High Performance Computational Fluid Dynamics on Clusters and Clouds : the ADAPT Experience

Imad Kissami

UNIVERSITÉ PARIS 13

THÈSE

En vue de l'obtention du

DOCTORAT DE l'UNIVERSITÉ PARIS XIII

Spécialité
INFORMATIQUE

Présentée et soutenue par

Imad KISSAMI

Sujet de la thèse:

High Performance Computational Fluid Dynamics on Clusters and Clouds: the ADAPT Experience

---

Laboratoire d'Informatique de Paris Nord (LIPN)
Laboratoire Analyse, Géométrie et Applications (LAGA)

Soutenue le: 28 février 2017

Devant le jury composé de:

Florian DE VUYST	Professeur, CMLA ENS Cachan - Rapporteur
Laurence HALPERN	Professeur, LAGA Institut Galilee Paris 13
Frederic MAGOULES	Professeur, Ecole Centrale de Paris
Fayssal BENKHALDOUN	Professeur, LAGA - IUTV Paris 13, co-encadrant
Marian VAJTERZIC	Professeur, Universität Salzburg, Autriche, rapporteur
Raphael COUTURIER	Professeur, FEMTO-ST, Université de Franche-Comté
Roberto WOLFER CALVO	Professeur, LIPN Paris 13
Christophe CÉRIN	Professeur, LIPN - IUTV Paris13, Directeur de thèse

Résumé

Dans cette thèse, nous présentons notre travail de recherche dans le domaine du calcul haute performance en mécanique des fluides (CFD) pour architectures de type cluster et cloud. De manière générale, nous nous proposons de développer un solveur efficace, appelé ADAPT, pour la résolution de problèmes de CFD selon une vue classique correspondant à des développements en MPI et selon une vue qui nous amène à représenter ADAPT comme un graphe de tâches destinées à être ordonnancées sur une plateforme de type cloud computing.

Comme première contribution, nous proposons une parallélisation de l'équation de diffusion-convection couplée à un système linéaire en 2D et en 3D à l'aide de MPI. Une parallélisation à deux niveaux est utilisée dans notre implémentation pour exploiter au mieux les capacités des machines multi-coeurs. Nous obtenons une distribution équilibrée de la charge de calcul en utilisant la décomposition du domaine à l'aide de METIS, ainsi qu'une résolution pertinente de notre système linéaire creux de très grande taille en utilisant le solveur parallèle MUMPS (Solveur MUltifrontal Massivement Parallèle).

Notre deuxième contribution illustre comment imaginer la plateforme ADAPT, telle que représentée dans la première contribution, comme un service. Nous transformons le framework ADAPT (en fait, une partie du framework) en DAG (Direct Acyclic Graph) pour le voir comme un workflow scientifique. Ensuite, nous introduisons de nouvelles politiques à l'intérieur du moteur de workflow RedisDG, afin de planifier les tâches du DAG, de manière opportuniste. Nous introduisons dans RedisDG la possibilité de travailler avec des machines dynamiques (elles peuvent quitter ou entrer dans le système de calcul comme elles veulent) et une approche multi-critères pour décider de la "meilleure" machine à choisir afin d'exécuter une tâche. Des expériences sont menées sur le workflow ADAPT pour illustrer l'efficacité de l'ordonnancement et des décisions d'ordonnancement dans le nouveau RedisDG.

Mots clés: calcul haute performance, MPI, partitionnement du maillage, résolution de systèmes linéaires creux, cloud computing

Abstract

In this thesis, we present our research work in the field of high performance computing in fluid mechanics (CFD) for cluster and cloud architectures. In general, we propose to develop an efficient solver, called ADAPT, for problem solving of CFDs in a classic view corresponding to developments in MPI and in a view that leads us to represent ADAPT as a graph of tasks intended to be ordered on a cloud computing platform.

As a first contribution, we propose a parallelization of the diffusion-convection equation coupled to a linear system in 2D and 3D using MPI. A two-level parallelization is used in our a implementation to take advantage of the current distributed multicore machines. A balanced distribution of the computational load is obtained by using the decomposition of the domain using METIS, as well as a relevant resolution of our very large linear system using the parallel solver MUMPS (Massive Parallel MUltifrontal Solver).

Our second contribution illustrates how to imagine the ADAPT framework, as depicted in the first contribution, as a Service. We transform the framework (in fact, a part of the framework) as a DAG (Direct Acyclic Graph) in order to see it as a scientific workflow. Then we introduce new policies inside the RedisDG workflow engine, in order to schedule tasks of the DAG, in an opportunistic manner. We introduce into RedisDG the possibility to work with dynamic workers (they can leave or enter into the computing system as they want) and a multi-criteria approach to decide on the "best" worker to choose to execute a task. Experiments are conducted on the ADAPT workflow to exemplify how fine is the scheduling and the scheduling decisions into the new RedisDG.

Key words: high performance computing, MPI, mesh partitionning, sparse direct solver, cloud computing

Contents

Chapter 1

Introduction

1.1 General background

The CFD, for Computational Fluid Dynamics is a branch of fluid mechanics that uses numerical methods to investigate and propose solutions to problems that involve fluid flows. This development aid tool becomes essential in all industrial areas related to fluid dynamics such as aerospace, plasma physics, porous media, meteorology, etc. Depending on the chosen approximations, which are the result of a compromise between the requirements in terms of accuracy of the physical representation, and computing resources available, the numerical solution of equations allows access to all instantaneous variables at each point of the computational domain.

Nowadays, CFD has reached a sufficient degree of maturity to be integrated with the industrial design process, in addition to the test campaigns because it limits the number of trials needed during a development cycle, and access all physical quantities on the whole computational domain. It alleviates the requirements to reduce design time and research performance solutions in a context where the objectives are multiple and often conflicting. With the ever increasing power of computing resources, it is now possible to achieve resolution levels for a detailed description of physical phenomenon.

1.2 Problem

Besides, scientists have always wanted to acquire a better knowledge of their environment. They conduct experiments to explain and predict the universe. With the emergence of computers, a new discipline that has greatly accelerated this process emerged: computer science. In addition to that, computers are capable of simulating experiments that can validate new concepts or designs with a precision never seen before.

Indeed, using numerical simulations, we can avoid the costly and complicated installation, time, or dangerous experiences. In addition, simulations can be used in various fields such as aerodynamics, meteorology, biology, finance, plasma physics, etc. This list is not exhaustive, but shows how simulations can affect many areas on a day to day database environment.

Thus, the numerical simulation is a very active area of research: testing must be both accurate and fast. Numerical calculations are often made as a complement to experiments and theoretical analysis, but can also be the only method suitable for a certain problem, for economical or technical reasons. One reason for the increasing use of numerical calculations is the development of robust and efficient numerical methods, another reason would be the price/performance for high performance computers.

In particular, nuclear fusion reactors are a promising area of research for our future as a new energy source, and require very precise simulations. Here, we look at the unbalanced ionization process (discharge) that occur when a neutral gas is exposed to the electric field of high intensity. They can be in various forms depending on the electric field and the pressure and the volume of the medium.

In this thesis, we will focus mainly on the streamers that are generated by ionization waves. Streamer discharge is used in several applications such as elimination of pollutants. The numerical simulation of the streamer diffusion is

based on a mathematical model of various levels of complexity, including the hydrodynamic approximation, leading to a system of equations which comprises the convection-diffusion-reaction equations for a charged particle coupled with the Poisson equation of the electric field.

Given the complexity of the phenomenon, it is essential to get realistic simulations that lead to find the best combination of models, numerical methods and computer resources. The numerical simulation of this physical phenomenon is then enriched by strong interactions between researchers in three disciplines : physics, mathematics and computer science. This research is based on the exploitation of computer science resources and, specifically, aim to develop parallelization techniques to effectively use resources calculation and storage.

1.3 Objective

Generally, this problem cannot be resolved continuously over the whole domain, but it must be studied in many points. Parallel computing intervenes in this case because it can perform calculations that would take too long if they were running on a single processor on a single machine by a single execution thread (one process, one thread). Most types of calculation must, for example, be made in a fixed and short time.

This is the case, for example, for the plasma phenomenon. A simulation should be performed daily and for the next day. In addition, the reliability of the results depends on the fineness of the calculation parameters, including that of the grid used to discretize the space and time: the increase in computing power to solve a problem involving more parameters, therefore gives a more accurate results. Similarly, many scientific applications require thousands of hours of computing: it would be unreasonable to expect the result calculated by a sequential program.

In our case, when we started to work on that research topic, the code took two days to capture the Streamer 2D and one month in case of 3D, hence the necessity of the transition to parallelization. Parallel computing promises to be effective and efficient in tackling these computational problems. However, parallel programming is different from and far more complex than conventional serial programming, and building efficient parallel programs is not an easy task. Furthermore, the fast evolution of parallel computing implies algorithms to be changed accordingly, and the diversity of parallel computing platforms also requires parallel algorithms and implementations to be written with consideration on the underlying hardware platform.

Recently, several researchers have been where interested in parallelizing fluid dynamics codes that include convection-diffusion-reaction equations coupled with Poisson equation. The general idea is to run the calculations using all available resources either using shared memory or distributed one. In the case of distributed memory architecture, the whole domain must be cut into subdomains using tools such as METIS or SCOTCH to do the calculations on each processor independently from any other. The resolution of the linear system deduced from the Poisson equation is made using parallel direct methods (MUMPS, PaStiX, SuperLU, ...) or parallel iterative methods (Pestc, GMRES, ...).

Furthermore, the emergence of new types of Internet-based computing such as cloud computing impacts seriously the field of High Performance Computing. Major companies, among them Google, Microsoft, Amazon... offer "computing as a service". Amazon started its AWS (Amazon Web Services) project more than 10 years ago.

One of Amazon Web Service's key components, EC2 (Elastic cloud Computing), was developed by a small team in a satellite development office in South Africa. Chris Pinkham, who worked as an engineer in charge of Amazon's global infrastructure in the early 2000s, noticed that "It struck us in the infrastructure engineering organisation that we really needed to decentralize the infrastructure by providing services to development teams". He also noticed "That was a big motivating factor."

Pinkham thought about this issue and, in 2003, started trying to build an "infrastructure service for the world". His hope was that he could develop a service that would not only deal with Amazon's infrastructure, but also help developers. Since its official launch in 2006, EC2's value has grown and it has become the cornerstone of Amazon's ecosystem of cloud services.

With cloud computing, computing is a service, among other services. The infrastructure is supposed to scale up and to scale down according to the demand and many other facilities have been proposed and adopted since 2006. We will discover some of them later on in our document.

The same appetite appeared for Google that now proposes many cloud services among them the google Cloud

Pub/Sub[1] which is a fully-managed real-time messaging service that allows you to send and receive messages between independent applications. You can leverage Cloud Pub/Sub's flexibility to decouple systems and components hosted on Google Cloud Platform or elsewhere on the Internet. As Google teams love to say: "By building on the same technology Google uses, Cloud Pub/Sub is designed to provide "at least once" delivery at low latency with on-demand scalability to 1 million messages per second (and beyond)".

In our work we also use the Pub/Sub paradigm in order to coordinate the components of a workflow engine able to execute scientific workflows in an opportunistic manner. This thesis contributes to the field of workflow scheduling and revisits HPC in terms of cloud computing. In some ways we improve an existing WaaS (Workflow as a Service) tool. For doing this, we need to adapt the way we decompose a scientific computational intensive problem. A practical use case demonstrates the potential of our approach for doing HPC in the cloud.

The world is really changing quickly since we are now faced to very easy ways to book resources (CPU, memory, disk), even coming from different cloud providers. For instance, the RosettaHub platform[2] allows you to book ressources alternatively on Amazon, Microsoft...clouds. RosettaHUB makes it possible for everyone to simultaneously use Python, R, Julia, SQL, Scala, Spark, Mathematica, ParaView, etc. as well as any computational code or library from within one single portal exposing web based and highly usable clouds management consoles, workbenches and notebooks.

1.4 Outline

This manuscript is organized as follows

We present in Chapter 2 an introduction to high performance computing. We detail the different architectures that have been developed to perform efficient computation. We begin with a brief presentation of the various architectures and optimization algorithms that result. Then, we present a state of the art of graph partitioning. Finally, we explain the different methods available to exploit the new heterogeneous parallel machines. Those methods that have been developed to address sparse linear systems are described, and we particularly focus on sparse direct algebra parallel solvers, which are the target of this thesis.

We will then present, in Chapter 3, state of art of cloud computing, Service Computing, and Workflow systems.

In Chapter 4, we introduce the streamer problem included in ADAPT framework and explain numerical schemes used to solve differential equations. Then we reduce the computational time cost of our ADAPT platform through parallelism. The parallel architecture exploited is: the cluster (set of network connected PCs). Furthermore, we parallelize 2D and 3D streamer codes using MPI, the most critical points are memory management and communications. The aim of this work is to adapt the code on this programming model, to optimize it and offer the possibility to do tests in reduced time.

In chapter 5, we discuss about the case of scheduling the ADAPT workflow according to the RedisDG workflow engine. More precisely, we schedule workflows of different sizes of a specific workflow solving one CFD problem of the ADAPT framework.

Finally, we will conclude and will present some areas for improvement of these methods in Chapter 6.

[1] https://cloud.google.com/pubsub/

[2] https://www.rosettahub.com/welcome

Part I

Scientific Computing

Chapter 2

Numerical Simulation

Contents

2.1 Context

A numerical simulation models generally the time evolution of a physical phenomenon in a domain of space. These simulations are based on a discretization in time and space. Often one uses an iterative scheme in time and space discretization via a geometric mesh elements (triangles, squares, tetrahedron, etc.). Concretely, in our case this numerical simulation is represented by an evolution equation coupled with Poisson equation (see equation (2.1)).

