  (simout,simlog) = simulation(steps,range,values,simulate_script);
  sims[i] = simout;
}

file stats_out<"output/average.out">;
file stats_log<"output/average.log">;
(stats_out, stats_log) = analyze(sims,stats_script);
```

FIGURE 2.21: Swift simulation code example

We must highlight that Swift builds the workflow considering the data dependencies between applications and exploits the maximum parallelism between applications. Thus, users must define the workflow regarding data dependencies between applications but do not need to build the workflow structure explicitly. Moreover, the applications can be any type of executable, like a shell does, and do not need to follow any convention (like MapReduce obligates users to define *Map* and *Reduce* functions).

2.2.2.4 COMP Superscalar (COMPSs)

Since this project is based in COMP Superscalar (COMPSs) [19] we have decided to provide a more in-depth view of it in Chapter 3.

Chapter 3

COMPSs overview

COMP Superscalar (COMPSs) [4] [33] is a task-based programming model that belongs to the family of Frameworks with implicit workflows. COMPSs applications consist of three parts: the application's code developed in a totally sequential manner, an application interface where the programmers specify which functions can be remotely executed (*tasks*) and a configuration file that describes the underlying infrastructure. With these three components, the COMPSs Runtime system exploits the inherent parallelism of the application at execution time by detecting the task calls and the data dependencies between them.

COMPSs natively supports Java applications but also provides bindings for Python (PyCOMPSs [23]) and C/C++. Furthermore, COMPSs allows applications to be executed on top of different infrastructures (such as multi-core machines, grids, clouds or containers) without modifying a single line of the application's code (see Figure 3.1). It also has fault-tolerant mechanisms for partial failures (with job resubmission and reschedule when task or resources fail), has a live monitoring tool through a built-in web interface, supports instrumentation using the Extrae [8] tool to generate post-mortem traces that can be analysed with Paraver [15], has an Eclipse IDE, and has pluggable cloud connectors and task schedulers.

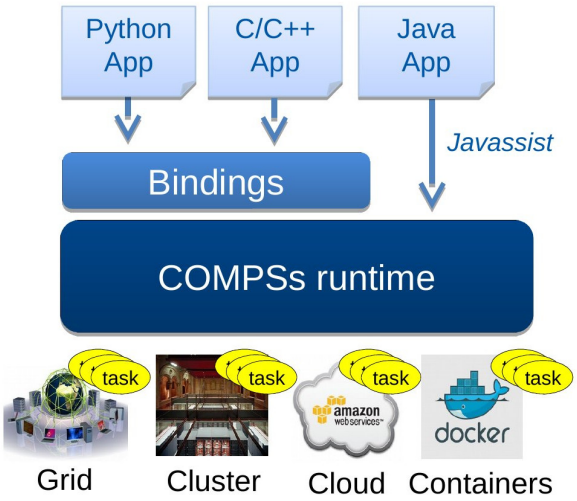

FIGURE 3.1: COMPSs overview

Additionally, the COMPSs model has three key characteristics:

- **Sequential Programming:** The users do not need to deal with any parallelization and distribution issue such as thread creation, synchronization, data distribution, messaging or fault-tolerance. COMPSs programmers only select which methods must be considered as tasks and the COMPSs Runtime spawns them asynchronously on a set of resources instead of executing them locally and sequentially.

- **Infrastructure Agnostic:** COMPSs model abstracts the application from the underlying infrastructure. Hence, COMPSs applications do not include any platform related detail such as deployment or resource management. This feature makes applications portable between infrastructures with different characteristics.
- **No APIs:** When using COMPSs native language, Java, the model does not require any special API, pragma or construct in the program. Since COMPSs instruments the application's code at execution time to detect the tasks, everything can be developed in the standard Java syntax and libraries.

3.1 Programming Model

The COMPSs programming model mainly involves choosing the right tasks for a sequential application. To illustrate how COMPSs applications are built from a sequential code, Figure 3.2 provides the main code and Figure 3.3 provides the helper methods of a sequential Increment application in Java. The Increment is a didactic application that takes *N* counters, initializes them to a random value and increments them by *U* units.

```
1  package increment;
2
3  import java.util.Random;
4
5
6  public class Increment {
7
8      private static final int MAX_INT = 10;
9
10     public static void main(String[] args) {
11         // Retrieve arguments
12         if (args.length != 2) {
13             System.err.println("[ERROR] Invalid number of arguments");
14             System.err.println("    Usage: increment.Increment <N> <U>");
15             System.exit(1);
16         }
17         int N = Integer.parseInt(args[0]);
18         int U = Integer.parseInt(args[1]);
19
20         // Initialize counters
21         Integer[] counters = new Integer[N];
22         for (int i = 0; i < N; ++i) {
23             counters[i] = new Random().nextInt(MAX_INT);
24             System.out.println("[LOG] Initial Counter " + i + " value is " + counters[i]);
25         }
26
27         // Increment U units each counter
28         for (int i = 0; i < U; ++i) {
29             for (int j = 0; j < N; ++j) {
30                 counters[j] = IncrementImpl.increment(counters[j]);
31             }
32         }
33
34         // Show final counter values
35         for (int i = 0; i < N; ++i) {
36             System.out.println("[LOG] Final Counter " + i + " value is " + counters[i]);
37         }
38     }
39 }
```

FIGURE 3.2: Increment main class

```java
package increment;

public class IncrementImpl {

    public static Integer increment(Integer counter) {
        // Return the increased counter
        return counter + 1;
    }

}
```

FIGURE 3.3: Increment helper methods class

After implementing the sequential code, the users must create an *Interface* to transform their sequential application into a COMPSs application. COMPSs requires this *Interface* to *annotate* which methods must be considered *tasks* that can be remotely executed.

COMPSs Interfaces must be defined inside a file with the same name than the main class of the users' application but with the "Itf" suffix (for instance, in the previous example, the Interface must be stored in the *IncrementItf.java* file). Regarding its content, the interface must contain one entry per task, which is annotated with the $@Method$ annotation [1]. For each *Method* annotation the users must also provide the *declaring class* and the parameters description. On the one hand, the declaring class of a function is the class containing the implementation of the task and its required to link the task to the method implementation. On the other hand, the parameters description is indicated by adding a $@Parameter$ annotation to every task parameter. This annotation is required to build the task dependency graph since COMPSs uses data-flow graphs. The mandatory contents of the *Parameter* annotation are the *Type* (that must refer to any Java basic type, a string, an object or a file) and the *Direction* (where the only valid values are *IN*, *OUT* and *INOUT*).

```java
package increment;

import integratedtoolkit.types.annotations.Parameter;
import integratedtoolkit.types.annotations.parameter.Direction;
import integratedtoolkit.types.annotations.parameter.Type;
import integratedtoolkit.types.annotations.task.Method;


public interface IncrementItf {

    @Method(declaringClass = "increment.IncrementImpl")
    Integer increment(
        @Parameter(type = Type.OBJECT, direction = Direction.IN) Integer counter
    );

}
```

FIGURE 3.4: Increment Interface

Figure 3.4 shows the Interface for the previous Increment application example. Notice that it only contains one task declaration, *increment*, with defined inside the *IncrementImpl* class and with two parameters. The first parameter is the return value of the function which has type *Integer* and, by default, has direction *OUT*. The second parameter is the *counter*

[1] There are more complex COMPSs annotations that are beyond the scope of this section. However, for any interested reader, the COMPSs annotations are described in-depth in the *COMPSs User Guide: Application Development* [19]

argument which has type *Integer* and direction IN because the function requires the input value of the parameter to increase it but does not modify it.

Since the COMPSs annotations do not interfere with the applications' code, all COMPSs applications can be sequentially executed. To do so, Figure 3.5 compiles the previous code and executes the application with $N = 2$ counters that must be increased by $U = 3$.

```
$ javac increment/*
$ jar cf increment.jar increment/
$ java -cp increment.jar increment.Increment 2 3
[LOG] Initial Counter 0 value is 7
[LOG] Initial Counter 1 value is 1
[LOG] Final Counter 0 value is 10
[LOG] Final Counter 1 value is 4
```

FIGURE 3.5: Sequential execution example of Increment

On the other hand, the code can also be executed with COMPSs without recompiling the application's code. To do so, the users must invoke the *runcompss* command instead of the traditional *java* command. When done, the COMPSs Runtime will be setup and the application will be distributedly executed. Figure 3.6 provides the execution output of the Increment application and the task graph generated by its execution.

```
$ runcompss -g increment.Increment 2 3
[  INFO] Using default execution type: compss
[  INFO] Using default location for project file
[  INFO] Using default location for resources file
[  INFO] Using default language: java

----------------- Executing increment.Increment --------------------------

WARNING: IT Properties file is null. Setting default values
[(790)    API]  -  Starting COMPSs Runtime v2.0
[LOG] Initial Counter 0 value is 5
[LOG] Initial Counter 1 value is 2
[LOG] Final Counter 0 value is 8
[LOG] Final Counter 1 value is 5
[(5090)    API]  -  Execution Finished

------------------------------------------------------------
```

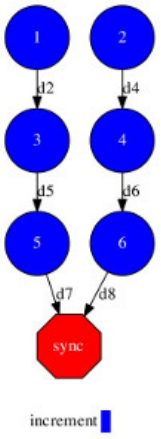

FIGURE 3.6: COMPSs execution example of Increment

The *runcompss* command has several command line arguments (that are fully detailed when executing *runcompss –help*) but Table 3.1 provides a short description of those we have found to be more useful.

Argument	Description
-d	Enables the debug mode
-g	Enables the final graph generation
-m	Enables the monitor tool
-t	Enables the tracing tool
- -sumary	Provides a task summary at the end of the execution
- -lang=<str>	Enables the Python and C/C++ bindings
- -project=<str>	Sets an specific project configuration file
- -resources=<str>	Sets an specific resources configuration file
- -classpath=<str>	Adds an specific classpath to the execution environment

TABLE 3.1: Useful arguments for the *runcompss* command

3.2 Runtime System

To abstract applications from the underlying infrastructure, COMPSs relies on its Runtime System to spawn a master process on the machine where the application is running and a worker process per available resource (see Figure 3.7). These processes are communicated through the network (using different communication adaptors) and can send messages to each other to orchestrate the distributed execution of the application.

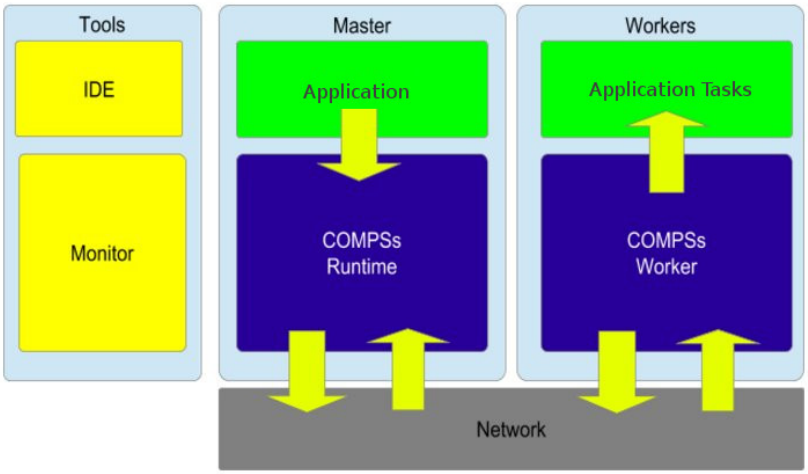

FIGURE 3.7: COMPSs structure

Once a Java application starts, the COMPSs Runtime [34] triggers a custom Java ClassLoader that uses Javassist [18] to instrument the application's main class. The instrumentation modifies the original code by inserting the necessary calls to the COMPSs API to generate tasks, handle data dependencies and add data synchronizations. To achieve the same purpose on Python applications, the Python Binding (PyCOMPSs) parses the decorators of the main code and adds the necessary calls to the COMPSs API. In the case of C/C++ applications, COMPSs also requires an Interface file that is used when compiling the application to generate stubs for the main code, add the required COMPSs API calls, and generate the

code for the tasks execution at the workers. In any case, as shown on the top of the Figure 3.8, the interaction between the application and the COMPSs Runtime is always made through the COMPSs API.

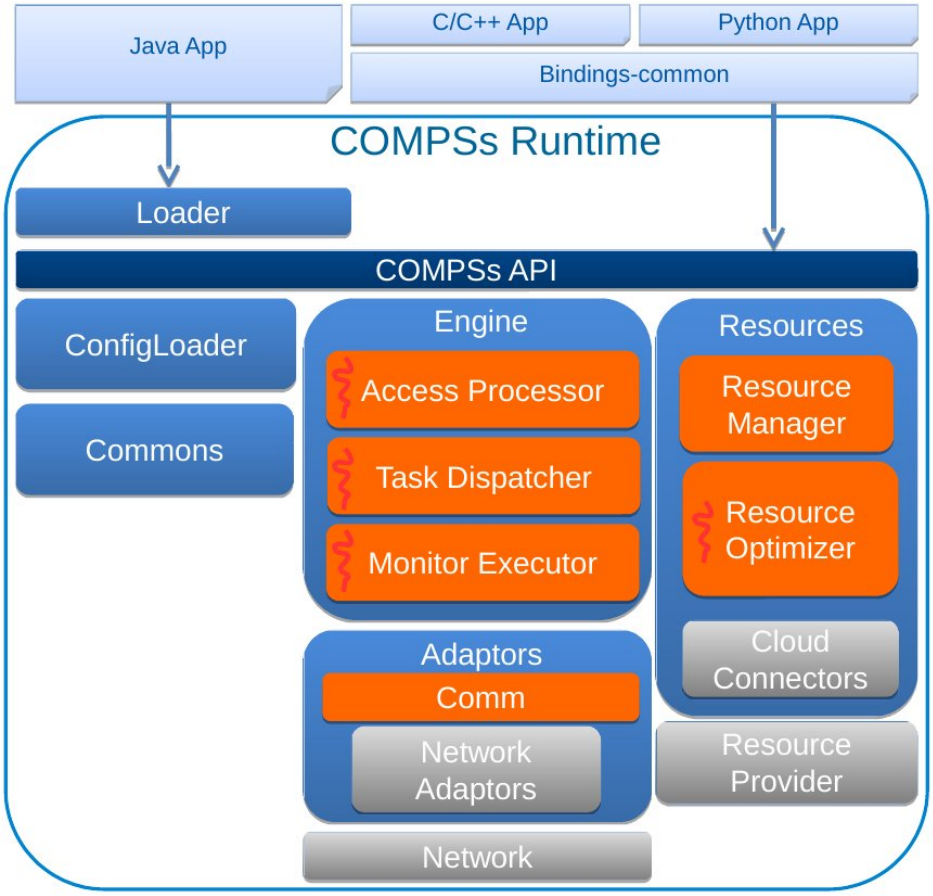

FIGURE 3.8: COMPSs Runtime overview

More in-depth, the COMPSs Runtime has five main components:

- **Commons** Contains the common structures used by all the Runtime components
- **ConfigLoader** Loads the project and the resources configuration files, the command line arguments, and the JVM configuration parameters.
- **Engine** Contains the submodules to handle the task detection, the data dependencies, and the task scheduling. More specifically, the *Access Processor* watches for the data accesses so that the Runtime can build the data dependencies between tasks, the *Task Dispatcher* controls the task life-cycle and the *Monitor Executor* controls the monitor structures for real-time and post-mortem monitoring.
- **Resources** Handles all the available resources in the underlying infrastructure. This component creates, destroys and monitors the state of all the available resources. Since COMPSs supports elasticity through cloud connectors, this component contains a *Resource Optimizer* subcomponent that takes care of creating and destroying resources.
- **Adaptors** Contains the different communication adaptors implementations. This layer is used to communicate the COMPSs Master and the COMPSs Workers and abstracts the rest of the Runtime from the different network adaptors.

3.3 Task Workflow

To clarify how COMPSs works when executing an application Figure 3.9 describes the task life-cycle. From the application's main code the COMPSs API registers the different tasks. Considering the registered tasks, COMPSs builds a task graph based on the data dependencies. This graph is then submitted to the *Task Dispatcher* that schedules the data-free tasks when possible. This means that a task is only scheduled when it is data-free, and there are enough free resources to execute it (each task can have different constraints, and thus, it is not scheduled if there is not a resource that satisfies the requirements).

Eventually, a task can be scheduled and, then, it is submitted to execution. This step includes the job creation, the transfer of the input data, the job transfer to the selected resource, the real task execution on the worker and the output retrieval from the worker back to the master. If any of these steps fail, COMPSs provides fault-tolerant mechanisms for partial failures.

Once the task has finished, COMPSs stores the monitoring data of the task, synchronizes any data required by the application, releases the data-dependant tasks so that they can be scheduled, and deletes the task.

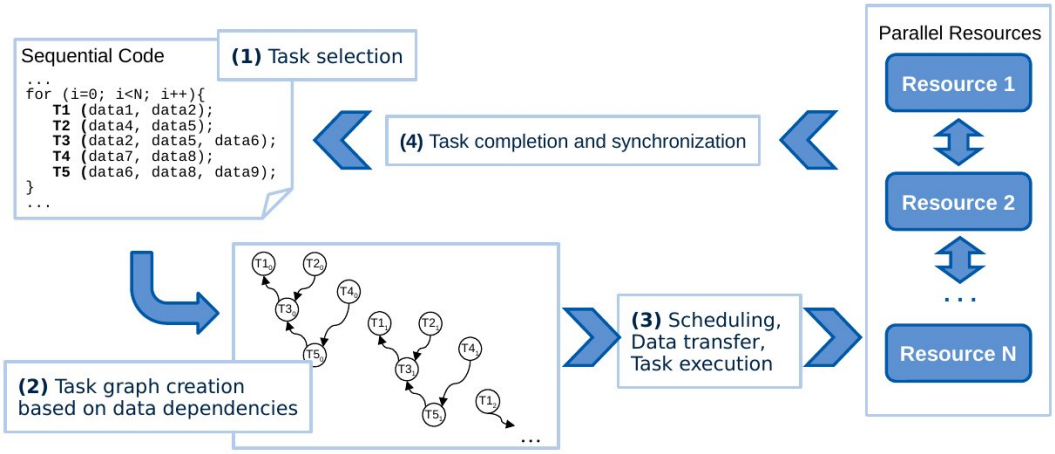

FIGURE 3.9: COMPSs task execution workflow

Chapter 4

Tools and methodology

On the one hand, this chapter lists the tools used to enhance COMPSs to enable analytic and HPC Workflows. Considering that this project is based on the extension of an already existing tool, COMPSs, most of the other tools used for its extension are already preset. On the other hand, Section 4.2, reports the development methodology of the project.

4.1 Tools

As stated, the principal tool used has been COMPSs as it was before this project (see Chapter 3). Since the COMPSs Runtime is developed in Java[39] language, the biggest part of the development of this project has been performed in Java language using the Eclipse[36] IDE and the Apache Maven[45] Software Project Management. The implementation has been completed with BASH[25] scripts to handle the communication between worker processes (see Section 5.3 for extended details).

For the results and the evaluation section, we have used Extrae[8] and Paraver[15] to validate the parallel executions and GnuPlot[27] to illustrate simulation results.

4.2 Methodology

We have selected a Design Research Method as the scientific method design, combined with a Test Driven Development strategy, always bearing in mind that the main goal of this project is to enhance COMPSs to enable analytic and HPC Workflows. Thus, we have selected conceptual use cases to theorize the COMPSs extensions (Section 4.2.1), carried them out following the development strategy (Section 4.2.2) and validated with real use cases (Section 4.2.3).

4.2.1 Scientific method design

In a first step, during the Relevance Cycle, we have analyzed analytic and HPC applications that can potentially be ported to COMPSs. In this sense, we have concluded that the COMPSs Programming Model lacks an easy way to execute binaries, MPI[38] and OmpSs[14] applications; which prevents some users from using COMPSs.

In a second step, during the Design Cycle, we have carefully selected two real use cases that represent the needs of the scientific community and which allow us to evaluate the requirements that must be fulfilled. The first application, *NMMB BSC-Dust* [12], is a model to predict the atmospheric life cycle of the desert dust for regional and global domains. The second application, *GUIDANCE* [9], is an integrated solution for Genome and Phenome association analysis.

However, the requirements of these applications turned out to be too complicated to be faced in a single development phase, which led us to find out simpler use cases and evaluate these during the validation phase. Consequently, we have faced the problem gradually

by dividing the requirements into two different categories. In the first category, we have considered the execution of simple binaries. To drain the standard binary execution requirements, we have analyzed a COMPSs version of the BLAST[6] application. To enhance COMPSs, we have proceeded gradually by defining incremental tests and finally achieving all the BLAST requirements. In the second category, we have considered a hybrid COMPSs and MPI version of the Matrix Multiplication, and we have enhanced COMPSs accordingly. As in the previous case, to ensure that all the requirements are fulfilled, we have performed incremental tests to finally execute this application.

Moreover, at the end of this second step, we have evaluated the real applications. We must highlight that we have gone through a validation process to evolve from the simpler use cases to the real ones.

Finally, during the Rigor Cycle, we have deployed the project development on a production environment, and we have largely documented the new artifacts to share the knowledge with the community.

4.2.2 Development strategy

The development has been based on the aforementioned proof of concept applications (BLAST and Matmul), and we have followed a Test Driven Development strategy. Although we have applied this methodology for the first time during this project, it turned out to be very robust, flexible and appropriate for the case.

More in-depth, we have defined several easy-to-run tests for each simple use case. Once identified the application requirements, we have implemented the features needed inside the COMPSs Runtime in a top-down strategy. The development has been performed in such a way, rather than following a bottom-up strategy, because the chosen use cases strongly define the requirements from the COMPSs API (method and parameter annotations) and are more flexible with regard to the COMPSs internals.

In this sense, from the first simple use case (BLAST), we have first enhanced the COMPSs Annotations to extend the method and the parameter annotation. Next, we have associated this information to the tasks and propagated it through the COMPSs Runtime up to the COMPSs Worker. Once all the information has been transferred to the COMPSs Worker, we have enhanced the task treatment (through new invokers) and, finally, we have implemented the binary executors.

From the second use case (Matmul), we have first extended our new annotation with an MPI method annotation. Then we have extended the COMPSs Schedulers to support multi-node tasks and, finally, we have extended the new invokers and executors to run MPI binaries.

We would like to highlight that the Test Driven Development has helped us significantly in making this process iterative without turning it into a nightmare. During the development, both use cases have been tested several times on the local machine to evaluate the development.

4.2.3 Validation strategy

Even if we have tested locally simple applications, an extensive validation has been performed to move from the simple use cases to the real applications. This validation process includes porting all the applications to the MareNostrum III [10] Supercomputer to execute and to perform an in-depth analysis of both types, the simple and the real of use cases (see Chapter 6 for more details).

Chapter 5

Implementation

5.1 Programming model annotations

As shown in Chapter 3, the COMPSs Programming model defines annotations that must be added to the sequential code in order to run the applications in parallel. These annotations can be splited into two groups:

- **Method Annotations** Annotations added to the sequential code methods to detect them as tasks and potentially execute them in parallel.
- **Parameter Annotations** Annotations added to the parameters of an annotated method to handle data dependencies and transfers.

Consequently, we have first extended the Method Annotations to detect Binary, MPI and OmpSs methods (from now on, *non-native* methods). Next, we have allowed the users to define annotations with environment variables. In this step, we have taken advantage of all these modifications to change also the annotations referring to the versioning and the scheduler hints. Finally, we have modified the parameter annotations to support specific binary needs.

5.1.1 New task annotations

As previously stated, we have decided to firstly extend the method annotations to recognize the different type of tasks. The task annotations are used by the COMPSs Runtime to detect the methods that the user wants to execute as tasks. The annotations are defined in a separated file (known as *interface*) that the COMPSs Loader uses to detect the tasks while instrumenting the application code. In this sense, the interface defines the signature of the method that the COMPSs Loader must transform into a task. Since a Java signature must contain the fully qualified name (that is, the declaring class of the method, the method name, and the parameter types), the COMPSs annotations provide a declaring class for each *Method* annotation.

To extend the annotations to support non-native tasks we have firstly modified the COMPSs Loader to consider as potential task any method defined inside the *binary.BINARY*, *mpi.MPI* and *ompss.OMPSS* classes. The methods found in these classes are then cross-validated with the tasks defined in the interface and selected as tasks if needed. This design decision is motivated by the fact that we consider that the annotation of non-native tasks must only refer to the real execution and, thus, we want to avoid a *declaringClass* field in the new annotation.

Secondly, we have extended the method annotations themselves to recognize the different type of tasks. For each non-native method that we are willing to support (execution of binaries, MPI binaries, and OmpSs binaries), we have created a new annotation that extends the information of the current *Method* annotation with the specificities of each type.

In the Binary case, we have created the $@Binary$ annotation that **must** define the associated binary file name through the *binary* field (see Figure 5.1). Moreover, this annotation can optionally define the *workingDir* field (to set up the working directory of the binary execution) and, as any *Method* annotation does, the *priority* and *constraints* fields (see Figure 5.2).

```
@Binary(binary = "binary")
void binaryTask();
```

FIGURE 5.1: Binary annotation using only mandatory fields

```
@Binary(binary = "binary",
        workingDir = "/tmp/",
        priority = true,
        constraints = @Constraints(computingUnits = "1"))
void binaryTask();
```

FIGURE 5.2: Binary annotation using all available fields

In the MPI case, we have created the $@MPI$ annotation that **must** define the MPI command to be run (also known as MPI Runner) through the *mpiRunner* field, the associated binary file name through the *binary* field, and the number of computing nodes to be reserved for the MPI execution through the *computingNodes* field (see Figure 5.3). Moreover, this annotation can optionally define the *workingDir* field (to set up the working directory of the binary execution) and, as any *Method* annotation does, the *priority* and *constraints* fields. Figure 5.4 shows a complete example of this annotation. Notice that the total number of requested Computing Units is $2\ nodes \cdot 4\ \frac{CUs}{node} = 8\ total\ CUs$.

```
@MPI(mpiRunner = "mpirun",
     binary = "mpiBinary",
     computingNodes = "2")
void mpiTask();
```

FIGURE 5.3: MPI annotation using only mandatory fields

```
@MPI(mpiRunner = "mpirun",
     binary = "mpiBinary",
     computingNodes = "2",
     workingDir = "/tmp/",
     priority = true,
     constraints = @Constraints(computingUnits = "4"))
void mpiTask();
```

FIGURE 5.4: MPI annotation using all available fields

In the OmpSs case, we have created the *@OmpSs* annotation that **must** define the associated binary through the *binary* field (see Figure 5.5). Moreover, this annotation can optionally define the *workingDir* field (to set up the working directory of the binary execution) and, as any *Method* annotation does, the *priority* and *constraints* fields (see Figure 5.6).

```
@OmpSs(binary = "ompssBinary")
void ompssTask();
```

FIGURE 5.5: OmpSs annotation using only mandatory fields

```
@OmpSs(binary = "ompssBinary",
       priority = true,
       workingDir = "/tmp/",
       constraints = @Constraints(computingUnits = "4"))
void ompssTask();
```

FIGURE 5.6: OmpSs annotation using all available fields

Only considering the modifications listed in this section, the current COMPSs annotations (for *Method* and *Service* tasks) were left intact. However, for the sake of clarity and to enhance the COMPSs capabilities, we have decided to modify the existant annotations to include environment variables as annotations' values, a clearer versioning annotation and new annotations for the upcoming schedulers.

5.1.2 Environment variables as annotations

A large variety of applications use different constraint values depending on the execution even if the code of the application remains the same. This is mostly because users want to adapt the task needs to the different data sizes and computational load. However, the current COMPSs version only allowed users to define different constraints for tasks by recompiling all the application with a different interface. Figure 5.7 shows the Interface of a clear example of this case: a Wordcount application that always defines two tasks (*wordCount* and *mergeResults*) with the same code. The execution load of the *wordCount* task is highly related to the size of the data to be treated. Thus, users want to specify a different memory requirement depending on the input data size (because the file must be totally loaded in memory). However, to change the value of the *memorySize* constraint, they must recompile the whole application.

```
public interface WordcountItf {
    @Method(declaringClass = "WordcountImpl")
    public HashMap<String, Integer> mergeResults(
        @Parameter HashMap<String, Integer> m1,
        @Parameter HashMap<String, Integer> m2
    );
    @Method(declaringClass = "WordcountImpl")
    @Constraints(memorySize = "4.0")
    public HashMap<String, Integer> wordCount(
        @Parameter(type = Type.FILE, direction = Direction.IN) String filePath
    );
}
```

FIGURE 5.7: Wordcount Interface

Figure 5.8 shows how a user must run two different executions of this application, one with a big file (for instance, needing 16 Gb of memory) and one with a small file (needing 1 Gb of memory).

```
# Modify the application Interface
$ vi WordcountItf.java # Change the constraint value to 1.0
# Clean and build the application
$ mvn clean package
# Run the application with COMPSs
$ runcompss wordcount.Wordcount /path/to/small/file
# Modify the application Interface
$ vi WordcountItf.java # Change the constraint value from 1.0 to 16.0
# Clean and build the application
$ mvn clean package
# Run the application with COMPSs
$ runcompss wordcount.Wordcount /path/to/big/file
```

FIGURE 5.8: Wordcount executions with different constraint values

In our new approach, the annotations' values can be specified through environment variables that are resolved on the COMPSs Master node in execution time. Following the previous example, Figure 5.9 shows the same application using an environment variable to define the memory size annotation. Figure 5.10 illustrates how the user can easily execute the application several times with different input data files with only redefining the environment variable (we must highlight, that the users no longer need to recompile the whole application between executions).

```
public interface WordcountItf {
  @Method(declaringClass = "WordcountImpl")
  public HashMap<String, Integer> mergeResults(
    @Parameter HashMap<String, Integer> m1,
    @Parameter HashMap<String, Integer> m2
  );
  @Method(declaringClass = "WordcountImpl")
  @Constraints(memorySize = "${MEMORY_SIZE}")
  public HashMap<String, Integer> wordCount(
    @Parameter(type = Type.FILE, direction = Direction.IN) String filePath
  );
}
```

FIGURE 5.9: Wordcount Interface with environment variables

```
# Clean and build the application
$ mvn clean package
# Modify the environment variable to be loaded by the Interface
export MEMORY_SIZE=1.0
# Run the application with COMPSs
$ runcompss wordcount.Wordcount /path/to/small/file
# Modify the environment variable to be loaded by the Interface
export MEMORY_SIZE=16.0
# Run the application with COMPSs
$ runcompss wordcount.Wordcount /path/to/big/file
```

FIGURE 5.10: Wordcount executions with environment variables as constraints

This behavior can be used with any already existing annotation. Next, Figure 5.11 provides a complete example of all the variables that can be resolved at execution time from environment variables (environment variables are detected by the use of the $ at the beginning of the field value). Notice that the only values that we are preventing the users from defining with environment variables are the *declaringClass* field inside the *Method* annotation and the parameter annotations.

```
public class MainItf {

    @Method(declaringClass = "MainImpl",
            isModifier = "${isModifier}",
            priority = "${isPrioritary}")
    @Binary(binary = "${binary},
            workingDir = "${wd}",
            priority = "${isPrioritary}")
    @MPI(mpiRunner = "${mpiRunner},
         binary = "${mpiBinary}",
         computingNodes = ${computingNodes},
         workingDir = "${wd}",
         priority = "${isPrioritary}")
    @OmpSs(binary = "${ompssBinary},
           workingDir = "${wd}",
           priority = "${isPrioritary}")
    @Constraints(computingUnits = "${computingUnits}",
                 processorName = "${processorName}",
                 processorSpeed = "${processorSpeed}",
                 processorArchitecture = "${processorArchitecture}",
                 processorPropertyName = "${processorPropertyName}",
                 processorPropertyValue = "${processorPropertyValue}",
                 memorySize = "${memorySize}",
                 memoryType = "${memoryType}",
                 storageSize = "${storageSize}",
                 storageType = "${storageType}",
                 operatingSystemType = "${operatingSystemType}",
                 operatingSystemDistribution = "${operatingSystemDistribution}",
                 operatingSystemVersion = "${operatingSystemVersion}",
                 appSoftware = "${appSoftware}",
                 hostQueues = "${hostQueues}",
                 wallClockLimit = "${wallClockLimit}")
    @SchedulerHints(isDistributed = "${isDistributed}",
                    isReplicated = "${isReplicated}")
    int task(
        @Parameter(type = Type.STRING, direction = Direction.IN) String message
    );
}
```

FIGURE 5.11: Example with an Interface with all the available environment variables

We must highlight that this modification seems to have low sides effects, but it is rather the other way around. For the end-users, these modifications imply that all the annotations' values are of type *string* rather than a specific type for each of them. For the COMPSs Runtime, it means that all the annotation values must be resolved and transformed to its real value (integer, float, string, etc.). Notice that while this transformation is simple when the final value is an integer or a float (because the constraint value can only be an integer/float value or an environment variable), it is not that simple when the final value is a string (because the constraint value can have zero, one or more than one environment variables). For instance, Figure 5.12 shows many different possibilities that the user can specify when providing the *workingDir* constraint.

```
    workingDir = "${wd}"
    workingDir = "$wd"
    workingDir = "/path/${wd}"
    workingDir = "${wd}/path"
    workingDir = "/path/$wd/path"
    workingDir = "/path/$wd1/path/${wd2}/path"
```

FIGURE 5.12: Example of complex environment variables on the *workingDir* field

Finally, when using environment variables as constraints, the Runtime will only raise an exception if the environment variable is not defined (*null*), empty or it cannot be parsed to its real type.

5.1.3 Versioning task annotation

Versioning is a mechanism that COMPSs provides for defining several implementations of the same task and deciding, at execution time, which is the best implementation to run.

In the previous COMPSs version, the user defines a task in the interface and defines more than one *declaringClass* field for the different implementations. Figures 5.13, 5.14 and 5.15 show how a task *sayHello* is called by the main code and two different implementations of this task with the same name but in separated files. Notice that the callee in the main code refers to the *Impl1* but in execution time the COMPSs Runtime will choose to execute any of the implementations (either *Impl1* or *Impl2*). That means that using the callee *Impl1.sayHello()* or *Impl2.sayHello()* will make no difference at execution time.

```
public class Hello {

    public static void main(String[] args) {
        System.out.println("Hello World (from main code)");
        // Launch task
        // We could use Impl2.sayHello() indistinctly
        Impl1.sayHello();
    }
}
```

FIGURE 5.13: Example of previous versionning main code: Hello.java

```
public class Impl1 {
    public static void sayHello() {
        System.out.println("Hello World (from Implementation 1)");
    }
}
```

FIGURE 5.14: Example of previous versionning Impl1.java

```
public class Impl2 {
    public static void sayHello() {
        System.out.println("Hello World (from Implementation 2)");
    }
}
```

FIGURE 5.15: Example of previous versionning Impl2.java

Following with the example, Figure 5.16 defines its interface. The task *sayHello* is annotated with two different implementations declared in classes *Impl1* and *Impl2* respectively.

```
public interface HelloItf {

    @Method(declaringClass = {"Impl1", "Impl2"})
    void sayHello();
}
```

FIGURE 5.16: Example of the Interface of previous versionning: HelloItf.java

The user has also available the *MultiConstraints* annotation to add the needed requirements for each implementation. The programming model supports a global constraint annotation for the common constraints of all the implementations and a *MultiConstraint* annotationfor the specific constraints of each implementation. Notice that the number of entries within the *MultiConstraint* annotation must be the same as the number of entries within the *declaringClass* field. Figure 5.17 redefines the Interface of the previous example to add a global constraint of 1 computing unit to both implementations and a specific memory constraint for each implementation. Obviously, this example can be extended to any constraint field.

```
public interface HelloItf {

  @Method(declaringClass = {"Impl1", "Impl2"})
  @Constraints(computingUnits = 1)
  @MultiConstraints({
    @Constraints(memorySize = 1),
    @Constraints(memorySize = 2)})
  void sayHello();
}
```

FIGURE 5.17: Example of the Interface of previous versionning with constraints: HelloItf.java

Extending this model to support non-native tasks is a problem since the different implementations are enclosed within the *Method* annotation (and the new *Binary*, *MPI* and *OmpSs* annotations are defined outside the *Method* annotation). Moreover, to check that the number of entries of the *MultiConstraints* annotation is the same than the number of declared implementations becomes expensive. Hence, our new implementation only allows to declare one implementation per method annotation, moves the *Constraints* annotation inside each task annotation to define the specific implementation requirements and allows the user to define one single *Constraint* clause to define the common requirements. Figure 5.18 shows how the previous example will be specified with the new COMPSs Annotations.

```
public interface NewHelloItf {

  @Method(declaringClass = "Impl1",
          constraints = @Constraints(memorySize = 1))
  @Method(declaringClass = "Impl2",
          constraints = @Constraints(memorySize = 2))
  @Constraints(computingUnits = 1)
  void sayHello();
}
```

FIGURE 5.18: Example of the new Annotation Interface: NewHelloItf.java

Notice that this new annotation syntax allows us to add the non-native tasks as different implementations easily. For example, Figure 5.19 adds a new binary implementation to the previous example (now, the *sayHello* method has three different implementations).

```
public interface NewHelloItf {

  @Method(declaringClass = "Impl1",
          constraints = @Constraints(memorySize = 1))
  @Method(declaringClass = "Impl2",
          constraints = @Constraints(memorySize = 2))
  @Binary(binary = "${BINARY}")
  @Constraints(computingUnits = 1)
  void sayHello();
}
```

FIGURE 5.19: Extended example of the new Annotation Interface: NewHelloItf.java

5.1.4 SchedulerHints task annotation

Since the previous modifications were already making the new COMPSs Annotations not backward compatible, we have profited to add a specific annotation for the user to define hints for the COMPSs Scheduler. In the current state, the new *SchedulerHints* annotation only supports the two fields shown in Table 5.1

Field	Type	Description
isDistributed	boolean	Forces the scheduler to schedule the instances of the task in a round-robin manner between nodes
isReplicated	boolean	Replicates the task execution in all the available nodes

TABLE 5.1: Available fields for the *SchedulerHints* annotation

We are conscient that these annotations may break, somehow, the programming model because users must be unaware of the underlying infrastructure and because the *isReplicated* field makes the parallel behavior different to the sequential behavior. However, these options are very useful for advanced users (for example, in cases where an initialization task must be executed in all the computational nodes) and may be, in the near future, very useful for scheduler enhancements (such as annotating map-reduce tasks).

Figure 5.20 illustrates how to use these annotations in the interface.

```
public interface MainItf {

    @Method(declaringClass = "MainImpl")
    @SchedulerHints(isReplicated = true)
    void replicatedTask();

    @Method(declaringClass = "MainImpl")
    @SchedulerHints(isDistributed = true)
    void distributedTask();
}
```

FIGURE 5.20: Example of an Interface with SchedulerHints: MainItf.java

5.1.5 New stream parameter annotation

Once the task annotations for non-native tasks was defined, we had to implement a way to communicate the Java application with the non-native tasks' execution. When executing standalone binaries, MPI processes or OmpSs processes the exit value of the processes is used as the return value. Thus, we have decided that the COMPSs non-native tasks must use the exit value of their internal binary as the return value of the task. In this sense, we have allowed the users to capture this value by defining the return type of the non-native task as an *int* (for implicit synchronization), as an *Integer* (for post-access synchronization) or to forget it (declaring the function as *void*). Figure 5.21 shows an Interface example of the three return types.

```
public interface MainItf {

    @Binary(binary = "${BINARY}")
    int binaryTask1();

    @Binary(binary = "${BINARY}")
    Integer binaryTask2();

    @Binary(binary = "${BINARY}")
    void binaryTask3();
}
```

FIGURE 5.21: Example of the different return types of the non-native tasks

However, the users not only need the process exit value to work with this kind of applications but need to set the Standard Input (*stdIn*) and capture the Standard Output (*stdOut*) and Error (*stdErr*). For this purpose, we have created a new parameter annotation, *stream*, that allows the users to set some parameters as I/O streams for the non-native tasks. Stream parameters are not passed directly to the binary command but rather they are set as *stdIn*, *stdOut* or *stdErr* of the binary process. Since this kind of redirection is restricted to files in *LINUX* Operating Systems, we have decided to keep the same restrictions to the annotation. Consequently, **all *stream* parameters must be files**.

Figure 5.22 shows the Interface of an application with two tasks that have a normal parameter (the first one, that will be sent directly to the binary execution), a file parameter to be used as *stdIn* of the process, a file parameter to be used as *stdOut* and a last file parameter to be used as *stdErr*. The difference between *task1* and *task2* in this example is that the first

task will overwrite the *fileOut* and *fileErr* content (since the files are opened in *write* mode), and the second task will append the *fileOut* and *fileErr* content at the end of the file (since the files are opened in *append* mode).

```
public interface StreamItf {

    @Binary(binary = "${BINARY}")
    Integer task1(
        @Parameter(type = Type.STRING, direction = Direction.IN) String normalParameter,
        @Parameter(type = Type.FILE, direction = Direction.IN, stream = Stream.STDIN)
    String fileIn,
        @Parameter(type = Type.FILE, direction = Direction.OUT, stream = Stream.STDOUT)
    String fileOut,
        @Parameter(type = Type.FILE, direction = Direction.OUT, stream = Stream.STDERR)
    String fileErr
    );

    @Binary(binary = "${BINARY}")
    Integer task2(
        @Parameter(type = Type.STRING, direction = Direction.IN) String normalParameter,
        @Parameter(type = Type.FILE, direction = Direction.IN, stream = Stream.STDIN)
    String fileIn,
        @Parameter(type = Type.FILE, direction = Direction.INOUT, stream = Stream.STDOUT)
    String fileOut,
        @Parameter(type = Type.FILE, direction = Direction.INOUT, stream = Stream.STDERR)
    String fileErr
    );
}
```

FIGURE 5.22: Example of the different stream annotations for non-native tasks

To summarize the last information retrieved from this example, Table 5.2 show the available modes for each stream type.

Type	Stream	Direction	Description
FILE	Stream.STDIN	Direction.IN	Sets the process *stdIn*. The file is opened in *read* mode
FILE	Stream.STDOUT	Direction.OUT	Sets the process *stdOut*. The file is opened in *write* mode
FILE	Stream.STDOUT	Direction.INOUT	Sets the process *stdOut*. The file is opened in *append* mode
FILE	Stream.STDERR	Direction.OUT	Sets the process *stdErr*. The file is opened in *write* mode
FILE	Stream.STDERR	Direction.INOUT	Sets the process *stdErr*. The file is opened in *append* mode

TABLE 5.2: Available stream types with their valid directions and execution behaviour

5.1.6 New prefix parameter annotation

Before going in-depth in this section, we must explain that COMPSs builds the task data dependencies graph taking into account the parameters annotated in the application Interface. Analyzing several binaries, we have found out that a non-negligible part of them use prefixes for each parameter. The prefixes used by binaries can be divided into two types:

- **Separated Prefix** A prefix that is written separately before the parameter value. This type of prefixes are of the form:

  $./binary\ -param1\ value\ --param2\ value\ -k\ value$

  In fact, there is not a strong need that the parameter prefix starts with a dash but its the common behavior for Linux binaries.

- **Joint Prefix** A prefix that is written with the parameter value without beeing separated or with a separation character that it is not an empty space. This types of prefix vary a lot but are of the form:

  $./binary\ -pValue\ -q=Value\ --r=Value\ s=Value$

The separated prefixes do not represent a problem for the COMPSs programming model since they can be defined as a standalone string parameter that is finally passed to the binary. However, the joint prefixes do represent a problem for COMPSs since the users must prepend the prefix to the parameter, breaking the data dependencies between the tasks. For the sake of clarity, consider the two tasks shown in Figure 5.23 and the main code shown in Figure 5.24. Since the second task requires a joint prefix, when calling it from the main code the users must modify its value and prepend the prefix to the *fileName* variable. This string modification causes a synchronization in the appliaction's main code instead of creating a data dependency between the two tasks.

```
public interface ExampleItf {

    // Must execute: ./tmp/bin1 fileName
    @Binary(binary = "/tmp/bin1")
    Integer task1(
        @Parameter(type = Type.FILE, direction = Direction.INOUT) String fileName
    );

    // Must execute: ./tmp/bin2 --file=fileName
    @Binary(binary = "/tmp/bin2")
    Integer task2(
        @Parameter(type = Type.FILE, direction = Direction.INOUT) String fileName
    );
}
```

FIGURE 5.23: Binary Tasks example for joint prefixes

```
public static void main(String[] args) {
    String fileName = "/tmp/file";
    BINARY.task1(fileName);
    BINARY.task2("--file=" + fileName);
}
```

FIGURE 5.24: Main code example for joint prefixes

Consequently, for this second type of prefixes, we have created a new parameter annotation *prefix* that allows the users to define the prefix separately to the parameter value and its prepended to the parameter value just before the binary execution. This modification allows COMPSs to handle the data dependencies between parameters (since prefixes are immutable strings that do not define data dependencies) and allows the binaries to receive the parameter prefixes and its value together as a single parameter.

```
public interface StreamItf {

    @Binary(binary = "binaryExample")
    void task1(
        @Parameter(type = Type.STRING, direction = Direction.IN)    String pPrefix,
        @Parameter(type = Type.FILE,   direction = Direction.IN)    String fileIn,
        @Parameter(type = Type.STRING, direction = Direction.IN)    String qPrefix,
        @Parameter(type = Type.FILE,   direction = Direction.INOUT) String fileInOut,
        @Parameter(type = Type.STRING, direction = Direction.IN)    String kPrefix,
        @Parameter(type = Type.INT,    direction = Direction.IN)    int k
    );

    @Binary(binary = "binaryExample")
    void task2(
        @Parameter(type = Type.FILE,   direction = Direction.IN,    prefix = "-p=")
            String fileIn,
        @Parameter(type = Type.FILE,   direction = Direction.INOUT, prefix = "--q=")
            String fileInOut,
        @Parameter(type = Type.INT,    direction = Direction.IN,    prefix = "k")
            int k
    );

    @Binary(binary = "binaryExample")
    void task3(
        @Parameter(type = Type.STRING, direction = Direction.IN)
            String pPrefix,
        @Parameter(type = Type.FILE,   direction = Direction.IN)
            String fileIn,
        @Parameter(type = Type.FILE,   direction = Direction.INOUT, prefix = "--q=")
            String fileInOut,
        @Parameter(type = Type.INT,    direction = Direction.IN,    prefix = "k")
            int k
    );
}
```

FIGURE 5.25: Interface example of an application with prefixes

Figure 5.25 shows the Interface of an application with three binary tasks, Figure 5.26 shows the main code calls to these tasks and Figure 5.27 shows the final binary command that is executed in the task. Notice that the first task, *task1*, only uses separated prefixes; the second task, *task2*, uses only joint prefixes and the third task, *task3*, is a hybrid example of both separated and joint prefixes.