These simulations, increasingly complex, require to be executed in parallel. The domain should be distributed among multiple computing units. The mesh is splitted into as many parts as processors used. The time discretization can calculate the system status at a given time from the previous state. More accurately, the state of an element is calculated from its previous state and the state of some other elements depending on the physical model; generally, it is the neighboring elements in the mesh, in addition to that, a linear system is computed at each iteration to solve the Poisson equation.

This distribution induces communications between processors when one of them requires data owned by others. These communications are synchronous and all processors must therefore advance the simulation at the same pace.

A good distribution or partition data should provide balanced parts and minimizing the number of data dependencies between processors, minimizing the time computing and communication.

A model for the problem is according to equation 2.1.

$$\begin{cases} \dfrac{\partial u}{\partial t} + F(V,u) = S, \\ \\ \Delta P = b. \end{cases} \tag{2.1}$$

given that $F(V,u) = div(u.\overrightarrow{V}) - \Delta u$, $S = 0$, and $V = \overrightarrow{\nabla} P$, the previous system gives:

$$\begin{cases} \dfrac{\partial u}{\partial t} + div(u.\overrightarrow{V}) = \Delta u \\ \\ \Delta P = b. \end{cases} \tag{2.2}$$

We aim to parallelize both the linear solver issued from the Poisson equation and the evolution equation using domain decomposition and mesh adaptation at "the same time". To our knowledge this is the first time that such a challenge is considered. Separating the two steps may add delays because we need to synchronize them and to align the execution time on the slowest processor. Considering the two steps simultaneously offers the potential to better overlap different computational steps. Some authors have tackled the parallelization of either the linear solvers or the evolution equation usually without mesh adaptation.

In [1] the authors consider the parallelization of a linear system of electromagnetic equation on non adaptive unstructured mesh. Their time integration method leads to the resolution of a linear system which requires a large memory capacity for a single processor.

In [2] the authors introduce a parallel code which was written in C++ augmented with MPI primitives and the LIS (Linear Iterative Solver) library. Several numerical experiments have been done on a cluster of 2.40 GHz Intel Xeon, with 12 GB RAM, connected with a Gigabit Ethernet switch. The authors note that a classical run of the sequential version of their code might easily exceed one month of calculation. The improvement of the parallel version was performed by the parallelization of three parts of the code: the diagonalization of the Schrödinger matrix, advancing one step in the Newton–Raphson iteration, and the Runge-Kutta integrator.

The authors in [3] focused only on the parallelization of a linear solver related to the discretization of self-adjoint elliptic partial differential equations and using multigrid methods. It should also be mentioned that authors in [4] have successfully studied similar problems to those presented in this paper.

The associated linear systems have been solved using iterative GMRES and CG solvers [5]. One difference is that in our work we consider direct methods based on LU decomposition using the MUMPS solver. Moreover, the direct solution methods generally involve the use of frontal algorithms in finite elements applications. The advent of multi—frontal solvers has greatly increased the efficiency of direct solvers for sparse systems. They make full use of high performance software layers such as invoking level 3 Basic Linear Algebra Subprograms (BLAS) [6, 7] library. Thus the memory requirement is greatly reduced and the computing speed greatly enhanced. Multi—frontal solvers have been successfully used both in the context of finite volumes, finite elements methods and in power system simulations.

2.2 Introduction to High Performance Computing (HPC)

2.2.1 Introduction

The demand of performances in numerical simulation is growing. Indeed, the application domains require to implement more and more calculations while considering increasing data sizes, assuming long execution time. As an illustration, we can mention programs for the simulation of weather forecasts and plasma physics, whose needs are growing.

The HPC (High Performance Computing) is a discipline that aims to tackle this problem. It involves the adaptation and execution of large programs in heavy computing on supercomputers. These machines typically include a large

number of processors or calculation units, allowing them to perform many tasks in parallel. They also make available a large memory space to handle large problems in data size. The input/output system is also designed to provide a wide bandwidth.

Running a program on a supercomputer requires application specific IT (Information Technology) developments. The original sequential program must first be optimized. Then, the implementation of the parallelization involves identifying and building computational tasks that can run simultaneously. This is called independence: tasks whose results do not depend on each other can be run simultaneously.

Some definitions are now presented as well as terminologies that are commonly found in HPC.

Task and parallel task

A task is a portion of a work to be carried by a computer. This is typically a set of instructions executed on a processor. A parallel task is a task that can run on multiple processors.

Process and thread

A process is a program in execution, it is created by the operating system. It contains one or more threads. A thread consists of one or more tasks. In a multi-threaded programming, each thread has its own memory and shared global memory of the parent thread.

Acceleration

Acceleration is the ratio of running time between sequential program and its parallel version. An ideal acceleration is the number of processors used (see scalability).

Massively parallel

This term is used to refer to hardware architectures that contain a large number of processors. This number is obviously constantly increasing, at the time of writing this manuscript it is around a ten million.

Communications

Communication is the exchange of data between parallel tasks. These communications are usually made through a network.

Synchronization

Synchronization is a breakpoint in a task that is released when other tasks have arrived at the same level. The aim is generally to coordinate tasks against communications.

granularity

Granularity is the ratio of computation time and the communication time. The grain is said coarse grain if a large amount of calculation is processed between communications and fine grain if instead there is a little calculation between communications.

scalability

The scalability of a parallelized system is its ability to provide an acceleration proportional to the number of processors. It depends on the hardware (bus speed, etc.), on the ability of the algorithm to be parallelized and also on programming. Ideal scalability is 1.

In the following, the taxonomy of Flynn will be introduced. Then, a quantification tool of the possible acceleration by parallelization will be presented, later we presents the two types of parallelism: parallelism on shared memory machine and on distributed memory machine.

2.2.2 The taxonomy of Flynn

As opposed to the archetype of a sequential machine, where instructions are executed one by one to the rhythm of clock cycles on a single datum, parallel machines can follow different execution models. These models have been proposed by Michael J. Flynn. [8]. They are based on the type of organization of the data and instruction streams.

- SISD Model (Single Instruction Stream, Single Data Stream): a sequential computer which exploits no parallelism, Von Neumann architecture.
- the model MISD (Multiple Instruction Stream, Single Data Stream): the same data is processed by several processors in parallel.
- SIMD model (Single Instruction stream, Multiple Data stream): table of processors, where each processor executes the same instruction on its own local data.

	Single Data	Multiple Data
Single Instruction	SISD	SIMD
Multiple Instruction	MISD	MIMD

Figure 2.1: Flynn classification

- The model MIMD (Multiple Instruction stream, Multiple Data stream): several processors process different data, each processor has a separate memory. The most used parallel architecture.

2.2.3 Limitations and cost of parallelism

Ideally, the acceleration of a parallelized program should be the number of computing cores used (optimal scalability). The reality is unfortunately not as simple because there are always sequential parts which can represent a significant cost. Amdahl proposed in 1967 [9] the following model to describe the acceleration that can provide a machine with n processors (2.3):

$$Acc = \frac{1}{r_s + \frac{r_p}{n}} \tag{2.3}$$

Where $r_s + r_p = 1$, r_s and r_p are respectively the proportions sequential and parallelized program. This model limits the acceleration that can make parallelization. Indeed, the limit as n approaches infinity is finished 2.2.

Gustafson proposed in 1988 [10] a reevaluation of the Amdahl's law because he considered it too pessimistic 2.3.

Parallelization is not just to reduce the computation time of a given problem, as Amdahl sees, but especially to deal with more important issues. In this case, the sequential proportion remains more or less constant. Indeed, in our problems, the sequential part typically contains the extraction of a mesh. The latter is of complexity N, where N is the number of nodes, for example. The physical problem to be solved can be meanwhile complexity of $N\ log\ N$ if it is of finite elements, or N^2 in the case of an integral method for example. The rising cost of sequential operations is minimal compared to those parallelizable. Gustafson therefore defines a linear acceleration law (2.4):

$$Acc = r_s + n.r_p \tag{2.4}$$

The truth is probably somewhere between the laws of Amdahl and Gustafson, Ni proposed a finer law [11] which also takes into account the communication time between processors, this law will not be presented in this manuscript.

2.2.4 Architectures of supercomputers

2.2.4.1 Multi-level parallelism

Technically, the introduction of the parallelization is based on the target technology. There are two categories particularly prevalent on current supercomputers:

- Shared memory architectures,

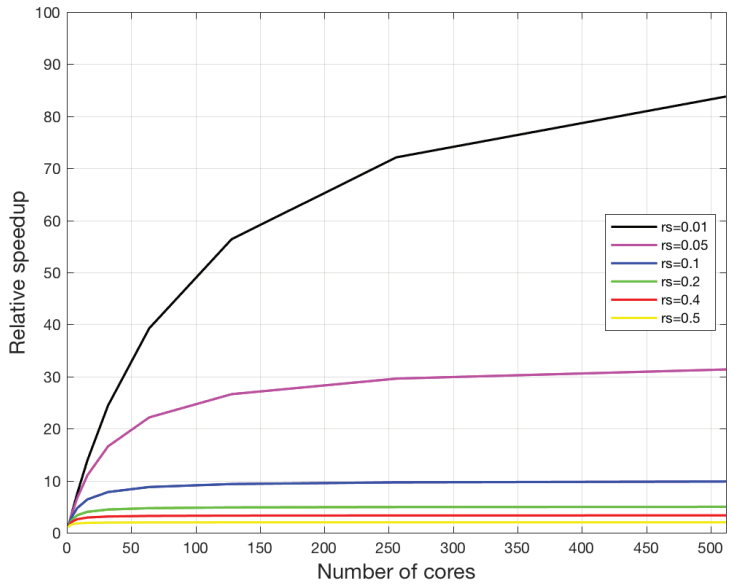


Figure 2.2: Amdahl's law, acceleration in the number of CPUs for different proportion of sequential program. Acceleration is limited here.

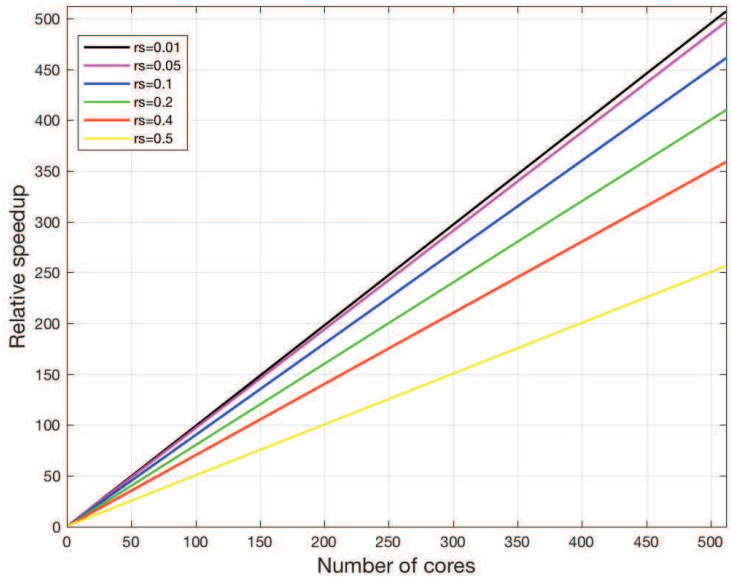


Figure 2.3: Gustafson's Law, acceleration depending on the number of processors for different proportion of sequential program. The acceleration follows a linear law.

- Distributed memory architectures.

It is possible to mix these technologies to form a hybrid parallel machine which are increasingly common in the

market for supercomputers. As an illustration, we can mention the chines Machine Sunway Taihulight (see Figure 2.4), located at Wuxi, Developed by the National Research Center of Parallel Computer Engineering & Technology (NRCPC) and installed at the National Supercomputing Center. It displays a computing power of 93 petaflop/s. The system has a theoretical peak performance of 125.4 Pflop/s with 10,649,600 cores and 1.31 PB of primary memory. It is a machine for a hybrid parallelism distributed memory between nodes, and shared memory between cores within the same node.

Figure 2.4: The supercomputer Sunway, Wuxi, China

There is a third type of prevalent architecture : the accelerator. This calculation means is designed to achieve high computational power while minimizing energy costs and space requirements compared to conventional multi-core processors. An accelerator consists mainly of the accumulation of a large number of processing units and a large memory space shared between these units. The processing units are less powerful and specialized in the calculation, therefore less versatile than those of a general-purpose processor. That is why parallel programming on this technology is requesting more substantial programming effort, the codes must be adapted depending on the architecture specific to the accelerator. As an example we can mention the graphics cards (GPGPU for general-purpose computing on graphics processing units) concatenating a large number of computing units in groups sharing resources (cache, instruction stack...).

In the following, we present the two types of parallelism of interest for this project, namely the parallelism on shared memory architecture and on distributed memory architecture.

- **The parallelism on shared memory machine**: Shared memory machines share the same memory area, accessible by all processing units. The shared memory machines are characterized by a set of processors which all have direct access to a common memory, through which they can communicate. This implies that we can define tasks to run in parallel, for example by distributing iterations of a loop via simple directives. The parallel tasks are then distributed on different threads, themselves distributed over the computing units of the shared memory machine considered. The directives in a code mean that we become intrusive in the code, but conversely, we quickly obtain a parallel code while keeping the code as close as possible to the original version.

- **The parallelism of distributed memory machine**: Typically, distributed memory machines are composed of a large number of processors or multiprocessor each with its own memory, also known as compute node. These nodes are interconnected by a fast communication network (InfiniBand technology, for example) so they can communicate. The MAGI cluster at Paris 13 University has been used in this thesis. It contains about 50 Dell FC430 blades with 10 Hyper-Threading dual processor cores (40 cores per blade), and 64 GB RAM per blade interconnected by InfiniBand.

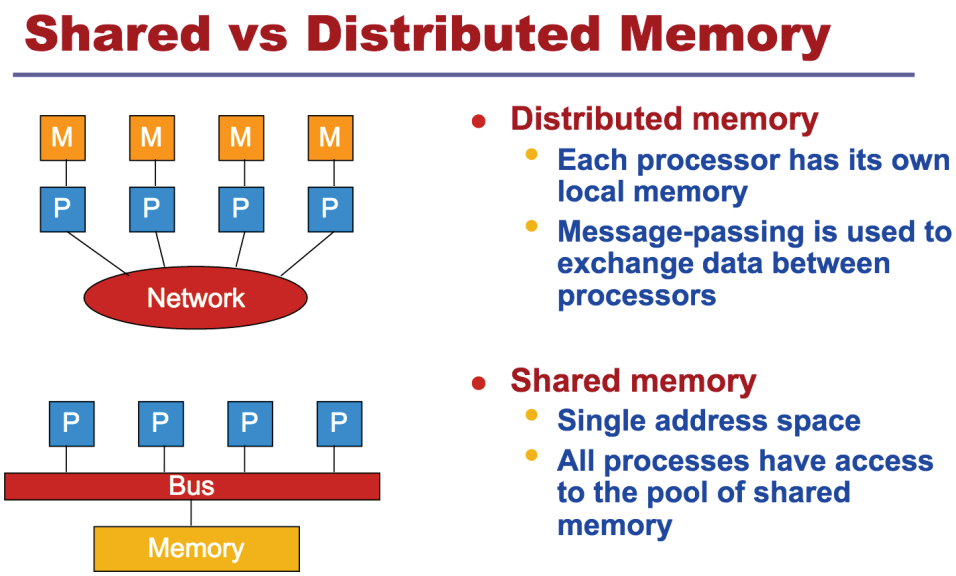


Figure 2.5: Shared vs Distributed memory

2.2.4.2 Multicore

Multicore processors are called processors that contain multiple processing units. Since 2012, it is the most common architecture, it also represents the new standard for PCs. The number of computing cores is usually between 2 and 8. It is a shared memory architecture because all cores have access to the RAM. To exploit this architecture, it is common to divide the problem into n threads, where n is the number of cores on the machine. This partitioning is fairly easy to implement because the number of partitions is low, and further, the memory is shared between threads.

2.2.4.3 SMT architecture

The SMT architecture (Simultaneous MultiThreading) optimizes the use of resources of multicore processors, such as wasted cycles (Figure 2.6). Pipelines are shared between multiple threads, registers and caches also. The individual performance of each thread is degraded, but the execution of all threads is improved [12] and [13]. For example, the Intel Core i7 2600, is an SMT architecture. Indeed, the processor has eight logical cores while there are physically four. This mechanism, that simulates the presence of more cores than there exists physically, is called hyperthreading by Intel.

2.2.4.4 NUMA architecture

The NUMA architecture (Non Uniform Memory Access or Non Uniform Memory Architecture) allows the grouping of processors, and processors are connected together by a specific bus. In a NUMA system, the memory access for a processor is fast regarding its own memory, but may be less fast to access the memory of others processors. The sharing of memory and peripherals imply that the NUMA multiprocessor technique exploits a sophisticated memory and cache coherence protocol across all processors attached to the platform. This protocol is a key component for efficient load-balancing strategies. Figure 2.7 shows the topology of a NUMA system composed of two 4 core multiprocessors on MAGI cluster and their dedicated memory, forming two NUMA nodes.

Compared to SMP (Symmetric MultiProcessing) technology, the NUMA architecture allows a greater number of processors simultaneously. The NUMA architecture requires a standalone operating system for each processor while in SMP, only one operating system is used jointly by all.

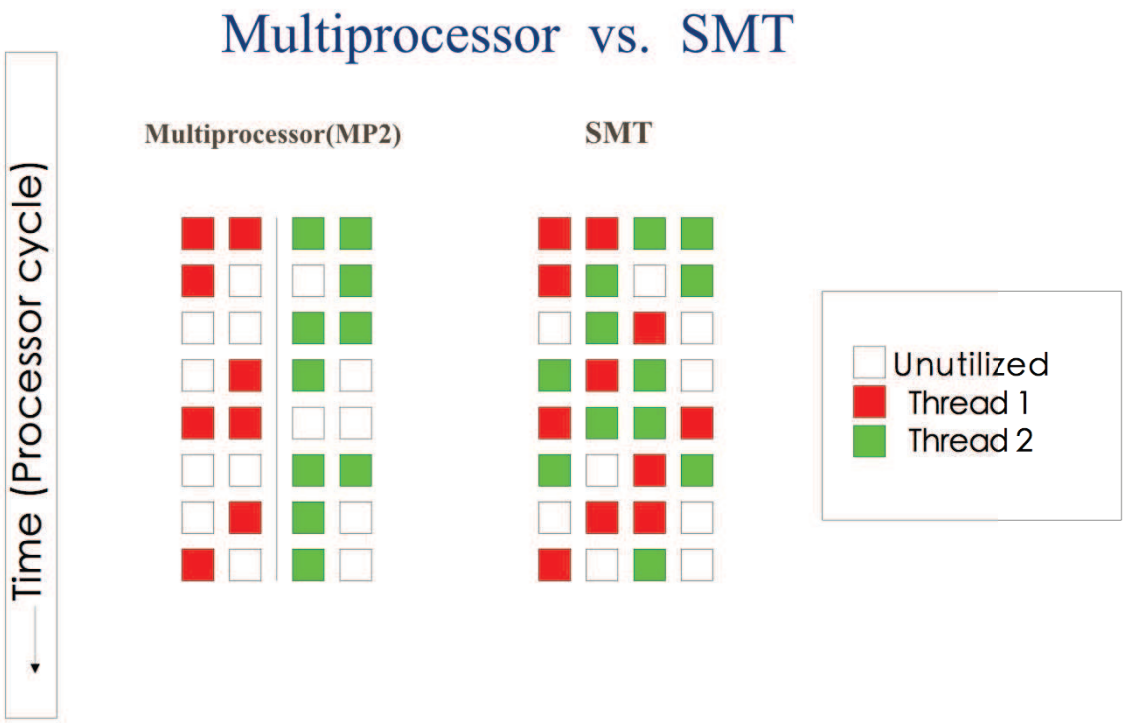


Figure 2.6: The SMT architecture optimizes the operation of computing saturating their pipelines processor resources

2.2.4.5 Computing cluster

A computing cluster means a collection of computers interconnected by a network. The scale of this network is local: a room or a building. This is a distributed memory architecture, each machine has no "direct" access to the memory of others. For a parallel computing code developer, he must also manage explicitly the communication which can become a bottleneck (see granularity).

The MAGI cluster

The MAGI cluster is located at the University of Paris 13.

- 50 Dell FC430 blades
  - 10 Hyper-Threading dual processor cores (40 cores per blade);
  - 64 GB RAM per blade;
  - Local disk is made of SSD;
  - InfiniBand for the low latency interconnect;
  - Total of 2000 cores.
- 3 servers “shared memory”
  - 16, 40 and 80 cores;
  - 512 GB RAM;
  - Local SATA disks;
  - Infiniband;

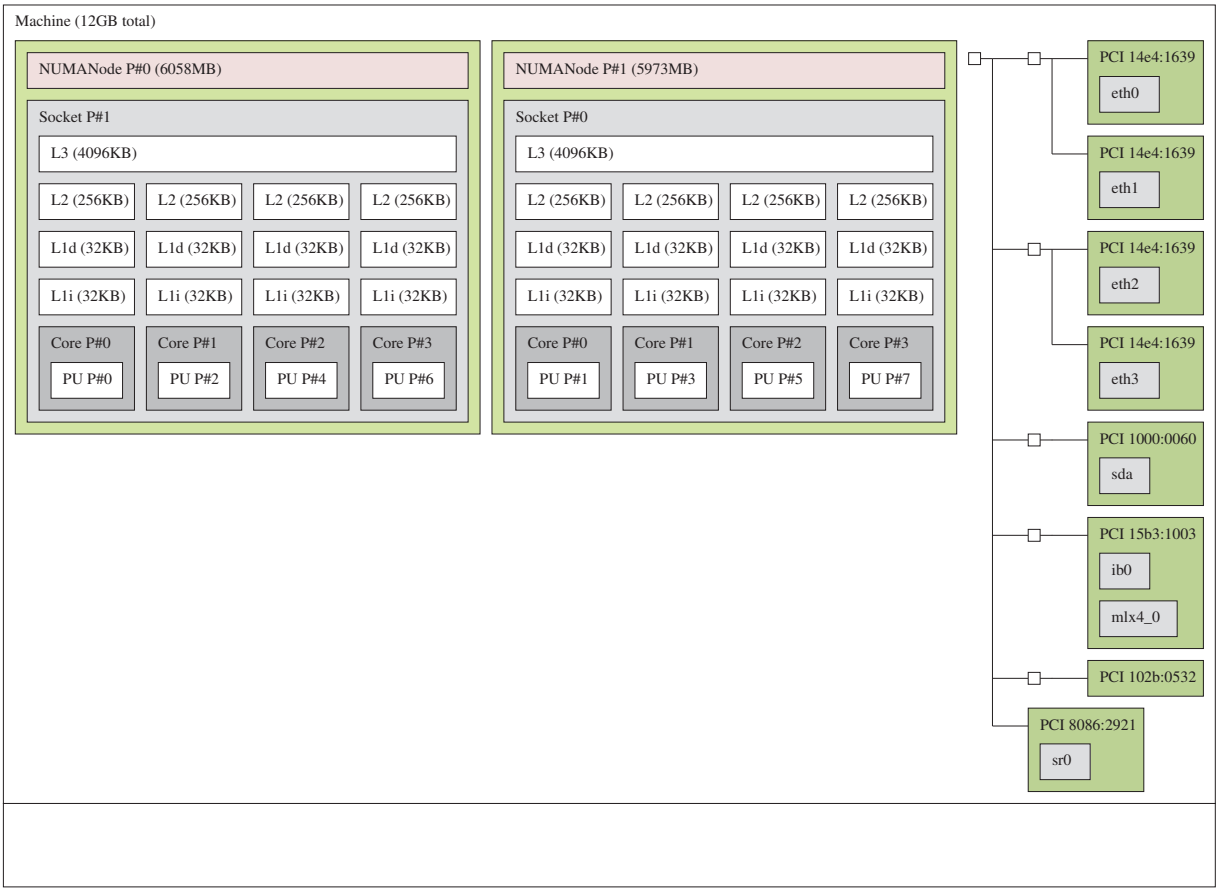


Figure 2.7: Topology of a computer with 2 NUMA nodes

  - Total of 136 cores.
- 1 GPU node
  - 4 Tesla cards K40m;
  - 40 intel cores;
  - 60 GB RAM;
  - Infiniband.
- 16 B500 Bull blades
  - 12 processors with 12 cores; 4 blades and 192 GB RAM;
  - 6 blades with 8 cores;
  - Total of 192 cores.

2.2.5 Parallel computing libraries

The few libraries presented here are among the best known to achieve performance.

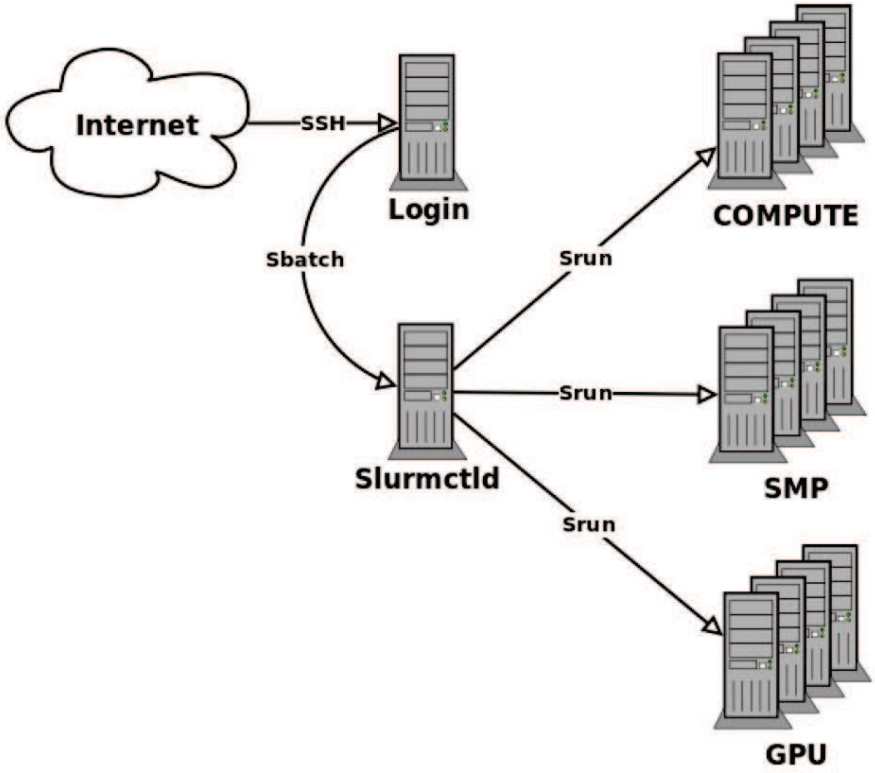


Figure 2.8: Topology of MAGI cluster

OpenMP

OpenMP is a multi-platform API for C/C++ and Fortran. It allows to easily carry parallel codes on shared memory architectures. The most common operation is to parallelize the loops provided each iteration in the loop is independent of the others.