```
public class BinaryPrefixesExample {

    public static void main(String[] args)
        String file1 = "file1.in"
        String file2 = "file2.inout"
        int kValue = 10;

        // Launch task 1
        task1("-p", file1, "--q", file2, "k", kValue);

        // Launch task 2
        task2(file1, file2, kValue);

        // Launch task 3
        task3("-p", file1, file2, kValue);
    }
}
```

FIGURE 5.26: Example of the main code calls to tasks with prefixes

```
# TASK 1
./binaryExample -p file1.in --q file2.inout k 10

# TASK 2
./binaryExample -p=file1.in --q=file2.inout k10

# TASK 3
./binaryExample -p file1.in --q=file2.inout k10
```

FIGURE 5.27: Example of the command executed inside each task using prefixes

5.2 Scheduling modifications

Before explaining the scheduler modifications, we must define how COMPSs handles the task creation, scheduling and execution. The COMPSs Runtime instruments the application's main code looking for invocations to the methods defined as tasks in the application interface. When these methods are detected, COMPSs creates a task that is submitted to the Task Analyzer component and substitutes the method call by an *executeTask()* call. When the Task Analyzer receives a new task, it computes its data dependencies and submits it to the Task Scheduler. Next, the Task Scheduler creates an Execution Action associated with the task and adds it to the execution queue. Eventually, the Execution Action will be scheduled and launched (this mechanism requires the task to be data-free and to have enough free resources to fulfill the task constraints). When the Execution Action is launched, a Job is created to monitor the task execution. This job includes the transfer of the job definition and all the input data to the target COMPSs Worker, the real task execution in the worker and the transfer of the output data back to the COMPSs Master. Once the job is completed, its data dependent Execution Actions are released (if any), and the job is destroyed (or, depending on the debug level, stored for post-mortem analysis).

5.2.1 Treatment of non-native tasks

Non-native tasks only represent a new way of executing tasks in the COMPSs Worker. Although the data structure that contains its information (*BinaryImplementation*, *MPIImplementation*, *OmpSsImplementation*) is quitely different than the one storing Method / Service tasks (*MethodImplementation* and *ServiceImplementation* respectively), all of them inherit a common super-structure (*Implementation*) that allows the COMPSs Master to treat any type of task in the same way.

Consequently, the Scheduler component is also independent of the task execution and, thus, no modification has been added to enable the execution of non-native tasks. However, we must emphasize that although the execution of MPI tasks itself has not caused any modification (because it also extends from the same interface), the fact of using more than one computational node did.

5.2.2 Multi-node execution actions

That beeing said, the first design choice to enable multi-node task executions has been to associate several execution actions to the same task. This mapping allows the Scheduler to treat the data-dependencies and the resource consumption as it was done before. However, the execution actions associated with the same task must have different behaviors during

the execution phase because only one of the actions must really launch the job.

Consequently, we have extended the *ExecutionAction* in a *MultiNodeExecutionAction* class that is only used when a task requires more than one computing node (otherwise the previous *ExecutionAction* implementation is used). When the task scheduler receives a new multi-node task (*ExecuteTaskRequest*) it creates a new *MultiNodeGroup* instance and *N MultiNodeExecutionAction* instances (beeing N the number of nodes requested by the task). The *MultiNodeGroup* instance is shared among all the actions assigned to the same task execution, and it handles the actions' id within the group. More in-depth, when the *MultiNodeExecutionActions* are created the *MultiNodeGroup* assigns a nullable identifier to all of them. Once the actions are scheduled and launched, the *MultiNodeGroup* assigns a unique valid identifier between 1 and N. This action identifier is used during the action execution to act as an execution slave node (when the assigned identifier is different to 1) or to act as an execution master node (when the assigned identifier is 1). When the *MultiNodeExecutionAction* is identified as a slave, it no longer triggers a job execution, but rather reserves the requested resources and waits for its master action to complete. When the *MultiNodeExecutionAction* is identified as a master, it retrieves all the hostnames of its slave actions (for the MPI command) and behaves as a normal ExecutionAction (launches a job to monitor the input data transfers, the real task execution on the node and the output data transfers).

On the one hand, Figure 5.28 shows an example of the normal process. A task *T1* requiring 1 node (normal task) is submitted to the scheduler through the *ExecuteTaskRequest* request. The request is then processed and an *ExecutionAction* is created as it was done before. Eventually, the action is scheduled, launched and finally executed, creating a new job that will monitor the task execution in the target node.

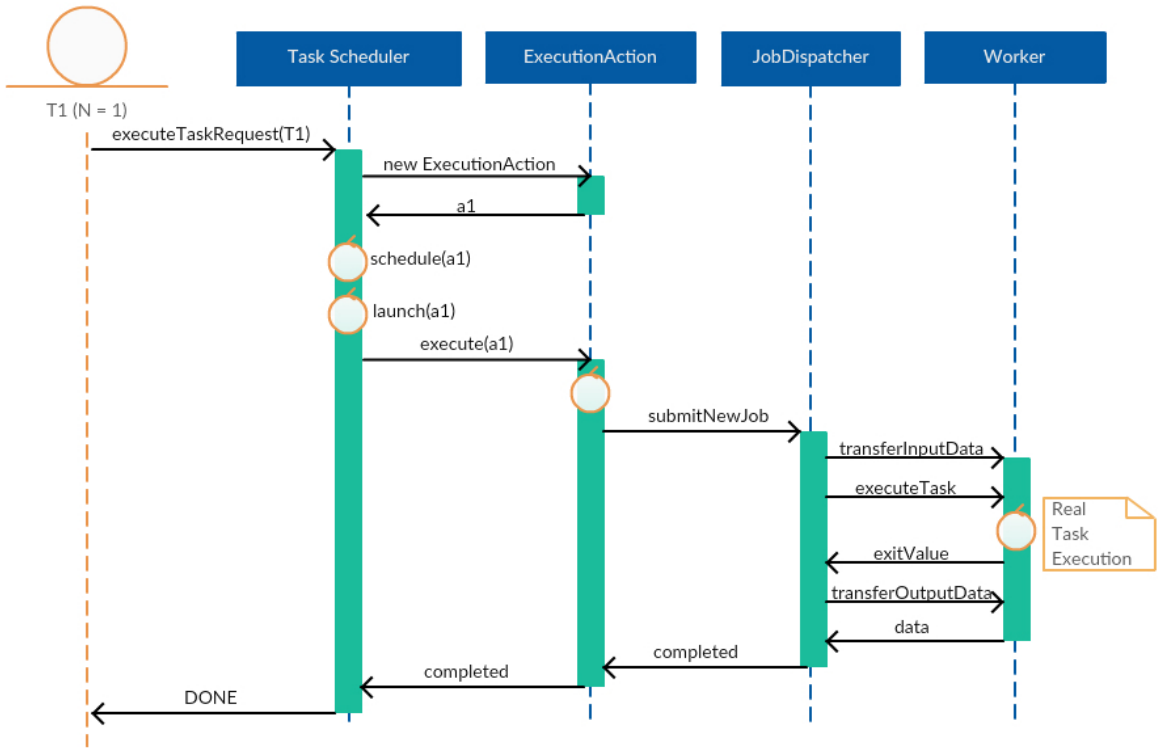

FIGURE 5.28: Example of a single node task flow

On the other hand, Figure 5.29 shows an example of the Multi-Node process. A task *T2* requiring 3 nodes (multi-node task) is submited to the scheduler through the same *ExecutionTaskRequest* request. The request is then processed: a new action group (lets say *g1*) is created (a new instance of the *MultiNodeGroup*) and 3 *MultiNodeAction* instances (lets say *a1*, *a2* and *a3*) are created. The action group *g1* is shared among all the three actions and assigns a nullable action identifier to all of them.

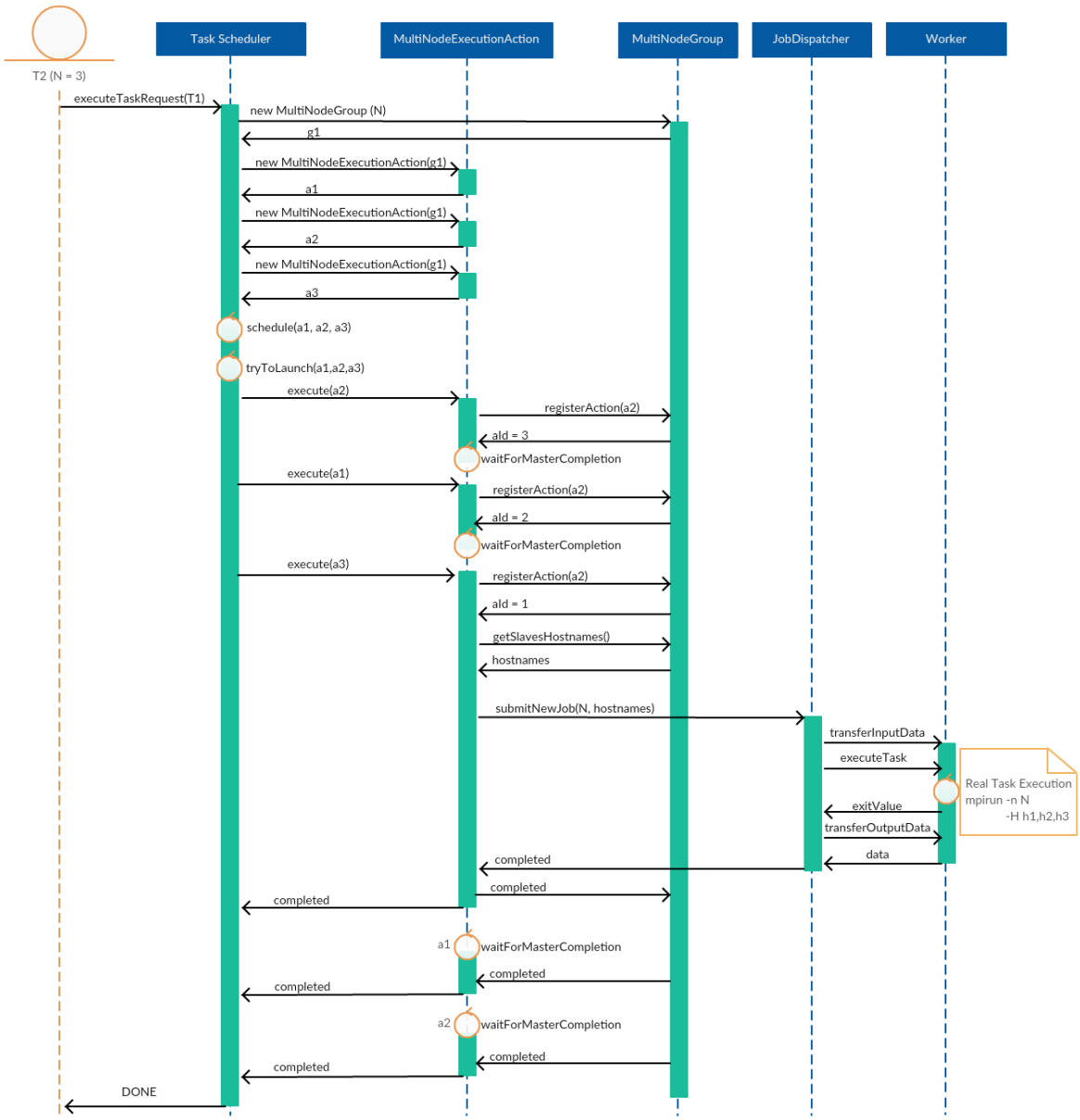

FIGURE 5.29: Example of a multi-node task flow

Eventually, *a2* is scheduled, launched and finally executed. On the execution phase, the action asks for an action identifier and the action group *g1* assigns it an $actionId = 3$ (because the group size is 3 and no action has previously requested an identifier). Since the action identifier classifies *a2* as a slave action, the execute phase only reserves the task constraints and waits for the master action completion.

Eventually, *a1* is also scheduled, launched and finally executed. Following the same process than the previous action, *a1* is granted with $actionId = 2$ (because the group size is 3 and

only one action has previously requested an identifier). Since the action *a1* is also classified as a slave, it reserves the task constraints and waits for the master action completion.

Finally, *a3* will also be scheduled, launched and finally executed. In this case, the action group assigns it an $actionId = 1$. Since it is the last action, it is now identified as master and during its execution phase it retrieves the hostnames of the resources assigned to all the actions inside the *g1* group (lets say, *h1* for *a1* and *h2* for *a2*) and launches the execution job. The job will be then executed (lets say that the host assigned to this action *a3* is *h3*) monitoring the input data transfers, performing the real task execution (for example, calling the MPI command inside the host *h3* with 3 nodes *h1*, *h2* and *h3*) and retrieving back the output data from *h3*.

Once the job is completed, the action *a3* is marked as completed (freeing all the reserved resources) and, then, it triggers its completion to all the slave actions registered in the group *g1*. Consequently, *a1* and *a2* are also marked as completed (and its resources are also freed). When all the actions within the group are marked as completed, the task is registered as *DONE* and follows the usual process: frees its data dependent tasks and it is stored for post-mortem analysis.

5.2.3 Treatment of SchedulerHints

Currently there are only two *SchedulerHints* available in COMPSs annotation: *isDistributed* and *isReplicated*. Both annotations are attached to the task definition and are treated when the *ExecuteTaskRequest* request is served.

When the *isDistributed* annotation is enabled, the request checks how many tasks of the same type have been already scheduled to each available worker. Then, it chooses the worker that has executed **less** tasks of the same type and forces the Execution Action to be scheduled to the selected worker. Notice that the computational cost of distributing a task in a Round Robin manner among the available workers is proportional to the number of available workers. To maintain consistency, when a resource is chosen as the target worker of an Execution Action during the *schedule* phase, a task counter in the target worker is increased. In this sense, notice that the memory cost is increased by Equation 5.1 since each available worker stores a list of counters of size equal to the number of different tasks registered in the Application Interface.

$$numWorkers \cdot numTypesTasks \cdot sizeof(int) \tag{5.1}$$

When the *isReplicated* annotation is enabled, the request creates one Execution Action per available worker and forces the actions to be executed in the selected workers. The task is then considered as *DONE* when all its Execution Actions are marked as *completed*. For this purpose, the task stores an execution counter initialized to the number of available workers that only releases the task (and its data dependent actions) when reaching zero.

As a final note, we highlight that the Scheduling Hints are evaluated during the scheduling phase. That means that the workers considered in both cases must be functional when the task scheduling is beeing treated (not when the task is really executed).

5.3 Worker enhancements

The Communication layer abstracts the Master node from the specific Communication Adaptors and thus, from the underlying infrastructure. However, the worker processes spawned by this layer are dependent on each Adaptor implementation. Currently, COMPSs supports the *NIO* and the *GAT* Communication Adaptors.

On the one hand, *GAT* Adaptor is built on top on the Java Grid Application Toolkit (JavaGAT) [48] which relies on the SSH connection between nodes. During the application execution, the Runtime spawns a new worker process per task execution. More specifically, when a task must be executed, the GAT Communication Adaptor creates a GAT Job, sends the job and the required data through SSH to the worker's resource, starts the worker process, executes the task itself, closes the worker process, and retrieves the job status, the job's log files, and the required output data. The *worker.sh* script orchestrates all the processes and launches a language dependant script for the real task execution (*GATWorker.java* for Java, *worker.py* for Python and *Worker* for C/C++). Although the implementation suffers from some performance overheads (because the overhead of spawning a new process on each task execution becomes non-negligible for small duration tasks), it provides a high connectivity interface since it only requires the SSH port to be opened.

On the other hand, *NIO* Adaptor is a more sophisticated implementation based on Java New I/O (NIO) library [41]. This adaptor spawns a persistent Java Worker Process per resource, rather than one per task execution, and the communication between Master and Workers is then made through Sockets. Hence, this Adaptor provides better performance than the *GAT* Adaptor but requires extra open ports between the available resources. Furthermore, the Worker processes persist during the full execution of the application, what also lets us have an object cache per worker, data communications between workers (rather than handling all the data in the Master resource) and thread binding mechanisms to map threads to specific cores of the machine. Finally, for the task execution, each worker has several *Executor* threads that can execute natively Java applications, or Python and C/C++ applications using a *ProcessBuilder*.

5.3.1 Invokers

To enable the execution of non-native tasks for any Communication Adaptor we have implemented a *GenericInvoker* class that provides an API for executing standard, MPI, and OmpSs binaries. This API is built on top of a *BinaryRunner* class that spawns, runs and monitors the execution of any binary command.

More specifically, the *BinaryRunner* class has two methods. Firstly, *createCMDParametersFromValues* serializes the received parameters to construct the binary arguments. This method is also in charge of processing the *Stream* annotations and redirecting the StdIn, StdOut, and StdErr when required. Secondly, *executeCMD* executes the received binary command (with all its parameters), monitors its execution and, finally, returns the exit value of the process.

On the other hand, the *GenericInvoker* class provide three functions: *invokeBinaryMethod*, *invokeMPIMethod*, and *invokeOmpSsMethod* to invoke respectively standard, MPI and OmpSs binaries. The three methods receive the binary path and the argument values, construct and execute the command by calling the *BinaryRunner* functions and return the exit value of the binary execution.

In a second step we have adapted each of the Communication Adaptor (*GAT* and *NIO*) to call this *GenericInvoker* when needed. In both cases, we have substituted the normal task execution by a *switch-case* that selects the required invoker considering the task's implementation type.

5.3.2 External executors enhancement

Currently, the PyCOMPSs binding is gaining relevance because Data Analytic Workflows can be easily designed in Python and thus, translated to PyCOMPSs. During the development of this project, we have found out that the COMPSs Workers loose performance because of the creation of the *ProcessBuilder* used to execute Python, C, and C++ tasks.

To solve this problem, we have completely redesigned the way the COMPSs Worker launches a Python, C or C++ task. First of all, during the Worker initialization, we use the *ProcessBuilder* to launch a single BASH script that creates N input pipes, N output pipes and N processes (beeing N the number of available *Executors* at the given Worker). These BASH processes persist until the Worker is stopped and use the pipes to communicate with the Java *Executors*. In fact, each Java *Executor* stores a pair of pipes so that the *Executors* are mapped one to one (see Figure 5.30).

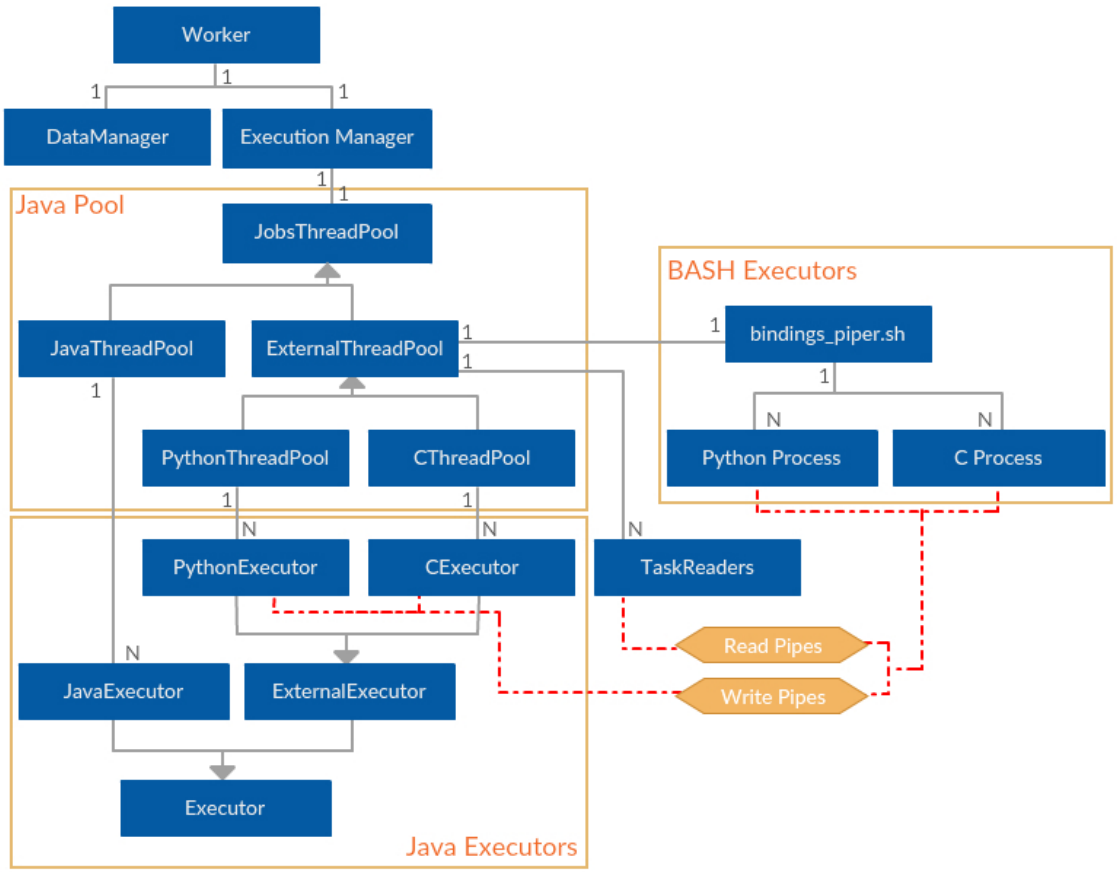

FIGURE 5.30: New structure of the COMPSs Worker Executors

When a task arrives at the Worker, it is processed by a Java *Executor* thread. Assuming that the task is a Python, C or C++ task, the Java *Executor* sends a command through its assigned input pipe to the BASH *Executor* with the task definition and waits for a completion message on the assigned output pipe. The BASH *Executor* then receives the task command, executes the task and sends its result back to the Java *Executor* through the output pipe. Notice that both executor threads are never active at the same time since, when one Java is processing the task the BASH process is listening, and when the BASH process is executing the task the Java process is listening.

At the end of the application execution, when the Worker is stopped, all the Java *Executors* send a *QUIT* command through the input pipe to kill its assigned BASH processes and then exit. To double check the shutdown process, the initial BASH script (spawned with the *ProcessBuilder* at the Worker creation) is killed with a bash *TRAP* to kill any remaining pipe or process.

Finally, Figure 5.31 shows the execution time in the y-axis versus the number of *Executors* (*N*) in the x-axis for an implementation with *ProcessBuilders* (blue) and an implementation with pipes (red for the total time including the spawn and the destruction of the first BASH script, and yellow for the task execution time). We must highlight that the implementation using pipes speeds-up significantly the pre and post actions that must be done in every task execution, getting us to a lower overhead when using the COMPSs Bindings.

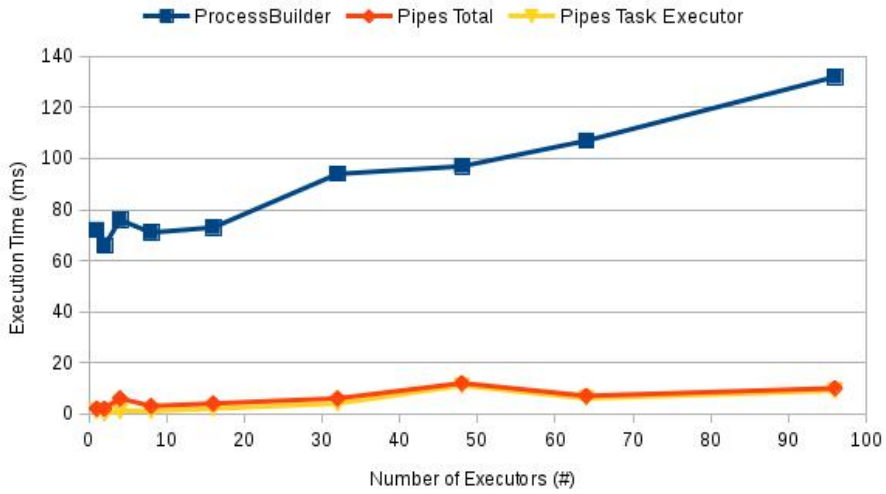

FIGURE 5.31: Execution time versus number of *ProcessBuilders* or Pipes

Chapter 6

Results and evaluation

6.1 Proofs of concept

6.1.1 BLAST

6.1.1.1 Application description

Basic Local Alignment Search Tool (BLAST) [6] is an algorithm to find regions of similarity between primary biological sequences. The program compares a nucleotide or protein (known as *query*) to a sequences' database and identifies sequences that resemble the *query* sequence above a certain threshold.

The COMPSs implementation of BLAST splits the *query* sequence on smaller *fragments*, comparing each fragment against the database and merging up the obtained results. Using different execution arguments, the users can select the *query* sequence, the number of fragments and the target database (Figure 6.1 provides the complete list of execution arguments).

```
blast.BLAST <debug> <database> <query> <nFrags> <tmpDir> <outputFile> <blastBinaryArgs>
    - debug            Enables and disables the debug mode
    - database         The sequences' database path to compare with
    - query            The path where the query sequence is stored
    - nFrags           The number of fragments used to divide the sequence (N)
    - tmpDir           A temporal directory for intermediate results
    - outputFile       The final output file with the sequence matches
    - blastBinaryArgs  The extra arguments to pass to the BLAST binary
```

FIGURE 6.1: Execution arguments of the COMPSs BLAST application

Regarding the code, it is divided in three main blocks:

- **Split:** Splits the *query* sequence in **N** fragments
- **Alignment:** Compares each fragment of the *query* sequence against the database invoking the BLAST binary
- **Assembly:** Merges all the intermediate files into a single file to produce the final result

The BLAST interface provides three types of tasks, one per block. Firstly, the input file is splited by using **N** *splitPartitions* tasks (beeing N the number of fragments). Next, each fragment is processed using the *align* task. Finally, the partial results are merged by using **N-1** *assemblyPartitions* tasks. Figure 6.2 shows an example with **N = 8**.

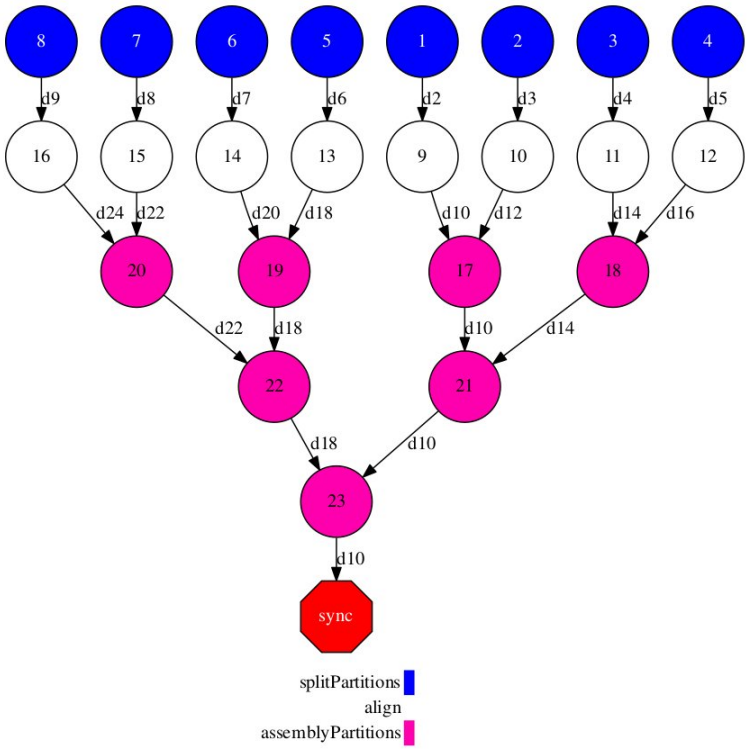

FIGURE 6.2: Example of BLAST execution with N = 8

6.1.1.2 Purpose

This proof of concept must demonstrate that our implementation in COMPSs eases the execution of binary files without degrading its performance. Moreover, the users must have enough flexibility to execute the binaries with different parameters, to synchronize (or not) the exit value, and to capture (or not) the *StdOut* and *StdErr*.

6.1.1.3 Evaluation

When porting the BLAST implementation with COMPSs to the new *Binary* annotation, we have left intact the execution parameters and the application behavior (blocks and number of tasks spawned per block). The only modifications that we have introduced are:

- ***align* task implementation:** A new dummy implementation of the *align* task is defined in the *binary.BINARY* file (see Figure 6.3). This implementation replaces the old one defined in the *BLASTImpl* file (see Figure 6.4), which is totally removed.

```
public static Integer align(String pFlag, String pMode, String dFlag, String database,
            String iFlag, String partitionFile, String oFlag,
            String partitionOutput, String extraCMDArgs) {

    return -1;
}
```

FIGURE 6.3: COMPSs BLAST application: new *align* task implementation

```
public static void align(String databasePath, String partitionFile, String partitionOutput,
        String blastBinary, String extraCMDArgs) throws BlastException {

    Long startAlignment = System.currentTimeMillis();
    String cmd = blastBinary
            + " -p blastx"
            + " -d " + databasePath
            + " -i " + partitionFile
            + " -o " + partitionOutput
            + " " + extraCMDArgs;
    Process p = Runtime.getRuntime().exec(cmd);
    int exitValue = p.waitFor();
    Long alignmentTime = (System.currentTimeMillis() - startAlignment) / 1_000;
    System.out.println("Alignment time " + alignmentTime + " s");

    if (exitValue != 0) {
        try (BufferedInputStream bisErr = new BufferedInputStream(p.getErrorStream());
             BufferedOutputStream bosErr = new BufferedOutputStream(
                     new FileOutputStream(partitionFile + ".err"));) {
            byte[] b = new byte[1_024];
            int read;
            while ((read = bisErr.read(b)) >= 0) {
                bosErr.write(b, 0, read);
            }
            bisErr.close();
            bosErr.close();
        } catch (IOException ioe) {
            String msg = "ERROR: Cannot retrieve CMD error content";
            System.err.println(msg);
            throw new BlastException(msg, ioe);
        }

        String msg = "Error executing Blast job, exit value is: " + exitValue;
        System.err.println(msg);
        throw new BlastException(msg);
    }
}
```

FIGURE 6.4: COMPSs BLAST application: old *align* task implementation

- ***align* task call:** Adapt the main code to the new *align* task call. The new task no longer wraps the binary execution, but rather spawns the *blast* binary directly. Thus, we must add the binary arguments explicitly and recover the exit value of the task. Figure 6.5 shows the new task call loop and Figure 6.6 shows the old one.

```
private static void alignSequences() throws BLASTException {
    final String pFlag = "-p";
    final String pMode = "blastx";
    final String dFlag = "-d";
    final String iFlag = "-i";
    final String oFlag = "-o";
    int numAligns = BLAST.partialInputs.size();
    Integer[] exitValues = new Integer[numAligns];
    for (int i = 0; i < numAligns; i++) {
        exitValues[i] = BINARY.align(pFlag, pMode, dFlag, BLAST.databasePath, iFlag,
            BLAST.partInputs.get(i), oFlag, BLAST.partOutputs.get(i), BLAST.cmdArgs);
    }
}
```

FIGURE 6.5: COMPSs BLAST application: new *align* task call

```
private static void alignSequences() throws BLASTException {
    for (int i = 0; i < partialInputs.size(); i++) {
        BLASTImpl.align(databaseName, partialInputs.get(i), partialOutputs.get(i),
                        blastBinary, commandArgs);
    }
}
```

FIGURE 6.6: COMPSs BLAST application: old *align* task call

- ***align* interface' annotation:** We delete the previous *align* method and add a new task with the Binary annotation. Figure 6.7 shows the new annotation and 6.8 shows the previous one.

```
@Binary(binary = "${BLAST_BINARY}")
Integer align(
    @Parameter(type = Type.STRING, direction = Direction.IN) String pFlag,
    @Parameter(type = Type.STRING, direction = Direction.IN) String pMode,
    @Parameter(type = Type.STRING, direction = Direction.IN) String dFlag,
    @Parameter(type = Type.STRING, direction = Direction.IN) String database,
    @Parameter(type = Type.STRING, direction = Direction.IN) String iFlag,
    @Parameter(type = Type.FILE, direction = Direction.IN) String partitionFile,
    @Parameter(type = Type.STRING, direction = Direction.IN) String oFlag,
    @Parameter(type = Type.FILE, direction = Direction.OUT) String partitionOutput,
    @Parameter(type = Type.STRING, direction = Direction.IN) String extraCMDArgs
);
```

FIGURE 6.7: COMPSs BLAST application: new *align*'s interface annotation

```
@Method(declaringClass = "blast.BLASTImpl")
void align(
    @Parameter(type = Type.STRING, direction = Direction.IN) String databasePath,
    @Parameter(type = Type.FILE, direction = Direction.IN) String partitionFile,
    @Parameter(type = Type.FILE, direction = Direction.OUT) String partitionOutput,
    @Parameter(type = Type.STRING, direction = Direction.IN) String blastBinary,
    @Parameter(type = Type.STRING, direction = Direction.IN) String extraCMDArgs
);
```

FIGURE 6.8: COMPSs BLAST application: old *align*'s interface annotation

Although the previous figures highlight the major modifications, Appendix A provides the full code of the BLAST implementation using the new COMPSs annotations. Notice that the annotation and the task call are more complex than the previous version (since there are more parameters), but the task's implementation is totally suppressed. This means, the users no longer need to create a new *ProcessBuilder* from the commands, spawn a process, wait for its completion, retrieve the exit value, and read the process' output and error streams. Thus, we consider that these annotations provide a significant advantage regarding programmability.

Concerning performance, we have measured the execution time of the *align* task using both implementations. On the one hand, for the *old* implementation, we have measured the

total task time inside the Runtime and the time spent on the real binary execution in the task code inside the application. On the other hand, for the *new* implementation, we have measured the total task time and the time spent on the real binary execution instrumenting the Runtime.

The experiments have been run on the *MinoTauro* [11] machine that is a heterogeneous cluster hosted at the Barcelona Supercomputing Center (BSC) that has 61 Bull B505 blades with 2 Intel E5649 (6-Core) processors at 2.53 GHz, 24 GB of Main memory, 250 GB SSD and 2 Infiniband QDR (40 Gbit each), and 39 Bullx R421-E4 servers with 2 Intel Xeon E5-2630 v3 8-core processors at 2.4 GHz, 128 Gb of main memory, 120 Gb SSD and 1 PCIe 3.0 x8 8GT/s. For the experimentation, we have only requested two nodes (one COMPSs Master and one COMPSs Worker) of the first type, and we have only launched one task at a time (1 maximum task per node) to avoid distortion of the task execution time caused by the execution of other processes. Moreover, we have launched 10 BLAST executions with the parameters shown in Table 6.1.

Parameter	Value
debug	disabled
database	swissprot
query	sargasso.fasta (43 Kb)
nFrags	8
tmpDir	/scratch/tmp/
Output file	output.txt
Binary Arguments	None

TABLE 6.1: Execution parameters of the BLAST application

Notice that the total number of executed tasks is 80, since the *query* sequence is divided in 8 fragments (see Equation 6.1).

$$1\,\frac{align\,task}{fragment}\;\cdot\;8\,\frac{fragment}{execution}\;\cdot\;10\,execution\;=\;80\,align\,task \tag{6.1}$$

Table 6.2 shows the minimum, the maximum and the mean execution times of the *align* task when running **10 times** the *BLAST* application under the aforementioned conditions. In terms of absolute values, the difference between the two implementations is negligible for the total task execution time (less than 1%) and for the binary execution time (less than 0.5%). Regarding the overhead, the old implementation adds 16.25 ms on average to the binary execution and the new implementation adds 15.50 ms on average. Hence, as expected, the overhead introduced by the new implementation can be considered the same than the old implementation, and there is no performance degradation when using the new Binary annotation.

	NEW		**OLD**	
	Total task	**Binary**	**Total task**	**Binary**
Minimum	9141	9119	9196	9164
Mean	15224	15208	15027	15010
Maximum	31204	31178	30463	30435

TABLE 6.2: Task times

6.1.2 Matmul

6.1.2.1 Application description

The Matrix Multiplication (*Matmul*) is a common operation in diverse fields such as numerical methods, earth science, industrial simulations, machine learning or bioinformatics. In fact, this operation has become so common that most of the underlying libraries (i.e. Intel Math Kernel Library - MKL [29]) provide an easy-to-use API call to an optimized implementation.

The common parallel approach to the Matrix Multiplication is to divide the matrix in smaller matrixes, called *blocks* (see Figure 6.9). When multiplying two block matrixes (i.e. *A* and *B*), the result matrix (i.e. *C*) is calculated block by block. Notice that the C blocks can be calculated independently and thus, in parallel.

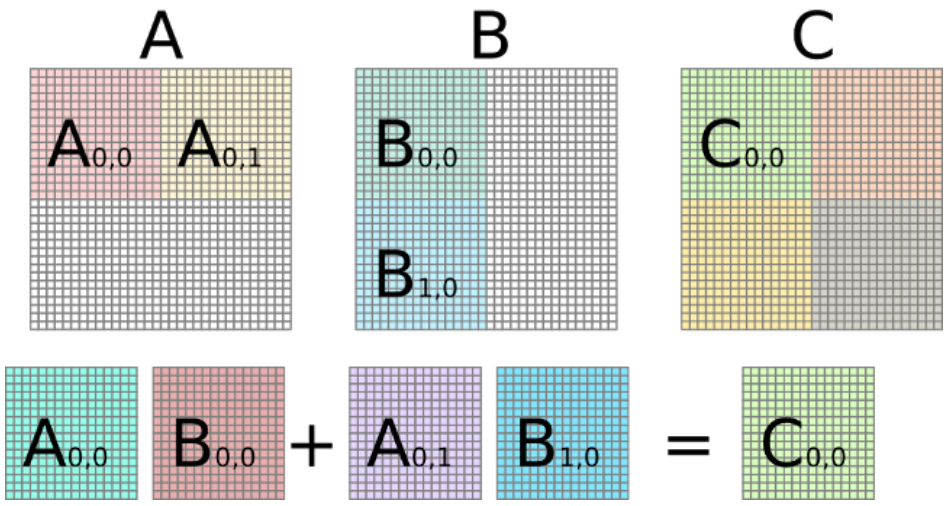

FIGURE 6.9: Multiplication of a Matrix divided in blocks
Source : Dongrui She, GPU Assignment 5KK70

The COMPSs implementation of *Matmul* constructs two float matrixes, *A* and *B*, with *MSIZE* blocks of size *BSIZE*. Both parameters are used passed through the application arguments. Notice that the matrixes are squared, and its real size is stated in Equation 6.2.

$$(MSIZE \cdot BSIZE) \times (MSIZE \cdot BSIZE) \ floats \tag{6.2}$$

Consequently, the result matrix, *C*, is also squared and has the same size.

Regarding the code, it is divided in two main blocks:

- **Initialization:** Initialization of each block of matrixes A and B to random float values using files to store each block.
- **Multiplication Loop:** Multiplies and accumulates the two matrixes block by block using the IKJ-algorithm which has been proven to be the best algorithm in terms of performance [31] without using external libraries.

The initialization of each block is performed inside the *initializeBlock* method, and the multiplication and accumulation is done in the *multiplyAccumulative* method. Both methods are annotated in the *Matmul* Interface so that COMPSs spawn them as tasks. Notice

that since the multiplication and accumulation of each block of C is independent, the implementation spawns $MSIZE \cdot MSIZE$ chains (number of total blocks of C) of depth $MSIZE$ (number of accumulations of a block of C, this is, the length of a row of A and the length of a column of B). Figure 6.10 shows the task execution graph of a *Matmul* of $MSIZE = 2, BSIZE = 128$ where the tasks before the barrier point are the initialization of the matrixes A, B and C, and the task-chains are the previously explained multiplications.

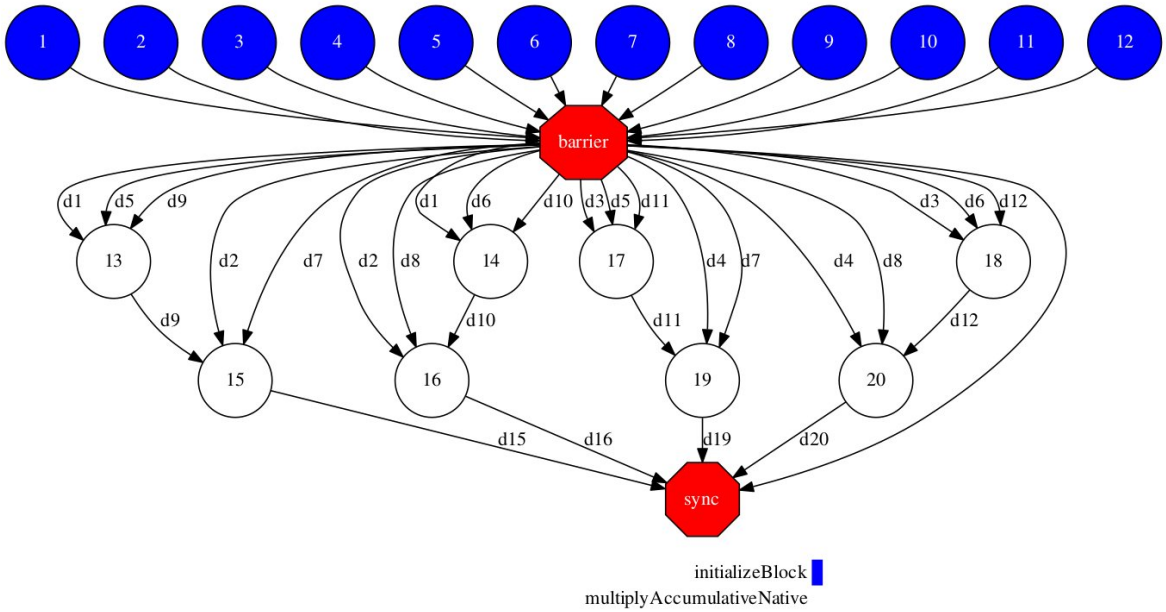

FIGURE 6.10: Task execution graph of a *Matmul* example with MSIZE = 2

6.1.2.2 Purpose

This proof of concept must demonstrate that our enhancement of the COMPSs Runtime eases the execution of MPI tasks by obtaining a hybrid COMPSs-MPI version of the *Matmul*. This new version must be capable of orchestrating several MPI executions in more than one computational node without introducing a significant overhead to the MPI execution. Moreover, the users must have enough flexibility to execute the MPI binaries with different parameters, to synchronize (or not) the exit value, and to capture (or not) the *StdOut* and *StdErr*.

6.1.2.3 Hybrid COMPSs and MPI Matmul implementation

The idea behind building a Hybrid COMPSs and MPI *Matmul* implementation is to divide the matrix into bigger blocks and multiply them in parallel using MPI. To do so, we add an extra MPI layer under COMPSs that multiplies one block of matrixes A and B using P MPI processes, and accumulates the result on one block of C (see Figure 6.11). Since, for this MPI layer, each COMPSs block is treated as a single matrix, the MPI implementation is independent of the COMPSs implementation. In this sense, we have chosen the standard MPI Matmul implementation that divides the A matrix into line-blocks and broadcasts the entire B matrix.

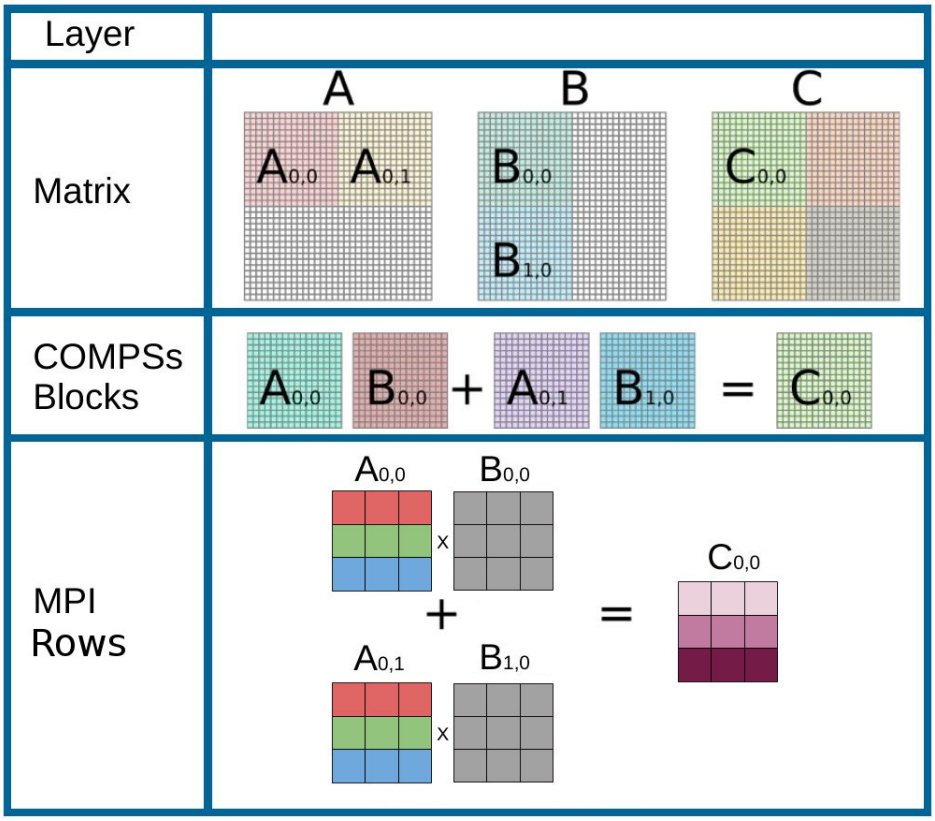

FIGURE 6.11: Hybrid COMPSs and MPI block layers

For this purpose, we have substituted in the previous implementation the *multiplyAccumulative* task by an MPI task that invokes any of the MPI implementation of the *Matmul*. Figure 6.12 shows the main multiplication loop invoking the MPI task and retrieving the exit value.

```
private static void computeMultiplication() {
    Integer[][][] exitValues = new Integer[MSIZE][MSIZE][MSIZE];

    // Launch tasks
    for (int i = 0; i < MSIZE; ++i) {
        for (int k = 0; k < MSIZE; ++k) {
            for (int j = 0; j < MSIZE; ++j) {
                exitValues[i][j][k] = MPI.multiplyAccumulative(BSIZE, AfileNames[i][k],
                                      BfileNames[k][j], CfileNames[i][j]);
            }
        }
    }
}
```

FIGURE 6.12: Main multiplication loop of the Hybrid *Matmul*

Figure 6.13 shows the interface annotation of the MPI *multiplyAccumulative* task. Notice that the binary path, the number of computing nodes, and the number of computing units per node are retrieved from environment variables to easily parametrize the MPI binary path and the number of MPI processes during the experimentation phase. Moreover, the total number of processes available for the MPI *Matmul* is $P = CN \cdot CUS$. Regarding the

task parameters, we use the MPI matrix size (this is, the block size for the COMPSs layer) and the file path for the contents of the matrixes A, B and C (which are, for the COMPSs layer, the matrix blocks).