MPI

Message Passing Interface is an API for C/C++ and Fortran. Contrary to OpenMP, it exploits the distributed memory architectures by passing messages between machines. This is now the standard on distributed memory parallel systems such as clusters and Grids.

CUDA

Cuda is an API developed by Nvidia that allows to exploit the graphics card brand to perform calculations.

OpenCL

Open Computing Language is an API developed by the Khronos Group in 2008. First introduced as an open alternative to CUDA, OpenCL is much as it handles both the GPU multi-CPU platforms.

In this thesis, the standard Message Passing Interface is used. As the name suggests, MPI is a programming tool for exchanging messages. There are several implementations and MPI libraries, for instance the Open MPI library (open source) which is used in Chapter 4 of this thesis. The MPI programming model is to distribute the total workload across multiple processes that can be run on multiple computing nodes, each with a memory space of its own. It is the responsibility of the programmer to explicitly divide the workload, for example by distributing the total volume of data to be processed on the various processes within the framework of solving a partial differential equation.

When running an MPI process, all necessary variables in this process are private and only visible in this process. This is why the MPI library provides various communication functions, blocking or not, for a group or point to point exchange between multiple processes. This is also the programmer bears the task of organizing communication between processes as needed. From the perspective of development, the establishment of such parallelism is particularly intrusive, involving many changes in the case of algorithms for solving partial differential equations. These changes are mainly the distribution of data, calculations and organization of communications.

2.3 Graph partitioning

2.3.1 Modelization

Meshes partitioning needs the calculation of dependencies between elements to construct a graph. Some partitioners use a model of hyper-graph, more general than the graph model. A graph partitioning tool or hyper-graph calculates scores by taking into account two objectives:

1. The parts must be balanced, that is to say about the same size. The partitioners create parts roughly balanced within a "unbalanced factor". An imbalance factor of 5% means that a chunk may reach up to 1.05 times its ideal size.

2. A cost function, called cutting of the graph, is minimized. This cost function varies according to the model and it generally deals with the communication time between processors.

To apply the graph partitioning or hyper-graph to a given problem, it is necessary to model it. It is important to choose a model adapted to the problem. The partitioner can create a partition based on optimizing the most adapted criteria for this simulation. The most common approach is to model the materials to be distributed in the form of a graph. The graph constructed must model the constraints of distributed computing: load balancing and reducing communications. A graph is defined either by a set of vertices and a set of edges connecting the vertices in pairs.

The edges of the graph are used to represent the dependencies between "elements" of the model. For example, the calculation of a feature may require data from adjacent elements in the mesh. A node in the graph is connected to all vertices representing neighboring elements in the mesh.

If elements are associated with different calculation loads, it is possible to associate weights with the vertices representing these calculation loads. In the same way, if some communications are more costly than others, a weight can be assigned to the edges. A graph partitioning tool optimizes two objectives: the balancing and cutting. The size of part is the number of vertices, in case they are balanced the size is the sum of their weights. The performance function most generally used in the graph partitioning tool is the number of cut edges (or the sum of their weight). An edge is called “cut” when both ends are assigned to different parts. The aim is to reduce the volume of data communication between processors.

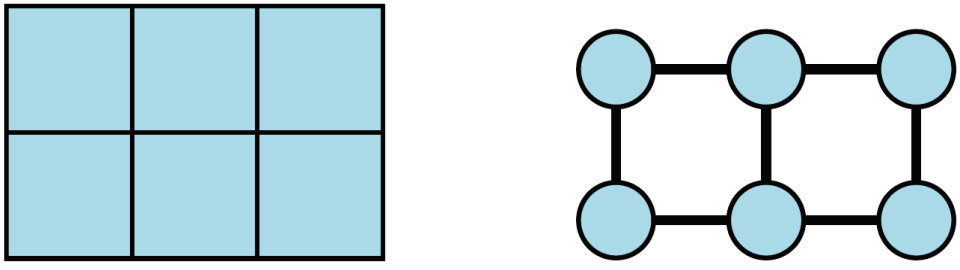


Figure 2.9: A mesh of 6 elements and the associated graph modeling the dependencies

Figure 2.9 shows the modeling of a 2 x 3 square grid as a graph. Each vertex of the graph corresponds to a square of the grid and is connected to the vertices corresponding to its neighbors in the mesh.

This model, although widespread, does not always perfectly model the communication volume required for the induced distribution. Since an edge connects two vertices, each cut edge induces two vertices to communicate. But a vertex can have several cut edges of the same part and the graph cutting may overestimate the communication volume induced by the partition. In Figure 2.10, a mesh of 6 elements is partitioned into two parts : a blue part and a green part. 4 elements have at least one neighbor in the other part, it's the elements that need to be shared with the other processor. But 3 edges are divided by the partition.

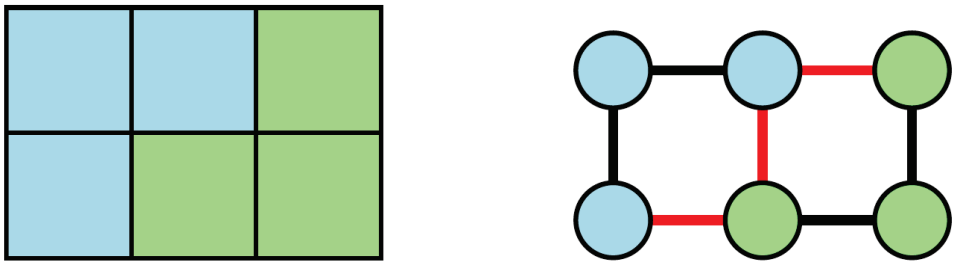


Figure 2.10: Mesh and graph splitted into two

Many tools have been proposed for the partitioning of graphs/meshes to solve this problem, such as Chaco [14], SCOTCH [15] METIS [16], etc. ADAPT in this operation is done using METIS.

2.3.2 METIS

METIS is a software package for partitioning large irregular graphs, partitioning large meshes, and computing fill reducing orderings of sparse matrices. The algorithms in METIS are based on multilevel graph partitioning described in [17, 18, 19], and based on the multilevel recursive-bisection, multilevel k-way, and multi-constraint partitioning schemes. The multi-level method reduces the size of the original graph, performs a partition on this and then finally uncoarsens the graph to find a partition for the original graph. METIS can be used as a suite of stand alone partitioning applications or by linking a user's Fortran or C application to the METIS library.

In Figures 2.12 and 2.13 we can see that the mesh is divided by METIS into two domains in both 2D and 3D cases, the number of cells in each subdomain are comparatively the same.

2.4 Parallel sparse linear algebra methods

2.4.1 Sparse linear algebra methods

Solving linear systems of equations has always been a precious approach to solve real life problems in numerous domains such as physics, geometry, etc. About four thousand years ago, Babylonians had already discovered how to solve a 2 x 2 linear system. In the early 19th century, Carl Gauss developed a method called "Gaussian elimination" in order to solve linear systems of equations.

The scientific community has developed multiple methods to efficiently solve sparse linear systems of the form

$$Ax = b \tag{2.5}$$

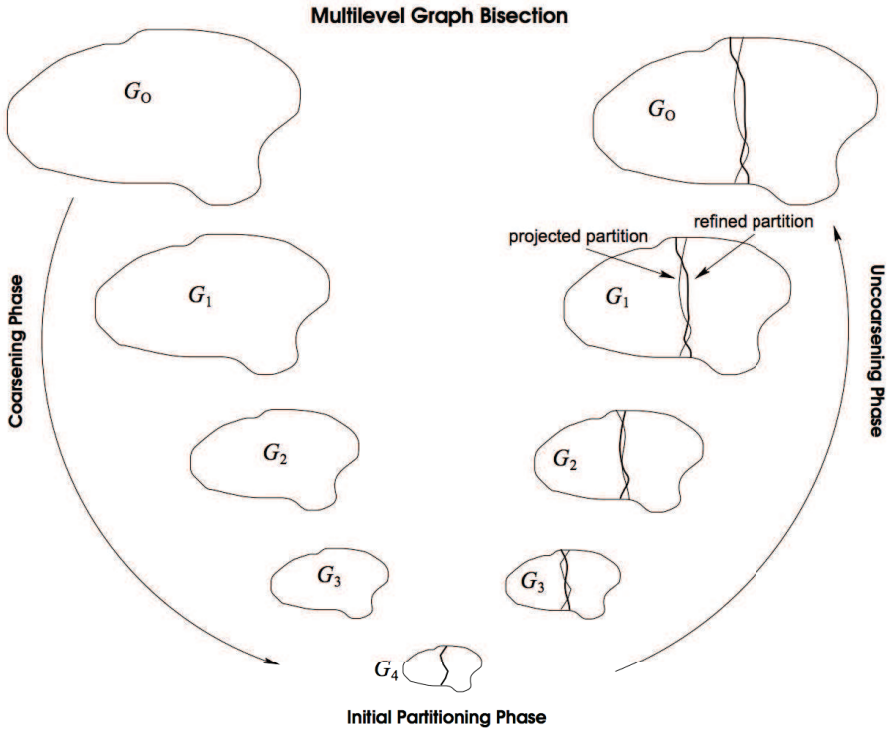


Figure 2.11: The various phases of the multilevel graph bisection. During the coarsening phase, the size of the graph is successively decreased; during the initial partitioning phase, a bisection of the smaller graph is computed; and during the uncoarsening phase, the bisection is successively refined as it is projected to the larger graphs. During the uncoarsening phase the light lines indicate projected partitions, and dark lines indicate partitions that were produced after refinement.

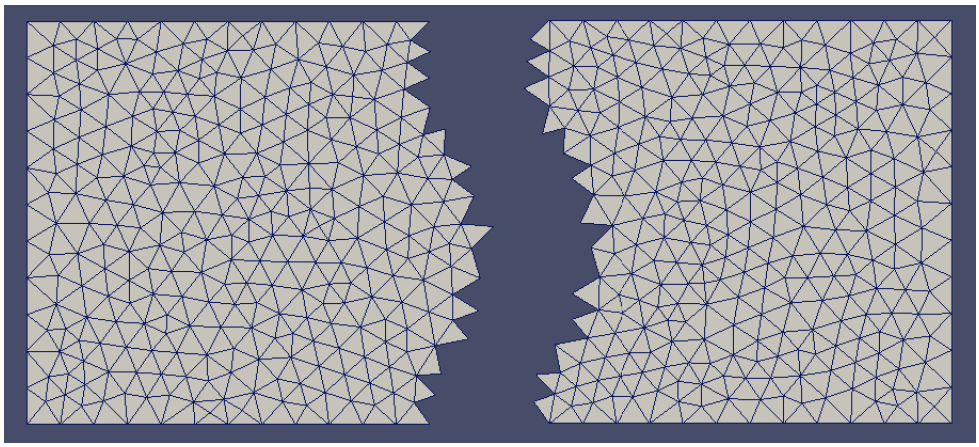


Figure 2.12: Unstructured 2D mesh

where A is a sparse matrix of size n. A is said to be a sparse matrix if it contains only few non zero terms. This low

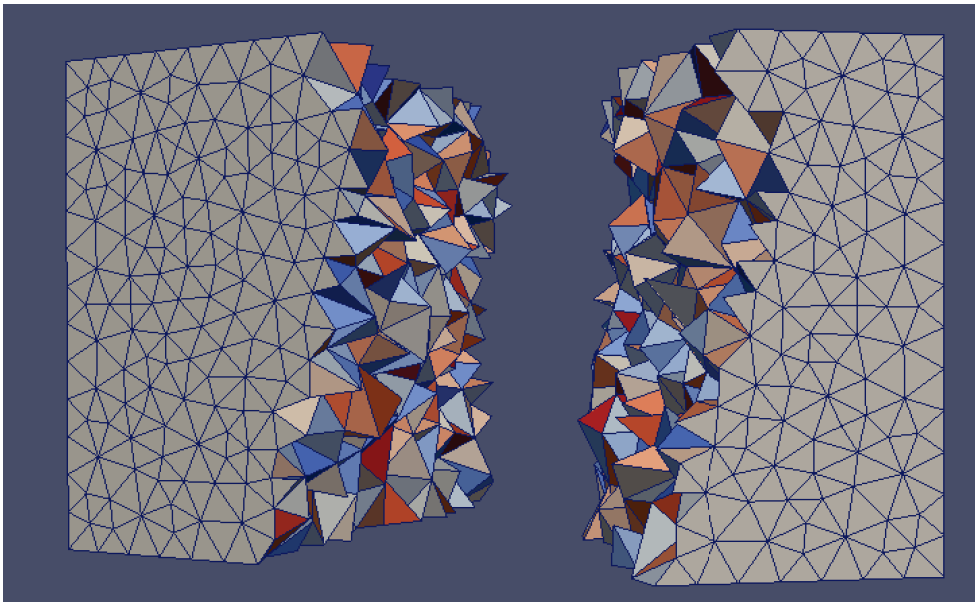


Figure 2.13: Unstructured 3D mesh

fill-in is owed to either nonexistent or neglected interactions in the mathematical model used by the simulation. Indeed, the coefficients of the matrix usually represent the interactions between two points resulting from the discretization of the studied problem. Frequently, interactions between two remote points can be neglected and zero terms appear in the matrix. x and b are two n sized vectors. x is the unknown of the system to be solved and b is called the right-hand side. x can then be obtained with the formula $b = A^{-1}x$.

However, the computation of the matrix A^{-1} is generally avoided. Indeed, we have no a priori knowledge of the structure of A^{-1}, and it might require being stored in an expensive dense matrix. This computation would require too much memory and computational time. The scientific community developed many other methods to obtain the solution x. These methods are adapted to the characteristics and the constraints of the simulation code. Indeed, the need for accuracy, the structural and numerical stability of the matrix at each time step, and its numerical complexity depends on the studied problem and the method used for the simulation.

There are two main methods for solving sparse linear systems: direct methods and iterative methods.

2.4.1.1 Direct methods

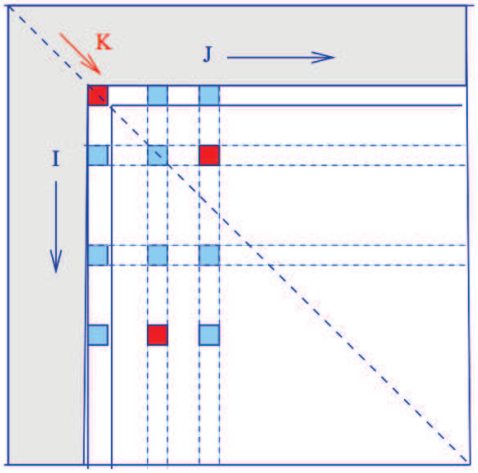
This section is devoted to direct resolution methods. It presents the graph models for the analysis of factorization process and the steps needed to build a successful direct factorization. One may refer to [20, 21, 22] and references therein for a more complete view of the direct methods. The direct resolution methods involve calculating the solution x of the system (2.5) by the successive resolutions of triangular or diagonal systems obtained by the factorization of matrix A. There are different types of factorizations, foremost among which are:

- LU factorization : a regular matrix A can be decomposed as $A = L.U$ where L is a lower triangular matrix with unit diagonal and U an upper triangular matrix;
- Cholesky factorization : a positive definite symmetric matrix A can be decomposed in the form $A = L.L^T$ where L is a lower triangular matrix;
- Crout factorization : a positive definite symmetric matrix A can be decomposed in the form $A = L.D.L^T$ where L is a lower triangular matrix with unit diagonal and D a diagonal matrix. Note that it is possible to use the $L.D.L^T$ factorization for positive non-definite symmetric matrices. In this case the D matrix is block-diagonal.

After factorization, the initial system solution can be obtained by successively solving triangular systems. For example, in the case of a factorization "LU", we solve:

$$\begin{cases} L.y = b \\ U.x = y \end{cases} \tag{2.6}$$

Algorithm 1: Scalar LU factorization



1

Inputs : $A = (a_{ij})$ regular matrix of dimension n Outputs : matrices L and U such as $L.U=A$

```
for k:=1 to N do
    for i:=k+1 to N-1 do
        a_ik = a_ik/a_kk
        for j:=k+1 to N do
            a_ij = a_ij - a_ik * a_kj
        end
    end
end
```

Algorithm 1 presents the factorization algorithm $A = L.U$. The matrix A is factorized locally by replacing its terms by those of two triangular matrices L and U. When the matrix is sparse, only non-zero elements are stored and operations on zero elements are removed. But when assigning the line 7 of the algorithm, if a_{ij} was zero before appointment, a new non-zero element is introduced into the factorized matrix $L.U$. During the factorization process, new non-zero terms are created and factors are fuller than the initial matrix.

2.4.1.2 Iterative methods

An iterative method is to solve a problem using an initial value x^0 that are refined to gradually get closer to the solution. So we construct a series of vectors $x^1, x^2, \cdots, x^k$ which converges towards the solution vector $x = (x_1, x_2, \cdots, x_n)$. The first traces of iterative methods for solving linear systems appear with the work of Gauss, Jacobi, Seidel, Nekrasov and in the nineteenth century (see [23] for details). The years 50-70 are dominated by the use of stationary iterative methods. These methods include modifying the components of the approximation of the solution in a certain order to achieve a satisfactory solution accuracy. the four major stationary iterative methods are:

- The Jacobi method is to calculate at step k a new approximation of each component $(x_1, x_2, \cdots, x_n)$ of the solution vector x independently. A new approximation x_i the ith equation of the system was extracted using the previous approximation x^{k-1} to the other terms of x present in the equation. A resolution is obtained by unknowns at each iteration from the solution of the previous iteration. This method is easy to implement but its convergence is slow.

- The Gauss-Seidel method is very similar to the Gauss method. It just allows to take into account new approximations of the components as they are calculated.

- The method of relaxation (SOR for successive over-relaxation) is an improvement of the previous method which introduces a relaxation parameter to improve the method of convergence speed.

In spite of their theoretical elegance, these iterative methods suffer from serious limitations such as lack of generics. Their convergence depends on parameters a priori difficult to estimate such as, for example, the spectrum of the matrix. In many practical problems, these methods hardly diverge or hardly converge. They are still used as multi-grid methods smoothers or as Krylov methods preconditioners.

Either a method is direct (LU, QR, etc.) and therefore is quick and accurate, but consumes lot of memory. Either it is iterative (GMRES, GC, etc.), memory-efficient, but slow enough to achieve a sufficient quality solution.

In this thesis, we are interested in the resolution of our large parallel sparse linear equations using the direct method. This method is the most expensive both in term of memory and computational power. However, as we have explained, it is widely used because it can reach a very high accuracy. We find many academic and industrial libraries dedicated to solving large sparse linear systems on massively parallel computers. Some of these libraries are specialized in direct methods. In general, this is the most consuming step of calculation time. Among the tools available as parallel direct methods we can mention : SuperLU, PARDISO, PaStiX and finally MUMPS that we use and which is among the fastest hollow parallel matrix solvers.

2.4.2 Parallel solver MUMPS

MUMPS has been developed by teams of three laboratories: CERFACS, ENSEEIHT-IRIT and RAL. Since the public (royalty-free) finalized version (MUMPS 5.0.2 July 2016). These substantive developments solve anomalies, extend the scope of use, improve ergonomics and above all, enrich the functionalities. MUMPS [24, 25, 26] is therefore a sustainable product, developped and maintained by teams of IRIT, CNRS, INRIA and CERFACS (half a dozen people).

It uses multi-frontal method to solve the system of linear equations on parallel computers in a distributed environment and using MPI. The multi-frontal method for solving sparse matrix linear systems is a direct method based on the LU factorization of the matrix. In what follows, we will consider multi-frontal method that solves the linear system $Ax = b$; either if A is symmetric or not. In both cases, the structure of the matrix is first analyzed to determine an order which, in the absence of any digital pivoting will preserve parsimony in the factors. The approximation of the minimum level is used for model $A+A^T$. The resolution of the linear system equation is done in 3 basic steps:

1. During analysis, preprocessing (see Subsection 3.2), including an ordering based on the symmetrized pattern $A + A^T$, and a symbolic factorization are performed. During the symbolic factorization, a mapping of the multi-frontal computational graph, the so called elimination tree[33], is computed and used to estimate the number of operations and memory necessary for factorization and solution. Both parallel and sequential implementations of the analysis phase are available.

2. During factorization $A_{pre} = LU$ or $A_{pre} = LDL^T$, depending on the symmetry of the preprocessed matrix, is computed. The original matrix is first distributed (or redistributed) onto the processors depending on the mapping computed during the analysis. The numerical factorization is then a sequence of dense factorization on so called frontal matrices. In addition to standard threshold pivoting and two-by-two pivoting (not so standard in distributed memory codes) there is an option to perform static pivoting. The elimination tree also expresses independency between tasks and enables multiple fronts to be processed simultaneously. This approach is called multi-frontal approach. After factorization, the factor matrices are kept distributed (in-core memory or on disk); they will be used at the solution phase.

3. The solution x_{pre} of $LUx_{pre} = b_{pre}$ or $LDL^T\ x_{pre} = b_{pre}$ where x_{pre} and b_{pre} are respectively the transformed solution x and right-hand side b associated to the preprocessed matrix A_{pre}, is obtained through a forward elimination step.

$$Ly = b_{pre} \quad or \quad LD_y = b_{pre} \tag{2.7}$$

   followed by a backward elimination step

$$Ux_{pre} = y \quad or \quad L^T x_{pre} = y \tag{2.8}$$

   The solution x_{pre} is finally postprocessed to obtain the solution x of the original system $Ax = b$, where x is either assembled on an identified processor (the host) or kept distributed on the working processors. Iterative refinement and backward error analysis are also postprocessing options of the solution phase.

All these steps can be called up separately or together. This can be exploited to minimize the computation solution for the subsequent iterations in the solution of a set of nonlinear equations. For example, if the matrix structure does not change during iteration, the analyzing step can be skipped after the evaluation. Similarly if the matrix does not change, part of that analysis and factoring can be neglected.

2.4.2.1 Symbolic factorization and elimination tree

The symbolic factorization phase is not concerned with numerical values. Here we look for a permutation matrix that will reduce the number of operations and memory requirements in the subsequent phase and then calculate a dependency graph associated with factoring. This symbolic factorization organizes the factors. The removal of tree represents the task dependency calculation, it gives a partial order in which the columns can be eliminated.

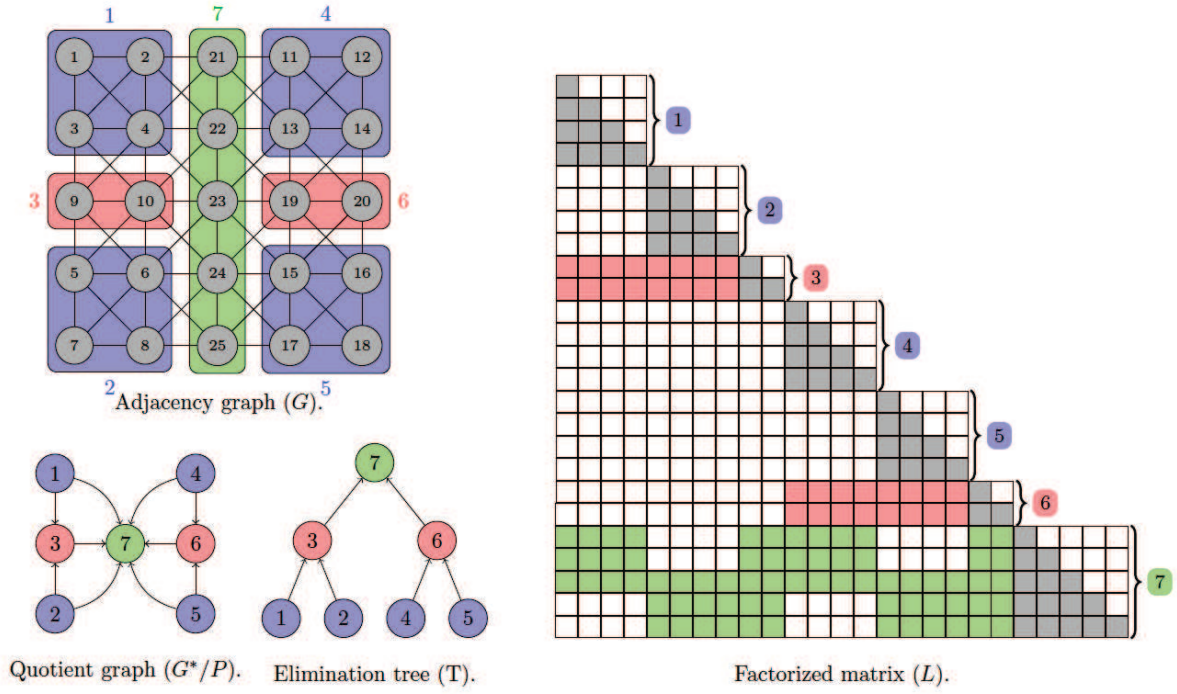


Figure 2.14: Symbolic factorization and elimination tree

2.4.2.2 Multifrontal method

You can see the multi-frontal method as a combination of left-looking and right-looking approches where all updates are sent after factoring in the current node, but only to his father. Figure 2.15 illustrates the principle of multi-frontal method to reduce the communication between the nodes of the tree.

Algorithm 2 presents the sequential multi-frontal LU algorithm. For each block column, the column block associated matrix is allocated and assembled using initial values and the sons contributions. The diagonal-block is factorized, the off-diagonal system is solved and the frontal matrix is updated. This frontal matrix is then passed to the column-block's father in the elimination tree. The latter adds up all the frontal matrices he receives with its own Schur complement before sending the result to its father and so on.