```
@MPI(binary = "${MATMUL_BINARY}",
     mpiRunner = "mpirun",
     computingNodes = "${CN}")
@Constraints(computingUnits = "${CUS}")
Integer multiplyAccumulative(
   @Parameter() int bsize,
   @Parameter(type = Type.FILE, direction = Direction.IN) String aIn,
   @Parameter(type = Type.FILE, direction = Direction.IN) String bIn,
   @Parameter(type = Type.FILE, direction = Direction.INOUT) String cOut
);
```

FIGURE 6.13: *multiplyAccumulative*'s interface annotation for the Hybrid COMPSs and MPI *Matmul*

Appendix B contains the complete code of this Hybrid COMPSs and MPI *Matmul* implementation.

6.1.2.4 Evaluation

To evaluate the new Matrix Multiplication implementation, we have run all the experiments on the *Nord III* [13] cluster, hosted at the Barcelona Supercomputing Center (BSC) composed of one compute rack from the MareNostrum III [10]. This supercomputer has 1344 Intel SandyBridge EP-2670 cores at 2.6 GHz, 10.5 TB of main memory, a peak performance of 28 GigaFlops and InfiniBand interconnection.

We have compared the standalone MPI implementation and the Hybrid (COMPSs and MPI) implementation for a different number of processes and a fixed matrix size to evaluate the overhead caused by including a new COMPSs layer on top of the MPI. Regarding the number of processes, we have chosen multiples of 16 to fill the computational nodes completely. Concerning the matrix size, it has been fixed to 16384 long float elements because it is the biggest matrix that can fit inside the memory of a single node (uses up to 29 Gb of main memory).

On the one hand, we have performed a set of experiments with a single MPI execution but with an increasing number of processes. Figure 6.14 shows the strong scaling analysis of both, the MPI and the Hybrid implementations. The left y-axis represents the execution time (in seconds), the right y-axis represents the percent overhead of the Hybrid implementation with respect to the MPI implementation, and the x-axis represents the number of processes. Notice that none of the implementations scales because the execution time includes the reading, the computation and the writing times, and, in our implementations, the reading and the writing phases are performed only by the master process. Although there are more sophisticated implementations that initialize and store the data in parallel, this is beyond the scope of this section since we are only willing to evaluate the overhead of including a new layer on top of MPI. In this sense, notice that the overhead of wrapping the MPI Matmul implementation with COMPSs is below the 5% for all the cases.

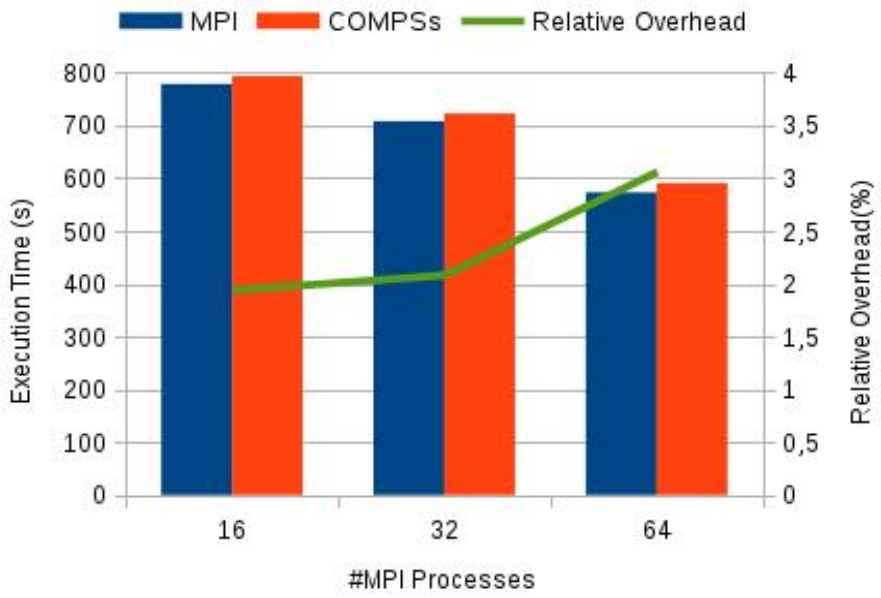

FIGURE 6.14: *Matmul* Strong Scaling analysis

On the other hand, we have performed a set of experiments with multiple MPI executions with the same amount of MPI threads. We have fixed the MPI threads to 16 to use one node completely, but as future work, it could be interesting to perform a more in-depth experimentation with an MPI thread size that does not fit properly to the node slots.

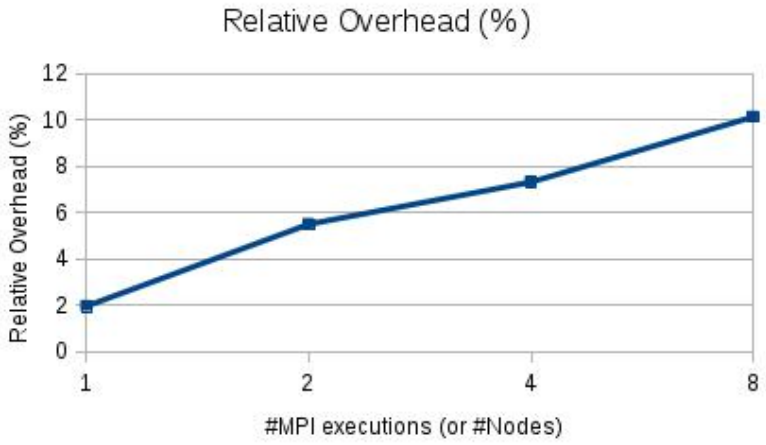

FIGURE 6.15: *Matmul* Weak Scaling analysis

Figure 6.15 shows the weak scaling analysis of the Hybrid implementation. The y-axis represents the percent overhead with respect to the standalone MPI implementation with 16 processes and matrix size 16384 long float elements, and the x-axis represents the number of simultaneous MPI executions of 16 threads each (or, what is equivalent, the number of nodes). Notice that, when using COMPSs to orchestrate several MPI executions, the overhead remains negligible (less than 8%) for up to 4 MPI executions (each of them using a complete node). However, when orchestrating 8 or more MPI executions, the overhead of

scheduling the tasks and managing the available resources becomes significant and requires a high-load computation in the MPI layer.

To conclude this section, the first set of experiments have demonstrated that including a new COMPSs layer on top of the MPI layer does not cause a significant overhead. Moreover, in the second set of experiments, we have shown that our enhancement of the COMPSs Runtime is capable of orchestrating up to 4 different MPI executions of 16 threads each without a significant overhead.

6.2 Use cases

6.2.1 NMMB/BSC-Dust

6.2.1.1 Application description

NMMB/BSC-Dust model [12] is pluggable component of the Non-hydrostatic Multiscale Model (NMMB) designed and developed by the Earth Sciences Department of the Barcelona Supercomputing Center (BSC) in collaboration with NOAA/National Centers for Environmental Prediction (NCEP), NASA Goddard Institute for Space Studies and the International Research Institute for Climate and Society (IRI). This model provides short to medium-range weather and dust forecasts for regional and global domains (Figure 6.16 shows a simulation example of four-hour average AOD of the North African Domain on 16 May 2006).

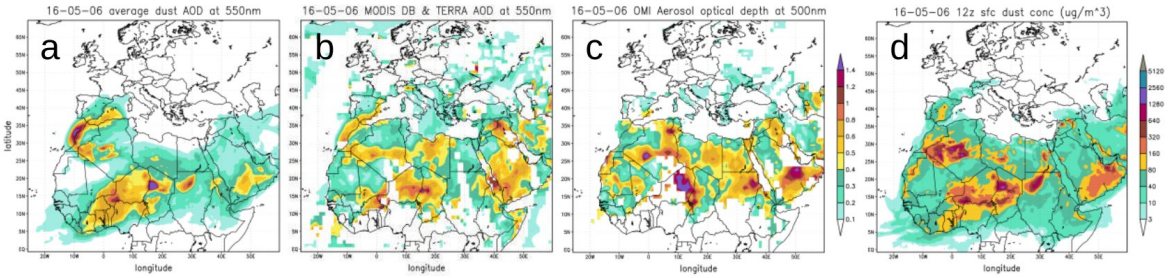

FIGURE 6.16: Example of four hour average AOD from NMMB/BSC-Dust
Source: NMMB/BSC-Dust model [12]

NMMB/BSC-Dust model appears in the frame of quantifying the global dust emission from arid and semi-arid areas and its spatial distribution. Far from being useless, *"dust significantly affects the global and regional energy balance by absorbing and scattering shortwave and longwave radiation, dust transported by winds modifies atmospheric heating rates, temperature and stability, influences the hydrological cycle and impacts the human health"* [1]. In fact, in the past decades, several models have been developed to reproduce the dust cycle and estimate its influence on the climate system.

More in-depth, the NMMB/BSC-Dust model defines a complex analytic workflow in a *BASH* script of 778 lines that handles the execution of an initialization step (called *FIXED*) and a main loop that is executed for each timestep of the simulation period. The main loop is defined by three internal steps: the *VARIABLE* pre-process, the *NMMB/BSC-Dust model simulation* and the post-process step. Figure 6.17 depicts the general NMMB/BSC-Dust workflow. Although each of these steps spawns a large number of binary calls (the biggest part of them written in *Fortran 77* and *Fortran 90*), the only one that actually runs in parallel is the MPI simulation inside the *UMO_MODEL* step. Furthermore, the script is already prepared to support different workflow options by redefining some variables inserted at its beginning (i.e. enabling and disabling a specific step, defining a global or a regional domain, changing the domain size, or changing the model data folder or parameters).

6.2.1.2 Purpose

The purpose is to evaluate the contributions of this thesis by porting a real data science application to COMPSs so that it can benefit of the COMPSs programming model abstractions. The porting must transform the current NMMB/BSC-Dust implementation (a complex sequential workflow defined in *BASH*) into an easy, parallel, and portable code without any loss of performance.

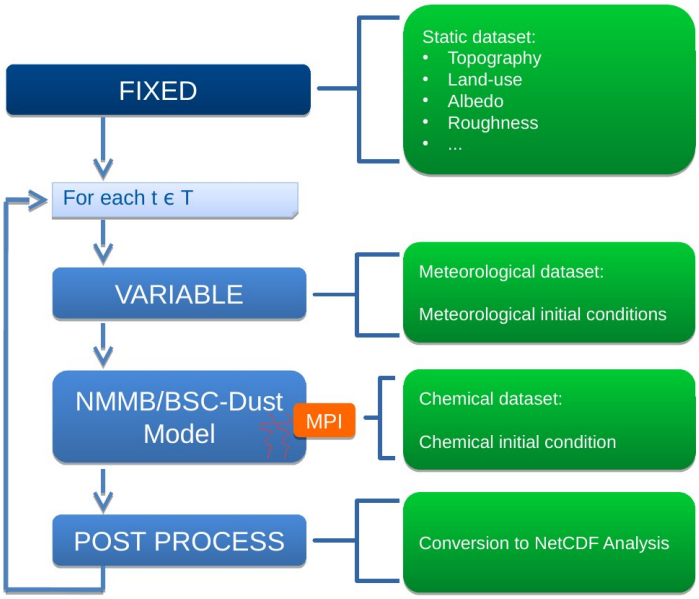

FIGURE 6.17: NMMB/BSC-Dust step workflow

6.2.1.3 NMMB/BSC-Dust implementation with COMPSs

From our point of view, having a complex workflow defined in such a long *BASH* script lacks reliability. On the one hand, because users need to modify the code to change the simulation parameters and, on the other hand, because maintaining this script becomes hard for unexperienced users.

Consequently, we have ported the NMMB/BSC-Dust workflow to a Java application. The new workflow has a main class to define the main step workflow (*Nmmb.java*), the application interface for COMPSs (*NmmbItf.java*, the binary and MPI tasks dummy definitions (*BINARY.java* and *MPI.java*), two classes to load and handle the simulation parameters (*NMMBConfigManager.java* and *NMMBParameters.java*), a class to handle the environment variables (*NMMBEnvironment.java*), a class to store all the constant values (*NMMBConstants.java*) and five util classes to encapsulate the file management, the logger printers and the *BASH* and *Fortran* executors. Figure 6.3 shows the code summary for the implementation and Appendix C contains the main file and the interface of the application.

Language	Files	Blank	Comment	Code
Fortran 90	23	394	2806	7581
Fortran 77	8	182	3568	6518
Java	18	558	887	2688
Bourne Shell	12	155	117	669
Maven	1	10	16	162
Bourne Again Shell	3	19	14	64
XML	1	0	0	13
SUM	66	1318	7408	17695

TABLE 6.3: NMMB/BSC-Dust code summary

This implementation offers three principal advantages. First, the execution parameters

are loaded from a configuration file that can be modified without recompiling the application (and without risking of involuntary modifications on the workflow behavior). Secondly, as it is built on top of COMPSs, it can rely on the power of the Runtime to abstract from the underlying infrastructure. In this sense, it can benefit from the *runcompss* and *enqueue_compss* commands to run over different infrastructures without modifying the application code. Finally, the binaries inside each step have been parallelized so that the available resources are exploited as much as possible.

6.2.1.4 Evaluation

To evaluate the new NMMB/BSC-Dust implementation we have run the experiments on the *Nord III* [13] cluster, hosted at the Barcelona Supercomputing Center (BSC) composed of one compute rack from the MareNostrum III [10]. This supercomputer has 1344 Intel SandyBridge EP-2670 cores at 2.6 GHz, 10.5 TB of main memory, a peak performance of 28 GigaFlops and InfiniBand interconnection. For the experimentation, we have run several simulations of 1 day (a single loop iteration) of a global domain with 64 cores available for the MPI simulation (4 nodes). In the case of the COMPSs implementation, we have requested 4 nodes for computation (COMPSs Workers) and an extra node for the COMPSs Master processes.

First, we have checked that the final result produced by both implementations is the same. The NMMB/BSC-Dust application produces a forecast binary file (*CASE.nc*) that can be visualized with the NCVIEW [20] visual browser for *netCDF* format files. As expected, in both cases, the file produced weights 210 Mb, and its visualization results are the same. Figures 6.18, 6.19 and 6.20 show three different views of the forecast result with both implementations (at right, the original result and, at left, with the new COMPSs implementation) to demonstrate that the visualizations are the same.

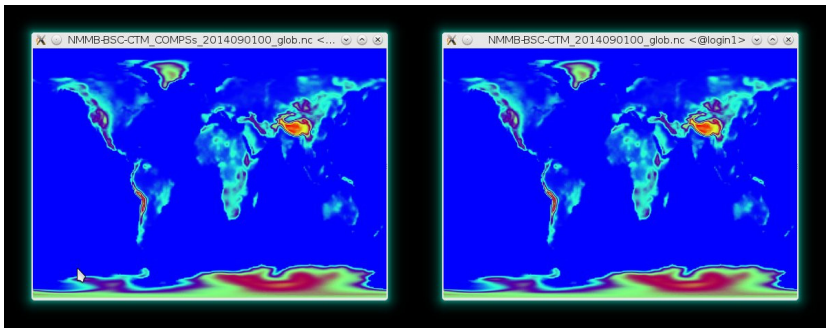

FIGURE 6.18: NCVIEW plot of FIS 3D variable for both implementations

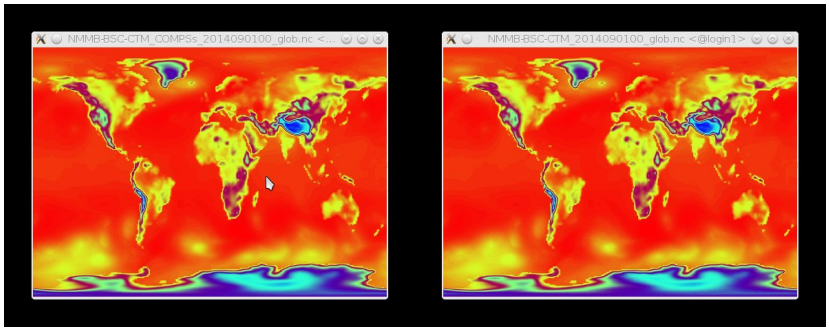

FIGURE 6.19: NCVIEW plot of PS 3D variable for both implementations

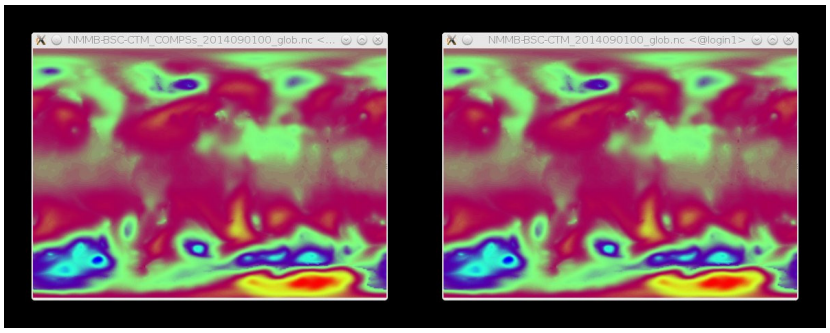

FIGURE 6.20: NCVIEW plot of SLP 3D variable for both implementations

Next, In terms of performance, we have measured the execution time of each step and the total execution time for both implementations. On the one hand, for the *old* implementation we have measured the time inserting *time* calls on the main BASH script. On the other hand, for the implementation with COMPSs, we have measured the time adding system calls on the main Java class. Table 6.4 summarizes the execution times for each implementation and the relative speed-ups. Notice that the implementation with COMPSs has an overall speed-up of $s = 1.45$ because the fixed, the variable and the post-process steps are splitted in tasks that can run in parallel. Although we will proceed to a more in-depth study, we can advance that this speed-up is limited by the data dependencies between the tasks inside each step. Moreover, the execution time of the Model Simulation step remains the same, allowing us to ensure that the current implementation of the *MPI* annotations do not introduce a significant overhead.

Steps	**Execution Times (s)**		**Speed-up (u)**
	Previous Impl.	**COMPSs Impl.**	
Fixed	290	117	2.48
Variable	26	19	1.37
Model simulation	244	242	1.01
Post process	38	34	1.12
Total	**601**	**413**	**1.45**

TABLE 6.4: Execution times of the different NMMB/BSC-Dust implementations for the simulation of 1 day of global domain with 64 cores

Finally, we have proceeded to analyze in-depth the COMPSs implementation. Figure 6.21 shows the task graph of the NMMB/BSC-Dust when executed with COMPSs. For the sake of clarity, we have highlighted each of the different steps.

Notice that the *fortranCompile* tasks open a significant parallelism at the beginning of the FIXED and the VARIABLE phases, but the rest of the tasks define a quite complex graph with lots of dependencies. In fact, the current implementation defines 37 different tasks and executes 58 tasks for a single iteration of the main loop (31 for the fixed step, 24 for the variable step, 1 for the model simulation and 2 for the post-process). Furthermore, the tasks have at least 3 parameters, at most 47 parameters (*allprep* task) and 7 parameters in average.

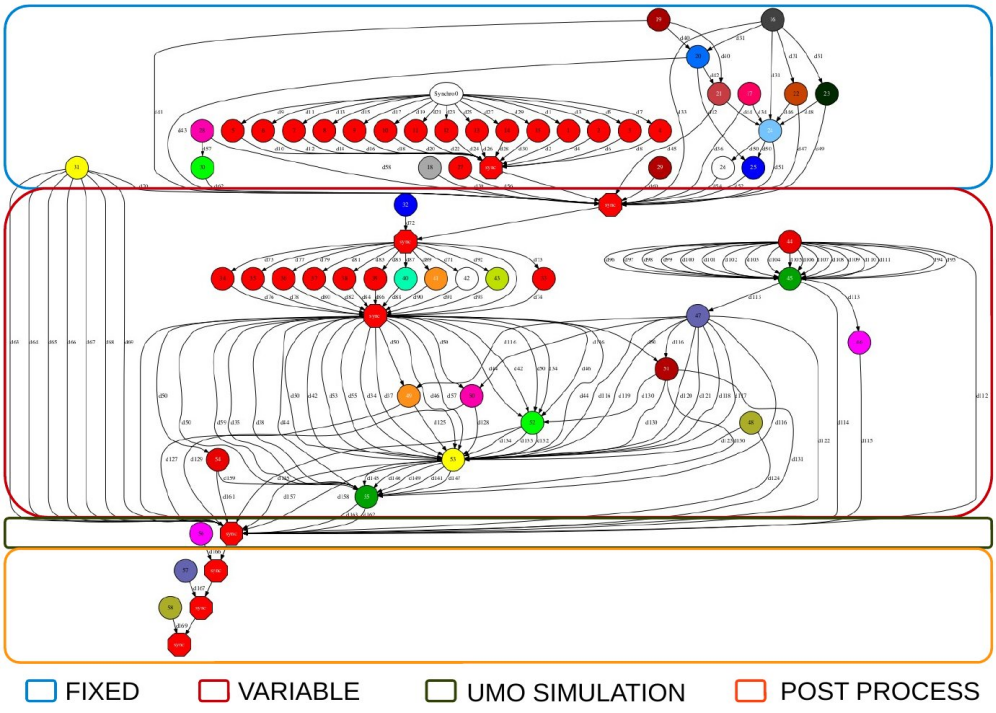

FIGURE 6.21: Tasks graph of NMMB/BSC-Dust with COMPSs

Thanks to the integration of COMPSs with the Extrae tool we have also obtained post-mortem trace of the application's execution. Figure 6.22 shows the Paraver view of the task trace of the NMMB/BSC-Dust implementation with COMPSs. For the sake of clarity, we have highlighted each of the steps at the bottom of the image.

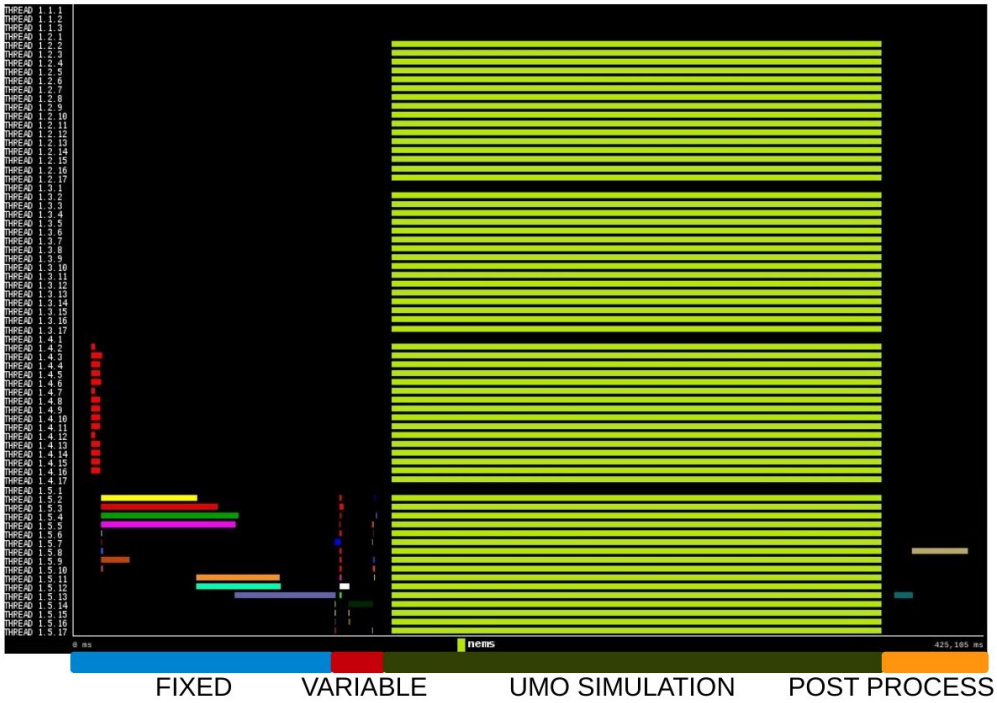

FIGURE 6.22: Paraver task view of the NMMB/BSC-Dust execution

Notice that the first node is reserved for the COMPSs Master and that the first thread of each node is reserved for the COMPSs Worker. For the rest of the threads, the application does not fill all the available resources during the *fixed*, *variable* and *post-process* steps because there are not enough tasks to run in parallel. In fact, the first compilation tasks (in red) are run in the fourth resource (threads 1.4.2 - 1.4.17) because it is the first available worker in the execution. However, the rest of the *fixed* and *variable* tasks are executed in the fifth resource (threads 1.5.2 - 1.5.17) because the first task has been arbitrarilly scheduled to this resource and the rest of them is scheduled to the same resource because of data locality (notice that, as shown in the task graph, there is a synchronization point rather than a data dependency between the first compilation tasks and the rest of the tasks). Regarding the Model Simulation, the scheduler has reserved all the available resources because it requires 64 cores. On the *post-process* step, the two tasks are also scheduled to the fifth resource because of data locality.

As a final note, this implementation is a prototype of the NMMB/BSC-Dust running with COMPSs. For next implementations, since there are free resources during the *fixed*, the *variable* and the *post-process* steps, it would be worth to avoid synchronizations between them and try to overlap the execution of tasks of different steps. It will also be interesting to use dynamic resource management to release resources during these steps and acquire them again during the execution of the Model Simulation (since the requirements of this MPI step are the ones that obligate the application to request so many resources).

6.2.2 GUIDANCE

6.2.2.1 Application description

GUIDANCE [9] is a framework for large-scale genome and phenome-wide association studies on parallel computing platforms developed by the Computational Genomics Group at the Barcelona Supercomputing Center (BSC). This integrated framework provides an easy solution to perform Genome and Phenome association analysis, allowing the users to perform all the steps in a single execution or in a modular way with optional user intervention. The current GUIDANCE's implementation is already based on COMPSs to make the application integrable to multiple parallel and distributed platforms, and to ensure the efficient usage of the computing resources.

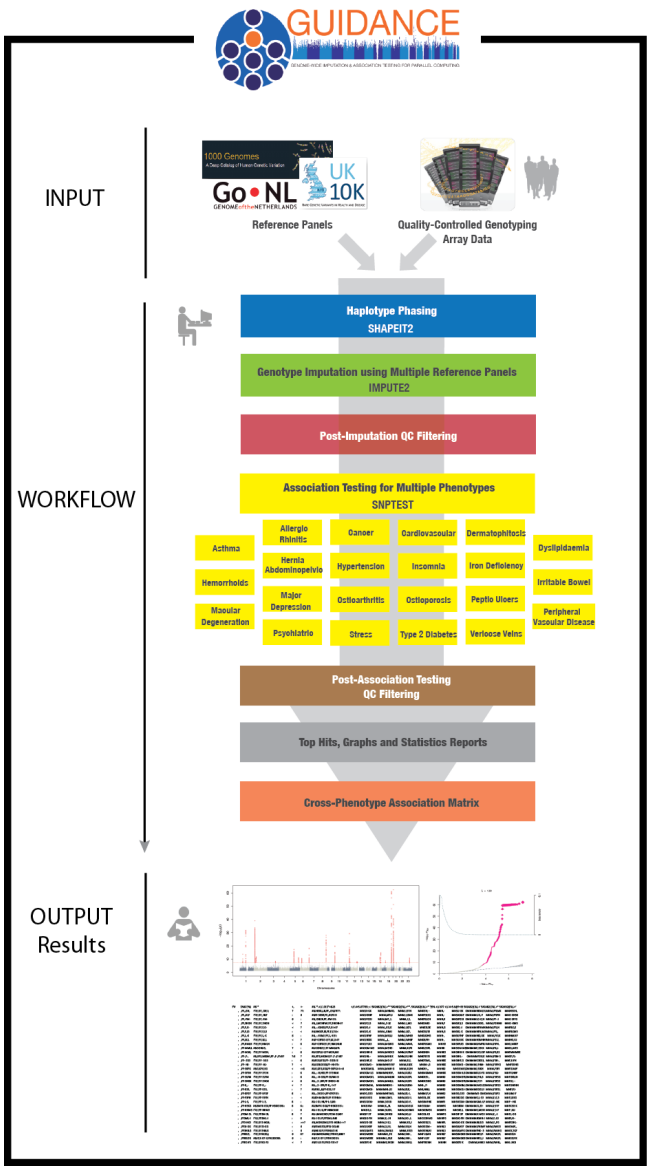

FIGURE 6.23: GUIDANCE's schematic representation of the typical complete Genome and Phenome association analysis. Source: [9]

Figure 6.23 provides a schematic representation of the steps performed by the typical complete Genome and Phenome association analysis. The application's workflow starts

retrieving Quality Controlled genetic data. Next, it goes through phasing and imputation using multiple panels and performs an association test considering multiple phenotypes (*SNPTEST*). Finally, GUIDANCE provides summary statistics and graphical representations of the results.

6.2.2.2 Purpose

The purpose is to evaluate the advantages of this thesis' developments with respect to the previous COMPSs model by upgrading a real data science application that already uses COMPSs. During the past year, the previous GUIDANCE implementation has become a huge workflow that can be hardly managed, mainly because of the invocation of many binary files. The upgrade must simplify the previous implementation without any loose of performance so that future engineers can keep improving the GUIDANCE framework in an easy way.

6.2.2.3 GUIDANCE implementation with COMPSs

GUIDANCE's main code is implemented in Java and defines a task workflow of both binary invocations and native methods. The main file (*Guidance.java*) contains the workflow description and a wrapper function for each task defined inside the *GuidanceImpl.java* file. All the tasks are duly annotated inside the *GuidanceItf.java* file.

```
public static void binaryTaskExample(String[] cmdParams) throws Exception {
    /**** SOME PRE-PROCESS (if any) ****/

    // Command construction
    String cmd = BINARY + FLAGS + cmdParams;
    ProcessBuilder pb = new ProcessBuilder(cmd.split(" "));
    // Process spawn
    Process p = pb.start();
    // Handling the streams so that dead lock situation never occurs
    byte[] b = new byte[1024];
    int read;
    BufferedInputStream bisInp = new BufferedInputStream(p.getInputStream());
    BufferedOutputStream bosInp = new BufferedOutputStream(
    new FileOutputStream(outputFile + ".stdout"));
    while ((read = bisInp.read(b)) >= 0) {
       bosInp.write(b, 0, read);
    }
    BufferedInputStream bisErr = new BufferedInputStream(p.getErrorStream());
    BufferedOutputStream bosErr = new BufferedOutputStream(
    new FileOutputStream(errorFile + ".stderr"));
    while ((read = bisErr.read(b)) >= 0) {
       bosErr.write(b, 0, read);
    }
    bisErr.close();
    bisInp.close();
    bosErr.close();
    bosInp.close();
    // Check the proper ending of the process
    int exitValue = p.waitFor();
    if (exitValue != 0) {
       throw new Exception("[ERROR] Task error, exit value is: " + exitValue);
    }

    /**** SOME POST-PROCESS (if any) ****/
}
```

FIGURE 6.24: Example of previous GUIDANCE binary task implementation

The previous implementation defines 24 different tasks; where 9 of them spawn a binary execution. Considering only these 9 tasks that spawn a binary execution, all of them have a common structure (see Figure 6.24): they construct the binary command, spawn a *ProcessBuilder*, launch the binary execution, capture the command output and error, and retrieve the process exit value. In some specific cases, there is also some pre-process to parse or decompress the given arguments, and some post-process to rename or compress the output files. Notice that this structure is repeated several times among the code and gets the code dirty for the final user.

Regarding the execution, GUIDANCE's smallest test lasts 9780 seconds (less than 3 hours) and spawns 1010 tasks. The test size cannot be smaller because the test must cover all the possibilities within the workflow. As depicted in the cropped task graph shown in Figure 6.25, the GUIDANCE implementation has an initialization phase with phew parallelism, an intermediate phase with a scatter/gather structure, and a final phase to analyze the results.

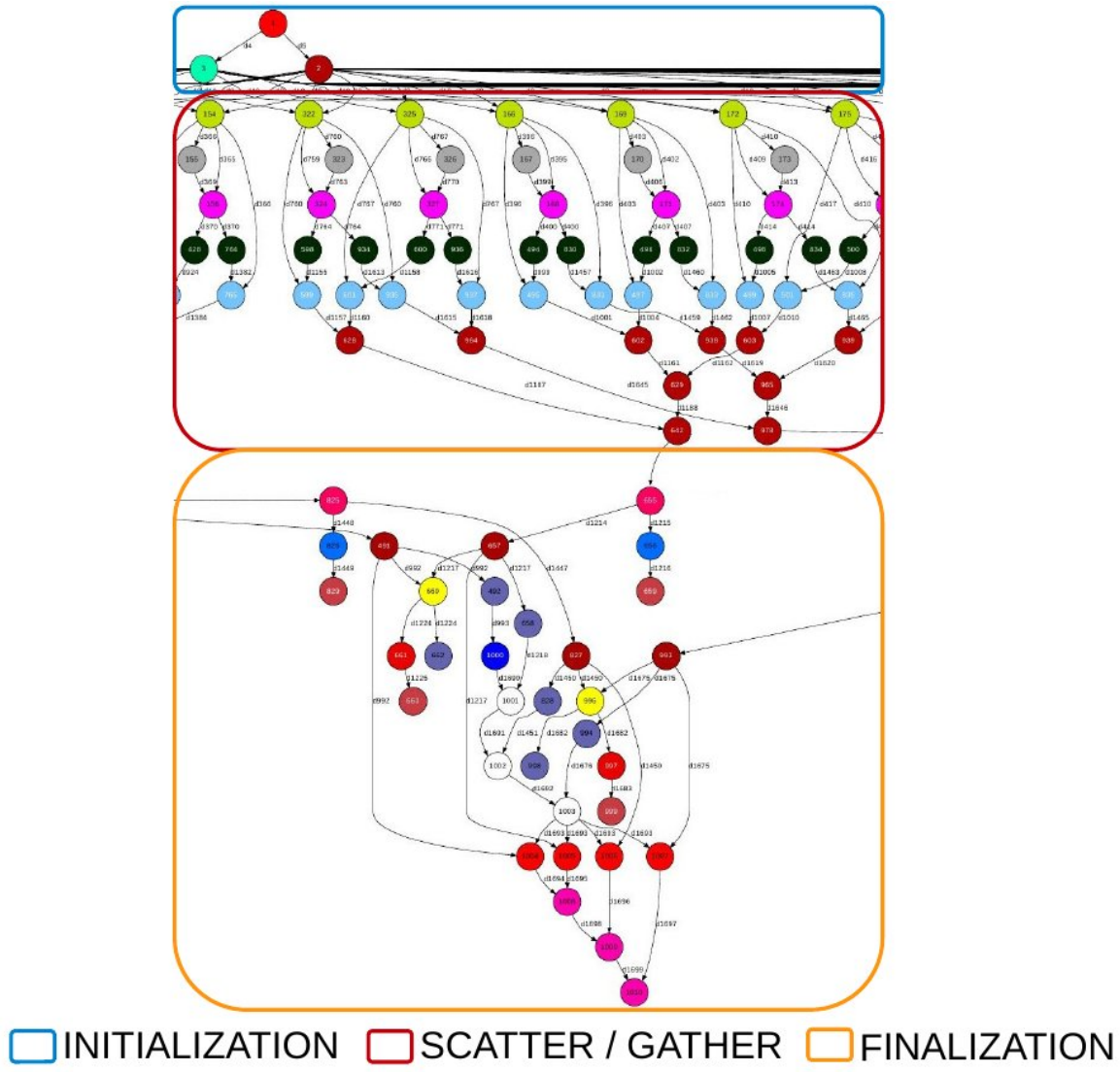

FIGURE 6.25: GUIDANCE's partial task graph

6.2.2.4 Evaluation

Regarding programmability, the new annotations allow the users to remove all the *ProcessBuilder* management required to launch binaries. In the GUIDANCE case, 9 tasks from the 24 available have been completely removed and substituted by a task with a binary annotation. Since some of this tasks require pre or post steps (such as compressing and decompressing files), we have added 4 additional pre/post tasks. Appendix D contains the main file and the interface of the GUIDANCE application.

Taking all into consideration, as shown in Figure 6.5 the tasks' code (*GuidanceImpl*) has been reduced by three and the application's main code (*Guidance*) has been reduced by a 20 %. On the other hand, the application's annotation interface has grown since the binary tasks have more parameters (the previous implementation had an average of 6.5 parameters per task, and the new implementation has an average of 9.0 parameters per task). The overall GUIDANCE's code has been reduced a 23.28 %, mostly due to the removal of the *ProcessBuilder* management.

File	Implementation	Num. Files	Blank	Comment	Code
Guidance	NEW	1	327	749	1296
GuidanceImpl	NEW	1	193	445	1074
BINARY	NEW	1	20	210	55
GuidanceItf	NEW	1	32	0	374
TOTAL	NEW	43	1862	4108	7125
Guidance	OLD	1	457	531	1637
GuidanceImpl	OLD	1	1066	1053	3488
GuidanceItf	OLD	1	54	41	352
TOTAL	OLD	14	2619	2891	9288

TABLE 6.5: GUIDANCE code summary

From our point of view, this new implementation is way easier to maintain than the previous one since the programmers do not need to handle with the binary invocations directly (but rather rely on the COMPSs *@Binary* annotation). Moreover, we have included an abstraction layer on the application's main code so that the programmers can add new steps to the workflow in an easy templatized way.

Finally, due to time constraints and the shutdown of the MareNostrum III Supercomputer this application has only been evaluated in terms of programmability. As future work, we plan to validate the implementation with big runs in the new MareNostrum IV supercomputer. However, since previous cases did not show up any lose of performance, we do not expect any improvement nor deterioration of the execution times.

Chapter 7

Conclusions and Future work

This Master Thesis provides a first adaptation of the COMPSs programming model to the needs of the Big-Data Ecosystems. Concerning the COMPSs programming model, this thesis provides a new set of task annotations to easily integrate Java workflows with the execution of binaries, and *MPI* and *OmpSs* applications. To provide full support to this integration, we have also implemented two new parameter annotations to add string prefixes and Linux stream redirections. Taking advantage of the new annotations, we have profited to redesign the versioning annotations, to extend the task annotations to support environment variables, and added new annotations for scheduler hints.

Regarding the COMPSs internals, this project has extended the schedulers to support multi-node actions in a transparent and non-blocking fashion. Furthermore, we have redesigned the COMPSs Worker Executors to support the execution of non-native tasks and to reduce the overhead of executing bindings' tasks (Python, C, and C++ tasks).

The previous implementations have been validated against two proof of concept applications and two real uses cases. On the one side, the BLAST and the Matrix Multiplication applications have demonstrated that the new annotations reduce the code complexity while maintaining the performance. On the other side, the NMMB/BSC-Dust and the GUIDANCE applications have shown that the new COMPSs features adapt to the Big-Data Ecosystem requirements; providing a simpler, portable and efficient implementation.

During the development of this project, the COMPSs Bindings (for Python, C, and C++ languages) have increasingly gained importance to provide a comfortable integration with the data science workflows. Hence, as future work, we plan to integrate the new COMPSs annotations with the COMPSs bindings. Moreover, since the community has reacted positively to the overhead's reduction that the new COMPSs Worker Executors provide when executing binding tasks, we plan to design persistent Python, C and C++ workers that can communicate through pipes with the current Java persistent worker and reduce even more this overhead.

Regarding the COMPSs Runtime itself, as future work, we also plan to implement pluggable scheduler politics so that users can select the suitable scheduler for their applications. In this sense, the new schedulers could profit from the new scheduler hints annotation to have more information about the tasks and take more complex decisions.

Finally, concerning the real uses cases, on the one hand, we want to perform executions of the NMMB/BSC-Dust application in other supercomputers to take advantage of the portability that the new COMPSs implementation provides. On the other hand, we want to do a performance analysis of the new GUIDANCE implementation to validate and compare it against the previous implementation.

Bibliography

[1] Haustein et al. Perez et al. "Atmospheric dust modeling from meso to global scales with the online NMMB/BSC-Dust model – Part 2: Experimental campaigns in Northern Africa". In: *Atmospheric Chemistry and Physics Journal (ACP)* 12 (Mar. 2012), pp. 2933–2958. URL: http://www.atmos-chem-phys.net/12/2933/2012/.

[2] Jack Dongarra et al. "The international Exascale Software Project roadmap". In: *International Journal of High Performance Computing Applications* 25.1 (Feb. 2011), pp. 3–60. URL: http://hpc.sagepub.com/content/25/1/3.

[3] Krste Asanovic et al. "A view of the Parallel Computing Landscape". In: *Communications of the ACM* 52.10 (Oct. 2009), pp. 56–67. URL: http://dl.acm.org/citation.cfm?id=1562783.

[4] R. M. Badia et al. "COMP superscalar, an interoperable programming framework". In: *SoftwareX* 3 (Dec. 2015), pp. 32–36. URL: http://dx.doi.org/10.1016/j.softx.2015.10.004.

[5] Derik et al. Barseghian. "Workflows and extensions to the Kepler scientific workflow system to support environmental sensor data access and analysis". In: *Ecological Informatics* 5 (2010), pp. 42–50. URL: http://dx.doi.org/10.1016/j.ecoinf.2009.08.008.

[6] National Center for Biotechnology Information (NCBI). *Blast*. URL: https://blast.ncbi.nlm.nih.gov/Blast.cgi.

[7] Barcelona Supercomputing Center (BSC). *Barcelona Supercomputing Center (BSC)*. URL: http://www.bsc.es.

[8] Barcelona Supercomputing Center (BSC). *Extrae Tool*. URL: https://tools.bsc.es/extrae.

[9] Barcelona Supercomputing Center (BSC). *Guidance*. URL: http://cg.bsc.es/guidance/.

[10] Barcelona Supercomputing Center (BSC). *MareNostrum 3 User Guide*. URL: https://www.bsc.es/support/MareNostrum3-ug.pdf.

[11] Barcelona Supercomputing Center (BSC). *Minotauro*. URL: https://www.bsc.es/innovation-and-services/supercomputers-and-facilities/minotauro.

[12] Barcelona Supercomputing Center (BSC). *NMMB BSC-Dust*. URL: http://www.bsc.es/ESS/nmmb_bsc-dust.

[13] Barcelona Supercomputing Center (BSC). *Nord III*. URL: https://www.bsc.es/user-support/nord3.php.

[14] Barcelona Supercomputing Center (BSC). *OmpSs*. URL: https://pm.bsc.es/ompss.

[15] Barcelona Supercomputing Center (BSC). *Paraver Tool*. URL: https://tools.bsc.es/paraver.

[16] Universitat Politècnica de Catalunya (UPC). *Universitat Politècnica de Catalunya (UPC)*. URL: http://www.upc.es.

[17] Scala Center. *Scala Programming Language*. URL: https://www.scala-lang.org/.

[18] Shigeru Chiba. "Load-time Structural Reflection in Java". In: *ECOOP 2000 - Object-Oriented Programming* 1850 (May 2000), pp. 313–336. URL: http://dx.doi.org/10.1007/3-540-45102-1_16.

[19] COMP Superscalar (COMPSs). *COMP Superscalar (COMPSs)*. URL: http://compss.bsc.es.

[20] Scripps Institution of Oceanography David W. Pierce. *NCView*. URL: http://meteora.ucsd.edu/~pierce/ncview_home_page.html.

[21] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters". In: *Communications of the ACM* 51.1 (Jan. 2008), pp. 107–113. URL: http://dl.acm.org/citation.cfm?id=1327492.

[22] Ewa Deelman. "Big Data Analytics and High Performance Computing Convergence Through Workflows and Virtualization". In: *Big Data and Extreme-Scale Computing* (2016). URL: http://www.exascale.org/bdec/sites/www.exascale.org.bdec/files/whitepapers/deelman-bdec2016.pdf.

[23] Jesús Labarta et al. Enric Tejedor Rosa M. Badia. "PyCOMPSs: Parallel computational workflows in Python". In: *The International Journal of High Performance Computing Applications (IJHPCA)* 31 (2017), pp. 66–82. URL: http://dx.doi.org/10.1177/1094342015594678.

[24] R Foundation. *R Programming Language*. URL: https://www.r-project.org/.

[25] GNU. *Bash*. URL: https://www.gnu.org/software/bash/.

[26] GNU. *C Socket implementation*. URL: https://www.gnu.org/software/libc/manual/html_node/Sockets.html.

[27] GNU. *GNU Plot*. URL: http://www.gnuplot.info/.

[28] Galaxy Community HUB. *Galaxy Project*. URL: https://galaxyproject.org/.

[29] Intel. *Intel Math Kernel Library (Intel MKL)*. URL: https://software.intel.com/en-us/intel-mkl.

[30] Intel. *Intel MPI implementation*. URL: https://software.intel.com/en-us/intel-mpi-library.

[31] F. G. Gustavson J. J. Dongarra and A. Karp. "Implementing linear algebra algorithms for dense matrices on a vector pipeline machine". In: *SIAM Review* 26.1 (Jan. 1984), pp. 91–112. URL: http://dx.doi.org/10.1137/1026003.

[32] Anubhav et al. Jain. "FireWorks: a dynamic workflow system designed for high-throughput applications". In: *Concurrency and Computation: Practice and Experience* 27.17 (2015), pp. 5037–5059. URL: http://dx.doi.org/10.1002/cpe.3505.

[33] Rosa M. Badia Javier Conejero Sandra Corella and Jesus Labarta. "Task-based programming in COMPSs to converge from HPC to big data". In: *The International Journal of High Performance Computing Applications* (Apr. 2017), p. 1094342017701278. URL: https://doi.org/10.1177/1094342017701278.

[34] R. M. Badia et al. Lordan F. "ServiceSs: an interoperable programming framework for the Cloud". In: *Journal of Grid Computing* 12.1 (Mar. 2014), pp. 67–91. URL: https://digital.csic.es/handle/10261/132141.

[35] Andy Konwinski Matei Zaharia Patrick Wendell and Holden Karau. *Learning Spark*. 2015.

[36] Eclipse Members. *Eclipse IDE*. URL: https://eclipse.org/.

[37] MongoDB. *MongoDB*. URL: https://www.mongodb.com.

[38] *MPI: A Message-Passing Interface Standard*. June 2015. URL: http://mpi-forum.org/docs/.

[39] Oracle. *Java Programming Language*. URL: https://www.oracle.com/es/java/index.html.

[40] Oracle. *Java Socket implementation*. URL: https://docs.oracle.com/javase/7/docs/api/java/net/Socket.html.

[41] Oracle. *JavaNIO*. URL: http://www.oracle.com/technetwork/articles/javase/nio-139333.html.

[42] Oracle. *Socket Definition*. URL: https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html.

[43] Apache Organization. *Apache Hadoop*. URL: http://hadoop.apache.org/.

[44] Apache Organization. *Taverna*. URL: http://www.taverna.org.uk/.

[45] Apache Maven Organization. *Maven*. URL: https://maven.apache.org/.

[46] JSON Organization. *JSON*. URL: http://www.json.org/.

[47] Yaml Organization. *YAML*. URL: http://yaml.org/.

[48] IBIS Project. *JavaGAT*. URL: http://www.cs.vu.nl/ibis/javagat.html.

[49] Python. *Python Socket implementation*. URL: https://docs.python.org/3/howto/sockets.html.

[50] Daniel A. Reed and Jack Dongarra. "Exascale Computing and Big Data". In: *Communications of the ACM* 58.7 (July 2015), pp. 56–68. URL: http://cacm.acm.org/magazines/2015/7/188732-exascale-computing-and-big-data.

[51] Manchester Joint team from the universities of Southampton and Oxford in the UK. *myExperiment*. URL: https://www.myexperiment.org/.

[52] Apache Spark. *Apache Spark*. URL: http://spark.apache.org/.

[53] Swift. *Swift Programming Language*. URL: http://swift-lang.org/.

[54] Open MPI Development Team. *Open MPI implementation*. URL: https://www.open-mpi.org/.

Appendices

Appendix A

Blast: complete code

A.1 Blast.java