Algorithm 2: Multi-frontal factorization: $A = LU$.

for k:=*1* **to** N **do**
  /* Assemble the matrix using frontal matrices from sons */;
  Assemble the frontal matrix associated to $A_{k,k}$;
  /* Factorize diagonal block */;
  Factorize $A_{k,k}$ in $L_{k,k}.U_{k,k}$;
  /* Solve off-diagonal systems */;
  Solve $L_{(1-b_k),k}.U_{k,k} = A_{(1-b_k),k}$;
  Solve $L_{k,k}.U_{k,(1-bk)} = A_{k,(1-bk)}$;
  /* Update the Schur complement */;
  $A_{(1-bk),(1-bk)} = A_{(1-bk),(1-bk)} - L_{(1-bk),k}.U_{k,(1-bk)}$
end

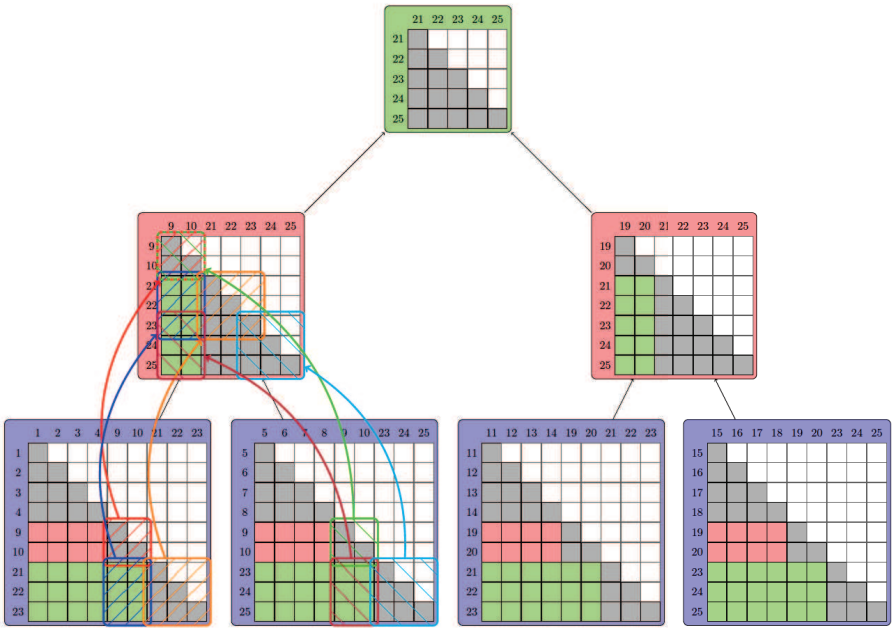


Figure 2.15: Multifrontal algorithm applied on the matrix from Figure 2.15 Level 2 frontal matrix is updated using the Schur complement of level 3 frontal matrices (represented with boxes and bent arrows for the most left bottom frontal matrices). And level 1 is updated by its two sons from level 2.

2.4.2.3 Pivot ordering

Suppose that each node in the tree corresponds to a task that consumes provisional data of children and produced preliminary data, which passes parental node. When the width of the tree is large, we have a good parallelism, many interim blocks to store and high memory usage. Moreover, when the tree is deep enough, we have less parallelism and memory usage is smaller. Figure 2.16 shows the different choices of order of the pivot.

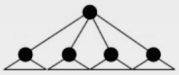
Reordering technique	Shape of the tree	observations
AMD		Deep well-balanced Large frontal matrices on top
AMF		Very deep balanced Small frontal matrices
PORD		Deep unbalanced Small frontal matrices
SCOTCH		Very wide well-balanced Large frontal matrices
METIS		Wide well-balanced Small frontal matrices(than SCOTCH)

Figure 2.16: Pivot ordering

2.5 CFD solvers

2.5.1 Background

The development of modern computation fluid dynamics (CFD) began with the advent of digital computer in the early 1950s mentioned in the book [27]. Recent progression in computing coupled with reduced costs of CFD software packages has advanced CFD as a viable technique to provide effective and efficient design solutions.

A partial differential equation is a relation between a function of several variables and its (partial) derivatives. Many problems in physics, engineering, mathematics and even banking are modeled by one or several partial differential equations. Freefem++ [28], trioCFD [29] and OpenFVM [30] are softwares to solve these equations numerically in dimensions two or three.

Library name	Finite Volumes method	Finite Elements method	Mesh adaptation	Mesh type
TrioCFD	No	Yes	Yes	Unstructured
FreeFem++	No	Yes	Yes	Unstructured
OpenFVM	Yes	No	No	Unstructured
ADAPT	Yes	Yes	Yes	Unstructured

Table 2.1: Main properties of ADAPT

2.5.2 Solving Fluid Dynamics problems

The numerical solution of Fluid Dynamics problems is done in these steps:

- Continuous physical problem is described by a continuous mathematical model (put into equations).
- Continuous mathematical model is discretized on the basis of one of the numerical methods.
- Discretized equations are approximated using the appropriate numerical schemes, the resolution algorithm is established.
- Algorithm is coded (C/C++, Fortran, Matlab, Java, ...).
- Code is executed on a computer.
- If all goes well, the approximate solution of the initial problem is obtained.

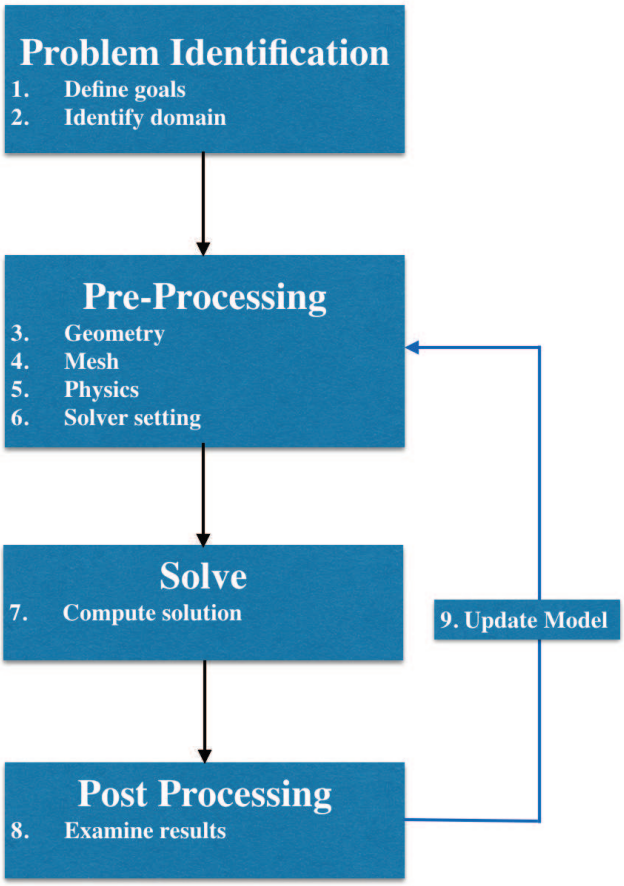


Figure 2.17: CFD Modeling Overview

Pre-Processing is the step where the modeling goals are determined and computational grid is created. In the second step numerical models and boundary conditions are set to start up the solver. Solver runs until the convergence is reached. When solver is terminated, the results are examined which is the post-processing part.

2.5.3 Classical discretization methods

A fundamental consideration for CFD code developers is the choice of suitable techniques to discretize the modeled fluid continuum. Of the many existing techniques, the most important include finite differences, finite elements and finite volumes. Although all these produce the same solution at high grid resolutions, the range of suitable problems is different for each. This means that the employed numerical technique is determined by the conceived range of code applications.

Finite differences methods (FDM) and finite elements methods (FEM), which are the basic tools used in the solution of partial differential equations in general and CFD in particular, have different origins. In 1910, at Royal Society of London, Richardson presented a paper of the first FDM solution for the stress analysis of a mansory dam. In contrast, the first FEM work was published in the Aeronautical Science Journal by Turner, Clough, and Topp in 1956s. Finite volume methods (FVM), because of their simple data structure, have become increasingly popular in recent years, their formulations being related to both FDM and FEM.

Finite differences techniques are of limited use in many engineering flows due to difficulties in their handling of complex geometries. This has led to increased use of finite elements and finite volumes, which employ suitable meshing structures to deal appropriately with arbitrary geometry. Finite elements [31] can be shown to have optimality properties for some types of equations. The first FEM work was published in the Aeronautical Science Journal by Turner, Clough, and Topp in 1956s.

When the governing equations are expressed through finite volumes, they form a physically intuitive method of achieving a systematic account of the changes in mass, momentum and energy as fluid crosses the boundaries of discrete spatial volumes within the computational domain [32]. The ease in the understanding, programming and versatility of finite volumes has meant that they are now the most commonly used techniques by CFD code developers. Table 2.2 summarize differences between the three methods.

Table 2.2: Classical discretization methods

	Finite differences method FDM	Finite volumes method FVM	Finite elements method FEM
Programmation	Simple implementation	Conservation of quantities (flux, mass, energy, ...)	Very precise High scalability
Convergence	Very slow computation Structured/Unstructured (adaptive) grids High detail require denser mesh	Faster computation High memory consumption High detail require denser mesh	Monstly unstructured grids High memory consumption Complex mesh implementation Complex mesh generation For strong deformations remeshing required
Typical application	Misc(fluids, solids, ...)	Fluids	Solids

Chapter 3

Computer systems and Services

Contents

3.1 Cloud computing

3.1.1 Introduction

Cloud computing corresponds to the concept of using a network of servers on the internet to process, store and manage data. The NIST[1] defines the concept as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models.

The NIST report says that the essential characteristics of cloud computing are:

1. On-demand self-service. A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider.

[1] http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

2. Broad network access. Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g., mobile phones, tablets, laptops, and workstations).

3. Resource pooling. The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter). Examples of resources include storage, processing, memory, and network bandwidth.

4. Rapid elasticity. Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.

5. Measured service. Cloud systems automatically control and optimize resource use by leveraging a metering capability1 at some level of abstraction appropriate to the type of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service.

According to the same report, the service models are:

1. Software as a Service (SaaS). The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure2. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user- specific application configuration settings.

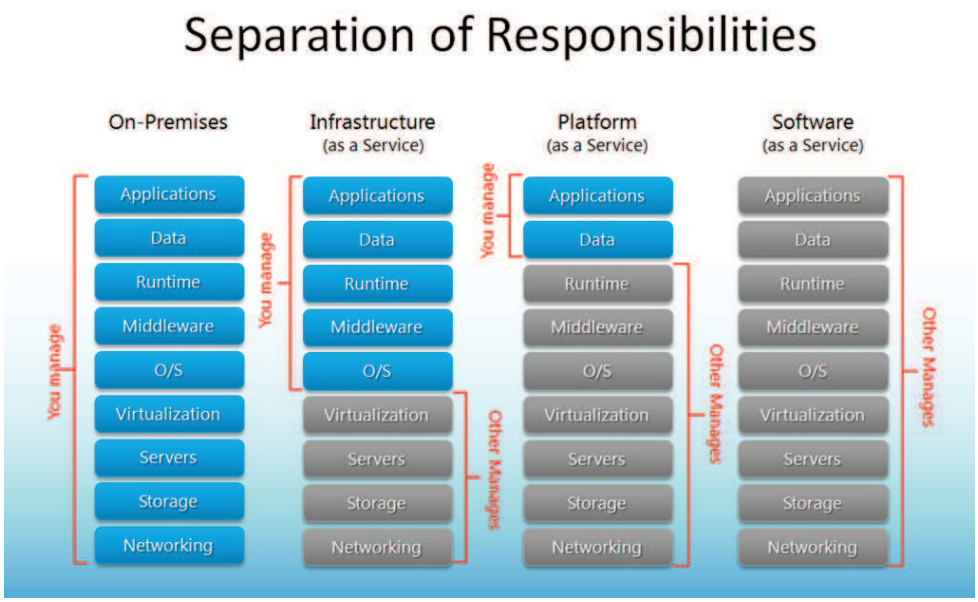


Figure 3.1: Separation of responsibilities

You manage means you are managing the system as an administrator system (low level). For the SaaS 3.1 the user has just to use the software and noting else.

2. Platform as a Service (PaaS). The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider.3 The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment.

3. Infrastructure as a Service (IaaS). The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls).

Since this time, the definition now includes similar concepts such as NaaS (Network as a Service) to refer to the ability to virtualize almost all the functionalities of a router.

Concerning the deployment models, the above mentioned NIST report considers:

1. Private cloud. The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises.

2. Community cloud. The cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be owned, managed, and operated by one or more of the organizations in the community, a third party, or some combination of them, and it may exist on or off premises.

3. Public cloud. The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider.

4. Hybrid cloud. The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds).

3.1.2 Others cloud deployment models

Distributed cloud is a cloud computing platform that can be assembled from a distributed set of machines in different locations, connected to a single network or hub service. It is possible to distinguish between two types of distributed clouds: public-resource computing and volunteer cloud:

- Public-resource computing—This type of distributed cloud results from an expansive definition of cloud computing, because they are more akin to distributed computing than cloud computing. Nonetheless, it is considered a sub-class of cloud computing, and some examples include distributed computing platforms such as BOINC and Folding@Home.

- Volunteer cloud—Volunteer cloud computing is characterized as the intersection of public-resource computing and cloud computing, where a cloud computing infrastructure is built using volunteered resources. Many challenges arise from this type of infrastructure, because of the volatility of the resources used to built it and the dynamic environment it operates in. It can also be called peer-to-peer clouds, or ad-hoc clouds. An interesting effort in such direction is Cloud@Home, it aims to implement a cloud computing infrastructure using volunteered resources providing a business-model to incentivize contributions through financial restitution.

The *Intercloud* concept is an interconnected global "cloud of clouds" and an extension of the Internet "network of networks" on which it is based. The focus is on direct interoperability between public cloud service providers, more so than between providers and consumers (as is the case for hybrid- and multi-cloud)

The *Multicloud* concept is to refer to the use of multiple cloud computing services in a single heterogeneous architecture to reduce reliance on single vendors, increase flexibility through choice, mitigate against disasters, etc. It differs from hybrid cloud in that it refers to multiple cloud services, rather than multiple deployment modes (public, private, legacy).