```
1  package blast;
2
3  import java.io.BufferedReader;
4  import java.io.File;
5  import java.io.FileReader;
6  import java.io.IOException;
7
8  import java.util.UUID;
9  import java.util.ArrayList;
10 import java.util.LinkedList;
11 import java.util.List;
12
13 import binary.BINARY;
14
15 import blast.BlastImpl;
16 import blast.exceptions.BlastException;
17 import blast.utils.FilesManagement;
18
19 /**
20  * BLAST implementation
21  *
22  */
23 public class Blast {
24
25   private static final String ENV_BLAST_BINARY = "BLAST_BINARY";
26
27   private static boolean debug;
28   private static String databasePath;
29   private static String inputFileName;
30   private static int numFragments;
31   private static String tmpDir;
32   private static String outputFileName;
33   private static String commandArgs;
34
35   private static List<String> partialOutputs = null;
36   private static List<String> partialInputs = null;
37
38   /**
39    * MAIN CODE
40    *
41    * @param args
42    * @throws BlastException
43    */
44   public static void main(String[] args) throws BlastException {
45     // Parse application parameters
46     parseArgs(args);
47
48     // -----------------------------------------------------
49     // Start execution
50     try {
51         // Split sequence input file
52         splitSequenceFile();
53         // Submit tasks
```

```
        alignSequences();
        // Assembly process
        String lastMerge=assembleSequences();
        // Move result to expected output file
        FilesManagement.copyResult(lastMerge,Blast.outputFileName, Blast.debug);
    } catch (BlastException be) {
      throw be;
      } finally {
        // Clean up partial results
        cleanUp();
      }
  }

  /**
   * Parses the input parameters and initializes the class atributes
   *
   * @param args
   */
  private static void parseArgs(String[] args) {
    Blast.debug = Boolean.parseBoolean(args[0]);
    Blast.databasePath = args[1];
    Blast.inputFileName = args[2];
    Blast.numFragments = Integer.parseInt(args[3]);
    Blast.tmpDir = args[4];
    Blast.outputFileName = args[5];

    Blast.commandArgs = "";
    for (int i = 6; i < args.length; i++) {
      Blast.commandArgs += args[i] + " ";
    }
  }

  /**
   * Splits the input sequence file into separated files
   *
   * @throws BlastException
   */
  private static void splitSequenceFile() throws BlastException {
    System.out.println("Split sequence file");

    // Read number of different sequences
    int nsequences = 0;
    try (BufferedReader bf = new BufferedReader(
            new FileReader(Blast.inputFileName))) {
      String line = null;
      while ((line = bf.readLine()) != null) {
        if (line.contains(">")) {
          nsequences++;
        }
      }
    } catch (IOException ioe) {
      String msg = "ERROR: Cannot read input file " + Blast.inputFileName;
      System.err.print(msg);
      throw new BlastException(msg, ioe);
    }

    System.out.println("- The total number of sequences is: " + nsequences);

    // Calculate seqs per fragment and needed files
    int seqsPerFragment = (int) Math
            .round(((double) nsequences / (double) Blast.numFragments));
    Blast.partialInputs = new ArrayList<>(Blast.numFragments);
    Blast.partialOutputs = new ArrayList<>(Blast.numFragments);

    if (Blast.debug) {
      System.out.println(
              "- The total number of sequences of a fragment is: "
                      + seqsPerFragment);
      System.out
              .println("\n- Splitting sequences among fragment files...");
    }

```

```
    // Split into files
127   for (int frag = 0; frag < Blast.numFragments; ++frag) {
      // Creating fragment
129   UUID index = UUID.randomUUID();
      String partitionFile = Blast.tmpDir + "seqFile" + index + ".sqf";
131   String partitionOutput = Blast.tmpDir + "resFile" + index
              + ".result.txt";
133
      // Initialize partial fragment information
135   BlastImpl.splitPartitions(Blast.inputFileName, partitionFile,
              Blast.numFragments, frag);
137
      // Store fileNames
139   Blast.partialInputs.add(partitionFile);
      Blast.partialOutputs.add(partitionOutput);
141   }

143   if (Blast.debug) {
      System.out.println("Input Files are: ");
145   for (int i = 0; i < Blast.partialInputs.size(); ++i) {
        System.out.println("   - " + Blast.partialInputs.get(i));
147   }
      System.out.println("Output Files are: ");
149   for (int i = 0; i < Blast.partialOutputs.size(); ++i) {
        System.out.println("   - " + Blast.partialOutputs.get(i));
151   }
    }
153 }

155 /**
     * Aligns each fragment
157  *
     */
159 private static void alignSequences() throws BlastException {
      System.out.println("");
161   System.out.println("Aligning Sequences:");

163   final String pFlag = "-p";
      final String pMode = "blastx";
165   final String dFlag = "-d";
      final String iFlag = "-i";
167   final String oFlag = "-o";
      int numAligns = Blast.partialInputs.size();
169   Integer[] exitValues = new Integer[numAligns];
      for (int i = 0; i < numAligns; i++) {
171     if (Blast.commandArgs != null && !Blast.commandArgs.isEmpty()) {
          exitValues[i] = BINARY.align(pFlag, pMode, dFlag,
173               Blast.databasePath, iFlag, Blast.partialInputs.get(i),
                  oFlag, Blast.partialOutputs.get(i), Blast.commandArgs);
175     } else {
          exitValues[i] = BINARY.align(pFlag, pMode, dFlag,
177               Blast.databasePath, iFlag, Blast.partialInputs.get(i),
                  oFlag, Blast.partialOutputs.get(i));
179     }
      }
181
      if (Blast.debug) {
183     System.out.println("");
        System.out.println(" - Number of fragments to assemble -> "
185             + Blast.partialOutputs.size());
      }
187 }

189 /**
     * Creates reduce tasks
191  *
     * @return fileName of last reduce
193  */
    private static String assembleSequences() {
195   // MERGE-REDUCE
      LinkedList<Integer> q = new LinkedList<>();
197   for (int i = 0; i < Blast.partialOutputs.size(); i++) {
```

```
      q.add(i);
199   }

201   int x = 0;
      while (!q.isEmpty()) {
203     x = q.poll();
        if (!q.isEmpty()) {
205       int y = q.poll();

207       if (debug) {
            System.out.println(
209                 " - Merging files -> " + Blast.partialOutputs.get(x)
                            + " and " + Blast.partialOutputs.get(y));
211       }
          BlastImpl.assemblyPartitions(Blast.partialOutputs.get(x),
213               Blast.partialOutputs.get(y));
          q.add(x);
215     }
      }
217

      return Blast.partialOutputs.get(0);
219   }

221   /**
       * Cleans up the intermediate files
223    *
       */
225   private static void cleanUp() {
        // Cleaning intermediate sequence input files
227     for (int i = 0; i < Blast.partialInputs.size(); i++) {
          File fSeq = new File(Blast.partialInputs.get(i));
229       fSeq.delete();
      }
231

      for (int i = 0; i < Blast.partialOutputs.size(); i++) {
233     File fres = new File(Blast.partialOutputs.get(i));
        fres.delete();
235   }
    }
237

}
```

A.2 BlastItf.java

```
  package blast;
2

  import integratedtoolkit.types.annotations.Parameter;
4 import integratedtoolkit.types.annotations.parameter.Direction;
  import integratedtoolkit.types.annotations.parameter.Type;
6 import integratedtoolkit.types.annotations.task.Binary;
  import integratedtoolkit.types.annotations.task.Method;
8

  public interface BlastItf {
10

    @Method(declaringClass = "blast.BlastImpl")
12  void splitPartitions(
          @Parameter(type = Type.FILE, direction = Direction.IN) String inputFileName,
14        @Parameter(type = Type.FILE, direction = Direction.OUT) String partitionFile,
          @Parameter() int nFrags, @Parameter() int myFrag
16  );

18  @Binary(binary = "${BLAST_BINARY}")
    Integer align(
20    @Parameter(type = Type.STRING, direction = Direction.IN) String pFlag,
          @Parameter(type = Type.STRING, direction = Direction.IN) String pMode,
22        @Parameter(type = Type.STRING, direction = Direction.IN) String dFlag,
          @Parameter(type = Type.STRING, direction = Direction.IN) String database,
24        @Parameter(type = Type.STRING, direction = Direction.IN) String iFlag,
```

```
        @Parameter(type = Type.FILE, direction = Direction.IN) String partitionFile,
        @Parameter(type = Type.STRING, direction = Direction.IN) String oFlag,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String partitionOutput,
        @Parameter(type = Type.STRING, direction = Direction.IN) String extraCMDArgs
  );

  @Binary(binary = "${BLAST_BINARY}")
  Integer align(
    @Parameter(type = Type.STRING, direction = Direction.IN) String pFlag,
        @Parameter(type = Type.STRING, direction = Direction.IN) String pMode,
        @Parameter(type = Type.STRING, direction = Direction.IN) String dFlag,
        @Parameter(type = Type.STRING, direction = Direction.IN) String database,
        @Parameter(type = Type.STRING, direction = Direction.IN) String iFlag,
        @Parameter(type = Type.FILE, direction = Direction.IN) String partitionFile,
        @Parameter(type = Type.STRING, direction = Direction.IN) String oFlag,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String partitionOutput
  );

  @Method(declaringClass = "blast.BlastImpl")
  void assemblyPartitions(
        @Parameter(type = Type.FILE, direction = Direction.INOUT) String partialFileA,
        @Parameter(type = Type.FILE, direction = Direction.IN) String partialFileB
  );

}
```

A.3 BlastImpl.java

```
package blast;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

import blast.exceptions.BlastException;

public class BlastImpl {

  /**
   * Splits the input file into the fragment myFrag
   *
   * @param inputFileName
   * @param partitionFile
   * @param nFrags
   * @param myFrag
   * @throws BlastException
   */
  public static void splitPartitions(String inputFileName,
          String partitionFile, int nFrags, int myFrag)
          throws BlastException {
    int frag = 0;
    try (BufferedReader bf = new BufferedReader(
            new FileReader(inputFileName));
            BufferedWriter bw = new BufferedWriter(
                    new FileWriter(partitionFile, true))) {

      String line = null;
      while ((line = bf.readLine()) != null) {
        if (line.contains(">")) {
          frag++;
        }
        if (frag % nFrags == myFrag) {
          bw.write(line);
          bw.newLine();
        }
      }
```

```
    } catch (IOException ioe) {
      String msg = "ERROR: Cannot read input file " + inputFileName;
      System.err.print(msg);
      throw new BlastException(msg, ioe);
    }
  }

  /**
   * Assembles the two partial files
   *
   * @param partialFileA
   * @param partialFileB
   */
  public static void assemblyPartitions(String partialFileA,
          String partialFileB) {
    System.out.println("Assembling partial outputs -> " + partialFileA
            + " to " + partialFileB);

    String line = null;
    boolean append = true;
    try (BufferedWriter bw = new BufferedWriter(
            new FileWriter(partialFileA, append));
            BufferedReader bfB = new BufferedReader(
                    new FileReader(partialFileB))) {

      while ((line = bfB.readLine()) != null) {
        bw.write(line);
        bw.newLine();
      }
    } catch (IOException ioe) {
      System.err.println("ERROR: Exception assembling partitions");
      ioe.printStackTrace();
    }
  }

}
```

A.4 BINARY.java

```
package binary;

/**
 * Dummy class for BINARY tasks
 *
 */
public class BINARY {

  public static Integer align(String pFlag, String pMode, String dFlag,
          String database, String iFlag, String partitionFile, String oFlag,
          String partitionOutput, String extraCMDArgs) {

    return -1;
  }

  public static Integer align(String pFlag, String pMode, String dFlag,
          String database, String iFlag, String partitionFile, String oFlag,
          String partitionOutput) {

    return -1;
  }

}
```

Appendix B

Matmul: complete code

B.1 Matmul.java

```
1  package matmul.files;
2
3  import java.io.IOException;
4
5  import integratedtoolkit.api.COMPSs;
6  import mpi.MPI;
7
8  /**
9   * MATMUL Implementation
10  *
11  */
12 public class Matmul {
13
14   private static int TYPE;
15   private static int MSIZE;
16   private static int BSIZE;
17
18   private static String[][] AfileNames;
19   private static String[][] BfileNames;
20   private static String[][] CfileNames;
21
22   /**
23    * MAIN CODE
24    *
25    * @param args
26    * @throws Exception
27    */
28   public static void main(String[] args) throws Exception {
29     // Check and get parameters
30     if (args.length != 3) {
31       System.out.println(
32                "Usage: matmul.files.Matmul <type> <MSize> <BSize>");
33       throw new Exception("[ERROR] Incorrect number of parameters");
34     }
35     TYPE = Integer.parseInt(args[0]);
36     MSIZE = Integer.parseInt(args[1]);
37     BSIZE = Integer.parseInt(args[2]);
38
39     // Initialize matrices
40     System.out.println("[LOG] TYPE parameter value = " + TYPE);
41     System.out.println("[LOG] MSIZE parameter value = " + MSIZE);
42     System.out.println("[LOG] BSIZE parameter value = " + BSIZE);
43     initializeVariables();
44     initializeMatrix(AfileNames, true);
45     initializeMatrix(BfileNames, true);
46     initializeMatrix(CfileNames, false);
47
48     // Wait for initialization
49     COMPSs.barrier();
50
51     // Compute matrix multiplication C = A x B
52     long startTime = System.currentTimeMillis();
53     computeMultiplication();
```

```
    long estimatedTime = System.currentTimeMillis() - startTime;
    System.out.println("[TIME] EXECUTION TIME = " + estimatedTime);

    // End
    System.out.println("[LOG] Main program finished.");
  }

  /**
   * Initializes the filenames
   *
   */
  private static void initializeVariables() {
    AfileNames = new String[MSIZE][MSIZE];
    BfileNames = new String[MSIZE][MSIZE];
    CfileNames = new String[MSIZE][MSIZE];
    for (int i = 0; i < MSIZE; i++) {
      for (int j = 0; j < MSIZE; j++) {
        AfileNames[i][j] = "A." + i + "." + j;
        BfileNames[i][j] = "B." + i + "." + j;
        CfileNames[i][j] = "C." + i + "." + j;
      }
    }
  }

  /**
   * Initializes each fileName with random values or 0s
   *
   * @param fileNames
   * @param initRand
   * @throws IOException
   */
  private static void initializeMatrix(String[][] fileNames, boolean initRand)
          throws IOException {
    for (int i = 0; i < MSIZE; ++i) {
      for (int j = 0; j < MSIZE; ++j) {
        MatmulImpl.initializeBlock(fileNames[i][j], BSIZE, initRand);
      }
    }
  }

  /**
   * Main loop of matrix multiplication
   *
   */
  private static void computeMultiplication() {
    System.out.println("[LOG] Computing result");
    Integer[][][] exitValues = new Integer[MSIZE][MSIZE][MSIZE];

    // Launch tasks
    for (int i = 0; i < MSIZE; ++i) {
      for (int k = 0; k < MSIZE; ++k) {
        for (int j = 0; j < MSIZE; ++j) {
          switch (TYPE) {
            case 1:
              exitValues[i][j][k] = MatmulImpl
                      .multiplyAccumulativeNative(BSIZE,
                              AfileNames[i][k], BfileNames[k][j],
                              CfileNames[i][j]);
              break;
            case 2:
              exitValues[i][j][k] = MPI.multiplyAccumulativeMPI(
                      BSIZE, AfileNames[i][k], BfileNames[k][j],
                      CfileNames[i][j]);
              break;
            default:
              System.err.println("[ERROR] Invalid type");
              System.exit(1);
              break;
          }
        }
      }
    }
```

```
127     // Sync: Wait loop
        for (int i = 0; i < MSIZE; i++) {
129       for (int j = 0; j < MSIZE; j++) {
            for (int k = 0; k < MSIZE; k++) {
131           if (exitValues[i][j][k] != 0) {
                System.err.println(
133                     "[ERROR] Some task failed with exitValue "
                                + exitValues[i][j][k]);
135           }
            }
137       }
        }
139   }

141 }
```

B.2 MatmulItf.java

```
1  package matmul.files;

3  import integratedtoolkit.types.annotations.Constraints;
   import integratedtoolkit.types.annotations.Parameter;
5  import integratedtoolkit.types.annotations.parameter.Direction;
   import integratedtoolkit.types.annotations.parameter.Type;
7  import integratedtoolkit.types.annotations.task.MPI;
   import integratedtoolkit.types.annotations.task.Method;
9

11 public interface MatmulItf {

13     @Method(declaringClass = "matmul.files.MatmulImpl")
       @Constraints(computingUnits = "1")
15     void initializeBlock(
           @Parameter(type = Type.FILE, direction = Direction.OUT) String filename,
17         @Parameter() int BSIZE,
           @Parameter() boolean initRand
19     );

21     @Method(declaringClass = "matmul.files.MatmulImpl")
       @Constraints(computingUnits = "1")
23     Integer multiplyAccumulativeNative(
           @Parameter() int bsize,
25         @Parameter(type = Type.FILE, direction = Direction.IN) String aIn,
           @Parameter(type = Type.FILE, direction = Direction.IN) String bIn,
27         @Parameter(type = Type.FILE, direction = Direction.INOUT) String cOut
       );
29
       @MPI(binary = "${MATMUL_BINARY}",
31          mpiRunner = "mpirun",
            computingNodes = "${NODES_PER_MPI_TASK}")
33     @Constraints(computingUnits = "${CUS_PER_MPI_TASK}")
       Integer multiplyAccumulativeMPI(
35        @Parameter() int bsize,
          @Parameter(type = Type.FILE, direction = Direction.IN) String aIn,
37        @Parameter(type = Type.FILE, direction = Direction.IN) String bIn,
          @Parameter(type = Type.FILE, direction = Direction.INOUT) String cOut
39     );

41 }
```

B.3 MatmulImpl.java

```
package matmul.files;

import java.io.FileOutputStream;
import java.io.IOException;

/**
 * Implementation of Task methods
 *
 */
public class MatmulImpl {

  private static final byte[] NEW_LINE = "\n".getBytes();

  /**
   * Initializes a block of size @BSIZE and stores it to @filename
   *
   * @param filename
   * @param BSIZE
   * @param initRand
   * @throws IOException
   */
  public static void initializeBlock(String filename, int BSIZE,
          boolean initRand) throws IOException {
    try (FileOutputStream fos = new FileOutputStream(filename)) {
      for (int iblock = 0; iblock < BSIZE; ++iblock) {
        for (int jblock = 0; jblock < BSIZE; ++jblock) {
          double value = (double) 0.0;
          if (initRand) {
            value = (double) (Math.random() * 10.0);
          }
          fos.write(String.valueOf(value).getBytes());
          fos.write(" ".getBytes());
        }
        fos.write(NEW_LINE);
      }
      fos.write(NEW_LINE);
    } catch (IOException e) {
      throw new IOException("[ERROR] Error initializing matrix", e);
    }
  }

  /**
   * Multiplies blocks @aFile and @bFile, accumulates result on block @cFile
   * and stores it to disk
   *
   * @param BSIZE
   * @param aFile
   * @param bFile
   * @param cFile
   * @return
   */
  public static Integer multiplyAccumulativeNative(int BSIZE, String aFile,
          String bFile, String cFile) {
    Block a = new Block(BSIZE, aFile);
    Block b = new Block(BSIZE, bFile);
    Block c = new Block(BSIZE, cFile);

    c.multiplyAccum(a, b);
    c.blockToDisk(cFile);

    return 0;
  }

}
```

B.4 MPI.java

```
package mpi;

/**
 * Dummy class for MPI tasks
 *
 */
public class MPI {

  /**
   * Dummy implementation to call the matmul MPI binary
   *
   * @param bSize
   * @param aIn
   * @param bIn
   * @param cOut
   * @return
   */
  public static Integer multiplyAccumulativeMPI(int bSize, String aIn,
          String bIn, String cOut) {
    return -1;
  }

}
```

B.5 Matmul.c

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>

#define MASTER 0               /* taskid of first task */
#define FROM_MASTER 1          /* setting a message type */
#define FROM_WORKER 2          /* setting a message type */


void fillMatrix(const char* fileName, int matrixSize, double* mat) {
    int i, j;

    FILE *file;
    file = fopen(fileName, "r");

    // printf("  - Open file %s with size %d.\n", fileName, matrixSize);
    for(i = 0; i < matrixSize; i++) {
        for(j = 0; j < matrixSize; j++) {
            if (!fscanf(file, "%lf", &mat[i*matrixSize + j])) {
                break;
            }
            // printf("%lf\n", mat[i*matrixSize + j]);
        }
    }
    fclose(file);
}

void storeMatrix(const char* fileName, int matrixSize, const double* mat) {
    int i, j;

    FILE *file;
    file = fopen(fileName, "w");

    // printf("  - Store matrix in file %s with size %d.\n", fileName,
    //   matrixSize);
    for(i = 0; i < matrixSize; i++) {
        for(j = 0; j < matrixSize; j++) {
```

```
            fprintf(file, "%lf ", mat[i*matrixSize + j]);
            // printf("%lf\n", mat[i*matrixSize + j]);
        }
        fprintf(file, "\n");
    }
    fclose(file);
}

int main (int argc, char *argv[]) {
    /**************************** Initialize ******************/
    if (argc != 5) {
        // printf("Incorrect usage: matmul <matrixSize> <Ain> <Bin> <Cout>. Quitting...\n")
    ;
        exit(1);
    }
    int matrixSize = atoi(argv[1]);     // Number of rows/columns in matrixes
    char* ain = argv[2];                // FileName of Ain
    char* bin = argv[3];                // FileName of Bin
    char* cout = argv[4];               // FileName of Cout

    double a[matrixSize*matrixSize];   // Matrix A to be multiplied
    double b[matrixSize*matrixSize];   // Matrix B to be multiplied
    double c[matrixSize*matrixSize];   // Result matrix C

    // Initialize MPI env
    int mpiProcs;                       // Number of MPI Nodes
    int taskid;                         // Task identifier
    MPI_Status status;                  // Status for MPI communications
    MPI_Request send_request;           // For async calls

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &taskid);
    MPI_Comm_size(MPI_COMM_WORLD, &mpiProcs);

    // Misc variables
    int i, j, k, dest, rows;

    /**************************** master task ************************/
    if (taskid == MASTER) {
        // printf("Matmul with %d MPI nodes.\n", mpiProcs);

        // Initialize arrays
        // printf("Initialize matrixes.\n");
        fillMatrix(ain, matrixSize, a);
        fillMatrix(bin, matrixSize, b);
        fillMatrix(cout, matrixSize, c);

        // Send matrix data to the worker tasks
        // printf("Send matrixes to workers.\n");
        int averow = matrixSize/mpiProcs;
        int extra = matrixSize%mpiProcs;
        int offset = 0;
        int mtype = FROM_MASTER;
        for (dest = 0; dest < mpiProcs; dest++) {
            rows = (dest < extra) ? averow+1 : averow;
            MPI_Isend(&offset, 1, MPI_INT, dest, mtype, MPI_COMM_WORLD,
      &send_request);
            MPI_Isend(&rows, 1, MPI_INT, dest, mtype, MPI_COMM_WORLD,
      &send_request);
            MPI_Isend(&a[offset*matrixSize], rows*matrixSize, MPI_DOUBLE, dest,
      mtype, MPI_COMM_WORLD, &send_request);
            MPI_Isend(&b, matrixSize*matrixSize, MPI_DOUBLE, dest, mtype,
      MPI_COMM_WORLD, &send_request);
            MPI_Isend(&c, rows*matrixSize, MPI_DOUBLE, dest, mtype,
      MPI_COMM_WORLD, &send_request);
            offset = offset + rows;
        }
    }

    /**************************** worker task ************/
    // Receive matrix
    // printf("Receive IN matrixes on process %d.\n", taskid);
```

```
109     int offset = 0;
        int mtype = FROM_MASTER;
111     MPI_Recv(&offset, 1, MPI_INT, MASTER, mtype, MPI_COMM_WORLD, &status);
        MPI_Recv(&rows, 1, MPI_INT, MASTER, mtype, MPI_COMM_WORLD, &status);
113     MPI_Recv(&a, rows*matrixSize, MPI_DOUBLE, MASTER, mtype, MPI_COMM_WORLD,
        &status);
115     MPI_Recv(&b, matrixSize*matrixSize, MPI_DOUBLE, MASTER, mtype,
        MPI_COMM_WORLD, &status);
117     MPI_Recv(&c[offset*matrixSize], rows*matrixSize, MPI_DOUBLE, MASTER, mtype,
        MPI_COMM_WORLD, &status);
119
        // Perform multiply accumulative
121     // printf("Perform multiply accumulative on process %d.\n", taskid);
        for (i = 0; i < rows; i++) {
123         for (k = 0; k < matrixSize; k++) {
                for (j = 0; j < matrixSize; j++) {
125                 c[i*matrixSize + j] = c[i*matrixSize + j]
      + a[i*matrixSize + k]*b[k*matrixSize + j];
127             }
            }
129     }

131     // Send back result to master
        // printf("Send result back to master on process %d.\n", taskid);
133     mtype = FROM_WORKER;
        MPI_Isend(&offset, 1, MPI_INT, MASTER, mtype, MPI_COMM_WORLD,
135     &send_request);
        MPI_Isend(&rows, 1, MPI_INT, MASTER, mtype, MPI_COMM_WORLD, &send_request);
137     MPI_Isend(&c, rows*matrixSize, MPI_DOUBLE, MASTER, mtype, MPI_COMM_WORLD,
        &send_request);
139


141     /*************************** master task ********************/
        if (taskid == MASTER) {
143         // Receive results from worker tasks
            // printf("Receive results.\n");
145         mtype = FROM_WORKER;
            for (i = 0; i < mpiProcs; i++) {
147             MPI_Recv(&offset, 1, MPI_INT, i, mtype, MPI_COMM_WORLD, &status);
                MPI_Recv(&rows, 1, MPI_INT, i, mtype, MPI_COMM_WORLD, &status);
149             MPI_Recv(&c[offset*matrixSize], rows*matrixSize, MPI_DOUBLE, i,
      mtype, MPI_COMM_WORLD, &status);
151             // printf("  - Received results from task %d\n", i);
            }
153
            // Print result
155         //printMatrix(matrixSize, c);

157         // Store result to file
            // printf("Store result matrix.\n");
159         storeMatrix(cout, matrixSize, c);

161         printf ("Done.\n");
        }
163
        /*************************** Finalize ***********************************/
165     MPI_Finalize();
    }
```

B.6 Block.java

```
    package matmul.files;
2
    import java.io.BufferedReader;
4   import java.io.FileNotFoundException;
    import java.io.FileOutputStream;
6   import java.io.FileReader;
    import java.io.IOException;
```

```
8  import java.util.StringTokenizer;
10 public class Block {
12   private static final byte[] NEW_LINE = "\n".getBytes();
14   private final int bSize;
     private final double[][] data;
16
     /**
18    * Initialization of a block of size @bSize from file @filename
      *
20    * @param bSize
      * @param filename
22    */
     public Block(int bSize, String filename) {
24     this.bSize = bSize;
       this.data = new double[this.bSize][this.bSize];
26
       // Retrieve values from file
28     try (BufferedReader br = new BufferedReader(new FileReader(filename))) {
         StringTokenizer tokens;
30       String nextLine;

32       for (int i = 0; i < this.bSize; i++) {
           nextLine = br.readLine();
34         tokens = new StringTokenizer(nextLine);
           for (int j = 0; j < bSize && tokens.hasMoreTokens(); j++) {
36           this.data[i][j] = Double.parseDouble(tokens.nextToken());
           }
38       }
       } catch (FileNotFoundException fnfe) {
40       fnfe.printStackTrace();
         System.exit(-1);
42     } catch (IOException ioe) {
         ioe.printStackTrace();
44       System.exit(-1);
       }
46
     }
48
     /**
50    * Dumps the content of the block to file @filename
      *
52    * @param filename
      */
54   public void blockToDisk(String filename) {
       try (FileOutputStream fos = new FileOutputStream(filename)) {
56       for (int i = 0; i < this.bSize; i++) {
           for (int j = 0; j < this.bSize; j++) {
58           String str = String.valueOf(this.data[i][j]) + " ";
             fos.write(str.getBytes());
60         }
           fos.write(NEW_LINE);
62       }
       } catch (FileNotFoundException fnfe) {
64       fnfe.printStackTrace();
         System.exit(-1);
66     } catch (IOException ioe) {
         ioe.printStackTrace();
68       System.exit(-1);
       }
70   }

72   /**
      * Multiplicates blocks @a and @b and accumulates result on the current
74    * block
      *
76    * @param a
      * @param b
78    */
     public void multiplyAccum(Block a, Block b) {
```

```
80      for (int i = 0; i < this.bSize; ++i) {
          for (int k = 0; k < this.bSize; ++k) {
82          for (int j = 0; j < this.bSize; ++j) {
              this.data[i][j] += a.data[i][k] * b.data[k][j];
84          }
          }
86      }
    }
88
}
```

Appendix C

NMMB/BSC-Dust: code highlights

C.1 Nmmb.java

```
1  package nmmb;
2
3  import java.io.File;
4  import java.util.Date;
5
6  import org.apache.commons.configuration.ConfigurationException;
7  import org.apache.logging.log4j.LogManager;
8  import org.apache.logging.log4j.Logger;
9
10 import binary.BINARY;
11 import mpi.MPI;
12 import nmmb.configuration.NMMBConfigManager;
13 import nmmb.configuration.NMMBConstants;
14 import nmmb.configuration.NMMBEnvironment;
15 import nmmb.configuration.NMMBParameters;
16 import nmmb.exceptions.MainExecutionException;
17 import nmmb.exceptions.TaskExecutionException;
18 import nmmb.loggers.LoggerNames;
19 import nmmb.utils.FileManagement;
20 import nmmb.utils.FortranWrapper;
21 import nmmb.utils.MessagePrinter;
22
23 public class Nmmb {
24
25   // Loggers
26   private static final Logger LOGGER_MAIN = LogManager.getLogger(LoggerNames.NMMB_MAIN);
27   private static final Logger LOGGER_FIXED = LogManager
28           .getLogger(LoggerNames.NMMB_FIXED);
29   private static final Logger LOGGER_VARIABLE = LogManager
30           .getLogger(LoggerNames.NMMB_VARIABLE);
31   private static final Logger LOGGER_UMO_MODEL = LogManager
32           .getLogger(LoggerNames.NMMB_UMO_MODEL);
33   private static final Logger LOGGER_POST = LogManager.getLogger(LoggerNames.NMMB_POST);
34
35   /**
36    * Prints the usage
37    *
38    */
39   private static void usage() {
40     LOGGER_MAIN.info("Invalid parameters for nmmb.Nmmb");
41     LOGGER_MAIN.info("    Usage: nmmb.Nmmb <configFilePath>");
42   }
43
44   /*
45    * FIXED STEP
46    */
47   private static void doFixed(NMMBParameters nmmbParams) throws TaskExecutionException {
48     LOGGER_FIXED.info("Enter fixed process");
49
50     // Prepare execution
51     nmmbParams.prepareFixedExecution();
52     MessagePrinter fixedMP = new MessagePrinter(LOGGER_FIXED);
53
```

```
    // Build the fortran executables
    if (nmmbParams.isCompileBinaries()) {
      Integer[] compilationEvs = new Integer[FortranWrapper.FIXED_FORTRAN_F90_FILES.length
              + FortranWrapper.FIXED_FORTRAN_F_FILES.length];
      int i = 0;
      fixedMP.printInfoMsg("Building fixed executables");
      for (String fortranFile : FortranWrapper.FIXED_FORTRAN_F90_FILES) {
        String executable = NMMBEnvironment.FIX + fortranFile
                + FortranWrapper.SUFFIX_EXE;
        String src = NMMBEnvironment.FIX + fortranFile
                + FortranWrapper.SUFFIX_F90_SRC;

        compilationEvs[i++] = BINARY.fortranCompiler(FortranWrapper.MC_FLAG,
                FortranWrapper.SHARED_FLAG, FortranWrapper.CONVERT_PREFIX,
                FortranWrapper.CONVERT_VALUE, FortranWrapper.TRACEBACK_FLAG,
                FortranWrapper.ASSUME_PREFIX, FortranWrapper.ASSUME_VALUE,
                FortranWrapper.OPT_FLAG, FortranWrapper.FPMODEL_PREFIX,
                FortranWrapper.FPMODEL_VALUE, FortranWrapper.STACK_FLAG,
                FortranWrapper.OFLAG, executable, src);
      }
      for (String fortranFile : FortranWrapper.FIXED_FORTRAN_F_FILES) {
        String executable = NMMBEnvironment.FIX + fortranFile
                + FortranWrapper.SUFFIX_EXE;
        String src = NMMBEnvironment.FIX + fortranFile
                + FortranWrapper.SUFFIX_F_SRC;
        compilationEvs[i++] = BINARY.fortranCompiler(FortranWrapper.MC_FLAG,
                FortranWrapper.SHARED_FLAG, FortranWrapper.CONVERT_PREFIX,
                FortranWrapper.CONVERT_VALUE, FortranWrapper.TRACEBACK_FLAG,
                FortranWrapper.ASSUME_PREFIX, FortranWrapper.ASSUME_VALUE,
                FortranWrapper.OPT_FLAG, FortranWrapper.FPMODEL_PREFIX,
                FortranWrapper.FPMODEL_VALUE, FortranWrapper.STACK_FLAG,
                FortranWrapper.OFLAG, executable, src);
      }
      // Sync master to wait for compilation
      for (i = 0; i < compilationEvs.length; ++i) {
        LOGGER_FIXED.debug("Compilation of " + i + " binary ended with status "
                + compilationEvs[i]);
        if (compilationEvs[i] != 0) {
          throw new TaskExecutionException(
                  "[ERROR] Error compiling binary " + i);
        }
      }
      fixedMP.printInfoMsg("Finished building fixed executables");
    }

    // Begin binary calls
    fixedMP.printHeaderMsg("BEGIN");

    final int NUM_BINARIES = 16;
    Integer[] fixedBinariesEvs = new Integer[NUM_BINARIES];
    int i = 0;

    fixedMP.printInfoMsg("Generate DEM height and sea mask files");
    String topoDir = NMMBEnvironment.GEODATA_DIR + "topo1kmDEM" + File.separator;
    String seamaskDEM = NMMBEnvironment.OUTPUT + "seamaskDEM";
    String heightDEM = NMMBEnvironment.OUTPUT + "heightDEM";
    fixedBinariesEvs[i++] = BINARY.smmount(topoDir, seamaskDEM, heightDEM);

    fixedMP.printInfoMsg("Generate landuse file");
    String landuseDataDir = NMMBEnvironment.GEODATA_DIR + "landuse_30s"
            + File.separator;
    String landuse = NMMBEnvironment.OUTPUT + "landuse";
    String kount_landuse = NMMBEnvironment.OUTPUT + "kount_landuse";
    fixedBinariesEvs[i++] = BINARY.landuse(landuseDataDir, landuse, kount_landuse);

    fixedMP.printInfoMsg("Generate landusenew file");
    String landusenew = NMMBEnvironment.OUTPUT + "landusenew";
    String kount_landusenew = NMMBEnvironment.OUTPUT + "kount_landusenew";
    fixedBinariesEvs[i++] = BINARY.landusenew(NMMBEnvironment.GTOPO_DIR, landusenew,
            kount_landusenew);

    fixedMP.printInfoMsg("Generate mountains");
```

```
        String topo30sDir = NMMBEnvironment.GEODATA_DIR + "topo_30s" + File.separator;
127     String heightmean = NMMBEnvironment.OUTPUT + "heightmean";
        fixedBinariesEvs[i++] = BINARY.topo(topo30sDir, heightmean);
129
        fixedMP.printInfoMsg("Generate standard deviation of topography height");
131     String stdh = NMMBEnvironment.OUTPUT + "stdh";
        fixedBinariesEvs[i++] = BINARY.stdh(heightmean, seamaskDEM, topo30sDir, stdh);
133
        fixedMP.printInfoMsg("Generate envelope mountains");
135     String height = NMMBEnvironment.OUTPUT + "height";
        fixedBinariesEvs[i++] = BINARY.envelope(heightmean, stdh, height);
137
        fixedMP.printInfoMsg("Generate top soil type file");
139     String soiltypeDir = NMMBEnvironment.GEODATA_DIR + "soiltype_top_30s"
                + File.separator;
141     String topsoiltype = NMMBEnvironment.OUTPUT + "topsoiltype";
        fixedBinariesEvs[i++] = BINARY.topsoiltype(seamaskDEM, soiltypeDir, topsoiltype);
143
        fixedMP.printInfoMsg("Generate bottom soil type file");
145     String soiltypePath = NMMBEnvironment.GEODATA_DIR + "soiltype_bot_30s"
                + File.separator;
147     String botsoiltype = NMMBEnvironment.OUTPUT + "botsoiltype";
        fixedBinariesEvs[i++] = BINARY.botsoiltype(seamaskDEM, soiltypePath, botsoiltype);
149
        fixedMP.printInfoMsg("Generate sea mask and reprocess mountains");
151     String seamask = NMMBEnvironment.OUTPUT + "seamask";
        fixedBinariesEvs[i++] = BINARY.toposeamask(seamaskDEM, seamask, height, landuse,
153             topsoiltype, botsoiltype);

155     fixedMP.printInfoMsg("Reprocess standard deviation of topography height");
        fixedBinariesEvs[i++] = BINARY.stdhtopo(seamask, stdh);
157
        fixedMP.printInfoMsg("Generate deep soil temperature");
159     String soiltempPath = NMMBEnvironment.GEODATA_DIR + "soiltemp_1deg"
                + File.separator;
161     String deeptemperature = NMMBEnvironment.OUTPUT + "deeptemperature";
        fixedBinariesEvs[i++] = BINARY.deeptemperature(seamask, soiltempPath,
163             deeptemperature);

165     fixedMP.printInfoMsg("Generate maximum snow albedo");
        String maxsnowalbDir = NMMBEnvironment.GEODATA_DIR + "maxsnowalb"
167             + File.separator;
        String snowalbedo = NMMBEnvironment.OUTPUT + "snowalbedo";
169     fixedBinariesEvs[i++] = BINARY.snowalbedo(maxsnowalbDir, snowalbedo);

171     fixedMP.printInfoMsg("Generate vertical coordinate");
        String dsg = NMMBEnvironment.OUTPUT + "dsg";
173     fixedBinariesEvs[i++] = BINARY.vcgenerator(dsg);

175     fixedMP.printInfoMsg("Generate highres roughness length for africa and asia");
        String roughnessDir = NMMBEnvironment.GEODATA_DIR + "roughness_025s"
177             + File.separator;
        String roughness = NMMBEnvironment.OUTPUT + "roughness";
179     fixedBinariesEvs[i++] = BINARY.roughness(roughnessDir, roughness);

181     fixedMP.printInfoMsg("Generate co2 files");
        String co2_data_dir = NMMBEnvironment.GEODATA_DIR + "co2data" + File.separator;
183     String co2_trans = NMMBEnvironment.OUTPUT + "co2_trans";
        fixedBinariesEvs[i++] = BINARY.gfdlco2(dsg, co2_data_dir, co2_trans);
185
        fixedMP.printInfoMsg(
187             "Generate lookup tables for aerosol scavenging collection efficiencies");
        String lookup_aerosol2_rh00 = NMMBEnvironment.OUTPUT + "lookup_aerosol2.dat.rh00";
189     String lookup_aerosol2_rh50 = NMMBEnvironment.OUTPUT + "lookup_aerosol2.dat.rh50";
        String lookup_aerosol2_rh70 = NMMBEnvironment.OUTPUT + "lookup_aerosol2.dat.rh70";
191     String lookup_aerosol2_rh80 = NMMBEnvironment.OUTPUT + "lookup_aerosol2.dat.rh80";
        String lookup_aerosol2_rh90 = NMMBEnvironment.OUTPUT + "lookup_aerosol2.dat.rh90";
193     String lookup_aerosol2_rh95 = NMMBEnvironment.OUTPUT + "lookup_aerosol2.dat.rh95";
        String lookup_aerosol2_rh99 = NMMBEnvironment.OUTPUT + "lookup_aerosol2.dat.rh99";
195     fixedBinariesEvs[i++] = BINARY.run_aerosol(nmmbParams.isCompileBinaries(),
                nmmbParams.isCleanBinaries(), lookup_aerosol2_rh00, lookup_aerosol2_rh50,
197             lookup_aerosol2_rh70, lookup_aerosol2_rh80, lookup_aerosol2_rh90,
```

```
            lookup_aerosol2_rh95, lookup_aerosol2_rh99);

    // Wait for binaries completion and check exit value
    for (i = 0; i < fixedBinariesEvs.length; ++i) {
      LOGGER_FIXED.debug("Execution of " + i + " binary ended with status "
              + fixedBinariesEvs[i]);
      if (fixedBinariesEvs[i] != 0) {
        throw new TaskExecutionException("[ERROR] Error executing binary " + i);
      }
    }

    // Clean Up binaries
    if (nmmbParams.isCleanBinaries()) {
      fixedMP.printInfoMsg("Clean up executables");
      for (String fortranFile : FortranWrapper.FIXED_FORTRAN_F90_FILES) {
        String executable = NMMBEnvironment.FIX + fortranFile
                + FortranWrapper.SUFFIX_EXE;
        File f = new File(executable);
        if (f.exists()) {
          f.delete();
        }
      }
      for (String fortranFile : FortranWrapper.FIXED_FORTRAN_F_FILES) {
        String executable = NMMBEnvironment.FIX + fortranFile
                + FortranWrapper.SUFFIX_EXE;
        File f = new File(executable);
        if (f.exists()) {
          f.delete();
        }
      }
    }

    /* End ********************************************/
    fixedMP.printHeaderMsg("END");

    LOGGER_FIXED.info("Fixed process finished");
  }

  /*
   * VARIABLE STEP
   */
  private static void compileVariable() throws TaskExecutionException {
    // Build the fortran objects
    Integer[] depCompilationEvs = new Integer[FortranWrapper.VARIABLE_FORTRAN_F90_DEP_FILES
    .length];
    int objectIndex = 0;
    for (String fortranFile : FortranWrapper.VARIABLE_FORTRAN_F90_DEP_FILES) {
      String moduleDir = NMMBEnvironment.VRB;
      String object = NMMBEnvironment.VRB + fortranFile
              + FortranWrapper.SUFFIX_OBJECT;
      String src = NMMBEnvironment.VRB + fortranFile
              + FortranWrapper.SUFFIX_F90_SRC;

      depCompilationEvs[objectIndex++] = BINARY.fortranCompileObject(
              FortranWrapper.MC_FLAG, FortranWrapper.SHARED_FLAG,
              FortranWrapper.CONVERT_PREFIX, FortranWrapper.CONVERT_VALUE,
              FortranWrapper.TRACEBACK_FLAG, FortranWrapper.ASSUME_PREFIX,
              FortranWrapper.ASSUME_VALUE, FortranWrapper.OPT_FLAG,
              FortranWrapper.FPMODEL_PREFIX, FortranWrapper.FPMODEL_VALUE,
              FortranWrapper.STACK_FLAG, FortranWrapper.CFLAG, src,
              FortranWrapper.OFLAG, object, FortranWrapper.MODULE_FLAG, moduleDir);
    }
    // Sync to check compilation status (dependency with task object is
    // also respected if this sync is erased)
    for (int i = 0; i < depCompilationEvs.length; ++i) {
      LOGGER_VARIABLE.debug("Compilation of " + i
              + " dependant binary ended with status " + depCompilationEvs[i]);
      if (depCompilationEvs[i] != 0) {
        throw new TaskExecutionException("[ERROR] Error compiling binary " + i);
      }
    }
```

```
269     // Build the fortran executables
        Integer[] compilationEvs = new Integer[FortranWrapper.VARIABLE_FORTRAN_F90_FILES.length
271             + FortranWrapper.VARIABLE_FORTRAN_F_FILES.length
                + FortranWrapper.VARIABLE_GFORTRAN_F_FILES.length
273             + FortranWrapper.VARIABLE_FORTRAN_F_FILES_WITH_W3.length
                + FortranWrapper.VARIABLE_FORTRAN_F_FILES_WITH_DEPS.length + 1];
275
        int executableIndex = 0;
277     for (String fortranFile : FortranWrapper.VARIABLE_FORTRAN_F90_FILES) {
          String executable = NMMBEnvironment.VRB + fortranFile
279               + FortranWrapper.SUFFIX_EXE;
          String src = NMMBEnvironment.VRB + fortranFile
281               + FortranWrapper.SUFFIX_F90_SRC;

283       compilationEvs[executableIndex++] = BINARY.fortranCompiler(
                FortranWrapper.MC_FLAG, FortranWrapper.SHARED_FLAG,
285             FortranWrapper.CONVERT_PREFIX, FortranWrapper.CONVERT_VALUE,
                FortranWrapper.TRACEBACK_FLAG, FortranWrapper.ASSUME_PREFIX,
287             FortranWrapper.ASSUME_VALUE, FortranWrapper.OPT_FLAG,
                FortranWrapper.FPMODEL_PREFIX, FortranWrapper.FPMODEL_VALUE,
289             FortranWrapper.STACK_FLAG, FortranWrapper.OFLAG, executable, src);
        }
291     for (String fortranFile : FortranWrapper.VARIABLE_FORTRAN_F_FILES) {
          String executable = NMMBEnvironment.VRB + fortranFile
293               + FortranWrapper.SUFFIX_EXE;
          String src = NMMBEnvironment.VRB + fortranFile + FortranWrapper.SUFFIX_F_SRC;
295       compilationEvs[executableIndex++] = BINARY.fortranCompiler(
                FortranWrapper.MC_FLAG, FortranWrapper.SHARED_FLAG,
297             FortranWrapper.CONVERT_PREFIX, FortranWrapper.CONVERT_VALUE,
                FortranWrapper.TRACEBACK_FLAG, FortranWrapper.ASSUME_PREFIX,
299             FortranWrapper.ASSUME_VALUE, FortranWrapper.OPT_FLAG,
                FortranWrapper.FPMODEL_PREFIX, FortranWrapper.FPMODEL_VALUE,
301             FortranWrapper.STACK_FLAG, FortranWrapper.OFLAG, executable, src);
        }
303
        for (String fortranFile : FortranWrapper.VARIABLE_GFORTRAN_F_FILES) {
305       String executable = NMMBEnvironment.VRB + fortranFile
                + FortranWrapper.SUFFIX_EXE;
307       String src = NMMBEnvironment.VRB + fortranFile + FortranWrapper.SUFFIX_F_SRC;
          compilationEvs[executableIndex++] = BINARY.gfortranCompiler(
309             FortranWrapper.BIG_O_FLAG, src, FortranWrapper.OFLAG, executable);
        }
311
        for (String fortranFile : FortranWrapper.VARIABLE_FORTRAN_F_FILES_WITH_W3) {
313       String executable = NMMBEnvironment.VRB + fortranFile
                + FortranWrapper.SUFFIX_EXE;
315       String src = NMMBEnvironment.VRB + fortranFile + FortranWrapper.SUFFIX_F_SRC;
          String w3LibFlag = "-L" + NMMBEnvironment.UMO_LIBS
317               + FortranWrapper.W3_LIB_DIR;
          String bacioLibFlag = "-L" + NMMBEnvironment.UMO_LIBS
319               + FortranWrapper.BACIO_LIB_DIR;
          compilationEvs[executableIndex++] = BINARY.fortranCompilerWithW3(
321             FortranWrapper.MC_FLAG, FortranWrapper.SHARED_FLAG,
                FortranWrapper.CONVERT_PREFIX, FortranWrapper.CONVERT_VALUE,
323             FortranWrapper.TRACEBACK_FLAG, FortranWrapper.ASSUME_PREFIX,
                FortranWrapper.ASSUME_VALUE, FortranWrapper.OPT_FLAG,
325             FortranWrapper.FPMODEL_PREFIX, FortranWrapper.FPMODEL_VALUE,
                FortranWrapper.STACK_FLAG, FortranWrapper.OFLAG, executable, src,
327             w3LibFlag, bacioLibFlag, FortranWrapper.W3_FLAG,
                FortranWrapper.BACIO_FLAG);
329     }
        for (String fortranFile : FortranWrapper.VARIABLE_FORTRAN_F_FILES_WITH_DEPS) {
331       String executable = NMMBEnvironment.VRB + fortranFile
                + FortranWrapper.SUFFIX_EXE;
333       String src = NMMBEnvironment.VRB + fortranFile + FortranWrapper.SUFFIX_F_SRC;
          String object = NMMBEnvironment.VRB + FortranWrapper.MODULE_FLT
335               + FortranWrapper.SUFFIX_OBJECT;
          compilationEvs[executableIndex++] = BINARY.fortranCompileWithObject(
337             FortranWrapper.MC_FLAG, FortranWrapper.SHARED_FLAG,
                FortranWrapper.CONVERT_PREFIX, FortranWrapper.CONVERT_VALUE,
339             FortranWrapper.TRACEBACK_FLAG, FortranWrapper.ASSUME_PREFIX,
                FortranWrapper.ASSUME_VALUE, FortranWrapper.OPT_FLAG,
```

```
341             FortranWrapper.FPMODEL_PREFIX, FortranWrapper.FPMODEL_VALUE,
                FortranWrapper.STACK_FLAG, FortranWrapper.OFLAG, executable, src,
343             object);
    }
345   String source = NMMBEnvironment.VRB + FortranWrapper.READ_PAUL_SOURCE
            + FortranWrapper.SUFFIX_F_SRC;
347   String executable = NMMBEnvironment.VRB + FortranWrapper.READ_PAUL_SOURCE
            + FortranWrapper.SUFFIX_EXE;
349   compilationEvs[executableIndex++] = BINARY.compileReadPaulSource(source,
            executable);
351
    // Sync master to wait for compilation
353   for (int i = 0; i < compilationEvs.length; ++i) {
      LOGGER_VARIABLE.debug("Compilation of " + i + " binary ended with status "
355             + compilationEvs[i]);
      if (compilationEvs[i] != 0) {
357     throw new TaskExecutionException("[ERROR] Error compiling binary " + i);
      }
359   }
  }
361
  private static void cleanUpVariableExe() {
363   for (String fortranFile : FortranWrapper.VARIABLE_FORTRAN_F90_DEP_FILES) {
      String executable = NMMBEnvironment.VRB + fortranFile
365             + FortranWrapper.SUFFIX_EXE;
      File f = new File(executable);
367     if (f.exists()) {
        f.delete();
369     }
    }
371   for (String fortranFile : FortranWrapper.VARIABLE_FORTRAN_F90_FILES) {
      String executable = NMMBEnvironment.VRB + fortranFile
373             + FortranWrapper.SUFFIX_EXE;
      File f = new File(executable);
375     if (f.exists()) {
        f.delete();
377     }
    }
379   for (String fortranFile : FortranWrapper.VARIABLE_FORTRAN_F_FILES) {
      String executable = NMMBEnvironment.VRB + fortranFile
381             + FortranWrapper.SUFFIX_EXE;
      File f = new File(executable);
383     if (f.exists()) {
        f.delete();
385     }
    }
387   for (String fortranFile : FortranWrapper.VARIABLE_GFORTRAN_F_FILES) {
      String executable = NMMBEnvironment.VRB + fortranFile
389             + FortranWrapper.SUFFIX_EXE;
      File f = new File(executable);
391     if (f.exists()) {
        f.delete();
393     }
    }
395   for (String fortranFile : FortranWrapper.VARIABLE_FORTRAN_F_FILES_WITH_W3) {
      String executable = NMMBEnvironment.VRB + fortranFile
397             + FortranWrapper.SUFFIX_EXE;
      File f = new File(executable);
399     if (f.exists()) {
        f.delete();
401     }
    }
403   for (String fortranFile : FortranWrapper.VARIABLE_FORTRAN_F_FILES_WITH_DEPS) {
      String executable = NMMBEnvironment.VRB + fortranFile
405             + FortranWrapper.SUFFIX_EXE;
      File f = new File(executable);
407     if (f.exists()) {
        f.delete();
409     }
    }
411   String readPaulSource = NMMBEnvironment.VRB + FortranWrapper.READ_PAUL_SOURCE
            + FortranWrapper.SUFFIX_EXE;
```

```
413     File f = new File(readPaulSource);
        if (f.exists()) {
415       f.delete();
        }
417   }

419   private static void doVariable(NMMBParameters nmmbParams, Date currentDate)
            throws TaskExecutionException {
421     LOGGER_VARIABLE.info("Enter variable process");

423     // Prepare execution
        nmmbParams.prepareVariableExecution(currentDate);
425     MessagePrinter variableMP = new MessagePrinter(LOGGER_VARIABLE);

427     /* Compile ************************************************/
        if (nmmbParams.isCompileBinaries()) {
429       variableMP.printInfoMsg("Building variable executables");
          compileVariable();
431       variableMP.printInfoMsg("Finished building variable executables");
        }
433
        // Set variables for binary calls
435     variableMP.printHeaderMsg("BEGIN");

437     String CW = NMMBEnvironment.OUTPUT + "00_CW.dump";
        String ICEC = NMMBEnvironment.OUTPUT + "00_ICEC.dump";
439     String SH = NMMBEnvironment.OUTPUT + "00_SH.dump";
        String SOILT2 = NMMBEnvironment.OUTPUT + "00_SOILT2.dump";
441     String SOILT4 = NMMBEnvironment.OUTPUT + "00_SOILT4.dump";
        String SOILW2 = NMMBEnvironment.OUTPUT + "00_SOILW2.dump";
443     String SOILW4 = NMMBEnvironment.OUTPUT + "00_SOILW4.dump";
        String TT = NMMBEnvironment.OUTPUT + "00_TT.dump";
445     String VV = NMMBEnvironment.OUTPUT + "00_VV.dump";
        String HH = NMMBEnvironment.OUTPUT + "00_HH.dump";
447     String PRMSL = NMMBEnvironment.OUTPUT + "00_PRMSL.dump";
        String SOILT1 = NMMBEnvironment.OUTPUT + "00_SOILT1.dump";
449     String SOILT3 = NMMBEnvironment.OUTPUT + "00_SOILT3.dump";
        String SOILW1 = NMMBEnvironment.OUTPUT + "00_SOILW1.dump";
451     String SOILW3 = NMMBEnvironment.OUTPUT + "00_SOILW3.dump";
        String SST_TS = NMMBEnvironment.OUTPUT + "00_SST_TS.dump";
453     String UU = NMMBEnvironment.OUTPUT + "00_UU.dump";
        String WEASD = NMMBEnvironment.OUTPUT + "00_WEASD.dump";
455
        String GFS_file = NMMBEnvironment.OUTPUT + "131140000.gfs";
457
        String deco = NMMBEnvironment.VRB;
459
        String llspl000 = NMMBEnvironment.OUTPUT + "llspl.000";
461     String outtmp = NMMBEnvironment.OUTPUT + "llstmp";
        String outmst = NMMBEnvironment.OUTPUT + "llsmst";
463     String outsst = NMMBEnvironment.OUTPUT + "llgsst";
        String outsno = NMMBEnvironment.OUTPUT + "llgsno";
465     String outcic = NMMBEnvironment.OUTPUT + "llgcic";

467     String llgsst05 = NMMBEnvironment.OUTPUT + "llgsst05";
        String sstfileinPath = NMMBEnvironment.OUTPUT + "sst2dvar_grb_0.5";
469
        String seamask = NMMBEnvironment.OUTPUT + "seamask";
471     String albedo = NMMBEnvironment.OUTPUT + "albedo";
        String albedobase = NMMBEnvironment.OUTPUT + "albedobase";
473     String albedomnth = NMMBEnvironment.GEODATA_DIR + "albedo" + File.separator
              + "albedomnth";
475
        String albedorrtm = NMMBEnvironment.OUTPUT + "albedorrtm";
477     String albedorrtm1degDir = NMMBEnvironment.GEODATA_DIR + "albedo_rrtm1deg"
              + File.separator;
479
        String vegfrac = NMMBEnvironment.OUTPUT + "vegfrac";
481     String vegfracmnth = NMMBEnvironment.GEODATA_DIR + "vegfrac" + File.separator
              + "vegfracmnth";
483
        String landuse = NMMBEnvironment.OUTPUT + "landuse";
```

```
485     String topsoiltype = NMMBEnvironment.OUTPUT + "topsoiltype";
        String height = NMMBEnvironment.OUTPUT + "height";
487     String stdh = NMMBEnvironment.OUTPUT + "stdh";
        String z0base = NMMBEnvironment.OUTPUT + "z0base";
489     String z0 = NMMBEnvironment.OUTPUT + "z0";
        String ustar = NMMBEnvironment.OUTPUT + "ustar";
491
        String sst05 = NMMBEnvironment.OUTPUT + "sst05";
493     String deeptemperature = NMMBEnvironment.OUTPUT + "deeptemperature";
        String snowalbedo = NMMBEnvironment.OUTPUT + "snowalbedo";
495     String landusenew = NMMBEnvironment.OUTPUT + "landusenew";
        String llgsst = NMMBEnvironment.OUTPUT + "llgsst";
497     String llgsno = NMMBEnvironment.OUTPUT + "llgsno";
        String llgcic = NMMBEnvironment.OUTPUT + "llgcic";
499     String llsmst = NMMBEnvironment.OUTPUT + "llsmst";
        String llstmp = NMMBEnvironment.OUTPUT + "llstmp";
501     String albedorrtmcorr = NMMBEnvironment.OUTPUT + "albedorrtmcorr";
        String dzsoil = NMMBEnvironment.OUTPUT + "dzsoil";
503     String tskin = NMMBEnvironment.OUTPUT + "tskin";
        String sst = NMMBEnvironment.OUTPUT + "sst";
505     String snow = NMMBEnvironment.OUTPUT + "snow";
        String snowheight = NMMBEnvironment.OUTPUT + "snowheight";
507     String cice = NMMBEnvironment.OUTPUT + "cice";
        String seamaskcorr = NMMBEnvironment.OUTPUT + "seamaskcorr";
509     String landusecorr = NMMBEnvironment.OUTPUT + "landusecorr";
        String landusenewcorr = NMMBEnvironment.OUTPUT + "landusenewcorr";
511     String topsoiltypecorr = NMMBEnvironment.OUTPUT + "topsoiltypecorr";
        String vegfraccorr = NMMBEnvironment.OUTPUT + "vegfraccorr";
513     String z0corr = NMMBEnvironment.OUTPUT + "z0corr";
        String z0basecorr = NMMBEnvironment.OUTPUT + "z0basecorr";
515     String emissivity = NMMBEnvironment.OUTPUT + "emissivity";
        String canopywater = NMMBEnvironment.OUTPUT + "canopywater";
517     String frozenprecratio = NMMBEnvironment.OUTPUT + "frozenprecratio";
        String smst = NMMBEnvironment.OUTPUT + "smst";
519     String sh2o = NMMBEnvironment.OUTPUT + "sh2o";
        String stmp = NMMBEnvironment.OUTPUT + "stmp";
521     String dsg = NMMBEnvironment.OUTPUT + "dsg";
        String fcst = NMMBEnvironment.OUTPUT + "fcst";
523     String fcstDir = NMMBEnvironment.OUTPUT + "fcst";
        String bocoPrefix = NMMBEnvironment.OUTPUT + "boco.";
525     String llsplPrefix = NMMBEnvironment.OUTPUT + "llspl.";

527     String source = NMMBEnvironment.OUTPUT + "source";
        String sourceNETCDF = NMMBEnvironment.OUTPUT + "source.nc";
529     String sourceNCIncludeDir = NMMBEnvironment.VRB_INCLUDE_DIR;

531     String soildust = NMMBEnvironment.OUTPUT + "soildust";
        String kount_landuse = NMMBEnvironment.OUTPUT + "kount_landuse";
533     String kount_landusenew = NMMBEnvironment.OUTPUT + "kount_landusenew";
        String roughness = NMMBEnvironment.OUTPUT + "roughness";
535
        // Begin binary calls
537     final int NUM_BINARIES = 12;
        Integer[] variableBinariesEvs = new Integer[NUM_BINARIES];
539     int binaryIndex = 0;

541     variableMP.printInfoMsg("degrib gfs global data");
        variableBinariesEvs[binaryIndex++] = BINARY.degribgfs_generic_05(CW, ICEC, SH,
543             SOILT2, SOILT4, SOILW2, SOILW4, TT, VV, HH, PRMSL, SOILT1, SOILT3, SOILW1,
                SOILW3, SST_TS, UU, WEASD);
545
        variableMP.printInfoMsg("GFS 2 Model");
547     variableBinariesEvs[binaryIndex++] = BINARY.gfs2model_rrtm(CW, ICEC, SH, SOILT2,
                SOILT4, SOILW2, SOILW4, TT, VV, HH, PRMSL, SOILT1, SOILT3, SOILW1, SOILW3,
549             SST_TS, UU, WEASD, GFS_file);

551     variableMP.printInfoMsg("INC RRTM");
        variableBinariesEvs[binaryIndex++] = BINARY.inc_rrtm(GFS_file, deco);
553
        variableMP.printInfoMsg("CNV RRTM");
555     variableBinariesEvs[binaryIndex++] = BINARY.cnv_rrtm(GFS_file, llspl000, outtmp,
                outmst, outsst, outsno, outcic);
```

```
557
    variableMP.printInfoMsg("Degrib 0.5 deg sst");
559     variableBinariesEvs[binaryIndex++] = BINARY.degribsst(llgsst05, sstfileinPath);

561     variableMP.printInfoMsg("Prepare climatological albedo");
        variableBinariesEvs[binaryIndex++] = BINARY.albedo(llspl000, seamask, albedo,
563             albedobase, albedomnth);

565     variableMP.printInfoMsg("Prepare rrtm climatological albedos");
        variableBinariesEvs[binaryIndex++] = BINARY.albedorrtm(llspl000, seamask,
567             albedorrtm, albedorrtm1degDir);

569     variableMP.printInfoMsg("Prepare climatological vegetation fraction");
        variableBinariesEvs[binaryIndex++] = BINARY.vegfrac(llspl000, seamask, vegfrac,
571             vegfracmnth);

573     variableMP.printInfoMsg("Prepare z0 and initial ustar");
        variableBinariesEvs[binaryIndex++] = BINARY.z0vegfrac(seamask, landuse,
575             topsoiltype, height, stdh, vegfrac, z0base, z0, ustar);

577     variableMP.printInfoMsg("Interpolate to model grid and execute allprep (fcst)");
        variableBinariesEvs[binaryIndex++] = BINARY.allprep(llspl000, llgsst05, sst05,
579             height, seamask, stdh, deeptemperature, snowalbedo, z0, z0base, landuse,
            landusenew, topsoiltype, vegfrac, albedorrtm, llgsst, llgsno, llgcic,
581         llsmst, llstmp, albedorrtmcorr, dzsoil, tskin, sst, snow, snowheight,
            cice, seamaskcorr, landusecorr, landusenewcorr, topsoiltypecorr,
583         vegfraccorr, z0corr, z0basecorr, emissivity, canopywater, frozenprecratio,
            smst, sh2o, stmp, dsg, fcst, albedo, ustar, fcstDir, bocoPrefix,
585         llsplPrefix);

587     variableMP.printInfoMsg("Prepare the dust related variable (soildust)");
        variableBinariesEvs[binaryIndex++] = BINARY.readpaulsource(seamask, source,
589             sourceNETCDF, sourceNCIncludeDir);

591     variableMP.printInfoMsg("Dust Start");
        variableBinariesEvs[binaryIndex++] = BINARY.dust_start(llspl000, soildust, snow,
593             topsoiltypecorr, landusecorr, landusenewcorr, kount_landuse,
            kount_landusenew, vegfrac, height, seamask, source, z0corr, roughness);
595
    // Wait for binaries completion and check exit value
597     for (int i = 0; i < variableBinariesEvs.length; ++i) {
      LOGGER_VARIABLE.debug("Execution of " + i + " binary ended with status "
599             + variableBinariesEvs[i]);
      if (variableBinariesEvs[i] != 0) {
601       throw new TaskExecutionException("[ERROR] Error executing binary " + i);
      }
603   }

605   variableMP.printHeaderMsg("END");

607   // Clean Up binaries
    if (nmmbParams.isCleanBinaries()) {
609     variableMP.printInfoMsg("Clean up executables");
      cleanUpVariableExe();
611   }

613   // Post execution
    String folderOutputCase = NMMBEnvironment.OUTNMMB + nmmbParams.getCase()
615             + File.separator;
    nmmbParams.postVariableExecution(folderOutputCase);
617
    LOGGER_VARIABLE.info("Variable process finished");
619 }

621 private static void copyFilesFromPreprocess(NMMBParameters nmmbParams)
        throws TaskExecutionException {
623   // Clean specific files
    final String[] outputFiles = new String[] { "isop.dat", "meteo-data.dat",
625             "chemic-reg", "main_input_filename", "main_input_filename2", "GWD.bin",
            "configure_file", "co2_trans", "ETAMPNEW_AERO", "ETAMPNEW_DATA" };
627   for (String file : outputFiles) {
      String filePath = NMMBEnvironment.UMO_OUT + file;
```

```
629       if (!FileManagement.deleteFile(filePath)) {
            LOGGER_UMO_MODEL.debug(
631                 "Cannot erase previous " + file + " because it doesn't exist.");
          }
633       }

635       // Clean regular expr files
          File folder = new File(NMMBEnvironment.UMO_OUT);
637       for (File file : folder.listFiles()) {
            if ((file.getName().startsWith("lai") && file.getName().endsWith(".dat"))
639                 || (file.getName().startsWith("pftp_")
                          && file.getName().endsWith(".dat"))
641                 || (file.getName().startsWith("PET")
                          && file.getName().endsWith("txt"))
643                 || (file.getName().startsWith("PET")
                          && file.getName().endsWith("File"))
645                 || (file.getName().startsWith("boco."))
                    || (file.getName().startsWith("boco_chem."))
647                 || (file.getName().startsWith("nmm_b_history."))
                    || (file.getName().startsWith("tr"))
649                 || (file.getName().startsWith("RRT"))
                    || (file.getName().endsWith(".TBL"))
651                 || (file.getName().startsWith("fcstdone."))
                    || (file.getName().startsWith("restartdone."))
653                 || (file.getName().startsWith("nmmb_rst_"))
                    || (file.getName().startsWith("nmmb_hst_"))) {
655
              if (!FileManagement.deleteFile(file)) {
657             LOGGER_UMO_MODEL.debug("Cannot erase previous " + file.getName()
                        + " because it doesn't exist.");
659           }

661       }
          }
663
          // Prepare UMO model files
665       String outputFolderPath = NMMBEnvironment.OUTPUT;
          File outputFolder = new File(outputFolderPath);
667       for (File file : outputFolder.listFiles()) {
            if (file.getName().startsWith("boco.")
669                 || file.getName().startsWith("boco_chem.")) {
              // Copy file
671           String targetPath = NMMBEnvironment.UMO_OUT + file.getName();
              if (!FileManagement.copyFile(file.getAbsolutePath(), targetPath)) {
673             throw new TaskExecutionException("[ERROR] Error copying file from "
                        + file.getName() + " to " + targetPath);
675           }
            }
677       }

679       String chemicRegSrc = NMMBEnvironment.OUTPUT + "chemic-reg";
          String chemicRegTarget = NMMBEnvironment.UMO_OUT + "chemic-reg";
681       if (!FileManagement.copyFile(chemicRegSrc, chemicRegTarget)) {
            // TODO: Really no error when file does not exist?
683         LOGGER_UMO_MODEL.debug("Cannot copy file from " + chemicRegSrc + " to "
                    + chemicRegTarget + ". Skipping...");
685       }

687       String gwdSrc = NMMBEnvironment.OUTPUT + "GWD.bin";
          String gwdTarget = NMMBEnvironment.UMO_OUT + "GWD.bin";
689       if (!FileManagement.copyFile(gwdSrc, gwdTarget)) {
            // TODO: Really no error when file does not exist?
691         LOGGER_UMO_MODEL.debug("Cannot copy file from " + gwdSrc + " to " + gwdTarget
                    + ". Skipping...");
693       }

695       String inputDomain1Src = NMMBEnvironment.OUTPUT + "fcst";
          String inputDomain1Target = NMMBEnvironment.UMO_OUT + "input_domain_01";
697       if (!FileManagement.copyFile(inputDomain1Src, inputDomain1Target)) {
            throw new TaskExecutionException("[ERROR] Error copying file from "
699                 + inputDomain1Src + " to " + inputDomain1Target);
          }
```

```
701
    String inputDomain2Src = NMMBEnvironment.OUTPUT + "soildust";
703 String inputDomain2Target = NMMBEnvironment.UMO_OUT + "main_input_filename2";
    if (!FileManagement.copyFile(inputDomain2Src, inputDomain2Target)) {
705   throw new TaskExecutionException("[ERROR] Error copying file from "
            + inputDomain2Src + " to " + inputDomain2Target);
707 }

709 // Copy aerosols scavenging coeff
    String lookupAerosol2RH00Src = NMMBEnvironment.OUTPUT
711         + "lookup_aerosol2.dat.rh00";
    String lookupAerosol2RH00Target = NMMBEnvironment.UMO_OUT + "ETAMPNEW_AERO_RH00";
713 if (!FileManagement.copyFile(lookupAerosol2RH00Src, lookupAerosol2RH00Target)) {
      throw new TaskExecutionException("[ERROR] Error copying file from "
715           + lookupAerosol2RH00Src + " to " + lookupAerosol2RH00Target);
    }
717
    String lookupAerosol2RH50Src = NMMBEnvironment.OUTPUT
719         + "lookup_aerosol2.dat.rh50";
    String lookupAerosol2RH50Target = NMMBEnvironment.UMO_OUT + "ETAMPNEW_AERO_RH50";
721 if (!FileManagement.copyFile(lookupAerosol2RH50Src, lookupAerosol2RH50Target)) {
      throw new TaskExecutionException("[ERROR] Error copying file from "
723           + lookupAerosol2RH50Src + " to " + lookupAerosol2RH50Target);
    }
725
    String lookupAerosol2RH70Src = NMMBEnvironment.OUTPUT
727         + "lookup_aerosol2.dat.rh70";
    String lookupAerosol2RH70Target = NMMBEnvironment.UMO_OUT + "ETAMPNEW_AERO_RH70";
729 if (!FileManagement.copyFile(lookupAerosol2RH70Src, lookupAerosol2RH70Target)) {
      throw new TaskExecutionException("[ERROR] Error copying file from "
731           + lookupAerosol2RH70Src + " to " + lookupAerosol2RH70Target);
    }
733
    String lookupAerosol2RH80Src = NMMBEnvironment.OUTPUT
735         + "lookup_aerosol2.dat.rh80";
    String lookupAerosol2RH80Target = NMMBEnvironment.UMO_OUT + "ETAMPNEW_AERO_RH80";
737 if (!FileManagement.copyFile(lookupAerosol2RH80Src, lookupAerosol2RH80Target)) {
      throw new TaskExecutionException("[ERROR] Error copying file from "
739           + lookupAerosol2RH80Src + " to " + lookupAerosol2RH80Target);
    }
741
    String lookupAerosol2RH90Src = NMMBEnvironment.OUTPUT
743         + "lookup_aerosol2.dat.rh90";
    String lookupAerosol2RH90Target = NMMBEnvironment.UMO_OUT + "ETAMPNEW_AERO_RH90";
745 if (!FileManagement.copyFile(lookupAerosol2RH90Src, lookupAerosol2RH90Target)) {
      throw new TaskExecutionException("[ERROR] Error copying file from "
747           + lookupAerosol2RH90Src + " to " + lookupAerosol2RH90Target);
    }
749
    String lookupAerosol2RH95Src = NMMBEnvironment.OUTPUT
751         + "lookup_aerosol2.dat.rh95";
    String lookupAerosol2RH95Target = NMMBEnvironment.UMO_OUT + "ETAMPNEW_AERO_RH95";
753 if (!FileManagement.copyFile(lookupAerosol2RH95Src, lookupAerosol2RH95Target)) {
      throw new TaskExecutionException("[ERROR] Error copying file from "
755           + lookupAerosol2RH95Src + " to " + lookupAerosol2RH95Target);
    }
757
    String lookupAerosol2RH99Src = NMMBEnvironment.OUTPUT
759         + "lookup_aerosol2.dat.rh99";
    String lookupAerosol2RH99Target = NMMBEnvironment.UMO_OUT + "ETAMPNEW_AERO_RH99";
761 if (!FileManagement.copyFile(lookupAerosol2RH99Src, lookupAerosol2RH99Target)) {
      throw new TaskExecutionException("[ERROR] Error copying file from "
763           + lookupAerosol2RH99Src + " to " + lookupAerosol2RH99Target);
    }
765
    // Copy coupling previous day (if required)
767 if (nmmbParams.getCoupleDustInit()) {
      String historySrc = NMMBEnvironment.OUTNMMB + nmmbParams.getCase()
769           + File.separator + "history_INIT.hhh";
      String historyTarget = NMMBEnvironment.UMO_OUT + "history_INIT.hhh";
771   if (!FileManagement.copyFile(historySrc, historyTarget)) {
        throw new TaskExecutionException("[ERROR] Error copying file from "
```

```
773                 + historySrc + " to " + historyTarget);
      }
775   }
  }
777
  /**
779  * Performs the UMO Model simulation step
   *
781  */
  private static void doUMOModel(NMMBParameters nmmbParams, Date currentDate)
783         throws TaskExecutionException {
    LOGGER_UMO_MODEL.info("Enter UMO Model process");
785
    // Prepare execution
787   copyFilesFromPreprocess(nmmbParams);
    nmmbParams.prepareUMOModelExecution(currentDate);
789   MessagePrinter umoModelMP = new MessagePrinter(LOGGER_UMO_MODEL);

791   // Begin MPI call
    umoModelMP.printHeaderMsg("BEGIN");
793   umoModelMP.printInfoMsg("Executing nmmb_esmf.x UMO-NMMb-DUST-RRTM model");

795   String stdOutFile = NMMBEnvironment.UMO_OUT + "nems.out";
    String stdErrFile = NMMBEnvironment.UMO_OUT + "nems.err";
797   Integer nemsEV = MPI.nems(stdOutFile, stdErrFile);

799   LOGGER_UMO_MODEL.debug("Execution of mpirun NEMS ended with status " + nemsEV);
    if (nemsEV != 0) {
801   throw new TaskExecutionException("[ERROR] Error executing mpirun nems");
    }
803   umoModelMP
          .printInfoMsg("Finished Executing nmmb_esmf.x UMO-NMMb-DUST-RRTM model");
805
    // Post execution
807   nmmbParams.postUMOModelExecution(currentDate);

809   umoModelMP.printHeaderMsg("END");

811   LOGGER_UMO_MODEL.info("UMO Model process finished");
  }
813
  /**
815  * Performs the POST step
   *
817  */
  private static void doPost(NMMBParameters nmmbParams, Date currentDate)
819         throws TaskExecutionException {
    // Define model output folder by case and date
821   String currentDateSTR = NMMBConstants.STR_TO_DATE.format(currentDate);
    String hourSTR = (nmmbParams.getHour() < 10)
823           ? "0" + String.valueOf(nmmbParams.getHour())
          : String.valueOf(nmmbParams.getHour());
825   String folderOutput = NMMBEnvironment.OUTNMMB + nmmbParams.getCase()
          + File.separator + currentDateSTR + hourSTR + File.separator;
827
    LOGGER_POST.info("Postproc_carbono process for DAY: " + currentDateSTR);
829
    // Prepare execution
831   nmmbParams.preparePostProcessExecution(currentDate);
    MessagePrinter postProcMP = new MessagePrinter(LOGGER_POST);
833
    // Deploy files and compile binaries if needed
835   Integer evCompile = BINARY.preparePost(nmmbParams.isCompileBinaries(),
          folderOutput);
837   if (evCompile != 0) {
    throw new TaskExecutionException("[ERROR] Error preparing post process");
839   }

841   // Begin POST call
    postProcMP.printHeaderMsg("BEGIN");
843
    String domainSTR = (nmmbParams.getDomain()) ? "glob" : "reg";
```

```
845     String dateHour = currentDateSTR + hourSTR;
        String pout = folderOutput + "new_pout_*.nc";
847     String CTM = folderOutput + "NMMB-BSC-CTM_" + dateHour + "_" + domainSTR + ".nc";

849     Integer ev = BINARY.executePostprocAuth(folderOutput, pout, CTM);
        LOGGER_POST.debug("Execution of NCRAT ended with status " + ev);
851     if (ev != 0) {
          throw new TaskExecutionException("[ERROR] Error executing post process");
853     }

855     // Post execution
        nmmbParams.cleanPostProcessExecution(folderOutput);
857
        postProcMP.printHeaderMsg("END");
859
        LOGGER_POST.info("Post process finished");
861   }

863   /**
       * Main NMMB Workflow
865    *
       * @param args
867    *            args[0] : Configuration file path
       *
869    * @throws MainExecutionException
       */
871   public static void main(String[] args) throws MainExecutionException {
        LOGGER_MAIN.info("Starting NMMB application");
873
        // Check and get arguments
875     if (args.length != 1) {
          usage();
877       System.exit(1);
        }
879     String configurationFile = args[0];

881     // Load configuration
        NMMBConfigManager nmmbConfigManager = null;
883     try {
          LOGGER_MAIN.info("Loading NMMB Configuration file " + configurationFile);
885       nmmbConfigManager = new NMMBConfigManager(configurationFile);
          LOGGER_MAIN.info("Configuration file loaded");
887     } catch (ConfigurationException ce) {
          LOGGER_MAIN.error("[ERROR] Cannot load configuration file: "
889               + configurationFile + ". Aborting...", ce);
          throw new MainExecutionException(ce);
891     }

893     // Compute the execution variables
        NMMBParameters nmmbParams = new NMMBParameters(nmmbConfigManager);
895
        // Prepare the execution
897     nmmbParams.prepareExecution();

899     // Fixed process (do before main time looping)
        if (nmmbParams.doFixed()) {
901       try {
            doFixed(nmmbParams);
903       } catch (TaskExecutionException tee) {
            LOGGER_FIXED.error("[ERROR] Task exception on fixed phase. Aborting...",
905                 tee);
            throw new MainExecutionException(tee);
907       }
        }
909
        // Start main time loop
911     Date currentDate = nmmbParams.getStartDate();
        while (!currentDate.after(nmmbParams.getEndDate())) {
913       String currentDateSTR = NMMBConstants.STR_TO_DATE.format(currentDate);
          LOGGER_MAIN.info(currentDateSTR + " simulation started");
915
          // Create output folders if needed
```

```
917       nmmbParams.createOutputFolders(currentDate);

919       // Vrbl process
          if (nmmbParams.doVariable()) {
921         try {
              doVariable(nmmbParams, currentDate);
923         } catch (TaskExecutionException tee) {
              LOGGER_VARIABLE
925                   .error("[ERROR] Task exception on variable phase at date "
                              + currentDateSTR + ". Aborting...", tee);
927           throw new MainExecutionException(tee);
            }
929       }

931       // UMO model run
          if (nmmbParams.doUmoModel()) {
933         try {
              doUMOModel(nmmbParams, currentDate);
935         } catch (TaskExecutionException tee) {
              LOGGER_UMO_MODEL
937                   .error("[ERROR] Task exception on UMO Model phase at date "
                              + currentDateSTR + ". Aborting...", tee);
939           throw new MainExecutionException(tee);
            }
941
          }
943
          // Post process
945       if (nmmbParams.doPost()) {
            try {
947           doPost(nmmbParams, currentDate);
            } catch (TaskExecutionException tee) {
949           LOGGER_POST.error("[ERROR] Task exception on Post phase at date "
                      + currentDateSTR + ". Aborting...", tee);
951         throw new MainExecutionException(tee);
            }
953       }

955       LOGGER_MAIN.info(currentDateSTR + " simulation finished");

957       // Getting next simulation day
          currentDate = Date.from(currentDate.toInstant()
959               .plusSeconds(NMMBConstants.ONE_DAY_IN_SECONDS));
        }
961   }

963 }
```

C.2 NmmbItf.java

```
1  package nmmb;

3  import integratedtoolkit.types.annotations.Constraints;
   import integratedtoolkit.types.annotations.Parameter;
5  import integratedtoolkit.types.annotations.parameter.Direction;
   import integratedtoolkit.types.annotations.parameter.Stream;
7  import integratedtoolkit.types.annotations.parameter.Type;
   import integratedtoolkit.types.annotations.task.Binary;
9  import integratedtoolkit.types.annotations.task.MPI;
   import nmmb.configuration.NMMBConstants;
11 import nmmb.utils.BinaryWrapper;
   import nmmb.utils.FortranWrapper;
13


15 public interface NmmbItf {

17     // COMPILE STEP
       @Binary(binary = FortranWrapper.FC)
```

```
19      Integer fortranCompiler(
            @Parameter(type = Type.STRING, direction = Direction.IN) String mcFlag,
21          @Parameter(type = Type.STRING, direction = Direction.IN) String sharedFlag,
            @Parameter(type = Type.STRING, direction = Direction.IN) String covertPrefix,
23          @Parameter(type = Type.STRING, direction = Direction.IN) String convertValue,
            @Parameter(type = Type.STRING, direction = Direction.IN) String tracebackFlag,
25          @Parameter(type = Type.STRING, direction = Direction.IN) String assumePrefix,
            @Parameter(type = Type.STRING, direction = Direction.IN) String assumeValue,
27          @Parameter(type = Type.STRING, direction = Direction.IN) String optFlag,
            @Parameter(type = Type.STRING, direction = Direction.IN) String fpmodelPrefix,
29          @Parameter(type = Type.STRING, direction = Direction.IN) String fpmodelValue,
            @Parameter(type = Type.STRING, direction = Direction.IN) String stackFlag,
31          @Parameter(type = Type.STRING, direction = Direction.IN) String oFlag,
            @Parameter(type = Type.STRING, direction = Direction.IN) String executable,
33          @Parameter(type = Type.FILE, direction = Direction.IN) String source
        );
35
        @Binary(binary = FortranWrapper.FC)
37      Integer fortranCompileObject(
            @Parameter(type = Type.STRING, direction = Direction.IN) String mcFlag,
39          @Parameter(type = Type.STRING, direction = Direction.IN) String sharedFlag,
            @Parameter(type = Type.STRING, direction = Direction.IN) String covertPrefix,
41          @Parameter(type = Type.STRING, direction = Direction.IN) String convertValue,
            @Parameter(type = Type.STRING, direction = Direction.IN) String tracebackFlag,
43          @Parameter(type = Type.STRING, direction = Direction.IN) String assumePrefix,
            @Parameter(type = Type.STRING, direction = Direction.IN) String assumeValue,
45          @Parameter(type = Type.STRING, direction = Direction.IN) String optFlag,
            @Parameter(type = Type.STRING, direction = Direction.IN) String fpmodelPrefix,
47          @Parameter(type = Type.STRING, direction = Direction.IN) String fpmodelValue,
            @Parameter(type = Type.STRING, direction = Direction.IN) String stackFlag,
49          @Parameter(type = Type.STRING, direction = Direction.IN) String cFlag,
            @Parameter(type = Type.STRING, direction = Direction.IN) String source,
51          @Parameter(type = Type.STRING, direction = Direction.IN) String oFlag,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String object,
53          @Parameter(type = Type.STRING, direction = Direction.IN) String moduleFlag,
            @Parameter(type = Type.STRING, direction = Direction.IN) String moduleDir
55      );

57      @Binary(binary = FortranWrapper.FC)
        Integer fortranCompileWithObject(
59          @Parameter(type = Type.STRING, direction = Direction.IN) String mcFlag,
            @Parameter(type = Type.STRING, direction = Direction.IN) String sharedFlag,
61          @Parameter(type = Type.STRING, direction = Direction.IN) String covertPrefix,
            @Parameter(type = Type.STRING, direction = Direction.IN) String convertValue,
63          @Parameter(type = Type.STRING, direction = Direction.IN) String tracebackFlag,
            @Parameter(type = Type.STRING, direction = Direction.IN) String assumePrefix,
65          @Parameter(type = Type.STRING, direction = Direction.IN) String assumeValue,
            @Parameter(type = Type.STRING, direction = Direction.IN) String optFlag,
67          @Parameter(type = Type.STRING, direction = Direction.IN) String fpmodelPrefix,
            @Parameter(type = Type.STRING, direction = Direction.IN) String fpmodelValue,
69          @Parameter(type = Type.STRING, direction = Direction.IN) String stackFlag,
            @Parameter(type = Type.STRING, direction = Direction.IN) String oFlag,
71          @Parameter(type = Type.STRING, direction = Direction.IN) String executable,
            @Parameter(type = Type.STRING, direction = Direction.IN) String source,
73          @Parameter(type = Type.FILE, direction = Direction.IN) String object
        );
75
        @Binary(binary = FortranWrapper.FC)
77      Integer fortranCompilerWithW3(
            @Parameter(type = Type.STRING, direction = Direction.IN) String mcFlag,
79          @Parameter(type = Type.STRING, direction = Direction.IN) String sharedFlag,
            @Parameter(type = Type.STRING, direction = Direction.IN) String covertPrefix,
81          @Parameter(type = Type.STRING, direction = Direction.IN) String convertValue,
            @Parameter(type = Type.STRING, direction = Direction.IN) String tracebackFlag,
83          @Parameter(type = Type.STRING, direction = Direction.IN) String assumePrefix,
            @Parameter(type = Type.STRING, direction = Direction.IN) String assumeValue,
85          @Parameter(type = Type.STRING, direction = Direction.IN) String optFlag,
            @Parameter(type = Type.STRING, direction = Direction.IN) String fpmodelPrefix,
87          @Parameter(type = Type.STRING, direction = Direction.IN) String fpmodelValue,
            @Parameter(type = Type.STRING, direction = Direction.IN) String stackFlag,
89          @Parameter(type = Type.STRING, direction = Direction.IN) String oFlag,
            @Parameter(type = Type.STRING, direction = Direction.IN) String executable,
```

```
91          @Parameter(type = Type.FILE, direction = Direction.IN) String source,
            @Parameter(type = Type.STRING, direction = Direction.IN) String w3libDir,
93          @Parameter(type = Type.STRING, direction = Direction.IN) String bacioLibDir,
            @Parameter(type = Type.STRING, direction = Direction.IN) String w3Lib,
95          @Parameter(type = Type.STRING, direction = Direction.IN) String bacioLib
        );
97
        @Binary(binary = FortranWrapper.GFC)
99      Integer gfortranCompiler(
            @Parameter(type = Type.STRING, direction = Direction.IN) String bigOFlag,
101         @Parameter(type = Type.FILE, direction = Direction.IN) String source,
            @Parameter(type = Type.STRING, direction = Direction.IN) String oFlag,
103         @Parameter(type = Type.STRING, direction = Direction.IN) String executable
        );
105
        @Binary(binary = BinaryWrapper.COMPILE_READ_PAUL_SOURCE)
107     Integer compileReadPaulSource(
            @Parameter(type = Type.FILE, direction = Direction.IN) String source,
109         @Parameter(type = Type.STRING, direction = Direction.IN) String executable
        );
111
        // FIXED STEP
113     @Binary(binary = "${" + NMMBConstants.ENV_NAME_FIX + "}/"
            + FortranWrapper.SMMOUNT + FortranWrapper.SUFFIX_EXE)
115     Integer smmount(
            @Parameter(type = Type.STRING, direction = Direction.IN) String topoDir,
117         @Parameter(type = Type.FILE, direction = Direction.OUT) String seamaskDEM,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String heightDEM
119     );

121     @Binary(binary = "${" + NMMBConstants.ENV_NAME_FIX + "}/"
            + FortranWrapper.LANDUSE + FortranWrapper.SUFFIX_EXE)
123     Integer landuse(
            @Parameter(type = Type.STRING, direction = Direction.IN) String landuseDir,
125         @Parameter(type = Type.FILE, direction = Direction.OUT) String landuse,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String kount_landuse
127     );

129     @Binary(binary = "${" + NMMBConstants.ENV_NAME_FIX + "}/"
            + FortranWrapper.LANDUSENEW + FortranWrapper.SUFFIX_EXE)
131     Integer landusenew(
            @Parameter(type = Type.STRING, direction = Direction.IN) String gtopDir,
133         @Parameter(type = Type.FILE, direction = Direction.OUT) String landusenew,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String kount_landusenew
135     );

137     @Binary(binary = "${" + NMMBConstants.ENV_NAME_FIX + "}/"
            + FortranWrapper.TOPO + FortranWrapper.SUFFIX_EXE)
139     Integer topo(
            @Parameter(type = Type.STRING, direction = Direction.IN) String topoDir,
141         @Parameter(type = Type.FILE, direction = Direction.OUT) String heightmean
        );
143
        @Binary(binary = "${" + NMMBConstants.ENV_NAME_FIX + "}/"
145         + FortranWrapper.STDH + FortranWrapper.SUFFIX_EXE)
        Integer stdh(
147         @Parameter(type = Type.FILE, direction = Direction.IN) String seamaskDEM,
            @Parameter(type = Type.FILE, direction = Direction.IN) String heightmean,
149         @Parameter(type = Type.STRING, direction = Direction.IN) String topoDir,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String stdh
151     );

153     @Binary(binary = "${" + NMMBConstants.ENV_NAME_FIX + "}/"
            + FortranWrapper.ENVELOPE + FortranWrapper.SUFFIX_EXE)
155     Integer envelope(
            @Parameter(type = Type.FILE, direction = Direction.IN) String heightmean,
157         @Parameter(type = Type.FILE, direction = Direction.IN) String stdh,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String height
159     );

161     @Binary(binary = "${" + NMMBConstants.ENV_NAME_FIX + "}/"
            + FortranWrapper.TOPSOILTYPE + FortranWrapper.SUFFIX_EXE)
```

```
163     Integer topsoiltype(
            @Parameter(type = Type.FILE, direction = Direction.IN) String seamaskDEM,
165         @Parameter(type = Type.STRING, direction = Direction.IN) String soiltypeDir,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String topsoiltype
167     );

169     @Binary(binary = "${" + NMMBConstants.ENV_NAME_FIX + "}/"
            + FortranWrapper.BOTSOILTYPE + FortranWrapper.SUFFIX_EXE)
171     Integer botsoiltype(
            @Parameter(type = Type.FILE, direction = Direction.IN) String seamaskDEM,
173         @Parameter(type = Type.STRING, direction = Direction.IN) String soiltypePath,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String botsoiltype
175     );

177     @Binary(binary = "${" + NMMBConstants.ENV_NAME_FIX + "}/"
            + FortranWrapper.TOPOSEAMASK + FortranWrapper.SUFFIX_EXE)
179     Integer toposeamask(
            @Parameter(type = Type.FILE, direction = Direction.IN) String seamaskDEM,
181         @Parameter(type = Type.FILE, direction = Direction.OUT) String seamask,
            @Parameter(type = Type.FILE, direction = Direction.INOUT) String height,
183         @Parameter(type = Type.FILE, direction = Direction.INOUT) String landuse,
            @Parameter(type = Type.FILE, direction = Direction.INOUT) String topsoiltype,
185         @Parameter(type = Type.FILE, direction = Direction.INOUT) String botsoiltype
        );
187
        @Binary(binary = "${" + NMMBConstants.ENV_NAME_FIX + "}/"
189         + FortranWrapper.STDHTOPO + FortranWrapper.SUFFIX_EXE)
        Integer stdhtopo(
191         @Parameter(type = Type.FILE, direction = Direction.IN) String seamask,
            @Parameter(type = Type.FILE, direction = Direction.INOUT) String stdh
193     );

195     @Binary(binary = "${" + NMMBConstants.ENV_NAME_FIX + "}/"
            + FortranWrapper.DEEPTEMPERATURE + FortranWrapper.SUFFIX_EXE)
197     Integer deeptemperature(
            @Parameter(type = Type.FILE, direction = Direction.IN) String seamask,
199         @Parameter(type = Type.STRING, direction = Direction.IN) String soiltempPath,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String deeptemperature
201     );

203     @Binary(binary = "${" + NMMBConstants.ENV_NAME_FIX + "}/"
            + FortranWrapper.SNOWALBEDO + FortranWrapper.SUFFIX_EXE)
205     Integer snowalbedo(
            @Parameter(type = Type.STRING, direction = Direction.IN) String maxsnowalbDir,
207         @Parameter(type = Type.FILE, direction = Direction.OUT) String snowalbedo
        );
209
        @Binary(binary = "${" + NMMBConstants.ENV_NAME_FIX + "}/"
211         + FortranWrapper.VCGENERATOR + FortranWrapper.SUFFIX_EXE)
        Integer vcgenerator(
213         @Parameter(type = Type.FILE, direction = Direction.OUT) String dsg
        );
215
        @Binary(binary = "${" + NMMBConstants.ENV_NAME_FIX + "}/"
217         + FortranWrapper.ROUGHNESS + FortranWrapper.SUFFIX_EXE)
        Integer roughness(
219         @Parameter(type = Type.STRING, direction = Direction.IN) String roughnessDir,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String roughness
221     );

223     @Binary(binary = "${" + NMMBConstants.ENV_NAME_FIX + "}/"
            + FortranWrapper.GFDLCO2 + FortranWrapper.SUFFIX_EXE)
225     Integer gfdlco2(
            @Parameter(type = Type.FILE, direction = Direction.IN) String dsg,
227         @Parameter(type = Type.STRING, direction = Direction.IN) String co2_data_dir,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String co2_trans
229     );

231     @Binary(binary = "${" + NMMBConstants.ENV_NAME_FIX + "}/lookup_tables/"
            + BinaryWrapper.RUN_AEROSOL)
233     Integer run_aerosol(
            @Parameter() boolean mustCompile,
```

```
235         @Parameter() boolean mustClean,
            @Parameter(type = Type.FILE, direction = Direction.OUT)
237         String lookup_aerosol2_rh00,
            @Parameter(type = Type.FILE, direction = Direction.OUT)
239         String lookup_aerosol2_rh50,
            @Parameter(type = Type.FILE, direction = Direction.OUT)
241         String lookup_aerosol2_rh70,
            @Parameter(type = Type.FILE, direction = Direction.OUT)
243         String lookup_aerosol2_rh80,
            @Parameter(type = Type.FILE, direction = Direction.OUT)
245         String lookup_aerosol2_rh90,
            @Parameter(type = Type.FILE, direction = Direction.OUT)
247         String lookup_aerosol2_rh95,
            @Parameter(type = Type.FILE, direction = Direction.OUT)
249         String lookup_aerosol2_rh99
        );
251

253     // VARIABLE STEP
        @Binary(binary = "${" + NMMBConstants.ENV_NAME_VRB + "}/"
255         + BinaryWrapper.DEGRIB_GFS_GENERIC)
        Integer degribgfs_generic_05(
257         @Parameter(type = Type.FILE, direction = Direction.OUT) String CW,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String ICEC,
259         @Parameter(type = Type.FILE, direction = Direction.OUT) String SH,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String SOILT2,
261         @Parameter(type = Type.FILE, direction = Direction.OUT) String SOILT4,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String SOILW2,
263         @Parameter(type = Type.FILE, direction = Direction.OUT) String SOILW4,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String TT,
265         @Parameter(type = Type.FILE, direction = Direction.OUT) String VV,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String HH,
267         @Parameter(type = Type.FILE, direction = Direction.OUT) String PRMSL,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String SOILT1,
269         @Parameter(type = Type.FILE, direction = Direction.OUT) String SOILT3,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String SOILW1,
271         @Parameter(type = Type.FILE, direction = Direction.OUT) String SOILW3,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String SST_TS,
273         @Parameter(type = Type.FILE, direction = Direction.OUT) String UU,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String WEASD
275     );

277     @Binary(binary = "${" + NMMBConstants.ENV_NAME_VRB + "}/"
            + FortranWrapper.GFS2MODEL + FortranWrapper.SUFFIX_EXE)
279     Integer gfs2model_rrtm(
            @Parameter(type = Type.FILE, direction = Direction.IN) String CW,
281         @Parameter(type = Type.FILE, direction = Direction.IN) String ICEC,
            @Parameter(type = Type.FILE, direction = Direction.IN) String SH,
283         @Parameter(type = Type.FILE, direction = Direction.IN) String SOILT2,
            @Parameter(type = Type.FILE, direction = Direction.IN) String SOILT4,
285         @Parameter(type = Type.FILE, direction = Direction.IN) String SOILW2,
            @Parameter(type = Type.FILE, direction = Direction.IN) String SOILW4,
287         @Parameter(type = Type.FILE, direction = Direction.IN) String TT,
            @Parameter(type = Type.FILE, direction = Direction.IN) String VV,
289         @Parameter(type = Type.FILE, direction = Direction.IN) String HH,
            @Parameter(type = Type.FILE, direction = Direction.IN) String PRMSL,
291         @Parameter(type = Type.FILE, direction = Direction.IN) String SOILT1,
            @Parameter(type = Type.FILE, direction = Direction.IN) String SOILT3,
293         @Parameter(type = Type.FILE, direction = Direction.IN) String SOILW1,
            @Parameter(type = Type.FILE, direction = Direction.IN) String SOILW3,
295         @Parameter(type = Type.FILE, direction = Direction.IN) String SST_TS,
            @Parameter(type = Type.FILE, direction = Direction.IN) String UU,
297         @Parameter(type = Type.FILE, direction = Direction.IN) String WEASD,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String GFS_file
299     );

301     @Binary(binary = "${" + NMMBConstants.ENV_NAME_VRB + "}/"
            + FortranWrapper.INC_RRTM + FortranWrapper.SUFFIX_EXE)
303     Integer inc_rrtm(
            @Parameter(type = Type.FILE, direction = Direction.IN) String GFS_file,
305         @Parameter(type = Type.STRING, direction = Direction.IN) String deco
        );
```

```
307
    @Binary(binary = "${" + NMMBConstants.ENV_NAME_VRB + "}/"
309         + FortranWrapper.CNV_RRTM + FortranWrapper.SUFFIX_EXE)
    Integer cnv_rrtm(
311         @Parameter(type = Type.FILE, direction = Direction.IN) String GFS_file,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String llspl000,
313         @Parameter(type = Type.FILE, direction = Direction.OUT) String outtmp,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String outmst,
315         @Parameter(type = Type.FILE, direction = Direction.OUT) String outsst,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String outsno,
317         @Parameter(type = Type.FILE, direction = Direction.OUT) String outcic
    );
319
    @Binary(binary = "${" + NMMBConstants.ENV_NAME_VRB + "}/"
321         + FortranWrapper.DEGRIB_SST + FortranWrapper.SUFFIX_EXE)
    Integer degribsst(
323         @Parameter(type = Type.FILE, direction = Direction.OUT) String llgsst05,
            @Parameter(type = Type.STRING, direction = Direction.IN) String sstfileinPath
325 );

327 @Binary(binary = "${" + NMMBConstants.ENV_NAME_VRB + "}/"
        + FortranWrapper.ALBEDO + FortranWrapper.SUFFIX_EXE)
329 Integer albedo(
        @Parameter(type = Type.FILE, direction = Direction.IN) String llspl000,
331     @Parameter(type = Type.FILE, direction = Direction.IN) String seamask,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String albedo,
333     @Parameter(type = Type.FILE, direction = Direction.OUT) String albedobase,
        @Parameter(type = Type.STRING, direction = Direction.IN) String albedomnth
335 );

337 @Binary(binary = "${" + NMMBConstants.ENV_NAME_VRB + "}/"
        + FortranWrapper.ALBEDO_RRTM_1DEG + FortranWrapper.SUFFIX_EXE)
339 Integer albedorrtm(
        @Parameter(type = Type.FILE, direction = Direction.IN) String llspl000,
341     @Parameter(type = Type.FILE, direction = Direction.IN) String seamask,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String albedorrtm,
343     @Parameter(type = Type.STRING, direction = Direction.IN) String albedorrtm1degDir
    );
345
    @Binary(binary = "${" + NMMBConstants.ENV_NAME_VRB + "}/"
347         + FortranWrapper.VEG_FRAC + FortranWrapper.SUFFIX_EXE)
    Integer vegfrac(
349         @Parameter(type = Type.FILE, direction = Direction.IN) String llspl000,
            @Parameter(type = Type.FILE, direction = Direction.IN) String seamask,
351         @Parameter(type = Type.FILE, direction = Direction.OUT) String vegfrac,
            @Parameter(type = Type.STRING, direction = Direction.IN) String vegfracmnth
353 );

355 @Binary(binary = "${" + NMMBConstants.ENV_NAME_VRB + "}/"
        + FortranWrapper.Z0_VEGUSTAR + FortranWrapper.SUFFIX_EXE)
357 Integer z0vegfrac(
        @Parameter(type = Type.FILE, direction = Direction.IN) String seamask,
359     @Parameter(type = Type.FILE, direction = Direction.IN) String landuse,
        @Parameter(type = Type.FILE, direction = Direction.IN) String topsoiltype,
361     @Parameter(type = Type.FILE, direction = Direction.IN) String height,
        @Parameter(type = Type.FILE, direction = Direction.IN) String stdh,
363     @Parameter(type = Type.FILE, direction = Direction.IN) String vegfrac,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String z0base,
365     @Parameter(type = Type.FILE, direction = Direction.OUT) String z0,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String ustar
367 );

369 @Binary(binary = "${" + NMMBConstants.ENV_NAME_VRB + "}/"
        + FortranWrapper.ALLPREP_RRTM + FortranWrapper.SUFFIX_EXE)
371 Integer allprep(
        @Parameter(type = Type.FILE, direction = Direction.IN) String llspl000,
373     @Parameter(type = Type.FILE, direction = Direction.IN) String llgsst05,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String sst05,
375     @Parameter(type = Type.FILE, direction = Direction.IN) String height,
        @Parameter(type = Type.FILE, direction = Direction.IN) String seamask,
377     @Parameter(type = Type.FILE, direction = Direction.IN) String stdh,
        @Parameter(type = Type.FILE, direction = Direction.INOUT) String deeptemperature,
```

```
379         @Parameter(type = Type.FILE, direction = Direction.INOUT) String snowalbedo,
            @Parameter(type = Type.FILE, direction = Direction.IN) String z0,
381         @Parameter(type = Type.FILE, direction = Direction.IN) String z0base,
            @Parameter(type = Type.FILE, direction = Direction.IN) String landuse,
383         @Parameter(type = Type.FILE, direction = Direction.IN) String landusenew,
            @Parameter(type = Type.FILE, direction = Direction.IN) String topsoiltype,
385         @Parameter(type = Type.FILE, direction = Direction.IN) String vegfrac,
            @Parameter(type = Type.FILE, direction = Direction.IN) String albedorrtm,
387         @Parameter(type = Type.FILE, direction = Direction.IN) String llgsst,
            @Parameter(type = Type.FILE, direction = Direction.IN) String llgsno,
389         @Parameter(type = Type.FILE, direction = Direction.IN) String llgcic,
            @Parameter(type = Type.FILE, direction = Direction.IN) String llsmst,
391         @Parameter(type = Type.FILE, direction = Direction.IN) String llstmp,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String albedorrtmcorr,
393         @Parameter(type = Type.FILE, direction = Direction.OUT) String dzsoil,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String tskin,
395         @Parameter(type = Type.FILE, direction = Direction.OUT) String sst,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String snow,
397         @Parameter(type = Type.FILE, direction = Direction.OUT) String snowheight,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String cice,
399         @Parameter(type = Type.FILE, direction = Direction.OUT) String seamaskcorr,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String landusecorr,
401         @Parameter(type = Type.FILE, direction = Direction.OUT) String landusenewcorr,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String topsoiltypecorr,
403         @Parameter(type = Type.FILE, direction = Direction.OUT) String vegfraccorr,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String z0corr,
405         @Parameter(type = Type.FILE, direction = Direction.OUT) String z0basecorr,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String emissivity,
407         @Parameter(type = Type.FILE, direction = Direction.OUT) String canopywater,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String frozenprecratio,
409         @Parameter(type = Type.FILE, direction = Direction.OUT) String smst,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String sh2o,
411         @Parameter(type = Type.FILE, direction = Direction.OUT) String stmp,
            @Parameter(type = Type.FILE, direction = Direction.IN) String dsg,
413         @Parameter(type = Type.FILE, direction = Direction.OUT) String fcst,
            @Parameter(type = Type.FILE, direction = Direction.IN) String albedo,
415         @Parameter(type = Type.FILE, direction = Direction.IN) String ustar,
            @Parameter(type = Type.STRING, direction = Direction.IN) String fcstDir,
417         @Parameter(type = Type.STRING, direction = Direction.IN) String bocoPrefix,
            @Parameter(type = Type.STRING, direction = Direction.IN) String llsplPrefix
419     );

421
    @Binary(binary = "${" + NMMBConstants.ENV_NAME_VRB + "}/"
423         + FortranWrapper.READ_PAUL_SOURCE + FortranWrapper.SUFFIX_EXE)
    Integer readpaulsource(
425         @Parameter(type = Type.FILE, direction = Direction.IN) String seamask,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String source,
427         @Parameter(type = Type.FILE, direction = Direction.OUT) String sourceNETCDF,
            @Parameter(type = Type.STRING, direction = Direction.IN) String sourceNCIncludeDir
429     );

431     @Binary(binary = "${" + NMMBConstants.ENV_NAME_VRB + "}/"
            + FortranWrapper.DUST_START + FortranWrapper.SUFFIX_EXE)
433     Integer dust_start(
            @Parameter(type = Type.FILE, direction = Direction.IN) String llspl000,
435         @Parameter(type = Type.FILE, direction = Direction.OUT) String soildust,
            @Parameter(type = Type.FILE, direction = Direction.IN) String snow,
437         @Parameter(type = Type.FILE, direction = Direction.IN) String topsoiltypecorr,
            @Parameter(type = Type.FILE, direction = Direction.IN) String landusecorr,
439         @Parameter(type = Type.FILE, direction = Direction.IN) String landusenewcorr,
            @Parameter(type = Type.FILE, direction = Direction.IN) String kount_landuse,
441         @Parameter(type = Type.FILE, direction = Direction.IN) String kount_landusenew,
            @Parameter(type = Type.FILE, direction = Direction.IN) String vegfrac,
443         @Parameter(type = Type.FILE, direction = Direction.IN) String height,
            @Parameter(type = Type.FILE, direction = Direction.IN) String seamask,
445         @Parameter(type = Type.FILE, direction = Direction.IN) String source,
            @Parameter(type = Type.FILE, direction = Direction.IN) String z0corr,
447         @Parameter(type = Type.FILE, direction = Direction.IN) String roughness
    );
449
    // UMO MODEL STEP
```

```
451     @MPI(mpiRunner = "mpirun",
            binary = "${" + NMMBConstants.ENV_NAME_UMO_PATH + "}/MODEL/exe/"
453             + FortranWrapper.NEMS + FortranWrapper.SUFFIX_EXE,
            workingDir = "${" + NMMBConstants.ENV_NAME_UMO_OUT + "}/",
455         computingNodes = "${NEMS_NODES}")
    @Constraints(computingUnits = "${NEMS_CUS_PER_NODE}")
457     Integer nems(
        @Parameter(type = Type.FILE, direction = Direction.OUT, stream = Stream.STDOUT)
459         String stdOutFile,
        @Parameter(type = Type.FILE, direction = Direction.OUT, stream = Stream.STDERR)
461         String stdErrFile
    );
463
    // POST PROC STEP
465 @Binary(binary = "${" + NMMBConstants.ENV_NAME_POST_CARBONO + "}/"
        + BinaryWrapper.PREPARE_POSTPROC_AUTH)
467 Integer preparePost(
        @Parameter() boolean mustCompile,
469     @Parameter(type = Type.STRING, direction = Direction.IN) String folderOutput
    );
471
    @Binary(binary = "${" + NMMBConstants.ENV_NAME_POST_CARBONO + "}/"
473     + BinaryWrapper.EXEC_POSTPROC_AUTH)
    Integer executePostprocAuth(
475     @Parameter(type = Type.STRING, direction = Direction.IN) String folderOutput,
        @Parameter(type = Type.STRING, direction = Direction.IN) String sourcesPath,
477     @Parameter(type = Type.FILE, direction = Direction.OUT) String destFile
    );
479
}
```