3.1.3 How to use?

Cloud computing is widely used, even for personal usages, as a means of storing information for users to be accessed wherever there is an internet connection, removing the need for an array of backups on a variety of devices

The servers are often maintained by cloud service providers such as Apple's iCloud service or Google Drive, who initially offer a limited amount of storage for free – iCloud offering an initial 5GB of storage which can later be upgraded into an indefinite amount for a fee which covers the cost of the storage facilities required for the function of network storage. almost the same is true for Amazon and its S3 service for data.

The cloud system consists of two portions – the front and the back end. The front end usually corresponds to the users device and network used to connect to the resources on the cloud system. The user utilises the application to access the cloud computing system through an interface, this could be through a web browser as for CUMULUS or a specific application such as Dropbox and once connection is established the user is free to use the service such as retrieving files or amending documents.

The backend of a cloud system consists of various computers, servers and storage systems. These devices perform and host the services the user wishes to access; such as a Web server, a database server... or, in our case, a system for computing. In our views, computation is also a service. Potentially, any kind of service can be implemented through a cloud solution.

The CUMULUS[2] cloud available at Sorbonne Paris cité is an example of private cloud. The members of the IDV (Imageries du Vivant) project uses it as a community cloud. On engineer is dedicated for the management of the virtual IDV data center which is a confined infrastructure inside CUMULUS.

The Figures 3.2 and 3.3 depict a session of the CUMULUS cloud that the user access through a browser. A Ubuntu Linux has been deployed, and inside this system, a browser has been launched in order to access to a service related to images. It is important to notice that all the services are running on CUMULUS (Ubuntu and the Web service for managing images) and only the display is done on the user's device. The benefit is that we have concentrated the data on the same location making more efficient the computation over data because of the principle of data-locality.

Cloud computing has a multitude of benefits and limitations which must be considered within computer systems. A major factor in cloud computing is cost efficiency, for instance licensing fees and electronic equipment can easily rack up a large expense. The fees for renting a service may be low if the user rent the service on-demand (pay-as-you-go model) and not for ever (on the user's PC the service is always available, but it is not necessary the case for cloud).

Negatives aspects of cloud systems include technical issues which can arise at any time, downtime of a server of a cloud service can render the data stored within unreachable. Until 2014, the LIPN laboratory at Paris 13 university participated to the IWGCR[3] (International Working Group on Cloud Computing Resiliency) project to inspect cloud availability.

3.1.4 Examples of cloud services

3.1.4.1 Microsoft Azure

Since it started back in 2010, Microsoft Azure was known as Windows Azure up until 2014. It has since grown tremendously with multiple data centres around the world and availability of its services in over 22 regions worldwide. Azure provides Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) and is considered to be a market lead in IaaS.

[2] http://cirrus.uspc.fr/cumulus/
[3] http://iwgcr.org

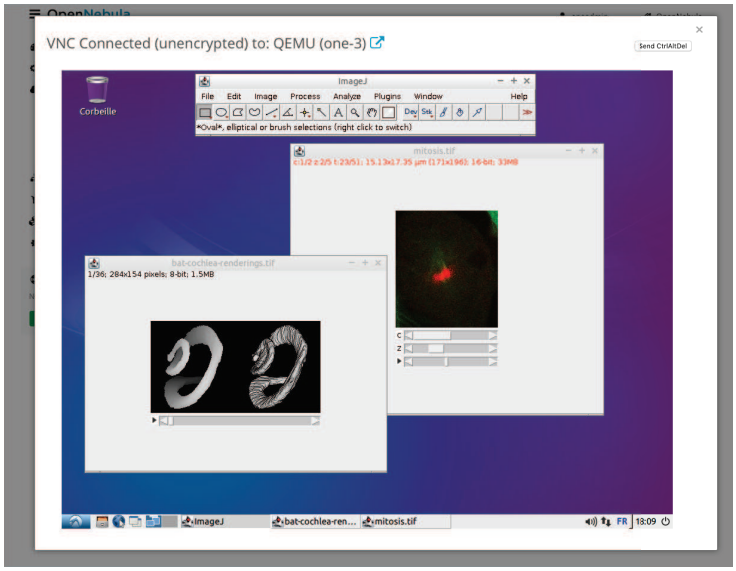


Figure 3.2: An example of a CUMULUS session: the ImageJ application is running inside a virtual machine

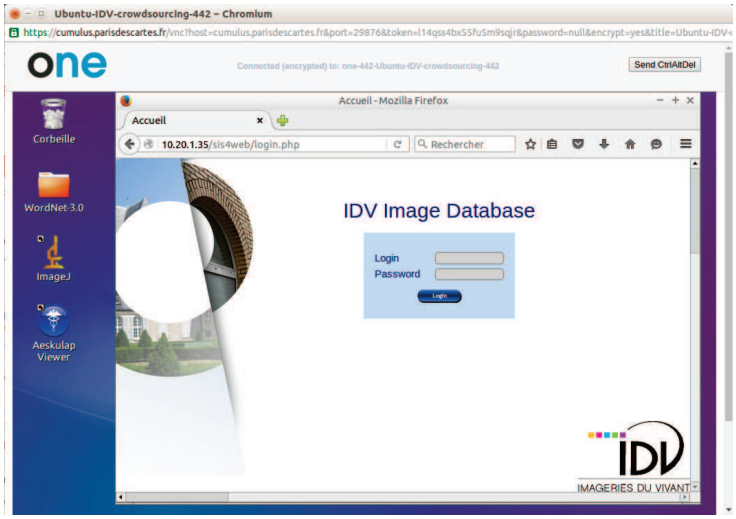


Figure 3.3: An example of a CUMULUS session: a navigator is running inside a virtual machine in order to access an image repository located inside the CUMULUS cloud.

From the services offered by Azure, users can run existing applications from the public cloud or develop new applications. A number of organizations opt to use Azure for disaster recovery, data backup and in some cases even as alternatives to local physical servers.

Azure's success has come due to the large variety of services available on the platform which include its ability to support a good number of programming languages. Its core services include computing, storage, networking and analytics.

3.1.4.2 Amazon Web Services (AWS)

AWS was developed in 2006, being among the first ever cloud computing platforms. Under amazon.com, the AWS platform was at first developed to offer online services for websites and client applications. AWS has multiple servers across the globe with central hubs and availability zones across different regions to minimize the impact of outages and ensure sustainability of the system.

AWS competes strongly with platforms like Microsoft Azure and Google Cloud Services. AWS offers multiple high speed, scalable and low-cost services like S3 for the storage.

Its four main features include networking, computing, storage and content delivery and database. These core features can further be divided into up to 36 services including to date;

3.1.4.2.1 S3 - Simple Storage Service. This is used for online backup and storage of application programs and other date.

3.1.4.2.2 CloudDrive . This allows users upload and access multiple files like documents, music, videos and photos provided they have internet-connected devices. This also includes streaming.

3.1.4.2.3 CloudSearch . This is a search service that enables customized search in other applications.

3.1.4.2.4 ElastiCache. This is a caching service that works to increase speeds of Web applications by avoiding the database load. It is compliant with Memcached which is a high performing, open source object caching system with distributed memory.

3.1.4.2.5 DDB - Dynamo Database. This service is for NoSQL (Not Only SQL) database and is known for scalability and low latencies.

3.1.4.2.6 RedShift. This is a data warehouse service for analytic workloads. It connects to SQL -based clients additional business tools.

3.1.4.2.7 Elastic Compute Cloud – EC2. This service serves as an unlimited group of virtual machines/VMs and helps business clients to access and run application programs.

3.1.4.3 Google Cloud Platform

Google says that "Google Cloud Platform lets you focus on what's next for your business. Google Cloud Platform frees you from the overhead of managing infrastructure, provisioning servers and configuring networks." The Google services have nicknames with clear meanings, among them the *Google App Engine* that lets you build, run, and scale applications without breaking a sweat, the *Google Compute Engine* delivers virtual machines running in Google's innovative data centers and worldwide fiber network. Compute Engine's tooling and workflow support enable scaling from single instances to global, load-balanced cloud computing, the *BigQuery service* which is Google's fully managed, petabyte scale, low cost analytics data warehouse. BigQuery is serverless, there is no infrastructure to manage and the user don't need a database administrator, so he can focus on analyzing data to find meaningful insights, use familiar SQL, and take advantage of the Google pay-as-you-go model.

The *Google Cloud Dataflow* delivers high-scale data processing through an Open Source API. Dataflow is a unified programming model and a managed service for developing and executing a wide range of data processing patterns including ETL, batch computation, and continuous computation. Cloud Dataflow frees you from operational tasks like resource management and performance optimization. Built upon services like Google Compute Engine, Dataflow is an operationally familiar compute environment that seamlessly integrates with Cloud Storage, Cloud Pub/Sub, Cloud Datastore, Cloud Bigtable, and BigQuery. The open source Java- and Python-based Cloud Dataflow SDKs enable developers to implement custom extensions and to extend Dataflow to alternate service environments. We will talk later on of the Pub/Sub paradigm because is it central to the workflow engine we will use to experiment with our HPC codes.

3.1.5 Cloud enabling technology

Cloud computing has evolved from number of existing technologies. These technologies are the predecessor of cloud computing and provide the much needed foundation. We shortly introduce Web 2.0 technologies and Grid computing, then, in the next section, we develop the concept of Service computing.

3.1.6 Web 2.0

The Web 2.0 is the successor of Web 1.0 and it provides environment for participation and collaboration for different entities having same interest. There are many definitions of Web 2.0; some of these definitions have been given as follows.

Web 2.0 takes place on Internet and not limited to a single software product. It is open, shared and users provide content and add value. Web 2.0 is about social publishing and not just developing web pages. It lowered the barrier of contribution from many users. In Web 2.0 data can be gathered, stored, analyzed and disseminated using online environment. It allows the end users to experience participation and interaction via a global, web based platform.

Aggregation services gathers the information from multiple sources and publish them in one place such as RSS feed. We intensively use the publish mechanism later on. These tools enable the web page to offer multiple pages from single site.

Data mashups are services capable to pull the data from multiple sources. Further, this data can be used by aggregators to filter the information.

All of these facilities are widely used in cloud computing for sharing the information among users and among different information providers.

3.1.7 Grid computing

The term grid computing originated in the early 1990s as a metaphor for making computer power as easy to access as an electric power grid. The power grid metaphor for accessible computing quickly became canonical when Ian Foster and Carl Kesselman published their seminal work, "The Grid: Blueprint for a new computing infrastructure" (1999).

Grid computing is a paradigm in which a number of network participate in sharing their resources. It can be viewed as a federation of clusters. Grid computing (see Figure 3.4) uses the resources of numerous computers in a network to work on a single problem at the same time. This is usually done to address a scientific or technical problem at the same time.

On a basic of volunteering, a well-known example of grid is the Search for Extraterritorial Intelligence (SETI) @Home project. In this project computer all over the world permits the SETI project to share the unused cycles of their computer to search for signs of intelligence.

The Desktop Grid Computing book [33] is a synthesis of work done during the past two decades. It covers many challenging problems, starting from the main components of a desktop grid. Later on, in this manuscript, we will provide with more information about them at the moment we will introduce the RedisDG workflow engine which is inspired by the ideas of desktop grids.

Desktop grids allows to build virtual supercomputers. Grid computing offers a way to solve Grand Challenge problems such as protein folding, financial modeling, earthquake simulation, and climate/weather modeling. The fastest virtual supercomputers are:

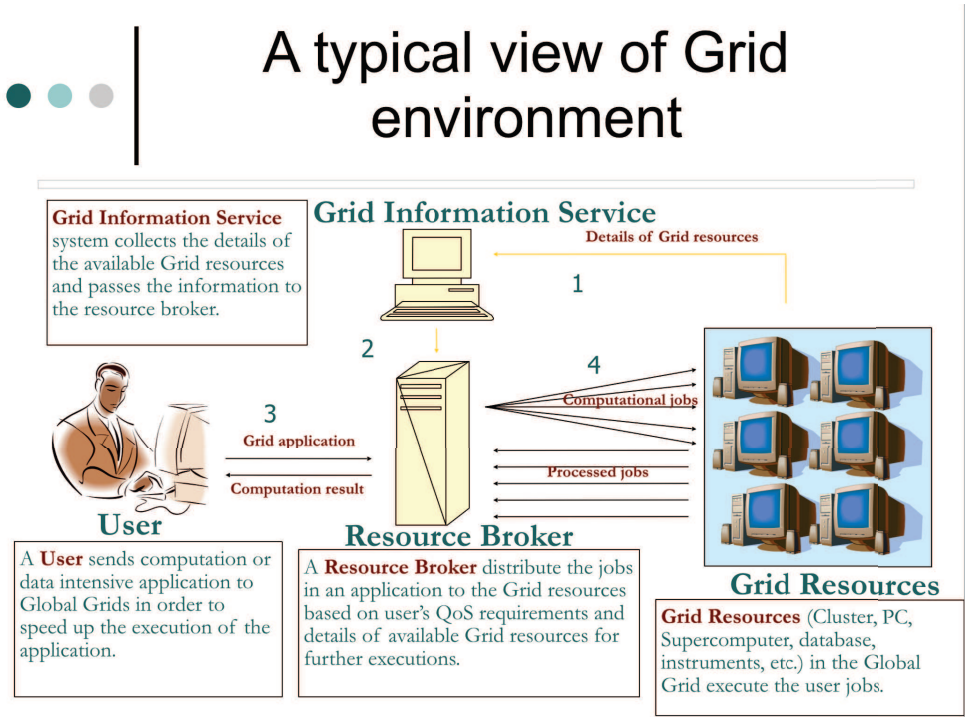


Figure 3.4: Grid computing

- As of August 2015, BOINC – 139 PFLOPS.
- As of August 2015, Folding@home – 36.3 x86-equivalent PFLOPS.
- As of August 2015, Einstein@Home 0.801 PFLOPS.
- As of August 2015, SETI@Home 0.677 PFLOPS.
- As of August 2015, MilkyWay@Home 0.381 PFLOPS.
- As of August 2015, GIMPS 0.235 PFLOPS.