Appendix D

GUIDANCE: code highlights

D.1 Guidance.java

```
package guidance;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import binary.BINARY;

import java.text.SimpleDateFormat;
import java.util.Date;

import guidance.GuidanceImpl;
import guidance.exceptions.EnvironmentVariableException;
import guidance.exceptions.GuidanceTaskException;
import guidance.files.AssocFiles;
import guidance.files.CombinedPanelsFiles;
import guidance.files.CommonFiles;
import guidance.files.ImputationFiles;
import guidance.files.MergeFiles;
import guidance.files.PhenomeAnalysisFiles;
import guidance.files.ResultsFiles;
import guidance.utils.ChromoInfo;
import guidance.utils.Environment;
import guidance.utils.Environment.ENV_VARS;
import guidance.utils.ParseCmdLine;


/**
 * The higher-level class of Guidance
 */
public class Guidance {

    private static final String PACKAGE_VERSION = "guidance_0.9.6";

    private static final long B_TO_GB = 1_048_576;

    private static final double PVA_THRESHOLD = 5e-8;
    private static final String PVA_THRS = Double.toString(PVA_THRESHOLD);


    /**
     * The main of Guidance begins here
     *
     * @param args
     * @throws EnvironmentVariableException
     * @throws Exception
     */
    public static void main(String args[]) throws EnvironmentVariableException,
      IOException {
```

```
54
        // Verify that all environment variables have been defined correctly
56      Environment.verify();

58      // Print information of Guidance version
        printGuidancePackageVersion();
60
        // Get the input arguments
62      ParseCmdLine parsingArgs = new ParseCmdLine(args);

64      // Verify and print the status of each stage
        printStagesStatus(parsingArgs);
66      System.out.println("\n[Guidance] Verifyed stages status.");

68      // Get the file name where the list of commands is going to be saved
        String listOfStagesFileName = parsingArgs.getListOfStagesFile();
70
        // Verify whether the file exists or not.
72      File listOfStages = new File(listOfStagesFileName);
        if (!listOfStages.exists()) {
74          System.out.println("\n[Guidance] File to store the tasks list: "
              + listOfStagesFileName);
76      } else {
            System.out.println("\n[Guidance] File to store the tasks list (overwritten): "
78            + listOfStagesFileName);
        }
80      listOfStages.createNewFile();

82      ArrayList<String> listOfCommands = new ArrayList<>();
        String datestring =new SimpleDateFormat("yyyy/MM/dd HH:mm:ss").format(new Date());
84      listOfCommands.add("#########################################################");
        listOfCommands.add("# List of tasks executed by Guidance workflow");
86      listOfCommands.add("# Date: " + datestring);
        listOfCommands.add("# Parameters of the execution: ");
88      listOfCommands.add("#########################################################");

90      // Get the names of the reference panel to be used in the execution
        ArrayList<String> rpanelTypes = new ArrayList<>(parsingArgs.getRpanelTypes());
92      String outDir = parsingArgs.getOutDir();
        ChromoInfo generalChromoInfo = new ChromoInfo();
94
        System.out.println("[Guidance] We print testTypes information");
96      int numberOfTestTypes = parsingArgs.getNumberOfTestTypeName();
        for (int kk = 0; kk < numberOfTestTypes; kk++) {
98          String tmp_test_type = parsingArgs.getTestTypeName(kk);
            String tmp_responseVar = parsingArgs.getResponseVar(kk);
100         String tmp_covariables = parsingArgs.getCovariables(kk);
            System.out.println("[Guidance] " + tmp_test_type + " = "
102             + tmp_responseVar + ":" + tmp_covariables);
        }
104
        // Main code of the workflow
106     // Now, we have to ask whether the GWAS process is mixed.
        // printEnVariables();
108     doMixed(parsingArgs, outDir, rpanelTypes, generalChromoInfo, listOfCommands);

110     // Finally, we print the commands in the output file defined for this.
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(listOfStages))) {
112         for (String str : listOfCommands) {
                writer.write(str);
114             writer.newLine();
                writer.newLine();
116         }

118         writer.flush();
        } catch (IOException ioe) {
120         throw ioe;
        }
122 }

124 /**
     * Method that performs the complete workflow when "mixed" type of GWAS is chosen
```

```
126      *
         * @param parsingArgs
128      * @param outDir
         * @param rpanelTypes
130      * @param generalChromoInfo
         * @param listOfCommands
132      */
     private static void doMixed(ParseCmdLine parsingArgs, String outDir,
134      List<String> rpanelTypes, ChromoInfo generalChromoInfo,
             ArrayList<String> listOfCommands) {
136
         // Create some general objects
138      int startChr = parsingArgs.getStart();
         int endChr = parsingArgs.getEnd();
140
         String exclCgatFlag = parsingArgs.getExclCgatSnp();
142      String exclSVFlag = parsingArgs.getExclSVSnp();
         int chunkSize = parsingArgs.getChunkSize();
144      String inputFormat = null;

146      // Create the names for the common files
         CommonFiles commonFilesInfo = new CommonFiles(parsingArgs, outDir);
148
         // Create the whole directory structure.
150      createDirStructure(parsingArgs, outDir, rpanelTypes, startChr, endChr);

152      // Create the names for mixed files
         ImputationFiles imputationFilesInfo = new ImputationFiles(parsingArgs,
154          generalChromoInfo, outDir, rpanelTypes);

156      // Create the names for Association files
         AssocFiles assocFilesInfo = new AssocFiles(parsingArgs, generalChromoInfo,
158          outDir, rpanelTypes);

160      // Create the names for Merge files
         MergeFiles mergeFilesInfo = new MergeFiles(parsingArgs, generalChromoInfo,
162          outDir, rpanelTypes);

164      // Create the names for Results Files. Take into account this class it to
         // generate file name for results
166      // of each testType and each rPanelType. The results for combined panels or
         // phenome analysis are created
168      // by other class.
         ResultsFiles resultsFilesInfo =new ResultsFiles(parsingArgs, outDir, rpanelTypes);
170      CombinedPanelsFiles combinedPanelsFilesInfo = new CombinedPanelsFiles(parsingArgs,
             outDir, rpanelTypes);
172      PhenomeAnalysisFiles phenomeAnalysisFilesInfo = new PhenomeAnalysisFiles(
             parsingArgs, outDir, rpanelTypes);
174
         // Create all file names used in the workflow.
176      // Also, define which of them will be temporal or permanent.
         setFinalStatusForCommonFiles(parsingArgs, commonFilesInfo);
178      setFinalStatusForImputationFiles(parsingArgs, imputationFilesInfo,
             generalChromoInfo, rpanelTypes);
180      setFinalStatusForAssocFiles(parsingArgs, assocFilesInfo, generalChromoInfo,
             rpanelTypes);
182
         // boolean refPanelCombine = parsingArgs.getRefPanelCombine();
184      // int numOfChromosToProcess = endChr - startChr + 1;
         for (int i = startChr; i <= endChr; i++) {
186          // We get the output pairs file name for mixed
             String mixedPairsFile = commonFilesInfo.getPairsFile(i);
188          String mixedSampleFile = commonFilesInfo.getSampleFile(i);

190          String theChromo = Integer.toString(i);
             String mixedGenFile = null;
192          String bedFile = null;
             String bimFile = null;
194          String famFile = null;
             String mixedBedFile = null;
196          String mixedBimFile = null;
             String mixedFamFile = null;
```

```
198             String mixedBedToBedLogFile = null;

200             String gmapFile = parsingArgs.getGmapDir() + File.separator
                    + parsingArgs.getGmapFileName(i);
202             String mixedShapeitHapsFile = commonFilesInfo.getShapeitHapsFile(i);
                String mixedShapeitSampleFile = commonFilesInfo.getShapeitSampleFile(i);
204             String mixedShapeitLogFile = commonFilesInfo.getShapeitLogFile(i);
                String mixedExcludedSnpsFile = commonFilesInfo.getExcludedSnpsFile(i);
206
                String mixedFilteredHaplotypesFile
208               = commonFilesInfo.getFilteredHaplotypesFile(i);
                String mixedFilteredHaplotypesSampleFile
210               = commonFilesInfo.getFilteredHaplotypesSampleFile(i);
                String mixedFilteredHaplotypesLogFile
212               = commonFilesInfo.getFilteredHaplotypesLogFile(i);
                String mixedFilteredHaplotypesVcfFile
214               = commonFilesInfo.getFilteredHaplotypesVcfFile(i);
                String mixedFilteredListOfSnpsFile
216               = commonFilesInfo.getListOfSnpsFile(i);

218             // Check if the input are in GEN o BED format
                inputFormat = parsingArgs.getInputFormat();
220             if (inputFormat.equals("BED")) {
                    bedFile = commonFilesInfo.getBedFile();
222                 bimFile = commonFilesInfo.getBimFile();
                    famFile = commonFilesInfo.getFamFile();
224
                    mixedBedFile = commonFilesInfo.getByChrBedFile(i);
226                 mixedBimFile = commonFilesInfo.getByChrBimFile(i);
                    mixedFamFile = commonFilesInfo.getByChrFamFile(i);
228                 mixedBedToBedLogFile = commonFilesInfo.getBedToBedLogFile(i);

230                 doConvertFromBedToBed(parsingArgs, listOfCommands, bedFile, bimFile,
                        famFile, theChromo, mixedBedFile, mixedBimFile,
232                         mixedFamFile, mixedBedToBedLogFile);

234                 // Create the RsId list of SNPs that are AT, TA, CG, or GC
                    // In that case, because inputType is BED we pass the newBimFile
236                 doCreateRsIdList(parsingArgs, listOfCommands, mixedBimFile, exclCgatFlag,
                        mixedPairsFile, inputFormat);
238
                    doPhasingBed(parsingArgs, listOfCommands, theChromo, mixedBedFile,
240                     mixedBimFile, mixedFamFile, gmapFile,
                            mixedExcludedSnpsFile, mixedShapeitHapsFile,
242                         mixedShapeitSampleFile, mixedShapeitLogFile,
                            mixedFilteredHaplotypesFile, mixedFilteredHaplotypesSampleFile,
244                         mixedFilteredHaplotypesLogFile, mixedFilteredHaplotypesVcfFile,
                            mixedFilteredListOfSnpsFile, exclCgatFlag, exclSVFlag);
246             } else { // The inputFormat is GEN
                    mixedGenFile = commonFilesInfo.getGenFile(i);
248                 // Task
                    doCreateRsIdList(parsingArgs, listOfCommands, mixedGenFile, exclCgatFlag,
250                     mixedPairsFile, inputFormat);

252                 doPhasing(parsingArgs, listOfCommands, theChromo, mixedGenFile,
                        mixedSampleFile, gmapFile, mixedExcludedSnpsFile,
254                         mixedShapeitHapsFile, mixedShapeitSampleFile,
                            mixedShapeitLogFile, mixedFilteredHaplotypesFile,
256                         mixedFilteredHaplotypesSampleFile, mixedFilteredHaplotypesLogFile,
                            mixedFilteredHaplotypesVcfFile, mixedFilteredListOfSnpsFile,
258                         exclCgatFlag, exclSVFlag);

260             } // End of inputFormat.equals("GEN")

262             // ********************************************
                // * Now perform the imputation tasks
264             // * over the data for the different reference
                // * panels that the user wants to include. The
266             // * list of the panels to be used are stored in
                // * the List array rpanelTypes
268             // ********************************************
                for (int kk = 0; kk < rpanelTypes.size(); kk++) {
```

```
270                 // String rpanelDir = parsingArgs.getRpanelDir(kk);
                    int maxSize = generalChromoInfo.getMaxSize(i);
272                 int minSize = generalChromoInfo.getMinSize(i);

274                 int lim1 = minSize;
                    int lim2 = lim1 + chunkSize - 1;
276                 int j = 0;
                    for (j = minSize; j < maxSize; j += chunkSize) {
278                     makeImputationPerChunk(parsingArgs, i, lim1, lim2, kk, gmapFile,
                            imputationFilesInfo, commonFilesInfo, listOfCommands);
280                     lim1 = lim1 + chunkSize;
                        lim2 = lim2 + chunkSize;
282                 }
                } // End for(kk=0; kk<rpanelTypes.size(); kk++)
284         } // End for(i=startChr;i <= endChr; i++)

286         // ********************************************************************
            // * COMPSs API Call to wait for all tasks *
288         // * comment out following line to include the synchronization point *
            // ********************************************************************
290         // COMPSs.barrier();

292         // Now we continue with the association
            int numberOfTestTypes = parsingArgs.getNumberOfTestTypeName();
294         for (int tt = 0; tt < numberOfTestTypes; tt++) {
                for (int kk = 0; kk < rpanelTypes.size(); kk++) {
296                 for (int i = startChr; i <= endChr; i++) {
                        int maxSize = generalChromoInfo.getMaxSize(i);
298                     int minSize = generalChromoInfo.getMinSize(i);

300                     int lim1 = minSize;
                        int lim2 = lim1 + chunkSize - 1;
302                     for (int j = minSize; j < maxSize; j += chunkSize) {
                            makeAssociationPerChunk(parsingArgs, tt, kk, i, lim1, lim2,
304                             imputationFilesInfo, commonFilesInfo, assocFilesInfo,
                                    listOfCommands);
306
                            lim1 = lim1 + chunkSize;
308                         lim2 = lim2 + chunkSize;
                        }
310
                        // Now we perform the merge of chunks for each chromosome
312                     makeMergeOfChunks(parsingArgs, listOfCommands, tt, kk, i, minSize,
                            maxSize, chunkSize, assocFilesInfo, mergeFilesInfo);
314
                        // We have to filterByAll the last merged File of each chromomose
316                     String theLastReducedFile
                          = mergeFilesInfo.getTheLastReducedFile(tt, kk, i);
318                     String filteredByAllFile
                          = mergeFilesInfo.getFilteredByAllFile(tt, kk, i);
320                     String condensedFile
                          = mergeFilesInfo.getCondensedFile(tt, kk, i);
322
                        doFilterByAll(parsingArgs, listOfCommands, theLastReducedFile,
324                         filteredByAllFile, condensedFile);
                    } // End for(i=startChr;i <= endChr; i++)
326
                    // Now we have to joint the condensedFiles of each chromosome. There is
328                 // not problem if chr23 is being processed, because
                    // the format of condensedFiles is the same for all chromosome.
330                 makeJointCondensedFiles(parsingArgs, listOfCommands, tt, kk, startChr,
                        endChr, mergeFilesInfo);
332
                    // Now we have to joint the filteredByAllFiles of each chromosome.
334                 // Here there is an additional complexity due to the chr23,
                    // because the file format changes if chr23 is being processed. This
336                 // situation is taken into account inside the next function.
                    String rpanelName = rpanelTypes.get(kk);
338
                    makeJointFilteredByAllFiles(parsingArgs, listOfCommands, tt, kk,
340                     rpanelName, startChr, endChr, mergeFilesInfo);
```

```
342                 String lastCondensedFile = mergeFilesInfo.getFinalCondensedFile(tt, kk);

344                 String lastFilteredByAllFile
                      = mergeFilesInfo.getFinalFilteredByAllFile(tt, kk);
346
                    // Lets extract top-hits results from the lastFilteredByAllFile
348                 String topHitsResults = resultsFilesInfo.getTopHitsFile(tt, kk);

350                 String filteredByAllXFile = null;
                    if (endChr == ChromoInfo.MAX_NUMBER_OF_CHROMOSOMES) {
352                     filteredByAllXFile
                          = mergeFilesInfo.getAdditionalFilteredByAllXFile(tt, kk, 0);
354                 } else {
                        filteredByAllXFile = lastFilteredByAllFile;
356                 }

358                 doGenerateTopHits(parsingArgs, listOfCommands, lastFilteredByAllFile,
                        filteredByAllXFile, topHitsResults, PVA_THRS);
360
                    String qqPlotPdfFile = resultsFilesInfo.getQqPlotPdfFile(tt, kk);
362                 String qqPlotTiffFile = resultsFilesInfo.getQqPlotTiffFile(tt, kk);

364                 String manhattanPlotPdfFile
                      = resultsFilesInfo.getManhattanPdfFile(tt, kk);
366                 String manhattanPlotTiffFile
                      = resultsFilesInfo.getManhattanTiffFile(tt, kk);
368
                    String correctedPvaluesFile
370                   = resultsFilesInfo.getCorrectedPvaluesFile(tt, kk);

372                 doGenerateQQManhattanPlots(parsingArgs, listOfCommands, lastCondensedFile,
                        qqPlotPdfFile, manhattanPlotPdfFile,
374                         qqPlotTiffFile, manhattanPlotTiffFile, correctedPvaluesFile);

376             } // End for(kk=0; kk<rpanelTypes.size(); kk++)

378             // Now we continue with the combining of the results of the different
                // reference panels.
380             // It is done if the refPanelCombine flag is true.
                makeCombinePanels(parsingArgs, mergeFilesInfo, combinedPanelsFilesInfo,
382                 rpanelTypes, tt, listOfCommands);

384         } // End for(tt=0; tt<numberOfTestTypes;tt++)

386         makePhenotypeAnalysis(parsingArgs, mergeFilesInfo, resultsFilesInfo,
                phenomeAnalysisFilesInfo, rpanelTypes, listOfCommands);
388
            System.out.println("\n[Guidance] All tasks are in execution, please wait...");
390     }

392     /**
         * Method that generates all the tasks for imputation and association
394      *
         * @param parsingArgs
396      * @param chrNumber
         * @param lim1
398      * @param lim2
         * @param panelIndex
400      * @param gmapFile
         * @param imputationFilesInfo
402      * @param commonFilesInfo
         * @param listOfCommands
404      */
        private static void makeImputationPerChunk(ParseCmdLine parsingArgs, int chrNumber,
406         int lim1, int lim2, int panelIndex, String gmapFile,
                ImputationFiles imputationFilesInfo, CommonFiles commonFilesInfo,
408             ArrayList<String> listOfCommands) {

410         int chunkSize = parsingArgs.getChunkSize();
            double infoThreshold = parsingArgs.getInfoThreshold();
412
            String knownHapFileName = parsingArgs.getRpanelHapFileName(panelIndex, chrNumber);
```

```
414         String rpanelDir = parsingArgs.getRpanelDir(panelIndex);
            String knownHapFile = rpanelDir + File.separator + knownHapFileName;
416
            String lim1S = Integer.toString(lim1);
418         String lim2S = Integer.toString(lim2);
            String chrS = Integer.toString(chrNumber);
420         String infoThresholdS = Double.toString(infoThreshold);
            String imputationTool = parsingArgs.getImputationTool();
422
            if (imputationTool.equals("impute")) {
424             String legendFileName=parsingArgs.getRpanelLegFileName(panelIndex, chrNumber);
                String legendFile = rpanelDir + File.separator + legendFileName;
426
                String mixedShapeitHapsFile = commonFilesInfo.getShapeitHapsFile(chrNumber);
428             String mixedShapeitSampleFile=commonFilesInfo.getShapeitSampleFile(chrNumber);
                String mixedPairsFile = commonFilesInfo.getPairsFile(chrNumber);
430             String mixedImputeFile = imputationFilesInfo.getImputedFile(panelIndex,
                    chrNumber, lim1, lim2, chunkSize);
432             String mixedImputeFileInfo =imputationFilesInfo.getImputedInfoFile(panelIndex,
                    chrNumber, lim1, lim2, chunkSize);
434             String mixedImputeFileSummary = imputationFilesInfo.getImputedSummaryFile(
                    panelIndex, chrNumber, lim1, lim2, chunkSize);
436             String mixedImputeFileWarnings = imputationFilesInfo.getImputedWarningsFile(
                    panelIndex, chrNumber, lim1, lim2, chunkSize);
438
                String mixedFilteredFile = imputationFilesInfo.getFilteredFile(panelIndex,
440                 chrNumber, lim1, lim2, chunkSize);
                String mixedFilteredLogFile = imputationFilesInfo.getFilteredLogFile(
442                 panelIndex, chrNumber, lim1, lim2, chunkSize);

444             // We create the list of rsId that are greater than or equal to the
                // infoThreshold value
446             String mixedFilteredRsIdFile = imputationFilesInfo.getFilteredRsIdFile(
                    panelIndex, chrNumber, lim1, lim2, chunkSize);
448
                doImputationWithImpute(parsingArgs, listOfCommands, chrS, gmapFile,
450                 knownHapFile, legendFile, mixedShapeitHapsFile,
                        mixedShapeitSampleFile, lim1S, lim2S, mixedPairsFile,
452                     mixedImputeFile, mixedImputeFileInfo, mixedImputeFileSummary,
                        mixedImputeFileWarnings);
454
                doFilterByInfo(parsingArgs, listOfCommands, mixedImputeFileInfo,
456                 mixedFilteredRsIdFile, infoThresholdS);

458             doQctoolS(parsingArgs, listOfCommands, mixedImputeFile,mixedFilteredRsIdFile,
                    mixedFilteredFile, mixedFilteredLogFile);
460         } else if (imputationTool.equals("minimac")) {
                String mixedFilteredHaplotypesFile
462               = commonFilesInfo.getFilteredHaplotypesFile(chrNumber);
                String mixedFilteredHaplotypesSampleFile
464               = commonFilesInfo.getFilteredHaplotypesSampleFile(chrNumber);
                String mixedFilteredListOfSnpsFile
466               = commonFilesInfo.getListOfSnpsFile(chrNumber);

468             String mixedImputedMMFileName = imputationFilesInfo.getImputedMMFile(
                    panelIndex, chrNumber, lim1, lim2, chunkSize);
470             String mixedImputedMMInfoFile = imputationFilesInfo.getImputedMMInfoFile(
                    panelIndex, chrNumber, lim1, lim2, chunkSize);
472             String mixedImputedMMErateFile = imputationFilesInfo.getImputedMMErateFile(
                    panelIndex, chrNumber, lim1, lim2, chunkSize);
474             String mixedImputedMMRecFile = imputationFilesInfo.getImputedMMRecFile(
                    panelIndex, chrNumber, lim1, lim2, chunkSize);
476             String mixedImputedMMDoseFile = imputationFilesInfo.getImputedMMDoseFile(
                    panelIndex, chrNumber, lim1, lim2, chunkSize);
478             String mixedImputedMMLogFile = imputationFilesInfo.getImputedMMLogFile(
                    panelIndex, chrNumber, lim1, lim2, chunkSize);
480
                doImputationWithMinimac(parsingArgs, listOfCommands, knownHapFile,
482                 mixedFilteredHaplotypesFile, mixedFilteredHaplotypesSampleFile,
                    mixedFilteredListOfSnpsFile, mixedImputedMMFileName,
484                 mixedImputedMMInfoFile, mixedImputedMMErateFile,
                    mixedImputedMMRecFile, mixedImputedMMDoseFile, mixedImputedMMLogFile,
```

```
                chrS, lim1S, lim2S);
        } else {
            System.err.println("\t[makeImputationPerChunk]: Error, the imputation tool "
              + imputationTool + " is not allowed...");
            System.exit(1);
        }
    }

    /**
     * Method that generates all the tasks for association
     *
     * @param parsingArgs
     * @param testTypeIndex
     * @param panelIndex
     * @param chrNumber
     * @param lim1
     * @param lim2
     * @param imputationFilesInfo
     * @param commonFilesInfo
     * @param assocFilesInfo
     * @param listOfCommands
     */
    private static void makeAssociationPerChunk(ParseCmdLine parsingArgs,
        int testTypeIndex, int panelIndex, int chrNumber, int lim1,
            int lim2, ImputationFiles imputationFilesInfo, CommonFiles commonFilesInfo,
            AssocFiles assocFilesInfo, ArrayList<String> listOfCommands) {

        int chunkSize = parsingArgs.getChunkSize();
        double mafThreshold = parsingArgs.getMafThreshold();
        double infoThreshold = parsingArgs.getInfoThreshold();
        double hweCohortThreshold = parsingArgs.getHweCohortThreshold();
        double hweCasesThreshold = parsingArgs.getHweCasesThreshold();
        double hweControlsThreshold = parsingArgs.getHweControlsThreshold();

        String responseVar = parsingArgs.getResponseVar(testTypeIndex);
        String covariables = parsingArgs.getCovariables(testTypeIndex);

        String chrS = Integer.toString(chrNumber);

        String mafThresholdS = Double.toString(mafThreshold);
        String infoThresholdS = Double.toString(infoThreshold);
        String hweCohortThresholdS = Double.toString(hweCohortThreshold);
        String hweCasesThresholdS = Double.toString(hweCasesThreshold);
        String hweControlsThresholdS = Double.toString(hweControlsThreshold);

        String snptestOutFile = assocFilesInfo.getSnptestOutFile(testTypeIndex,
            panelIndex, chrNumber, lim1, lim2, chunkSize);
        String snptestLogFile = assocFilesInfo.getSnptestLogFile(testTypeIndex,
            panelIndex, chrNumber, lim1, lim2, chunkSize);

        String mixedSampleFile = commonFilesInfo.getSampleFile(chrNumber);
        String mixedImputeFileInfo = imputationFilesInfo.getImputedInfoFile(panelIndex,
            chrNumber, lim1, lim2, chunkSize);

        String mixedFilteredFile = imputationFilesInfo.getFilteredFile(panelIndex,
            chrNumber, lim1, lim2, chunkSize);

        doSnptest(parsingArgs, listOfCommands, chrS, mixedFilteredFile, mixedSampleFile,
            snptestOutFile, snptestLogFile, responseVar, covariables);

        String summaryFile = assocFilesInfo.getSummaryFile(testTypeIndex, panelIndex,
            chrNumber, lim1, lim2, chunkSize);

        doCollectSummary(parsingArgs, listOfCommands, chrS, mixedImputeFileInfo,
            snptestOutFile, summaryFile, mafThresholdS, infoThresholdS,
                hweCohortThresholdS, hweCasesThresholdS, hweControlsThresholdS);
    }

    /**
     * Method to perform the merging fo chunks for each chromosome
     *
     * @param parsingArgs
```

```
558      * @param listOfCommands
         * @param ttIndex
560      * @param rpanelIndex
         * @param chr
562      * @param minSize
         * @param maxSize
564      * @param chunkSize
         * @param assocFilesInfo
566      * @param mergeFilesInfo
         */
568     private static void makeMergeOfChunks(ParseCmdLine parsingArgs,
            ArrayList<String> listOfCommands, int ttIndex, int rpanelIndex, int chr,
570             int minSize, int maxSize, int chunkSize, AssocFiles assocFilesInfo,
                MergeFiles mergeFilesInfo) {
572
            String theChromo = Integer.toString(chr);
574         int lim1 = minSize;
            int lim2 = lim1 + chunkSize - 1;
576
            int numberOfChunks = maxSize / chunkSize;
578         int module = maxSize % chunkSize;
            if (module != 0) {
580             numberOfChunks++;
            }
582         int indexA = 0;
            int indexC = 0;
584         String reducedA = null;
            String reducedB = null;
586         String reducedC = null;
            for (int processedChunks = 0; processedChunks < 2 * numberOfChunks - 2;
588             processedChunks = processedChunks + 2) {
                if (processedChunks < numberOfChunks) {
590                 reducedA = assocFilesInfo.getSummaryFile(ttIndex, rpanelIndex, chr,
                        lim1, lim2, chunkSize);
592                 lim1 = lim1 + chunkSize;
                    lim2 = lim2 + chunkSize;
594             } else {
                    reducedA=mergeFilesInfo.getReducedFile(ttIndex, rpanelIndex, chr, indexA);
596                 indexA++;
                }
598
                if (processedChunks < numberOfChunks - 1) {
600                 reducedB = assocFilesInfo.getSummaryFile(ttIndex, rpanelIndex, chr,
                        lim1, lim2, chunkSize);
602                 lim1 = lim1 + chunkSize;
                    lim2 = lim2 + chunkSize;
604             } else {
                    reducedB=mergeFilesInfo.getReducedFile(ttIndex, rpanelIndex, chr, indexA);
606                 indexA++;
                }
608             reducedC = mergeFilesInfo.getReducedFile(ttIndex, rpanelIndex, chr, indexC);
                doMergeTwoChunks(parsingArgs, listOfCommands, reducedA, reducedB, reducedC,
610                 theChromo);
                indexC++;
612         } // End of for of chunks
        }
614
        /**
616      * Method to perform the joint of condensed files of each rpanel
         *
618      * @param parsingArgs
         * @param listOfCommands
620      * @param ttIndex
         * @param rpanelIndex
622      * @param startChr
         * @param endChr
624      * @param mergeFilesInfo
         */
626     private static void makeJointCondensedFiles(ParseCmdLine parsingArgs,
            ArrayList<String> listOfCommands, int ttIndex, int rpanelIndex,
628             int startChr, int endChr, MergeFiles mergeFilesInfo) {
```

```
630         int indexA = 0;
            int indexC = 0;
632         String condensedA = null;
            String condensedB = null;
634         String condensedC = null;
            int numberOfChrs = endChr - startChr + 1;
636
            if (numberOfChrs == 1) { // There is only one chr to process.
638             condensedA = mergeFilesInfo.getCondensedFile(ttIndex, rpanelIndex, startChr);
                condensedB = condensedA;
640
                condensedC
642               = mergeFilesInfo.getAdditionalCondensedFile(ttIndex, rpanelIndex, indexC);
                doJointCondenseFiles(parsingArgs, listOfCommands, condensedA, condensedB,
644                 condensedC);
            } else {
646             for (int processedCondensed = 0; processedCondensed < 2 * numberOfChrs - 2;
                    processedCondensed = processedCondensed + 2) {
648                 if (processedCondensed < numberOfChrs) {
                        int i = startChr + processedCondensed;
650                     condensedA = mergeFilesInfo.getCondensedFile(ttIndex, rpanelIndex, i);
                    } else {
652                     condensedA = mergeFilesInfo.getAdditionalCondensedFile(ttIndex,
                            rpanelIndex, indexA);
654                     indexA++;
                    }
656
                    if (processedCondensed < numberOfChrs - 1) {
658                     int i = startChr + processedCondensed + 1;
                        condensedB = mergeFilesInfo.getCondensedFile(ttIndex, rpanelIndex, i);
660                 } else {
                        condensedB = mergeFilesInfo.getAdditionalCondensedFile(ttIndex,
662                         rpanelIndex, indexA);
                        indexA++;
664                 }
                    condensedC = mergeFilesInfo.getAdditionalCondensedFile(ttIndex,
666                     rpanelIndex, indexC);

668                 doJointCondenseFiles(parsingArgs, listOfCommands, condensedA,
                        condensedB, condensedC);
670                 indexC++;
                } // End for
672         }
        }
674
        /**
676      * Method to perform the joint of filteredByAll files of each rpanel
         *
678      * @param parsingArgs
         * @param listOfCommands
680      * @param ttIndex
         * @param rpanelIndex
682      * @param rpanelName
         * @param startChr
684      * @param endChr
         * @param mergeFilesInfo
686      */
        private static void makeJointFilteredByAllFiles(ParseCmdLine parsingArgs,
688         ArrayList<String> listOfCommands, int ttIndex, int rpanelIndex,
            String rpanelName, int startChr, int endChr, MergeFiles mergeFilesInfo) {
690
            int endChrNormal = endChr;
692         if (startChr < endChr) {
                if (endChr != ChromoInfo.MAX_NUMBER_OF_CHROMOSOMES) {
694                 endChrNormal = endChr;
                } else {
696                 endChrNormal = endChr - 1;
                }
698         }

700         int indexA = 0;
            int indexC = 0;
```

```
702         int processedFiltered = 0;
            String rpanelFlag = "NO";
704         String filteredA = null;
            String filteredB = null;
706         String filteredC = null;
            int numberOfChrs = endChrNormal - startChr + 1;
708
            if (numberOfChrs == 1) { // There is only one chr to process.
710             rpanelFlag = "YES";
                filteredA=mergeFilesInfo.getFilteredByAllFile(ttIndex, rpanelIndex, startChr);
712             filteredB = filteredA;

714             filteredC = mergeFilesInfo.getAdditionalFilteredByAllFile(ttIndex,
                    rpanelIndex, indexC);
716             if (startChr != ChromoInfo.MAX_NUMBER_OF_CHROMOSOMES) {
                    doJointFilteredByAllFiles(parsingArgs, listOfCommands, filteredA,
718                     filteredB, filteredC, rpanelName, rpanelFlag);
                }
720         } else {
                for (processedFiltered = 0; processedFiltered < 2 * numberOfChrs - 2;
722                 processedFiltered = processedFiltered + 2) {
                    if (processedFiltered < numberOfChrs) {
724                     int i = startChr + processedFiltered;
                        filteredA = mergeFilesInfo.getFilteredByAllFile(ttIndex,
726                         rpanelIndex, i);
                    } else {
728                     filteredA = mergeFilesInfo.getAdditionalFilteredByAllFile(ttIndex,
                            rpanelIndex, indexA);
730                     indexA++;
                    }
732
                    if (processedFiltered < numberOfChrs - 1) {
734                     int i = startChr + processedFiltered + 1;
                        filteredB=mergeFilesInfo.getFilteredByAllFile(ttIndex, rpanelIndex, i);
736                 } else {
                        filteredB = mergeFilesInfo.getAdditionalFilteredByAllFile(ttIndex,
738                         rpanelIndex, indexA);
                        indexA++;
740                     rpanelFlag = "YES";
                    }
742                 filteredC = mergeFilesInfo.getAdditionalFilteredByAllFile(ttIndex,
                        rpanelIndex, indexC);
744
                    doJointFilteredByAllFiles(parsingArgs, listOfCommands, filteredA,
746                     filteredB, filteredC, rpanelName, rpanelFlag);
                    indexC++;
748             } // End for
            }
750
            // Now we process the chr 23 if this is defined in this execution
752         if (endChr == ChromoInfo.MAX_NUMBER_OF_CHROMOSOMES) {
                rpanelFlag = "YES";
754             filteredA = mergeFilesInfo.getFilteredByAllFile(ttIndex, rpanelIndex, endChr);
                filteredC = mergeFilesInfo.getAdditionalFilteredByAllXFile(ttIndex,
756                 rpanelIndex, 0);

758             doJointFilteredByAllFiles(parsingArgs, listOfCommands, filteredA,
                    filteredA, filteredC, rpanelName, rpanelFlag);
760         }

762     }

764     /**
         * Method that performs the last part of the workflow corresponding to the merging,
766      * combining and drawing of results
         *
768      * @param parsingArgs
         * @param mergeFilesInfo
770      * @param combinedPanelsFilesInfo
         * @param rpanelTypes
772      * @param ttIndex
         * @param listOfCommands
```

```
774      */
     private static void makeCombinePanels(ParseCmdLine parsingArgs,
776          MergeFiles mergeFilesInfo, CombinedPanelsFiles combinedPanelsFilesInfo,
                 List<String> rpanelTypes, int ttIndex, ArrayList<String> listOfCommands) {
778
             int startChr = parsingArgs.getStart();
780          int endChr = parsingArgs.getEnd();
             int endChrNormal = endChr;
782          if (endChr == ChromoInfo.MAX_NUMBER_OF_CHROMOSOMES) {
                 endChrNormal = ChromoInfo.MAX_NUMBER_OF_CHROMOSOMES - 1;
784          }

786          String startChrS = Integer.toString(startChr);
             String endChrS = Integer.toString(endChr);
788          String endChrNormalS = Integer.toString(endChrNormal);

790          String filteredPanelA = null;
             String filteredPanelB = null;
792          String filteredPanelC = null;

794          String filteredXPanelA = null;
             String filteredXPanelB = null;
796          String filteredXPanelC = null;
             boolean refPanelCombine = parsingArgs.getRefPanelCombine();
798
             String combinedCondensedFile = null;
800
             // We have to ask if we have to combine results from different panels or not.
802          if (refPanelCombine == true) {
                 int startingIndex = 0;
804              filteredPanelA = mergeFilesInfo.getFinalFilteredByAllFile(ttIndex,
                     startingIndex);
806
                 if (endChr == ChromoInfo.MAX_NUMBER_OF_CHROMOSOMES) {
808                  filteredXPanelA = mergeFilesInfo.getAdditionalFilteredByAllXFile(ttIndex,
                         startingIndex, 0);
810              }

812              for (int kk = 1; kk < rpanelTypes.size(); kk++) {
                     filteredPanelB = mergeFilesInfo.getFinalFilteredByAllFile(ttIndex, kk);
814                  filteredPanelC = combinedPanelsFilesInfo.getCombinedFilteredByAllFile(
                         ttIndex, kk - 1);
816
                     if (startChr != ChromoInfo.MAX_NUMBER_OF_CHROMOSOMES) {
818                      doCombinePanelsComplex(parsingArgs, listOfCommands, filteredPanelA,
                             filteredPanelB, filteredPanelC, startChrS, endChrNormalS);
820                  }
                     filteredPanelA = filteredPanelC;
822
                     if (endChr == ChromoInfo.MAX_NUMBER_OF_CHROMOSOMES) {
824                      filteredXPanelB = mergeFilesInfo.getAdditionalFilteredByAllXFile(
                             ttIndex, kk, 0);
826                      filteredXPanelC
                           = combinedPanelsFilesInfo.getCombinedFilteredByAllXFile(
828                            ttIndex, kk - 1);

830                      doCombinePanelsComplex(parsingArgs, listOfCommands, filteredXPanelA,
                             filteredXPanelB, filteredXPanelC, endChrS, endChrS);
832                      filteredXPanelA = filteredXPanelC;
                     }
834              }

836              // Now we have to generate the combinedCondensedFile. We do it, by using the
                 // last filteredPanelC and the last filteredXPanelC
838              if (endChr != ChromoInfo.MAX_NUMBER_OF_CHROMOSOMES) {
                     filteredXPanelC = filteredPanelC;
840              }

842              if ((startChr == ChromoInfo.MAX_NUMBER_OF_CHROMOSOMES)) {
                     filteredPanelC = filteredXPanelC;
844              }
```

```
846             combinedCondensedFile
                  = combinedPanelsFilesInfo.getCombinedCondensedFile(ttIndex);
848
                doCombineCondensedFiles(parsingArgs, listOfCommands, filteredPanelC,
850                 filteredXPanelC, combinedCondensedFile);

852             // Finally, we create topHits, QQ and Manhattan plots.
                String topHitsCombinedResults
854               = combinedPanelsFilesInfo.getTopHitsFile(ttIndex);

856             doGenerateTopHits(parsingArgs, listOfCommands, filteredPanelC,
                    filteredXPanelC, topHitsCombinedResults, PVA_THRS);
858
                String combinedQqPlotPdfFile
860               = combinedPanelsFilesInfo.getQqPlotPdfFile(ttIndex);
                String combinedQqPlotTiffFile
862               = combinedPanelsFilesInfo.getQqPlotTiffFile(ttIndex);
                String combinedManhattanPlotPdfFile
864               = combinedPanelsFilesInfo.getManhattanPdfFile(ttIndex);
                String combinedManhattanPlotTiffFile
866               = combinedPanelsFilesInfo.getManhattanTiffFile(ttIndex);
                String combinedCorrectedPvaluesFile
868               = combinedPanelsFilesInfo.getCorrectedPvaluesFile(ttIndex);

870             doGenerateQQManhattanPlots(parsingArgs, listOfCommands, combinedCondensedFile,
                    combinedQqPlotPdfFile, combinedManhattanPlotPdfFile,
872                 combinedQqPlotTiffFile, combinedManhattanPlotTiffFile,
                    combinedCorrectedPvaluesFile);
874
            }
876     }

878     /**
         * Method that performs the last part of the workflow corresponding to the phenome
880      * Analysis, combining the results of each phenoType analysis
         *
882      * @param parsingArgs
         * @param mergeFilesInfo
884      * @param resultsFilesInfo
         * @param phenomeAnalysisFilesInfo
886      * @param rpanelTypes
         * @param listOfCommands
888      */
        private static void makePhenotypeAnalysis(ParseCmdLine parsingArgs,
890         MergeFiles mergeFilesInfo, ResultsFiles resultsFilesInfo,
            PhenomeAnalysisFiles phenomeAnalysisFilesInfo, List<String> rpanelTypes,
892         ArrayList<String> listOfCommands) {

894         int endChr = parsingArgs.getEnd();
            String endChrS = Integer.toString(endChr);
896
            String topHitsFile = null;
898         String filteredByAllFile = null;
            String filteredByAllXFile = null;
900         // String condensedFile = null;
            String phenomeFileA = null;
902         String phenomeFileB = null;
            String phenomeFileC = null;
904
            int numberOfTestTypes = parsingArgs.getNumberOfTestTypeName();
906         int numberOfRpanelsTypes = rpanelTypes.size();

908         // int maxPhenoIndex = numberOfTestTypes * numberOfRpanelsTypes - 1;
            int phenoIndex = 0;
910
            int ttIndex = 0;
912         int rpIndex = 0;
            String ttName = parsingArgs.getTestTypeName(ttIndex);
914         String rpName = rpanelTypes.get(rpIndex);

916         // Lets create the header for the initPhenoMatrix task
            ttIndex = 0;
```

```
        rpIndex = 0;
        ttName = parsingArgs.getTestTypeName(ttIndex);
        rpName = rpanelTypes.get(rpIndex);
        topHitsFile = resultsFilesInfo.getTopHitsFile(ttIndex, rpIndex);
        phenomeFileA = phenomeAnalysisFilesInfo.getPhenotypeIntermediateFile(phenoIndex);
        phenoIndex++;

        doInitPhenoMatrix(parsingArgs, listOfCommands, topHitsFile, ttName, rpName,
            phenomeFileA);
        for (ttIndex = 0; ttIndex < numberOfTestTypes; ttIndex++) {
            int startRp = 0;
            if (ttIndex == 0) {
                startRp = 1;
            }
            ttName = parsingArgs.getTestTypeName(ttIndex);
            for (rpIndex = startRp; rpIndex < numberOfRpanelsTypes; rpIndex++) {
                rpName = rpanelTypes.get(rpIndex);

                topHitsFile = resultsFilesInfo.getTopHitsFile(ttIndex, rpIndex);
                phenomeFileB
                  = phenomeAnalysisFilesInfo.getPhenotypeIntermediateFile(phenoIndex);

                doAddToPhenoMatrix(parsingArgs, listOfCommands, phenomeFileA, topHitsFile,
                    ttName, rpName, phenomeFileB);

                phenomeFileA = phenomeFileB;
                phenoIndex++;
            }
        }

        phenoIndex = 0;
        // Lets do the fillinout of the phenomeAnalysis.
        for (ttIndex = 0; ttIndex < numberOfTestTypes; ttIndex++) {
            ttName = parsingArgs.getTestTypeName(ttIndex);

            for (rpIndex = 0; rpIndex < numberOfRpanelsTypes; rpIndex++) {
                rpName = rpanelTypes.get(rpIndex);

                filteredByAllFile = mergeFilesInfo.getFinalFilteredByAllFile(ttIndex,
                    rpIndex);
                if (endChrS.equals("23")) {
                    filteredByAllXFile
                    = mergeFilesInfo.getAdditionalFilteredByAllXFile(ttIndex, rpIndex, 0);
                } else {
                    filteredByAllXFile = mergeFilesInfo.getFinalFilteredByAllFile(ttIndex,
                        rpIndex);
                }

                phenomeFileB = phenomeAnalysisFilesInfo.getPhenotypeFile(phenoIndex);

                doFilloutPhenoMatrix(parsingArgs, listOfCommands, phenomeFileA,
                    filteredByAllFile, filteredByAllXFile, endChrS, ttName,
                        rpName, phenomeFileB);

                phenoIndex++;
            }
        }

        // Last round to generate final results
        phenoIndex = 0;
        phenomeFileA = phenomeAnalysisFilesInfo.getPhenotypeFile(phenoIndex);

        phenoIndex++;

        // Lets do the finalization of the phenomeAnalysis.
        for (ttIndex = 0; ttIndex < numberOfTestTypes; ttIndex++) {
            int startRp = 0;
            if (ttIndex == 0) {
                startRp = 1;
            }

            ttName = parsingArgs.getTestTypeName(ttIndex);
```

```
990             for (rpIndex = startRp; rpIndex < numberOfRpanelsTypes; rpIndex++) {
                    rpName = rpanelTypes.get(rpIndex);
992                 phenomeFileB = phenomeAnalysisFilesInfo.getPhenotypeFile(phenoIndex);
                    phenomeFileC = phenomeAnalysisFilesInfo.getPhenotypeFinalFile(phenoIndex);
994                 doFinalizePhenoMatrix(parsingArgs, listOfCommands, phenomeFileA,
                        phenomeFileB, ttName, rpName, phenomeFileC);
996                 phenomeFileA = phenomeFileC;
                    phenoIndex++;
998             }
            }
1000
    }
1002
    /**
1004 * Method the wraps the execution of convertFromBedToBed tasks and store the command
     * in the listOfCommands
1006 *
     * @param parsingArgs
1008 * @param listOfCommands
     * @param bedFile
1010 * @param bimFile
     * @param famFile
1012 * @param theChromo
     * @param mixedBedFile
1014 * @param mixedBimFile
     * @param mixedFamFile
1016 * @param mixedBedToBedLogFile
     */
1018 private static void doConvertFromBedToBed(ParseCmdLine parsingArgs,
        ArrayList<String> listOfCommands, String bedFile, String bimFile,
1020        String famFile, String theChromo, String mixedBedFile, String mixedBimFile,
            String mixedFamFile, String mixedBedToBedLogFile) {
1022
        if (parsingArgs.getStageStatus("convertFromBedToBed") == 1) {
1024        // Prepare the parameter values
            String noWebFlag = "--noweb";
1026        String bedPrefix = "--bed";
            String bimPrefix = "--bim";
1028        String famPrefix = "--fam";
            String chromoPrefix = "--chr";
1030        String recodeFlag = "--recode";
            String outPrefix = "--out";
1032        String makeBedFlag = "--make-bed";
            String stdout = mixedBedFile + ".stdout";
1034        String stderr = mixedBedFile + ".stderr";

1036        // Store the command on the list
            String cmdToStore = ENV_VARS.PLINK_BINARY.value() + " " + noWebFlag + " "
1038            + bedPrefix + " " + bedFile + " " + bimPrefix + " "
                    + bimFile + " " + famPrefix + " " + famFile + " " + chromoPrefix
1040                + " " + theChromo + " " + recodeFlag + " " + outPrefix
                    + " " + mixedBedFile + " " + makeBedFlag + " #" + mixedBedFile
1042                + " #" + mixedBimFile + " #" + mixedFamFile + " #"
                    + mixedBedToBedLogFile + " stdout " + stdout + " stderr " + stderr;
1044        listOfCommands.add(cmdToStore);

1046        // Launch the task
            // We do not capture the ExitValue because the task will fail and
1048        // we do not want to synchronize
            BINARY.convertFromBedToBed(noWebFlag, bedPrefix, bedFile, bimPrefix, bimFile,
1050            famPrefix, famFile, chromoPrefix, theChromo, recodeFlag, outPrefix,
                mixedBedFile, makeBedFlag, mixedBimFile, mixedFamFile,
1052            mixedBedToBedLogFile, stdout, stderr);
        }
1054 }

1056 /**
     * Method the wraps the execution of createRsIdList task and store the command in
1058 * the listOfCommands
     *
1060 * @param parsingArgs
     * @param listOfCommands
```

```
1062      * @param mixedBimOrGenFile
          * @param exclCgatFlag
1064      * @param mixedPairsFile
          * @param inputFormat
1066      */
      private static void doCreateRsIdList(ParseCmdLine parsingArgs, ArrayList<String>
1068      listOfCommands, String mixedBimOrGenFile,
              String exclCgatFlag, String mixedPairsFile, String inputFormat) {
1070
          if (parsingArgs.getStageStatus("createRsIdList") == 1) {
1072          // Store the command on the list
              String cmdToStore = ENV_VARS.JAVA_HOME.value() + File.separator
1074              + "java createRsIdList " + mixedBimOrGenFile + " "
                      + exclCgatFlag + " " + mixedPairsFile + " " + inputFormat;
1076          listOfCommands.add(cmdToStore);

1078          // Launch the task
              try {
1080              GuidanceImpl.createRsIdList(mixedBimOrGenFile, exclCgatFlag,
                      mixedPairsFile, inputFormat);
1082          } catch (GuidanceTaskException gte) {
                  // COMPSs: This code is never reached since exception causes task to fail
1084              System.err.println("[Guidance] Exception createRsIdList task");
                  System.err.println(gte.getMessage());
1086          }
          }
1088  }

1090  /**
       * Method that wraps the execution of phasing task with bed input formats and store
1092   * the command in the listOfCommands
       *
1094   * @param parsingArgs
       * @param listOfCommands
1096   * @param theChromo
       * @param bedFile
1098   * @param bimFile
       * @param famFile
1100   * @param gmapFile
       * @param excludedSnpsFile
1102   * @param shapeitHapsFile
       * @param shapeitSampleFile
1104   * @param shapeitLogFile
       * @param filteredHaplotypesFile
1106   * @param filteredHaplotypesSampleFile
       * @param filteredHaplotypesLogFile
1108   * @param filteredHaplotypesVcfFile
       * @param listOfSnpsFile
1110   * @param exclCgatFlag
       * @param exclSVFlag
1112   */
      private static void doPhasingBed(ParseCmdLine parsingArgs, ArrayList<String>
1114      listOfCommands, String theChromo, String bedFile, String bimFile,
          String famFile, String gmapFile, String excludedSnpsFile,
1116      String shapeitHapsFile, String shapeitSampleFile, String shapeitLogFile,
          String filteredHaplotypesFile, String filteredHaplotypesSampleFile,
1118      String filteredHaplotypesLogFile, String filteredHaplotypesVcfFile,
          String listOfSnpsFile, String exclCgatFlag, String exclSVFlag) {
1120
          if (parsingArgs.getStageStatus("phasingBed") == 1) {
1122          // Prepare the parameter values
              String inputBedPrefix = "--input-bed";
1124          String inputMapPrefix = "--input-map";
              String chrXPrefix = ((theChromo.equals("23")) ? "--chrX" : "");
1126          String outputMaxPrefix = " --output-max ";
              String threadPrefix = "--thread";
1128          String effectiveSizePrefix = "--effective-size";
              String outputLogPrefix = "--output-log";
1130          int numThreads = 16;
              int effectiveSize = 20_000;
1132          String shapeitStdOut = shapeitHapsFile + ".stdout";
              String shapeitStdErr = shapeitHapsFile + ".stderr";
```

```
1134
            // Store the command on the list
1136        String cmdToStore = ENV_VARS.SHAPEIT_BINARY.value() + " " + inputBedPrefix
                + " " + bedFile + " " + bimFile + " " + famFile + " "
1138                + inputMapPrefix + " " + gmapFile + " " + chrXPrefix + " "
                + outputMaxPrefix + " " + shapeitHapsFile + " "
1140                + shapeitSampleFile + " " + threadPrefix + " " + numThreads + " "
                + effectiveSizePrefix + " " + effectiveSize + " "
1142                + outputLogPrefix + " " + shapeitLogFile + " stdout " + shapeitStdOut
                    + " stderr " + shapeitStdErr;
1144
            listOfCommands.add(cmdToStore);
1146
            // Launch the task
1148        // We do not capture the ExitValue because the task will fail and
            // we do not want to synchronize
1150        BINARY.phasingBed(inputBedPrefix, bedFile, bimFile, famFile, inputMapPrefix,
                gmapFile, chrXPrefix, outputMaxPrefix, shapeitHapsFile,
1152            shapeitSampleFile, threadPrefix, numThreads, effectiveSizePrefix,
                effectiveSize, outputLogPrefix, shapeitLogFile, shapeitStdOut,
1154            shapeitStdErr);
        }
1156
        if (parsingArgs.getStageStatus("createListOfExcludedSnps") == 1) {
1158        // Store the command on the list
            String cmdToStore = ENV_VARS.JAVA_BINARY.value() + " "
1160            + "createListOfExcludedSnps.jar" + " " + shapeitHapsFile + " "
                    + excludedSnpsFile + " " + exclCgatFlag + " " + exclSVFlag;
1162        listOfCommands.add(cmdToStore);

1164        // Launch the task
            try {
1166            GuidanceImpl.createListOfExcludedSnps(shapeitHapsFile, excludedSnpsFile,
                    exclCgatFlag, exclSVFlag);
1168        } catch (GuidanceTaskException gte) {
                // COMPSs: This code is never reached since exception causes task to fail
1170            System.err.println("[Guidance] Exception createListOfExcludedSnps task");
                System.err.println(gte.getMessage());
1172        }
        }
1174
        if (parsingArgs.getStageStatus("filterHaplotypes") == 1) {
1176        // Prepare the parameter values
            String convertFlag = "-convert";
1178        String inputHapsPrefix = "--input-haps";
            String excludeSnpPrefix = "--exclude-snp";
1180        String outputHapsPrefix = "--output-haps";
            String outputLogPrefix = "--output-log";
1182        String outputVcfPrefix = "--output-vcf";

1184        // Store the command on the list
            String cmdToStore = ENV_VARS.SHAPEIT_BINARY.value() + " " + convertFlag + " "
1186            + inputHapsPrefix + " " + shapeitHapsFile + " " + shapeitSampleFile
                + " " + excludeSnpPrefix + " " + excludedSnpsFile + " "
1188            + outputHapsPrefix + " " + filteredHaplotypesFile + " "
                + filteredHaplotypesSampleFile + " " + outputLogPrefix + " "
1190            + filteredHaplotypesLogFile + " " + outputVcfPrefix
                + filteredHaplotypesVcfFile;
1192        listOfCommands.add(cmdToStore);

1194        // Launch the task
            try {
1196            BINARY.filterHaplotypes(convertFlag, inputHapsPrefix, shapeitHapsFile,
                    shapeitSampleFile, excludeSnpPrefix, excludedSnpsFile,
1198                outputHapsPrefix, filteredHaplotypesFile,
                    filteredHaplotypesSampleFile, outputLogPrefix,
1200                    filteredHaplotypesLogFile, outputVcfPrefix,
                        filteredHaplotypesVcfFile);
1202
                GuidanceImpl.postFilterHaplotypes(filteredHaplotypesFile, listOfSnpsFile);
1204        } catch (GuidanceTaskException gte) {
                // COMPSs: This code is never reached since exception causes task to fail
```

```
1206                System.err.println("[Guidance] Exception createListOfExcludedSnps task");
                    System.err.println(gte.getMessage());
1208            }
            }
1210    }

1212    /**
         * Method that wraps the execution of phasing task with gen input formats and store
1214     * the command in the listOfCommands
         *
1216     * @param parsingArgs
         * @param listOfCommands
1218     * @param theChromo
         * @param genFile
1220     * @param sampleFile
         * @param gmapFile
1222     * @param excludedSnpsFile
         * @param shapeitHapsFile
1224     * @param shapeitSampleFile
         * @param shapeitLogFile
1226     * @param filteredHaplotypesFile
         * @param filteredHaplotypesSampleFile
1228     * @param filteredHaplotypesLogFile
         * @param filteredHaplotypesVcfFile
1230     * @param listOfSnpsFile
         * @param exclCgatFlag
1232     * @param exclSVFlag
         */
1234    private static void doPhasing(ParseCmdLine parsingArgs, ArrayList<String>
            listOfCommands, String theChromo, String genFile, String sampleFile,
1236        String gmapFile, String excludedSnpsFile, String shapeitHapsFile,
            String shapeitSampleFile, String shapeitLogFile, String filteredHaplotypesFile,
1238        String filteredHaplotypesSampleFile, String filteredHaplotypesLogFile,
                String filteredHaplotypesVcfFile, String listOfSnpsFile, String exclCgatFlag,
1240            String exclSVFlag) {

1242        if (parsingArgs.getStageStatus("pashing") == 1) {
                // Prepare the parameter values
1244            String inputGenPrefix = "--input-gen";
                String inputMapPrefix = "--input-map";
1246            String chromoFlag = ((theChromo.equals("23")) ? "--chrX" : "");
                String outputPrefix = " --output-max ";
1248            String threadPrefix = "--thread";
                String effectiveSizePrefix = "--effective-size";
1250            String outputLogPrefix = "--output-log";
                int numThreads = 16;
1252            int effectiveSize = 20_000;
                String shapeitStdOut = shapeitHapsFile + ".stdout";
1254            String shapeitStdErr = shapeitHapsFile + ".stderr";

1256            // Store the command on the list
                String cmdToStore = ENV_VARS.SHAPEIT_BINARY.value() + " " + inputGenPrefix
1258                + " " + genFile + " " + sampleFile + " "
                        + inputMapPrefix + " " + gmapFile + " " + chromoFlag + " "
1260                + outputPrefix + " " + shapeitHapsFile + " "
                        + shapeitSampleFile + " " + threadPrefix + " " + numThreads + " "
1262                + effectiveSizePrefix + " " + effectiveSize + " "
                        + outputLogPrefix + " " + shapeitLogFile + " stdout " + shapeitStdOut
1264                    + " stderr " + shapeitStdErr;
                listOfCommands.add(cmdToStore);
1266
                // Launch the task
1268            BINARY.phasing(inputGenPrefix, genFile, sampleFile, inputMapPrefix, gmapFile,
                    chromoFlag, outputPrefix, shapeitHapsFile, shapeitSampleFile,
1270                threadPrefix, numThreads, effectiveSizePrefix, effectiveSize,
                    outputLogPrefix, shapeitLogFile, shapeitStdOut, shapeitStdErr);
1272
            }
1274
            if (parsingArgs.getStageStatus("createListOfExcludedSnps") == 1) {
1276            // Store the command on the list
                String cmdToStore = ENV_VARS.JAVA_HOME.value() + File.separator
```

```
1278                + "java createListOfExcludedSnps.jar " + shapeitHapsFile + " "
                        + excludedSnpsFile + " " + exclCgatFlag + " " + exclSVFlag;
1280
                listOfCommands.add(cmdToStore);
1282
                // Launch the task
1284            try {
                    GuidanceImpl.createListOfExcludedSnps(shapeitHapsFile, excludedSnpsFile,
1286                    exclCgatFlag, exclSVFlag);
                } catch (GuidanceTaskException gte) {
1288                // COMPSs: This code is never reached since exception causes task to fail
                    System.err.println("[Guidance] Exception createListOfExcludedSnps task");
1290                System.err.println(gte.getMessage());
                }
1292        }

1294        if (parsingArgs.getStageStatus("filterHaplotypes") == 1) {
                // Prepare the parameter values
1296            String convertFlag = "-convert";
                String inputHapsPrefix = "--input-haps";
1298            String excludeSnpPrefix = "--exclude-snp";
                String outputHapsPrefix = "--output-haps";
1300            String outputLogPrefix = "--output-log";
                String outputVcfPrefix = "--output-vcf";
1302
                // Store the command on the list
1304            String cmdToStore = ENV_VARS.SHAPEIT_BINARY.value() + " " + convertFlag + " "
                    + inputHapsPrefix + " " + shapeitHapsFile + " " + shapeitSampleFile
1306                + " " + excludeSnpPrefix + " " + excludedSnpsFile + " "
                    + outputHapsPrefix + " " + filteredHaplotypesFile + " "
1308                + filteredHaplotypesSampleFile + " " + outputLogPrefix + " "
                    + filteredHaplotypesLogFile + " " + outputVcfPrefix
1310                + filteredHaplotypesVcfFile;
                listOfCommands.add(cmdToStore);
1312
                // Launch the task
1314            try {
                    BINARY.filterHaplotypes(convertFlag, inputHapsPrefix, shapeitHapsFile,
1316                    shapeitSampleFile, excludeSnpPrefix, excludedSnpsFile,
                        outputHapsPrefix, filteredHaplotypesFile,
1318                    filteredHaplotypesSampleFile, outputLogPrefix,
                            filteredHaplotypesLogFile, outputVcfPrefix,
1320                        filteredHaplotypesVcfFile);
                    GuidanceImpl.postFilterHaplotypes(filteredHaplotypesFile, listOfSnpsFile);
1322            } catch (GuidanceTaskException gte) {
                    // COMPSs: This code is never reached since exception causes task to fail
1324                System.err.println("[Guidance] Exception createListOfExcludedSnps task");
                    System.err.println(gte.getMessage());
1326            }
            }
1328    }

1330    /**
         * Method that wraps the execution of impute task and store the command in the
1332     * listOfCommands
         *
1334     * @param parsingArgs
         * @param listOfCommands
1336     * @param chrS
         * @param gmapFile
1338     * @param knownHapFile
         * @param legendFile
1340     * @param shapeitHapsFile
         * @param shapeitSampleFile
1342     * @param lim1S
         * @param lim2S
1344     * @param pairsFile
         * @param imputeFile
1346     * @param imputeFileInfo
         * @param imputeFileSummary
1348     * @param imputeFileWarnings
         */
```

```
1350    private static void doImputationWithImpute(ParseCmdLine parsingArgs, ArrayList<String>
            listOfCommands, String chrS, String gmapFile, String knownHapFile,
1352        String legendFile, String shapeitHapsFile, String shapeitSampleFile,
            String lim1S, String lim2S, String pairsFile, String imputeFile,
1354        String imputeFileInfo, String imputeFileSummary, String imputeFileWarnings) {

1356        if (parsingArgs.getStageStatus("imputeWithImpute") == 1) {
            // Prepare the parameter values
1358        String prephasedFlag = "use_prephased_g";
            String mPrefix = "-m";
1360        String hPrefix = "-h";
            String lPrefix = "-l";
1362        String knownHapsPrefix = "-known_haps_g";
            String samplePrefix = "-sample_g";
1364        String intPrefix = "-int";
            String chromoFlag = ((chrS.equals("23")) ? "-chrX" : "");
1366        String exludeSnpsPrefix = "-exclude_snps_g";
            String imputeExcludedFlag = "-impute_excluded";
1368        String nePrefix = "-Ne";
            int ne = 20_000;
1370        String oPrefix = "-o";
            String iPrefix = "-i";
1372        String rPrefix = "-r";
            String wPrefix = "-w";
1374        String noSampleQCFlag = "-no_sample_qc_info";
            String oGzFlag = "-o_gz";
1376        String stdOutFile = imputeFile + ".stdout";
            String stdErrorFile = imputeFile + ".stderr";
1378
            // Store the command on the list
1380        String cmdToStore = ENV_VARS.IMPUTE2_BINARY.value() + " " + prephasedFlag
                + " " + mPrefix + " " + gmapFile + " " + hPrefix + " " + knownHapFile
1382            + " " + lPrefix + " " + legendFile + " " + knownHapsPrefix + " "
                + shapeitHapsFile + " " + samplePrefix + " " + shapeitSampleFile + " "
1384            + intPrefix + " " + lim1S + " " + lim2S + " " + chromoFlag + " "
                + exludeSnpsPrefix + " " + pairsFile + " " + imputeExcludedFlag + " "
1386            + nePrefix + " " + ne + " " + oPrefix + " " + imputeFile + " "
                + iPrefix + " " + imputeFileInfo + " " + rPrefix + " "
1388            + imputeFileSummary + wPrefix + " " + imputeFileWarnings + " "
                    + noSampleQCFlag + " " + oGzFlag + " stdout " + stdOutFile
1390                + " stderr " + stdErrorFile;

1392        listOfCommands.add(cmdToStore);

1394        // Launch task
            BINARY.imputeWithImpute(prephasedFlag, mPrefix, gmapFile, hPrefix,
1396            knownHapFile, lPrefix, legendFile, knownHapsPrefix,
                    shapeitHapsFile, samplePrefix, shapeitSampleFile, intPrefix,
1398                lim1S, lim2S, chromoFlag, exludeSnpsPrefix, pairsFile,
                    imputeExcludedFlag, nePrefix, ne, oPrefix, imputeFile, iPrefix,
1400                imputeFileInfo, rPrefix, imputeFileSummary, wPrefix,
                    imputeFileWarnings, noSampleQCFlag, oGzFlag, stdOutFile,
1402                stdErrorFile);

1404    }
    }
1406
    /**
1408 * Method that wraps the execution of Imputation task with Minimac2 and store the
     * command in the listOfCommands
1410 *
     * @param parsingArgs
1412 * @param listOfCommands
     * @param knownHapFile
1414 * @param filteredHapsFile
     * @param filteredSampleFile
1416 * @param filteredListOfSnpsFile
     * @param imputedMMFileName
1418 * @param imputedMMInfoFile
     * @param imputedMMErateFile
1420 * @param imputedMMRecFile
     * @param imputedMMDoseFile
```

```
1422      * @param imputedMMLogFile
          * @param chrS
1424      * @param lim1S
          * @param lim2S
1426      */
      private static void doImputationWithMinimac(ParseCmdLine parsingArgs,
1428          ArrayList<String> listOfCommands, String knownHapFile,
              String filteredHapsFile, String filteredSampleFile,
1430          String filteredListOfSnpsFile, String imputedMMFileName,
              String imputedMMInfoFile, String imputedMMErateFile, String imputedMMRecFile,
1432          String imputedMMDoseFile, String imputedMMLogFile,
              String chrS, String lim1S, String lim2S) {
1434
          if (parsingArgs.getStageStatus("imputeWithMinimac") == 1) {
1436          // Prepare the parameter values
              String vcfReferenceFlag = "--vcfReference";
1438          String refHapsPrefix = "--refHaps";
              String snpsPrefix = "--snps";
1440          String shapeHapsPrefix = "--shape_haps";
              String samplePrefix = "--sample";
1442          String vcfStartPrefix = "--vcfstart";
              String vcfEndPrefix = "--vcfend";
1444          String chromoFlag = chrS.equals(MAX_NUMBER_OF_CHROMOSOMES_STR) ? "--chr" : "";
              String vcfWindowPrefix = "--vcfwindow";
1446          int vcfWindow = 250_000;
              String roundsPrefix = "--rounds";
1448          int rounds = 5;
              String statesPrefix = "--states";
1450          int states = 200;
              String prefixPrefix = "--prefix";
1452          String gzipFlag = "--gzip";
              String stdErrFile = imputedMMFileName + ".stderr";
1454
              // Store the command on the list
1456          String cmdToStore = ENV_VARS.MINIMAC_BINARY.value() + " " + vcfReferenceFlag
                  + " " + refHapsPrefix + " " + knownHapFile + " " + snpsPrefix + " "
1458              + filteredListOfSnpsFile + " " + shapeHapsPrefix + " "
                  + filteredHapsFile + " " + samplePrefix + " " + filteredSampleFile
1460              + " " + vcfStartPrefix + " " + lim1S + " " + vcfEndPrefix + " "
                  + lim2S + " " + chromoFlag + " " + chrS + " " + vcfWindowPrefix + " "
1462              + vcfWindow + " " + roundsPrefix + " " + rounds + " " + statesPrefix
                  + " " + states + " " + prefixPrefix + " " + imputedMMFileName + " "
1464              + gzipFlag + " # " + imputedMMInfoFile + " # " + imputedMMErateFile
                      + " # " + imputedMMRecFile + " # " + imputedMMDoseFile + " stdout "
1466              + imputedMMLogFile + " stderr " + stdErrFile;
              listOfCommands.add(cmdToStore);
1468
              // Launch task
1470          BINARY.imputeWithMinimac(vcfReferenceFlag, refHapsPrefix, knownHapFile,
                  snpsPrefix, filteredListOfSnpsFile, shapeHapsPrefix,
1472                    filteredHapsFile, samplePrefix, filteredSampleFile, vcfStartPrefix,
                      lim1S, vcfEndPrefix, lim2S, chromoFlag, chrS, vcfWindowPrefix,
1474                  vcfWindow, roundsPrefix, rounds, statesPrefix, states, prefixPrefix,
                      imputedMMFileName, gzipFlag, imputedMMInfoFile, imputedMMErateFile,
1476                  imputedMMRecFile, imputedMMDoseFile, imputedMMLogFile, stdErrFile);
          }
1478  }

1480  /**
       * Method that wraps the execution of filterByInfo task and store the command in
1482   * the listOfCommands
       *
1484   * @param parsingArgs
       * @param listOfCommands
1486   * @param imputeFileInfo
       * @param filteredRsIdFile
1488   * @param infoThresholdS
       */
1490  private static void doFilterByInfo(ParseCmdLine parsingArgs, ArrayList<String>
              listOfCommands, String imputeFileInfo, String filteredRsIdFile,
1492          String infoThresholdS) {
```

```
1494        if (parsingArgs.getStageStatus("filterByInfo") == 1) {
            // We create the list of rsId that are greater or equal to the infoThreshold
1496        // Store the command on the list
            String cmdToStore = ENV_VARS.JAVA_HOME.value() + File.separator
1498            + "java filterByInfo " + imputeFileInfo + " " + filteredRsIdFile
                    + " " + infoThresholdS;
1500        listOfCommands.add(cmdToStore);

1502        // Launch the task
            try {
1504            GuidanceImpl.filterByInfo(imputeFileInfo, filteredRsIdFile,
                    infoThresholdS);
1506        } catch (GuidanceTaskException gte) {
                // COMPSs: This code is never reached since exception causes task to fail
1508            System.err.println("[Guidance] Exception filterByInfo task for controls");
                System.err.println(gte.getMessage());
1510        }
        }
1512    }

1514    /**
         * Method that wraps the execution of qctoolS task and store the command in the
1516     * listOfCommands
         *
1518     * @param parsingArgs
         * @param listOfCommands
1520     * @param imputeFile
         * @param filteredRsIdFile
1522     * @param filteredFile
         * @param filteredLogFile
1524     */
    private static void doQctoolS(ParseCmdLine parsingArgs, ArrayList<String>
1526        listOfCommands, String imputeFile, String filteredRsIdFile,
            String filteredFile, String filteredLogFile) {
1528
        double mafThreshold = parsingArgs.getMafThreshold();
1530    String mafThresholdS = Double.toString(mafThreshold);

1532    if (parsingArgs.getStageStatus("qctoolS") == 1) {
            // Prepare the parameter values
1534        String gPrefix = "-g";
            String ogPrefix = "-og";
1536        String inclRSIdsPrefix = "-incl-rsids";
            String omitChromosomeFlag = "-omit-chromosome";
1538        String forceFlag = "-force";
            String logPrefix = "-log";
1540        String mafPrefix = "-maf";
            String oneFlag = "1";
1542        String stdOutFile = filteredFile + ".stdout";
            String stdErrorFile = filteredFile + ".stderr";
1544        String filteredFileGz = filteredFile + ".gz";

1546        // Store the command on the list
            String cmdToStore = ENV_VARS.QCTOOL_BINARY.value() + " " + gPrefix + " "
1548            + imputeFile + " " + ogPrefix + " " + filteredFile + " "
                + inclRSIdsPrefix + " " + filteredRsIdFile + " " + omitChromosomeFlag
1550            + " " + forceFlag + " " + logPrefix + " " + filteredLogFile + " "
                + mafPrefix + " " + mafThresholdS + " " + oneFlag + " stdout "
1552            + stdOutFile + " stderr " + stdErrorFile;
            listOfCommands.add(cmdToStore);
1554
            // Launch the task
1556        BINARY.qctoolS(gPrefix, imputeFile, ogPrefix, filteredFile, inclRSIdsPrefix,
                filteredRsIdFile, omitChromosomeFlag, forceFlag, logPrefix,
1558            filteredLogFile, mafPrefix, mafThresholdS, oneFlag, stdOutFile,
                stdErrorFile);
1560
            // Launch the post zip task
1562        try {
                GuidanceImpl.postSnptest(filteredFile, filteredFileGz);
1564        } catch (GuidanceTaskException gte) {
                // COMPSs: This code is never reached since exception causes task to fail
```

```
1566                System.err.println("[Guidance] Exception on zip post SNP Test files");
                    System.err.println(gte.getMessage());
1568            }
            }
1570    }

1572    /**
         * Method that wraps the execution of snptest task and store the command in the
1574     * listOfCommands
         *
1576     * @param parsingArgs
         * @param listOfCommands
1578     * @param chrS
         * @param mergedGenFile
1580     * @param mergedSampleFile
         * @param snptestOutFile
1582     * @param snptestLogFile
         * @param responseVar
1584     * @param covariables
         */
1586    private static void doSnptest(ParseCmdLine parsingArgs, ArrayList<String>
            listOfCommands, String chrS, String mergedGenFile, String mergedSampleFile,
1588        String snptestOutFile, String snptestLogFile, String responseVar,
            String covariables) {
1590
            if (parsingArgs.getStageStatus("snptest") == 1) {
1592            // Prepare the parameter values
                String covarsSTR = covariables.replace(',', ' ');
1594            String dataPrefix = "-data";
                String oPrefix = "-0";
1596            String phenoPrefix = "-pheno";
                String covarsFlag = covariables.equals("none") ? "":" -cov_names "+ covarsSTR;
1598            String hweFlag = "-hwe";
                String logPrefix = "-log";
1600            final String chromo23Flags
                  = "-method newml -assume_chromosome X -stratify_on sex -frequentist 1";
1602            final String chromoOtherFlags = "-method em -frequentist 1 2 3 4 5";
                String chromoFlag = chrS.equals(ChromoInfo.MAX_NUMBER_OF_CHROMOSOMES_STR)
1604                ? chromo23Flags : chromoOtherFlags;
                String stdOutFile = snptestLogFile + ".stdout";
1606            String stdErrFile = snptestLogFile + ".stderr";
                String snpTestOutFileGz = snptestOutFile + ".gz";
1608
                // Store the command on the list
1610            String cmdToStore = ENV_VARS.SNPTEST_BINARY.value() + dataPrefix + " "
                    + mergedGenFile + " " + mergedSampleFile + " " + oPrefix
1612                      + " " + snptestOutFile + " " + phenoPrefix + " " + responseVar
                          + " " + covarsFlag + " " + hweFlag + " " + logPrefix
1614                      + " " + snptestLogFile + " " + chromoFlag + " " + stdOutFile
                          + " " + stdErrFile;
1616            listOfCommands.add(cmdToStore);

1618            // Launch the task
                BINARY.snptest(dataPrefix, mergedGenFile, mergedSampleFile, oPrefix,
1620                snptestOutFile, phenoPrefix, responseVar, covarsFlag, hweFlag,
                    logPrefix, snptestLogFile, chromoFlag, stdOutFile, stdErrFile);
1622
                // Launch the post zip task
1624            try {
                    GuidanceImpl.postSnptest(snptestOutFile, snpTestOutFileGz);
1626            } catch (GuidanceTaskException gte) {
                    // COMPSs: This code is never reached since exception causes task to fail
1628                System.err.println("[Guidance] Exception on zip post SNP Test files");
                    System.err.println(gte.getMessage());
1630            }
            }
1632    }

1634    /**
         * Method that wraps the execution of collectSummary task and store the command in
1636     * the listOfCommands
         *
```

```
1638      * @param parsingArgs
          * @param listOfCommands
1640      * @param chrS
          * @param imputeFileInfo
1642      * @param snptestOutFile
          * @param summaryFile
1644      * @param mafThresholdS
          * @param infoThresholdS
1646      * @param hweCohortThresholdS
          * @param hweCasesThresholdS
1648      * @param hweControlsThresholdS
          */
1650     private static void doCollectSummary(ParseCmdLine parsingArgs, ArrayList<String>
              listOfCommands, String chrS, String imputeFileInfo, String snptestOutFile,
1652          String summaryFile, String mafThresholdS, String infoThresholdS,
              String hweCohortThresholdS, String hweCasesThresholdS,
1654          String hweControlsThresholdS) {

1656          if (parsingArgs.getStageStatus("collectSummary") == 1) {
                  // Submitting the collect_summary task per this chunk
1658
                  // Store the command on the list
1660              String cmdToStore = ENV_VARS.JAVA_HOME.value() + File.separator
                      + "java collectSummary " + chrS + " " + imputeFileInfo + " "
1662                       + snptestOutFile + " " + summaryFile + " " + mafThresholdS + " "
                      + infoThresholdS + " " + hweCohortThresholdS + " "
1664                       + hweCasesThresholdS + " " + hweControlsThresholdS;

1666              listOfCommands.add(cmdToStore);

1668              // Launch the task
                  try {
1670                  GuidanceImpl.collectSummary(chrS, imputeFileInfo, snptestOutFile,
                          summaryFile, mafThresholdS, infoThresholdS,
1672                           hweCohortThresholdS, hweCasesThresholdS, hweControlsThresholdS);
                  } catch (GuidanceTaskException gte) {
1674                  // COMPSs: This code is never reached since exception causes task to fail
                      System.err.println("[Guidance] Exception collectSummary task");
1676                  System.err.println(gte.getMessage());
                  }
1678          }
          }
1680
          /**
1682       * Method that wraps the jointCondensedFiles task and store the command in
           * the listOfCommands
1684       *
           * @param parsingArgs
1686       * @param listOfCommands
           * @param condensedA
1688       * @param condensedB
           * @param condensedC
1690       */
          private static void doJointCondenseFiles(ParseCmdLine parsingArgs,
1692          ArrayList<String> listOfCommands, String condensedA,
              String condensedB, String condensedC) {
1694
          if (parsingArgs.getStageStatus("jointCondensedFiles") == 1) {
1696          // Store the command on the list
              String cmdToStore = ENV_VARS.JAVA_HOME.value() + File.separator
1698              + "java jointCondensedFiles " + condensedA + " " + condensedB
                      + " " + condensedC;
1700          listOfCommands.add(cmdToStore);

1702          // Launch the task
              try {
1704              GuidanceImpl.jointCondensedFiles(condensedA, condensedB, condensedC);
              } catch (GuidanceTaskException gte) {
1706              // COMPSs: This code is never reached since exception causes task to fail
                  System.err.println("[Guidance] Exception jointCondensedFiles task");
1708              System.err.println(gte.getMessage());
              }
```

```
1710        }
    }
1712
    /**
     * Method that wraps the doJointFilteredByAllFiles task and store the command in
1714  * the listOfCommands
     *
1716  * @param parsingArgs
     * @param listOfCommands
1718  * @param filteredByAllA
     * @param filteredByAllB
1720  * @param filteredByAllC
     * @param rpanelName
1722  * @param rpanelFlag
     */
1724  private static void doJointFilteredByAllFiles(ParseCmdLine parsingArgs,
        ArrayList<String> listOfCommands, String filteredByAllA,
1726          String filteredByAllB, String filteredByAllC, String rpanelName,
            String rpanelFlag) {

1728        if (parsingArgs.getStageStatus("jointFilteredByAllFiles") == 1) {
            // Store the command on the list
1730          String cmdToStore = ENV_VARS.JAVA_HOME.value() + File.separator
                + "java jointFilteredByAllFiles " + filteredByAllA + " "
1732                  + filteredByAllB + " " + filteredByAllC + " " + rpanelName
                    + " " + rpanelFlag;
1734          listOfCommands.add(cmdToStore);

1736          // Launch the task
            try {
1738            GuidanceImpl.jointFilteredByAllFiles(filteredByAllA, filteredByAllB,
                    filteredByAllC, rpanelName, rpanelFlag);
1740          } catch (GuidanceTaskException gte) {
              // COMPSs: This code is never reached since exception causes task to fail
1742            System.err.println("[Guidance] Exception getBestSnps task");
                System.err.println("The error message here is " + gte.getMessage());
1744          }
        }
1746  }

1748  /**
     * Method that wraps the generateTopHits task and store the command in
1750  * the listOfCommands
     *
1752  * @param parsingArgs
     * @param listOfCommands
1754  * @param filteredFile
     * @param filteredXFile
1756  * @param topHitsResults
     * @param pvaThrS
1758  */
    private static void doGenerateTopHits(ParseCmdLine parsingArgs,
1760      ArrayList<String> listOfCommands, String filteredFile,
          String filteredXFile, String topHitsResults, String pvaThrS) {

1762      if (parsingArgs.getStageStatus("generateTopHits") == 1) {
          // Store the command on the list
1764        String cmdToStore = ENV_VARS.JAVA_HOME.value() + File.separator
              + "java generateTopHits " + filteredFile + " " + filteredXFile
1766                + " " + topHitsResults + " " + pvaThrS;
          listOfCommands.add(cmdToStore);

1768        // Launch the task
          try {
1770          GuidanceImpl.generateTopHitsAll(filteredFile, filteredXFile,
                  topHitsResults, pvaThrS);
1772        } catch (GuidanceTaskException gte) {
            // COMPSs: This code is never reached since exception causes task to fail
1774          System.err.println("[Guidance] Exception generateTopHits task");
              System.err.println(gte.getMessage());
1776        }
      }
```

```
1782    }
1784    /**
         * Method that wraps the generateQQManhattanPlots task and store the command in
1786     * the listOfCommands
         *
1788     * @param parsingArgs
         * @param listOfCommands
1790     * @param condensedFile
         * @param qqPlotFile
1792     * @param manhattanPlotFile
         * @param qqPlotTiffFile
1794     * @param manhattanPlotTiffFile
         * @param correctedPvaluesFile
1796     */
    private static void doGenerateQQManhattanPlots(ParseCmdLine parsingArgs,
1798        ArrayList<String> listOfCommands, String condensedFile,
            String qqPlotFile, String manhattanPlotFile, String qqPlotTiffFile,
1800        String manhattanPlotTiffFile, String correctedPvaluesFile) {

1802        if (parsingArgs.getStageStatus("generateQQManhattanPlots") == 1) {
            // Prepare the parameter values
1804        String stdOutFile = correctedPvaluesFile + ".stdout";
            String stdErrFile = correctedPvaluesFile + ".stderr";
1806
            // Store the command on the list
1808        String cmdToStore = ENV_VARS.R_SCRIPT_BIN_DIR.value() + File.separator
                + "Rscript " + ENV_VARS.R_SCRIPT_DIR.value() + File.separator
1810            + "qqplot_manhattan.R " + condensedFile + " " + qqPlotFile + " "
                + manhattanPlotFile + " " + qqPlotTiffFile + " "
1812            + manhattanPlotTiffFile + " " + correctedPvaluesFile + " stdout "
                + stdOutFile + " stderr " + stdErrFile;
1814        listOfCommands.add(cmdToStore);

1816        // Launch the unzip task
            String condensedFileUnzip = condensedFile + ".temp1";
1818        try {
                GuidanceImpl.preGenerateQQManhattanPlots(condensedFile,
1820                condensedFileUnzip);
            } catch (GuidanceTaskException gte) {
1822            // COMPSs: This code is never reached since exception causes task to fail
                System.err.println("[Guidance] Exception QQ Manhattan Plot files");
1824            System.err.println(gte.getMessage());
            }
1826
            // Launch the task
1828        BINARY.generateQQManhattanPlots(condensedFileUnzip, qqPlotFile,
                manhattanPlotFile, qqPlotTiffFile, manhattanPlotTiffFile,
1830                correctedPvaluesFile, stdOutFile, stdErrFile);
        }
1832    }

1834    /**
         * Method that wraps the combinePanelsComplex task and store the command in the
1836     * listOfCommands
         *
1838     * @param parsingArgs
         * @param listOfCommands
1840     * @param resultsPanelA
         * @param resultsPanelB
1842     * @param lastResultFile
         * @param startChrS
1844     * @param endChrS
         */
1846    private static void doCombinePanelsComplex(ParseCmdLine parsingArgs,
        ArrayList<String> listOfCommands, String resultsPanelA, String resultsPanelB,
1848    String lastResultFile, String startChrS, String endChrS) {

1850        if (parsingArgs.getStageStatus("combinePanelsComplex") == 1) {
            // Store the command on the list
1852        String cmdToStore = ENV_VARS.JAVA_HOME.value() + File.separator
                + "java combinePanelsComplex " + resultsPanelA + " " + resultsPanelB
```

```
1854                + " " + lastResultFile + " " + startChrS + " " + endChrS;
                listOfCommands.add(cmdToStore);
1856
                // Launch the task
1858            try {
                    GuidanceImpl.combinePanelsComplex(resultsPanelA, resultsPanelB,
1860                    lastResultFile, startChrS, endChrS);
                } catch (GuidanceTaskException gte) {
1862                // COMPSs: This code is never reached since exception causes task to fail
                    System.err.println("[Guidance] Exception combinePanelsComplex task");
1864                System.err.println(gte.getMessage());
                }
1866        }
    }
1868
    /**
1870 * Method that wraps the doCombineCondensedFiles task and store the command in
     * the listOfCommands
1872 *
     * @param parsingArgs
1874 * @param listOfCommands
     * @param filteredA
1876 * @param filteredX
     * @param combinedCondensedFile
1878 */
    private static void doCombineCondensedFiles(ParseCmdLine parsingArgs,
1880        ArrayList<String> listOfCommands, String filteredA,
            String filteredX, String combinedCondensedFile) {
1882
        double mafThreshold = parsingArgs.getMafThreshold();
1884    double infoThreshold = parsingArgs.getInfoThreshold();
        double hweCohortThreshold = parsingArgs.getHweCohortThreshold();
1886    double hweCasesThreshold = parsingArgs.getHweCasesThreshold();
        double hweControlsThreshold = parsingArgs.getHweControlsThreshold();
1888
        String mafThresholdS = Double.toString(mafThreshold);
1890    String infoThresholdS = Double.toString(infoThreshold);
        String hweCohortThresholdS = Double.toString(hweCohortThreshold);
1892    String hweCasesThresholdS = Double.toString(hweCasesThreshold);
        String hweControlsThresholdS = Double.toString(hweControlsThreshold);
1894
        if (parsingArgs.getStageStatus("combineCondensedFiles") == 1) {
1896        // Store the command on the list
            String cmdToStore = ENV_VARS.JAVA_HOME.value() + File.separator
1898            + "java combineCondensedFiles " + filteredA + " " + filteredX
                    + " " + combinedCondensedFile + " " + mafThresholdS + " "
1900            + infoThresholdS + " " + hweCohortThresholdS + " "
                    + hweCasesThresholdS + " " + hweControlsThresholdS;
1902        listOfCommands.add(cmdToStore);

1904        // Launch the task
            try {
1906            GuidanceImpl.combineCondensedFiles(filteredA, filteredX,
                    combinedCondensedFile, mafThresholdS, infoThresholdS,
1908                hweCohortThresholdS, hweCasesThresholdS, hweControlsThresholdS);
            } catch (GuidanceTaskException gte) {
1910            // COMPSs: This code is never reached since exception causes task to fail
                System.err.println("[Guidance] Exception combineCondensedFile task");
1912            System.err.println(gte.getMessage());
            }
1914    }
    }
1916
    /**
1918 * Method that wraps the mergeTwoChunks task and store the command in
     * the listOfCommands
1920 *
     * @param parsingArgs
1922 * @param listOfCommands
     * @param reduceA
1924 * @param reduceB
     * @param reduceC
```

```
1926      * @param theChromo
          */
1928     private static void doMergeTwoChunks(ParseCmdLine parsingArgs,
             ArrayList<String> listOfCommands, String reduceA, String reduceB,
1930             String reduceC, String theChromo) {

1932         if (parsingArgs.getStageStatus("mergeTwoChunks") == 1) {
                 // Store the command on the list
1934             String cmdToStore = ENV_VARS.JAVA_HOME.value() + File.separator
                     + "java mergeTwoChunks " + reduceA + " " + reduceB + " "
1936                         + reduceC + " " + theChromo;
                 listOfCommands.add(cmdToStore);
1938
                 // Launch the task
1940             try {
                     GuidanceImpl.mergeTwoChunks(reduceA, reduceB, reduceC, theChromo);
1942             } catch (GuidanceTaskException gte) {
                     // COMPSs: This code is never reached since exception causes task to fail
1944                 System.err.println("[Guidance] Exception mergeTwoChunks task");
                     System.err.println(gte.getMessage());
1946             }
             }
1948     }

1950     /**
          * Method that wraps the filterByAll task and store the command in the listOfCommands
1952      *
          * @param parsingArgs
1954      * @param listOfCommands
          * @param inputFile
1956      * @param outputFile
          * @param outputCondensedFile
1958      */
         private static void doFilterByAll(ParseCmdLine parsingArgs,
1960         ArrayList<String> listOfCommands, String inputFile, String outputFile,
             String outputCondensedFile) {
1962
         double mafThreshold = parsingArgs.getMafThreshold();
1964     double infoThreshold = parsingArgs.getInfoThreshold();
         double hweCohortThreshold = parsingArgs.getHweCohortThreshold();
1966     double hweCasesThreshold = parsingArgs.getHweCasesThreshold();
         double hweControlsThreshold = parsingArgs.getHweControlsThreshold();
1968
         String mafThresholdS = Double.toString(mafThreshold);
1970     String infoThresholdS = Double.toString(infoThreshold);
         String hweCohortThresholdS = Double.toString(hweCohortThreshold);
1972     String hweCasesThresholdS = Double.toString(hweCasesThreshold);
         String hweControlsThresholdS = Double.toString(hweControlsThreshold);
1974
         // Task
1976     if (parsingArgs.getStageStatus("filterByAll") == 1) {
             // Store the command on the list
1978         String cmdToStore = ENV_VARS.JAVA_HOME.value() + File.separator
                 + "java filterByAll " + inputFile + " " + outputFile + " "
1980                 + outputCondensedFile + " " + mafThresholdS + " " + infoThresholdS
                     + " " + hweCohortThresholdS + " " + hweCasesThresholdS + " "
1982                 + hweControlsThresholdS;
             listOfCommands.add(cmdToStore);
1984
             // Launch the task
1986         try {
                 GuidanceImpl.filterByAll(inputFile, outputFile, outputCondensedFile,
1988                 mafThresholdS, infoThresholdS, hweCohortThresholdS,
                         hweCasesThresholdS, hweControlsThresholdS);
1990         } catch (GuidanceTaskException gte) {
                 // COMPSs: This code is never reached since exception causes task to fail
1992             System.err.println("[Guidance] Exception filterByAll task");
                 System.err.println(gte.getMessage());
1994         }
         }
1996     }
```

```
1998     /**
         * Method that wraps the initPhenoMatrix task and store the command in the
2000     * listOfCommands
         *
2002     * @param parsingArgs
         * @param listOfCommands
2004     * @param topHitsFile
         * @param ttName
2006     * @param rpName
         * @param phenomeFile
2008     */
        private static void doInitPhenoMatrix(ParseCmdLine parsingArgs,
2010         ArrayList<String> listOfCommands, String topHitsFile, String ttName,
                String rpName, String phenomeFile) {
2012
            if (parsingArgs.getStageStatus("initPhenoMatrix") == 1) {
2014             // Store the command on the list
                String cmdToStore = ENV_VARS.JAVA_HOME.value() + File.separator
2016                 + "java initPhenoMatrix " + topHitsFile + " " + ttName + " "
                        + rpName + " " + phenomeFile;
2018             listOfCommands.add(cmdToStore);

2020             // Launch the task
                try {
2022                 GuidanceImpl.initPhenoMatrix(topHitsFile, ttName, rpName, phenomeFile);
                } catch (GuidanceTaskException gte) {
2024                 // COMPSs: This code is never reached since exception causes task to fail
                    System.err.println("[Guidance] Exception initPhenoMatrix task");
2026                 System.err.println(gte.getMessage());
                }
2028         }
        }
2030
        /**
2032     * Method that wraps the addToPhenoMatrix task and store the command in the
         * listOfCommands
2034     *
         * @param parsingArgs
2036     * @param listOfCommands
         * @param phenomeFileA
2038     * @param topHitsFile
         * @param ttName
2040     * @param rpName
         * @param phenomeFileB
2042     */
        private static void doAddToPhenoMatrix(ParseCmdLine parsingArgs,
2044         ArrayList<String> listOfCommands, String phenomeFileA,
                String topHitsFile, String ttName, String rpName, String phenomeFileB) {
2046
            if (parsingArgs.getStageStatus("addToPhenoMatrix") == 1) {
2048             // Store the command on the list
                String cmdToStore = ENV_VARS.JAVA_HOME.value() + File.separator
2050                 + "java addToPhenoMatrix " + phenomeFileA + " " + topHitsFile
                        + " " + ttName + " " + rpName + " " + phenomeFileB;
2052             listOfCommands.add(cmdToStore);

2054             // Launch the task
                try {
2056                 GuidanceImpl.addToPhenoMatrix(phenomeFileA, topHitsFile, ttName,
                        rpName, phenomeFileB);
2058             } catch (GuidanceTaskException gte) {
                    // COMPSs: This code is never reached since exception causes task to fail
2060                 System.err.println("[Guidance] Exception addToPhenoMatrix task");
                    System.err.println(gte.getMessage());
2062             }
            }
2064     }

2066     /**
         * Method that wraps the filloutPhenoMatrix task and store the command in
2068     * the listOfCommands
         *
```

```
2070      * @param parsingArgs
          * @param listOfCommands
2072      * @param phenomeFileA
          * @param filteredByAllFile
2074      * @param filteredByAllXFile
          * @param endChrS
2076      * @param ttName
          * @param rpName
2078      * @param phenomeFileB
          */
2080     private static void doFilloutPhenoMatrix(ParseCmdLine parsingArgs,
             ArrayList<String> listOfCommands, String phenomeFileA,
2082             String filteredByAllFile, String filteredByAllXFile, String endChrS,
                 String ttName, String rpName, String phenomeFileB) {
2084
             if (parsingArgs.getStageStatus("filloutPhenoMatrix") == 1) {
2086             // Store the command on the list
                 String cmdToStore = ENV_VARS.JAVA_HOME.value() + File.separator
2088                 + "java filloutPhenoMatrix " + phenomeFileA + " " + filteredByAllFile
                     + " " + filteredByAllXFile + " " + endChrS + " " + ttName + " "
2090                 + rpName + " " + phenomeFileB;
                 listOfCommands.add(cmdToStore);
2092
                 // Launch the task
2094             try {
                     GuidanceImpl.filloutPhenoMatrix(phenomeFileA, filteredByAllFile,
2096                     filteredByAllXFile, endChrS, ttName, rpName, phenomeFileB);
                 } catch (GuidanceTaskException gte) {
2098                 // COMPSs: This code is never reached since exception causes task to fail
                     System.err.println("[Guidance] Exception filloutPhenoMatrix task");
2100                 System.err.println(gte.getMessage());
                 }
2102         }
         }
2104
         /**
2106      * Method that wraps the finalizePhenoMatrix task and store the command in the
          * listOfCommands
2108      *
          * @param parsingArgs
2110      * @param listOfCommands
          * @param phenomeFileA
2112      * @param phenomeFileB
          * @param ttName
2114      * @param rpName
          * @param phenomeFileC
2116      */
         private static void doFinalizePhenoMatrix(ParseCmdLine parsingArgs,
2118         ArrayList<String> listOfCommands, String phenomeFileA,
                 String phenomeFileB, String ttName, String rpName, String phenomeFileC) {
2120
             if (parsingArgs.getStageStatus("finalizePhenoMatrix") == 1) {
2122             // Store the command on the list
                 String cmdToStore = ENV_VARS.JAVA_HOME.value() + File.separator
2124                 + "java finalizePhenoMatrix " + phenomeFileA + " "
                         + phenomeFileB + " " + ttName + " " + rpName + " " + phenomeFileC;
2126             listOfCommands.add(cmdToStore);

2128             // Launch the task
                 try {
2130                 GuidanceImpl.finalizePhenoMatrix(phenomeFileA, phenomeFileB, ttName,
                         rpName, phenomeFileC);
2132             } catch (GuidanceTaskException gte) {
                     // COMPSs: This code is never reached since exception causes task to fail
2134                 System.err.println("[Guidance] Exception finalizePhenoMatrix task");
                     System.err.println(gte.getMessage());
2136             }
             }
2138     }

2140     /**
          * Method to set the final status of each file that has been generated in the
```

```
2142      * execution So far, the three main status are: uncompressed, compressed, and deleted.
          * The default status is: uncompressed. That is the initial status assigned
2144      * to each file
          *
2146      * @param parsingArgs
          * @param commonFilesInfo
2148      */
        private static void setFinalStatusForCommonFiles(ParseCmdLine parsingArgs,
2150        CommonFiles commonFilesInfo) {
            // String removeTempFiles = parsingArgs.getRemoveTemporalFiles();
2152        String compressFiles = parsingArgs.getCompressFiles();
            String finalStatus = null;
2154        if (compressFiles.equals("NO")) {
                finalStatus = "uncompressed";
2156        } else {
                finalStatus = "compressed";
2158        }

2160        int startChr = parsingArgs.getStart();
            int endChr = parsingArgs.getEnd();
2162        for (int i = startChr; i <= endChr; i++) {
                commonFilesInfo.setPairsFileFinalStatus(i, finalStatus);
2164            commonFilesInfo.setShapeitHapsFileFinalStatus(i, finalStatus);
                commonFilesInfo.setShapeitSampleFileFinalStatus(i, finalStatus);
2166        }
        }
2168
        /**
2170     * Method to set the final status of each file that has been generated in the
         * execution So far, the three main status are: uncompressed, compressed, and deleted.
2172     * The default status is: uncompressed. That is the initial status assigned
         * to each file
2174     *
         * @param parsingArgs
2176     * @param imputationFilesInfo
         * @param generalChromoInfo
2178     * @param refPanels
         */
2180    private static void setFinalStatusForImputationFiles(ParseCmdLine parsingArgs,
            ImputationFiles imputationFilesInfo,
2182            ChromoInfo generalChromoInfo, List<String> refPanels) {

2184        int startChr = parsingArgs.getStart();
            int endChr = parsingArgs.getEnd();
2186        int chunkSize = parsingArgs.getChunkSize();
            String imputationTool = parsingArgs.getImputationTool();
2188
            // String removeTempFiles = parsingArgs.getRemoveTemporalFiles();
2190        String compressFiles = parsingArgs.getCompressFiles();
            String finalStatus = null;
2192        if (compressFiles.equals("NO")) {
                finalStatus = "uncompressed";
2194        } else {
                finalStatus = "compressed";
2196        }

2198        if (imputationTool.equals("impute")) {
                for (int j = 0; j < refPanels.size(); j++) {
2200                // String rPanel = refPanels.get(j);
                    for (int chromo = startChr; chromo <= endChr; ++chromo) {
2202                    int lim1 = 1;
                        int lim2 = lim1 + chunkSize - 1;
2204                    int numberOfChunks = generalChromoInfo.getMaxSize(chromo)/chunkSize;
                        int module = generalChromoInfo.getMaxSize(chromo) % chunkSize;
2206                    if (module != 0)
                            numberOfChunks++;
2208
                        for (int k = 0; k < numberOfChunks; k++) {
2210                        imputationFilesInfo.setImputedFileFinalStatus(j, chromo, lim1,
                                lim2, chunkSize, finalStatus);
2212                        imputationFilesInfo.setFilteredFileFinalStatus(j, chromo, lim1,
                                lim2, chunkSize, finalStatus);
```

```
                        imputationFilesInfo.setImputedInfoFileFinalStatus(j, chromo, lim1,
                            lim2, chunkSize, finalStatus);

                        lim1 = lim1 + chunkSize;
                        lim2 = lim2 + chunkSize;
                    }
                }
            }
        } else if (imputationTool.equals("minimac")) {
            for (int j = 0; j < refPanels.size(); j++) {
                // String rPanel = refPanels.get(j);
                for (int chromo = startChr; chromo <= endChr; ++chromo) {
                    int lim1 = 1;
                    int lim2 = lim1 + chunkSize - 1;
                    int numberOfChunks = generalChromoInfo.getMaxSize(chromo) / chunkSize;
                    int module = generalChromoInfo.getMaxSize(chromo) % chunkSize;
                    if (module != 0)
                        numberOfChunks++;

                    for (int k = 0; k < numberOfChunks; k++) {
                        imputationFilesInfo.setImputedMMInfoFileFinalStatus(j, chromo,
                            lim1, lim2, chunkSize, finalStatus);
                        lim1 = lim1 + chunkSize;
                        lim2 = lim2 + chunkSize;
                    }
                }
            }
        }

    }

    /**
     * Method to set the final status of each file that has been generated in the
     * execution. So far, the three main status are: uncompressed, compressed, and deleted
     * The default status is: uncompressed. That is the initial status assigned
     * to each file
     *
     * @param parsingArgs
     * @param assocFilesInfo
     * @param generalChromoInfo
     * @param refPanels
     */
    private static void setFinalStatusForAssocFiles(ParseCmdLine parsingArgs,
        AssocFiles assocFilesInfo, ChromoInfo generalChromoInfo,
            List<String> refPanels) {

        int startChr = parsingArgs.getStart();
        int endChr = parsingArgs.getEnd();
        int chunkSize = parsingArgs.getChunkSize();

        // String removeTempFiles = parsingArgs.getRemoveTemporalFiles();
        String compressFiles = parsingArgs.getCompressFiles();
        String finalStatus = null;
        if (compressFiles.equals("NO")) {
            finalStatus = "uncompressed";
        } else {
            finalStatus = "compressed";
        }

        // Now we continue with the association
        int numberOfTestTypes = parsingArgs.getNumberOfTestTypeName();
        for (int tt = 0; tt < numberOfTestTypes; tt++) {
            for (int j = 0; j < refPanels.size(); j++) {
                // String rPanel = refPanels.get(j);
                for (int chromo = startChr; chromo <= endChr; ++chromo) {
                    int maxSize = generalChromoInfo.getMaxSize(chromo);
                    int total_chunks = maxSize / chunkSize;
                    int lim1 = 1;
                    int lim2 = lim1 + chunkSize - 1;

                    for (int k = 0; k < total_chunks; k++) {
                        assocFilesInfo.setSnptestOutFileFinalStatus(tt, j, chromo,
```

```
2286                            lim1, lim2, chunkSize, finalStatus);
                            assocFilesInfo.setSummaryFileFinalStatus(tt, j, chromo,
2288                            lim1, lim2, chunkSize, finalStatus);

2290                        lim1 = lim1 + chunkSize;
                            lim2 = lim2 + chunkSize;
2292                    }
                    }
2294            }
        }
2296    }

2298 }
```

D.2 GuidanceItf.java

```
    package guidance;
2
    import guidance.utils.Environment;
4   import integratedtoolkit.types.annotations.Constraints;
    import integratedtoolkit.types.annotations.task.Binary;
6   import integratedtoolkit.types.annotations.task.Method;
    import integratedtoolkit.types.annotations.Parameter;
8   import integratedtoolkit.types.annotations.parameter.Direction;
    import integratedtoolkit.types.annotations.parameter.Stream;
10  import integratedtoolkit.types.annotations.parameter.Type;

12
    public interface GuidanceItf {
14
        @Binary(binary = "$" + Environment.EV_PLINKBINARY )
16      @Constraints(computingUnits = "1", memorySize = "1.0")
        Integer convertFromBedToBed(
18          @Parameter(type = Type.STRING, direction = Direction.IN) String webFlag,
            @Parameter(type = Type.STRING, direction = Direction.IN) String bedPrefix,
20          @Parameter(type = Type.FILE, direction = Direction.IN) String bedFile,
            @Parameter(type = Type.STRING, direction = Direction.IN) String bimPrefix,
22          @Parameter(type = Type.FILE, direction = Direction.IN) String bimFile,
            @Parameter(type = Type.STRING, direction = Direction.IN) String famPrefix,
24          @Parameter(type = Type.FILE, direction = Direction.IN) String famFile,
            @Parameter(type = Type.STRING, direction = Direction.IN) String chromoPrefix,
26          @Parameter(type = Type.STRING, direction = Direction.IN) String chromo,
            @Parameter(type = Type.STRING, direction = Direction.IN) String recodeFlag,
28          @Parameter(type = Type.STRING, direction = Direction.IN) String outPrefix,
            @Parameter(type = Type.FILE, direction = Direction.OUT)String newBedFile,
30          @Parameter(type = Type.STRING, direction = Direction.IN) String makeBedFlag,
            @Parameter(type = Type.FILE, direction = Direction.OUT, prefix = "#")
32          String newBimFile,
            @Parameter(type = Type.FILE, direction = Direction.OUT, prefix = "#")
34          String newFamFile,
            @Parameter(type = Type.FILE, direction = Direction.OUT, prefix = "#")
36          String newLogFile,
            @Parameter(type = Type.FILE, direction = Direction.OUT, stream = Stream.STDOUT)
38          String stdOutputFile,
            @Parameter(type = Type.FILE, direction = Direction.OUT, stream = Stream.STDERR)
40          String stdErrorFile
        );
42
        @Binary(binary = "$" + Environment.EV_SHAPEITBINARY)
44      @Constraints(computingUnits="16", memorySize = "20.0f")
        Integer phasingBed(
46          @Parameter(type = Type.STRING, direction = Direction.IN) String inputBedPrefix,
            @Parameter(type = Type.FILE, direction = Direction.IN) String bedFile,
48          @Parameter(type = Type.FILE, direction = Direction.IN) String bumFile,
            @Parameter(type = Type.FILE, direction = Direction.IN) String famFile,
50          @Parameter(type = Type.STRING, direction = Direction.IN) String inputMapPrefix,
            @Parameter(type = Type.FILE, direction = Direction.IN) String gmapFile,
52          @Parameter(type = Type.STRING, direction = Direction.IN) String chromoFlag,
```

```
        @Parameter(type = Type.STRING, direction = Direction.IN) String outputPrefix,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String shapeitHapsFile,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String shapeitSampleFile,
        @Parameter(type = Type.STRING, direction = Direction.IN) String threadPrefix,
        @Parameter(type = Type.INT, direction = Direction.IN) int numThreads,
        @Parameter(type = Type.STRING, direction = Direction.IN)
        String effectiveSizePrefix,
        @Parameter(type = Type.INT, direction = Direction.IN) int effectiveSize,
        @Parameter(type = Type.STRING, direction = Direction.IN) String outputLogPrefix,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String shapeitLogFile,
        @Parameter(type = Type.FILE, direction = Direction.OUT, stream = Stream.STDOUT)
        String stdOutputFile,
        @Parameter(type = Type.FILE, direction = Direction.OUT, stream = Stream.STDERR)
        String stdErrorFile
    );

    @Binary(binary = "$" + Environment.EV_SHAPEITBINARY)
    @Constraints(computingUnits="16", memorySize = "20.0f")
    Integer phasing(
        @Parameter(type = Type.STRING, direction = Direction.IN) String inputGenPrefix,
        @Parameter(type = Type.FILE, direction = Direction.IN) String inputGenFile,
        @Parameter(type = Type.FILE, direction = Direction.IN) String inputSampleFile,
        @Parameter(type = Type.STRING, direction = Direction.IN) String inputMapPrefix,
        @Parameter(type = Type.FILE, direction = Direction.IN) String gmapFile,
        @Parameter(type = Type.STRING, direction = Direction.IN) String chromoFlag,
        @Parameter(type = Type.STRING, direction = Direction.IN) String outputPrefix,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String shapeitHapsFile,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String shapeitSampleFile,
        @Parameter(type = Type.STRING, direction = Direction.IN) String threadPrefix,
        @Parameter(type = Type.INT, direction = Direction.IN) int numThreads,
        @Parameter(type = Type.STRING, direction = Direction.IN)
        String effectiveSizePrefix,
        @Parameter(type = Type.INT, direction = Direction.IN) int effectiveSize,
        @Parameter(type = Type.STRING, direction = Direction.IN) String outputLogPrefix,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String shapeitLogFile,
        @Parameter(type = Type.FILE, direction = Direction.OUT, stream = Stream.STDOUT)
        String stdOutFile,
        @Parameter(type = Type.FILE, direction = Direction.OUT, stream = Stream.STDERR)
        String stdErrorFile
    );

    @Binary(binary = "$" + Environment.EV_IMPUTE2BINARY)
    @Constraints(computingUnits="1", memorySize = "14.0f")
    Integer imputeWithImpute(
        @Parameter(type = Type.STRING, direction = Direction.IN) String prephasedFlag,
        @Parameter(type = Type.STRING, direction = Direction.IN) String mPrefix,
        @Parameter(type = Type.FILE, direction = Direction.IN) String gmapFile,
        @Parameter(type = Type.STRING, direction = Direction.IN) String hPrefix,
        @Parameter(type = Type.FILE, direction = Direction.IN) String knownHapFile,
        @Parameter(type = Type.STRING, direction = Direction.IN) String lPrefix,
        @Parameter(type = Type.FILE, direction = Direction.IN) String legendFile,
        @Parameter(type = Type.STRING, direction = Direction.IN) String knownHapsPrefix,
        @Parameter(type = Type.FILE, direction = Direction.IN) String shapeitHapsFile,
        @Parameter(type = Type.STRING, direction = Direction.IN) String samplePrefix,
        @Parameter(type = Type.FILE, direction = Direction.IN) String shapeitSampleFile,
        @Parameter(type = Type.STRING, direction = Direction.IN) String intPrefix,
        @Parameter(type = Type.STRING, direction = Direction.IN) String lim1S,
        @Parameter(type = Type.STRING, direction = Direction.IN) String lim2S,
        @Parameter(type = Type.STRING, direction = Direction.IN) String chromoFlag,
        @Parameter(type = Type.STRING, direction = Direction.IN) String exludeSnpsPrefix,
        @Parameter(type = Type.FILE, direction = Direction.IN) String pairsFile,
        @Parameter(type = Type.STRING, direction = Direction.IN)
        String imputeExcludedFlag,
        @Parameter(type = Type.STRING, direction = Direction.IN) String nePrefix,
        @Parameter(type = Type.INT, direction = Direction.IN) int ne,
        @Parameter(type = Type.STRING, direction = Direction.IN) String oPrefix,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String imputeFile,
        @Parameter(type = Type.STRING, direction = Direction.IN) String iPrefix,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String imputeFileInfo,
        @Parameter(type = Type.STRING, direction = Direction.IN) String rPrefix,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String imputeFileSummary,
        @Parameter(type = Type.STRING, direction = Direction.IN) String wPrefix,
```

```
        @Parameter(type = Type.FILE, direction = Direction.OUT) String imputeFileWarnings,
        @Parameter(type = Type.STRING, direction = Direction.IN) String noSampleQCFlag,
        @Parameter(type = Type.STRING, direction = Direction.IN) String oGzFlag,
        @Parameter(type = Type.FILE, direction = Direction.OUT, stream = Stream.STDOUT)
        String stdOutFile,
        @Parameter(type = Type.FILE, direction = Direction.OUT, stream = Stream.STDERR)
        String stdErrorFile
    );

    @Binary(binary = "$" + Environment.EV_MINIMACBINARY)
    @Constraints(computingUnits="1", memorySize = "3.0f")
    Integer imputeWithMinimac(
        @Parameter(type = Type.STRING, direction = Direction.IN) String vcfReferenceFlag,
        @Parameter(type = Type.STRING, direction = Direction.IN) String refHapsPrefix,
        @Parameter(type = Type.FILE, direction = Direction.IN) String knownHapFile,
        @Parameter(type = Type.STRING, direction = Direction.IN) String snpsPrefix,
        @Parameter(type = Type.FILE, direction = Direction.IN)
        String filteredListOfSnpsFile,
        @Parameter(type = Type.STRING, direction = Direction.IN) String shapeHapsPrefix,
        @Parameter(type = Type.FILE, direction = Direction.IN) String filteredHapsFile,
        @Parameter(type = Type.STRING, direction = Direction.IN) String samplePrefix,
        @Parameter(type = Type.FILE, direction = Direction.IN) String filteredSampleFile,
        @Parameter(type = Type.STRING, direction = Direction.IN) String vcfStartPrefix,
        @Parameter(type = Type.STRING, direction = Direction.IN) String lim1S,
        @Parameter(type = Type.STRING, direction = Direction.IN) String vcfEndPrefix,
        @Parameter(type = Type.STRING, direction = Direction.IN) String lim2S,
        @Parameter(type = Type.STRING, direction = Direction.IN) String chromoFlag,
        @Parameter(type = Type.STRING, direction = Direction.IN) String chromo,
        @Parameter(type = Type.STRING, direction = Direction.IN) String vcfWindowPrefix,
        @Parameter(type = Type.INT, direction = Direction.IN) int vcfWindow,
        @Parameter(type = Type.STRING, direction = Direction.IN) String roundsPrefix,
        @Parameter(type = Type.INT, direction = Direction.IN) int rounds,
        @Parameter(type = Type.STRING, direction = Direction.IN) String statesPrefix,
        @Parameter(type = Type.INT, direction = Direction.IN) int states,
        @Parameter(type = Type.STRING, direction = Direction.IN) String prefixPrefix,
        @Parameter(type = Type.STRING, direction = Direction.IN) String imputedMMFileName,
        @Parameter(type = Type.STRING, direction = Direction.IN) String gzipFlag,
        @Parameter(type = Type.FILE, direction = Direction.OUT, prefix = "#")
        String imputedMMInfoFile,
        @Parameter(type = Type.FILE, direction = Direction.OUT, prefix = "#")
        String imputedMMErateFile,
        @Parameter(type = Type.FILE, direction = Direction.OUT, prefix = "#")
        String imputedMMRecFile,
        @Parameter(type = Type.FILE, direction = Direction.OUT, prefix = "#")
        String imputedMMDoseFile,
        @Parameter(type = Type.FILE, direction = Direction.OUT, stream = Stream.STDOUT)
        String stdOutFile,
        @Parameter(type = Type.FILE, direction = Direction.OUT, stream = Stream.STDERR)
        String stdErrFile
    );

    @Method(declaringClass = "guidance.GuidanceImpl")
    @Constraints(computingUnits="1", memorySize = "14.0f")
    void preGenerateQQManhattanPlots(
        @Parameter(type = Type.FILE, direction = Direction.IN) String condensedFile,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String condensedFileUnzip
    );

    @Binary(binary = "$" + Environment.EV_RSCRIPTBINDIR
        + "/Rscript" + Environment.EV_RSCRIPTDIR + "/qqplot_manhattan.R")
    @Constraints(computingUnits="1", memorySize = "14.0f")
    Integer generateQQManhattanPlots(
        @Parameter(type = Type.FILE, direction = Direction.IN) String inputFile,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String qqPlotFile,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String manhattanPlotFile,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String qqPlotTiffFile,
        @Parameter(type = Type.FILE, direction = Direction.OUT)
        String manhattanPlotTiffFile,
        @Parameter(type = Type.FILE, direction = Direction.OUT)
        String correctedPvaluesFile,
        @Parameter(type = Type.FILE, direction = Direction.OUT, stream = Stream.STDOUT)
        String stdOutFile,
```

```
        @Parameter(type = Type.FILE, direction = Direction.OUT, stream = Stream.STDERR)
198     String stdErrorFile
    );
200
    @Binary(binary = "$" + Environment.EV_SNPTESTBINARY)
202 @Constraints(computingUnits="1", memorySize = "2.0f")
    Integer snptest(
204         @Parameter(type = Type.STRING, direction = Direction.IN) String dataPrefix,
            @Parameter(type = Type.FILE, direction = Direction.IN) String mergedGenFile,
206         @Parameter(type = Type.FILE, direction = Direction.IN)
            String mergedSampleFile,
208         @Parameter(type = Type.STRING, direction = Direction.IN) String oPrefix,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String snpTestOutFile,
210         @Parameter(type = Type.STRING, direction = Direction.IN) String phenoPrefix,
            @Parameter(type = Type.STRING, direction = Direction.IN) String responseVar,
212         @Parameter(type = Type.STRING, direction = Direction.IN) String covarsFlag,
            @Parameter(type = Type.STRING, direction = Direction.IN) String hweFlag,
214         @Parameter(type = Type.STRING, direction = Direction.IN) String logPrefix,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String snpTestLogFile,
216         @Parameter(type = Type.STRING, direction = Direction.IN) String chromoFlag,
            @Parameter(type = Type.FILE, direction = Direction.OUT,
218         stream = Stream.STDOUT) String stdOutFile,
            @Parameter(type = Type.FILE, direction = Direction.OUT,
220         stream = Stream.STDERR) String stdErrorFile
    );
222
    @Method(declaringClass = "guidance.GuidanceImpl")
224 @Constraints(computingUnits="1", memorySize = "2.0f")
    void postSnptest(
226     @Parameter(type = Type.FILE, direction = Direction.IN) String snpTestOutFile,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String snpTestOutFileZip
228 );

230 @Binary(binary = "$" + Environment.EV_QCTOOLBINARY)
    @Constraints(computingUnits="1", memorySize = "2.0f")
232 Integer qctoolS(
        @Parameter(type = Type.STRING, direction = Direction.IN) String gPrefix,
234     @Parameter(type = Type.FILE, direction = Direction.IN) String imputeFile,
        @Parameter(type = Type.STRING, direction = Direction.IN) String ogPrefix,
236     @Parameter(type = Type.FILE, direction = Direction.OUT) String filteredFile,
        @Parameter(type = Type.STRING, direction = Direction.IN) String inclRSIdsPrefix,
238     @Parameter(type = Type.FILE, direction = Direction.IN) String inclusionRsIdFile,
        @Parameter(type = Type.STRING, direction = Direction.IN)
240     String omitChromosomeFlag,
        @Parameter(type = Type.STRING, direction = Direction.IN) String forceFlag,
242     @Parameter(type = Type.STRING, direction = Direction.IN) String logPrefix,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String filteredLogFile,
244     @Parameter(type = Type.STRING, direction = Direction.IN) String mafPrefix,
        @Parameter(type = Type.STRING, direction = Direction.IN) String mafThresholdS,
246     @Parameter(type = Type.STRING, direction = Direction.IN) String oneFlag,
        @Parameter(type = Type.FILE, direction = Direction.OUT, stream = Stream.STDOUT)
248     String stdOutFile,
        @Parameter(type = Type.FILE, direction = Direction.OUT, stream = Stream.STDERR)
250     String stdErrorFile
    );
252
    @Method(declaringClass = "guidance.GuidanceImpl")
254 @Constraints(computingUnits="1", memorySize = "2.0f")
    void postQCTool(
256     @Parameter(type = Type.FILE, direction = Direction.IN) String filteredFile,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String filteredFileGz
258 );

260 @Binary(binary = "$" + Environment.EV_SHAPEITBINARY)
    @Constraints(computingUnits="1", memorySize = "4.0f")
262 void filterHaplotypes(
        @Parameter(type = Type.STRING, direction = Direction.IN) String convertFlag,
264     @Parameter(type = Type.STRING, direction = Direction.IN) String inputHapsPrefix,
        @Parameter(type = Type.FILE, direction = Direction.IN) String hapsFile,
266     @Parameter(type = Type.FILE, direction = Direction.IN) String sampleFile,
        @Parameter(type = Type.STRING, direction = Direction.IN) String exludeSnpPrefix,
268     @Parameter(type = Type.FILE, direction = Direction.IN) String excludedSnpsFile,
```

```
        @Parameter(type = Type.STRING, direction = Direction.IN) String outputHapsPrefix,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String filteredHapsFile,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String filteredSampleFile,
        @Parameter(type = Type.STRING, direction = Direction.IN) String outputLogPrefix,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String filteredLogFile,
        @Parameter(type = Type.STRING, direction = Direction.IN) String outputVcfPrefix,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String filteredHapsVcfFile
    );

    @Method(declaringClass = "guidance.GuidanceImpl")
    @Constraints(computingUnits="1", memorySize = "4.0f")
    void postFilterHaplotypes(
        @Parameter(type = Type.FILE, direction = Direction.IN) String filteredHapsFile,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String listOfSnpsFile
    );

    @Method(declaringClass = "guidance.GuidanceImpl")
    @Constraints(computingUnits="1", memorySize = "1.0f")
    void createRsIdList(
       @Parameter(type = Type.FILE,   direction = Direction.IN)  String genFile,
       @Parameter(type = Type.STRING, direction = Direction.IN)  String exclCgatFlag,
       @Parameter(type = Type.FILE,   direction = Direction.OUT) String pairsFile,
       @Parameter(type = Type.STRING, direction = Direction.IN)  String inputFormat
    );

    @Method(declaringClass = "guidance.GuidanceImpl")
    @Constraints(computingUnits="1", memorySize = "4.0f")
    void createListOfExcludedSnps(
       @Parameter(type = Type.FILE, direction = Direction.IN) String shapeitHapsFile,
       @Parameter(type = Type.FILE, direction = Direction.OUT) String excludedSnpsFile,
       @Parameter(type = Type.STRING, direction = Direction.IN) String exclCgatFlag,
       @Parameter(type = Type.STRING, direction = Direction.IN) String exclSVFlag
    );

    @Method(declaringClass = "guidance.GuidanceImpl")
    @Constraints(computingUnits="1", memorySize = "6.0f")
    void filterByAll(
       @Parameter(type = Type.FILE, direction = Direction.IN) String inputFile,
       @Parameter(type = Type.FILE, direction = Direction.OUT) String outputFile,
       @Parameter(type = Type.FILE, direction = Direction.OUT) String outputCondensedFile,
       @Parameter(type = Type.STRING, direction = Direction.IN) String mafThresholdS,
       @Parameter(type = Type.STRING, direction = Direction.IN) String infoThresholdS,
       @Parameter(type = Type.STRING, direction = Direction.IN)
       String hweCohortThresholdS,
       @Parameter(type = Type.STRING, direction = Direction.IN) String hweCasesThreshold,
       @Parameter(type = Type.STRING, direction = Direction.IN)
       String mafControlsThreshold
    );

    @Method(declaringClass = "guidance.GuidanceImpl")
    @Constraints(computingUnits="1", memorySize = "6.0f")
    void filterByInfo(
       @Parameter(type = Type.FILE, direction = Direction.IN) String imputeFileInfo,
       @Parameter(type = Type.FILE, direction = Direction.OUT) String filteredFile,
       @Parameter(type = Type.STRING, direction = Direction.IN) String threshold
    );

    @Method(declaringClass = "guidance.GuidanceImpl")
    @Constraints(computingUnits="1", memorySize = "6.0f")
    void jointFilteredByAllFiles(
       @Parameter(type = Type.FILE, direction = Direction.IN) String filteredByAllA,
       @Parameter(type = Type.FILE, direction = Direction.IN) String filteredByAllB,
       @Parameter(type = Type.FILE, direction = Direction.OUT) String filteredByAllC,
       @Parameter(type = Type.STRING, direction = Direction.IN) String rpanelName,
       @Parameter(type = Type.STRING, direction = Direction.IN) String rpanelFlag
    );

    @Method(declaringClass = "guidance.GuidanceImpl")
    @Constraints(computingUnits="1", memorySize = "6.0f")
    void jointCondensedFiles(
       @Parameter(type = Type.FILE, direction = Direction.IN) String inputAFile,
       @Parameter(type = Type.FILE, direction = Direction.IN) String inputBFile,
```

```
        @Parameter(type = Type.FILE, direction = Direction.OUT) String outputFile
    );

    @Method(declaringClass = "guidance.GuidanceImpl")
    @Constraints(computingUnits="1", memorySize = "8.0f")
    void collectSummary(
        @Parameter(type = Type.STRING, direction = Direction.IN) String chr,
        @Parameter(type = Type.FILE, direction = Direction.IN) String firstImputeFileInfo,
        @Parameter(type = Type.FILE, direction = Direction.IN) String snptestOutFile,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String reduceFile,
        @Parameter(type = Type.STRING, direction = Direction.IN) String mafThresholdS,
        @Parameter(type = Type.STRING, direction = Direction.IN) String infoThresholdS,
        @Parameter(type = Type.STRING, direction = Direction.IN)
        String hweCohortThresholdS,
        @Parameter(type = Type.STRING, direction = Direction.IN) String hweCasesThresholdS,
        @Parameter(type = Type.STRING, direction = Direction.IN)
        String hweControlsThresholdS
    );

    @Method(declaringClass = "guidance.GuidanceImpl")
    @Constraints(computingUnits="1", memorySize = "8.0f")
    void initPhenoMatrix(
        @Parameter(type = Type.FILE, direction = Direction.IN) String topHitsFile,
        @Parameter(type = Type.STRING, direction = Direction.IN) String ttName,
        @Parameter(type = Type.STRING, direction = Direction.IN) String rpName,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String phenomeFile
    );

    @Method(declaringClass = "guidance.GuidanceImpl")
    @Constraints(computingUnits="1", memorySize = "8.0f")
    void addToPhenoMatrix(
        @Parameter(type = Type.FILE, direction = Direction.IN) String phenomeAFile,
        @Parameter(type = Type.FILE, direction = Direction.IN) String topHitsFile,
        @Parameter(type = Type.STRING, direction = Direction.IN) String ttName,
        @Parameter(type = Type.STRING, direction = Direction.IN) String rpName,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String phenomeBFile
    );

    @Method(declaringClass = "guidance.GuidanceImpl")
    @Constraints(computingUnits="1", memorySize = "8.0f")
    void filloutPhenoMatrix(
        @Parameter(type = Type.FILE, direction = Direction.IN) String phenomeAFile,
        @Parameter(type = Type.FILE, direction = Direction.IN) String filteredByAllFile,
        @Parameter(type = Type.FILE, direction = Direction.IN) String filteredByAllXFile,
        @Parameter(type = Type.STRING, direction = Direction.IN) String thereisX,
        @Parameter(type = Type.STRING, direction = Direction.IN) String ttName,
        @Parameter(type = Type.STRING, direction = Direction.IN) String rpName,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String phenomeBFile
    );

    @Method(declaringClass = "guidance.GuidanceImpl")
    @Constraints(computingUnits="1", memorySize = "8.0f")
    void finalizePhenoMatrix(
        @Parameter(type = Type.FILE, direction = Direction.IN) String phenomeAFile,
        @Parameter(type = Type.FILE, direction = Direction.IN) String phenomeBFile,
        @Parameter(type = Type.STRING, direction = Direction.IN) String ttName,
        @Parameter(type = Type.STRING, direction = Direction.IN) String rpName,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String phenomeCFile
    );

    @Method(declaringClass = "guidance.GuidanceImpl")
    @Constraints(computingUnits="1", memorySize = "8.0f")
    void mergeTwoChunks(
        @Parameter(type = Type.FILE, direction = Direction.IN) String reduceFileA,
        @Parameter(type = Type.FILE, direction = Direction.IN) String reduceFileB,
        @Parameter(type = Type.FILE, direction = Direction.OUT) String reduceFileC,
        @Parameter(type = Type.STRING, direction = Direction.IN) String chrS
    );

    @Method(declaringClass = "guidance.GuidanceImpl")
    @Constraints(computingUnits="1", memorySize = "20.0f")
    void combinePanelsComplex(
```

```
414         @Parameter(type = Type.FILE, direction = Direction.IN) String resultsFileA,
            @Parameter(type = Type.FILE, direction = Direction.IN) String resultsFileB,
416         @Parameter(type = Type.FILE, direction = Direction.OUT) String resultsFileC,
            @Parameter(type = Type.STRING, direction = Direction.IN) String chromoStart,
418         @Parameter(type = Type.STRING, direction = Direction.IN) String chromoEnd
        );

420     @Method(declaringClass = "guidance.GuidanceImpl")
        @Constraints(computingUnits="1", memorySize = "6.0f")
422     void combineCondensedFiles(
            @Parameter(type = Type.FILE, direction = Direction.IN) String filteredA,
424         @Parameter(type = Type.FILE, direction = Direction.IN) String filteredX,
            @Parameter(type = Type.FILE, direction = Direction.OUT)
426         String combinedCondensedFile,
            @Parameter(type = Type.STRING, direction = Direction.IN) String mafThresholdS,
428         @Parameter(type = Type.STRING, direction = Direction.IN) String infoThresholdS,
            @Parameter(type = Type.STRING, direction = Direction.IN)
430         String hweCohortThresholdS,
            @Parameter(type = Type.STRING, direction = Direction.IN) String hweCasesThreshold,
432         @Parameter(type = Type.STRING, direction = Direction.IN)
            String mafControlsThreshold
434     );

436     @Method(declaringClass = "guidance.GuidanceImpl")
        @Constraints(computingUnits="1", memorySize = "8.0f")
438     void generateTopHitsAll(
            @Parameter(type = Type.FILE, direction = Direction.IN) String resultsAFile,
440         @Parameter(type = Type.FILE, direction = Direction.IN) String resultsBFile,
            @Parameter(type = Type.FILE, direction = Direction.OUT) String outputTopHitsFile,
442         @Parameter(type = Type.STRING, direction = Direction.IN) String pvaThreshold
        );
444
    }
```