Desktop grids offer computing power at low cost. Desktop grids (DGs) are built out of commodity PCs and use Internet as the communication layer. DGs also aim at exploiting the resources of idle machines over Internet. Indeed, DGs have important features that explain the large number of international projects aiming to better exploit this computational potential. Many DGs systems [33] have been developed using a centralized model. The most popular are BOINC [34], Condor [35], OurGrid [36] and Xtremweb [37].

To summary, in the mid-1990, grid computing was introduced to combine computers from multiple administrative domains to reach a common goal. Internet was the communication layer to federate the domains and this fact is noticeable.

One of the main strategies of grid computing is to use middleware to divide and apportion pieces of a program among several computers, sometimes up to many thousands. One can say that middleware were popularized into the grid computing world. At this time, many middleware were developed, among them Globus Toolkit, gLite, and UNICORE.

Globus online[4] can be seen as the successor of Globus and it has been merge with the Globus project. Globus Online addresses the challenges faced by researchers in moving, sharing, and archiving large volumes of data among

[4]https://www.globus.org

distributed sites. With Globus Online, you hand-off data movement tasks to a hosted service that manages the entire operation, monitoring performance and errors, retrying failed transfers, correcting problems automatically whenever possible, and reporting status to keep you informed while you focus on your research. Command line and web-based interfaces are available. The command line interface, which requires only ssh to be installed on the client, is the method of choice for grid-based workflows.

The Enabling Grids for E-sciencE project[5], based in the European Union and included sites in Asia and the United States, was a follow-up project to the European DataGrid (EDG) and evolved into the European Grid Infrastructure. This, along with the LHC Computing Grid (LCG), was developed to support experiments using the CERN Large Hadron Collider.

3.2 Service Computing

3.2.1 Introduction

A challenging cloud topic is on how to automate the process through which the user can provision resources on-demand. By minimizing user involvement, automation speeds up the process, reduces labor costs and reduces the possibility of human errors. Cloud computing adopts concepts from Service-oriented Architecture (SOA) that can help the user break these problems into services that can be integrated to provide a solution. Cloud computing provides all of its resources as services, and makes use of the well-established standards and best practices gained in the domain of SOA to allow global and easy access to cloud services in a standardized way.

Services Computing [38] has become a cross-discipline that covers, for our concerns, Web services and service-oriented architecture (SOA), cloud computing, business process modeling. This scope of Services Computing covers almost the whole life-cycle of services innovation research that includes business componentization, services modeling, services creation, services realization, services annotation, services deployment, services discovery, services composition, services delivery, service-to-service collaboration, services monitoring, services optimization, as well as services management. The goal of Services Computing is to enable IT services and computing technology to perform business services more efficiently and effectively.

More specifically, the IEEE Transactions on Services Computing (TSC) journal is a journal that focuses on research on the algorithmic, mathematical, statistical and computational methods that are central in services computing; the emerging field of Service Oriented Architecture, Web Services, Business Process Integration, Solution Performance Management, Services Operations and Management. The transactions covers but is not limited to the following topics: Mathematical foundation of Services Computing, Service-Oriented Architecture (SOA), Service creation, development, and management, Linkage between IT services and business services, Web services security and privacy, Web services agreement and contract, Web services discovery and negotiation, Web services management, Web services collaboration, Quality of Service for Web services, Web services modeling and performance management, Solution frameworks for building service-oriented applications, Composite Web service creation and enabling infrastructures, Business and scientific applications using Web services and SOA, Business process integration and management using Web services, Standards and specifications of Services Computing, Utility models and solution architectures, Resource acquisition models in Utility Computing, Mathematical foundation of business process modeling, integration and management, Business process modeling, integration, and collaboration.

Although service computing has become a new research are both in academia and industry, it has not a unified concept as we can note from the bove long list of terms.. As it is in a continuously developing process, its definition and connotation are also constantly changing.

Mike P. Papazoglou, who is from the software system design and development perspective, thinks that "Service computing is a way of developing application systems with services as the basic elements" [39]. Munindar P Singh and Michael N. Huhns, who are from the application of service technology, think that "Service computing is the set of technologies that combine service concept, service system architecture, service technology and service infrastructure together to guide how to use these services" [40]. Maria E. Orlowska and Sanjiva Weerawarana, who are from the distributed computing field, think that "Service computing is a kind of distributed computing paradigm evolved from

[5] https://www.egi.eu

object-oriented and component-oriented computing. It makes different application systems that are distributed, achieving rapid, flexible seamless integration and cooperation" [41]. Liangjie Zhang, who is from the discipline perspective, thinks that "Service computing is a basic discipline that crosses computers, information technology, commercial management and consultation services. Its objective is to use service science and service technology to eliminate the gap between commercial services and information technology services" [42, 43].

3.2.2 Service Oriented Architecture (SOA)

A service-oriented architecture (SOA) is a style of computer software where services are provided to the other components by application components, through a communication protocol over a network. The basic fundamental principles of service oriented architecture is independent of vendors, products and technologies. A service is a discrete unit of functionality that can be accessed remotely and acted upon and updated independently.

A service has four properties according to one of many definitions of SOA:

1. It logically represents a business activity with a specified outcome.
2. It is self-contained.
3. It is a black box for its consumers.
4. It may consist of other underlying services.

Different services can be used in conjunction to provide the functionality of a large software application. Service-oriented architecture makes it easier for software components to communicate and cooperate over the network, without requiring any human interaction or changes in the underlying program, so that service candidates can be redesigned before their implementation.

According to the Wikipedia definition of Service Oriented Architecture [6] each SOA building block can play any of the three roles:

- **Service provider**. It creates a web service and provides its information to the service registry. Each provider debates upon a lot of hows and whys likes which service to expose, whom to give more importance: security or easy availability, what price to offer the service for and many more. The provider also has to decide what category the service should be listed in for a given broker service and what sort of trading partner agreements are required to use the service.
- **Service broker**. Is is also known as the service registry. Its main functionality is to make the information regarding the web service available to any potential requester. Whoever implements the broker decides the scope of the broker. Public brokers are available anywhere and everywhere but private brokers are only available to a limited amount of public.
- **Service requester/consumer**. It locates entries in the broker registry using various find operations and then binds to the service provider in order to invoke one of its web services. Whichever service the service-consumers need, they have to take it into the brokers, bind it with respective service and then use it. They can access multiple services if the service provides multiple services.

The service consumer-provider relationship is governed by a service contract, which has a business part, a functional part and a technical part.

Later on, we will introduce the RedisDG workflow engine that follows, as much as possible, the above mentioned principles. This service is built upon a broker, the consumers can be view as the workers requesting work to do. RedisDG implement a very basic Service provider building block since in our case the cloud layer, above the RedisDG service, implements it. The consumer-provider relationship is also under the responsibility of the cloud layer.

[6] https://fr.wikipedia.org/wiki/Architecture_orienté_services

3.2.3 Microservices

Microservices is a specialization of and implementation approach for service-oriented architectures (SOA) used to build flexible, independently deployable software systems. As with SOA, services in a microservice architecture (MSA) are processes that communicate with each other over a network in order to fulfill a goal. Also, like SOA, these services use technology-agnostic protocols. The microservices approach is a first realization of SOA that followed the introduction of DevOps[7] and is becoming more popular for building continuously deployed systems.

In a microservices architecture, services should have a small granularity and the protocols should be lightweight. A central microservices property that appears in multiple definitions is that services should be independently deployable. The benefit of distributing different responsibilities of the system into different smaller services is that it enhances the cohesion and decreases the coupling. This makes it easier to change and add functions and qualities to the system at any time. It also allows the architecture of an individual service to emerge through continuous refactoring.

To summarize, our personal view of microservices is the following and constitutes one possible definition:

1. Services are designed to perform a single function;

2. The project organization should consider AUTOMATION, DEPLOYMENT and testing;

3. Each service is ELASTIC, RESILIENT, modular, minimal and complete;

We put in capital letters, some keywords that are common between the worlds of microservices and cloud computing. This explains what cloud computing can bring to the field of microservices. But microservices approach builds on several best practices, standards and patterns for software design, architecture, and DevOps style organization that can also cross-fertilize the field of cloud computing. At least, microservices requires expertise in distributed programming in particular for the interaction of microservices and in the context of 'faults'.

The contradiction is that some people think that the key principles and issues with microservices are not exactly the above-mentioned properties, and they argue:

1. Microservices is hype; subroutines, modular programing, object orientation, functional programming, aspect-oriented programming, distributed services, service-oriented architecture... All are in some way microservices;

2. The first modularization technique is probably the subroutine library across the machine room of EDSAC at Cambridge (May 1949) – It was later that monolithic programs appeared;

3. The major advantage of (micro)services is functional decomposition; This is important for maintenance (understandability, easier to locate errors), for re-use of code and for matching the code to the enterprise requirements;

4. A problem with (micro)services is that it is hard to maintain the NFRs (Non Functional Requirements) consistently across the services;

5. Another common feature is connections by message passing, not shared main memory; this has advantages for distribution but not for performance;

   3,4,5 are all relevant to cloud computing;

The RedisDG workflow engine we will introduce later on tries to follow the ideas of micro-services in the sense that, for instance, the scheduling services implement multiple policies that can be view as single and independent services. It is very easy to switch from one policy to another one.

[7] DevOps is a culture, movement or practice that emphasizes the collaboration and communication of both software developers and other information-technology (IT) professionals while automating the process of software delivery and infrastructure changes.

3.3 Workflow systems

3.3.1 Introduction

A scientific workflow system is a specialized form of a workflow management system designed specifically to compose and execute a series of computational or data manipulation steps, or workflow, in a scientific application. In this thesis 'workflow' and 'scientific workflow' are considered equivalent for sake of simplicity.

The simplest computerized scientific workflows are scripts that call in data, programs, and other inputs and produce outputs that might include visualizations and analytical results. These may be implemented in programs such as R or MATLAB, or using a scripting language such as Python or Perl with a command-line interface.

By focusing on the scientists, the focus of designing scientific workflow system shifts away from the workflow scheduling activities, typically considered by grid computing environments in the past and now by cloud computing environments for optimizing the execution of complex computations on predefined resources, to a domain-specific view of what data types, tools and distributed resources should be made available to the scientists and how can one make them easily accessible and with specific Quality of Service (QoS) requirements.

In our case, we will see later that QoS means mastering the load of the system and/or the fairness of the system and/or the energy consumed by the workflow system. We shift from a vision based only on the execution time to a vision with multiple objectives, all of them are measurable.

3.3.2 The RedisDG workflow engine

Resource utilization in computer systems is usually below 10% [44] due primarily but not only to resource over-provisioning. Efficient resource sharing in cloud is, as a consequence, challenging. Resource under-provisioning will inevitably scale-down performance and resource over-provisioning can result in idle time, thereby incurring unnecessary costs for the user. In this paper, in order to address the problem of finding a 'good' resource utilization, we use an old-fashionable computer engineering principle which is multiplexing processes on the same compute unit. But since the compute units have been allocated by a cloud system and are spread over multiple sites we also need to pay attention to the impact of the network performance on the effective sharing of resources.

The cloud user needs to be sure that all the reserved computing units are utilized and also that they are fully used (the processor load is as high as possible). These two conflicting objectives are under concern in the RedisDG framework. They are conflicting because either you distribute the tasks as much as possible on the different computing units but they are running under a low CPU utilization metric or you consolidate tasks on few computing units and you get first a better utilization and resource sharing on a subset of computing unit and, second you get idle time spent in the other subset of the computing units.

The usage scenario that has been implemented is as follows. We assume that a user has booked N compute units in the cloud. Now the user submit a scientific workflow to the cloud. The cloud system deploys the workflow engine (RedisDG in our case) on the N compute units and starts the processing of the workflow's tasks. There is no elasticity in the classical sense where new compute units are dynamically added/removed and no user intervention for instance to configure auto-scaling indicators and thresholds.

The RedisDG [45, 46, 47] system is a desktop grid middleware which is a light desktop grid middleware that has been formally designed and easy to cloudify. It makes possible to offer 'Desktop Grid as a Service' in a cloud context. Note that the cloudification of BOINC or Condor is also possible by at a very high price for the integration [48].

RedisDG has been integrated (cloudiflied) inside SlapOS [49] which is a distributed Cloud Computing system designed by Université de Paris 13 and NEXEDI. Marketed in Europe, Japan and China, it comprises deployment functions and automatic orchestration used in production by SANEF, Mitsubishi, Airbus Defence... From a research point of view, SlapOS has been studied in many contexts [50, 51, 52] ranging from resiliency to energy optimization and Software as a Service Data movement.

The goal and the difficulties are to define a desktop grid middleware able to support workflows that is light enough to be integrated easily into a cloud and using current Web technologies (in our case the Publish-Subscribe paradigm).

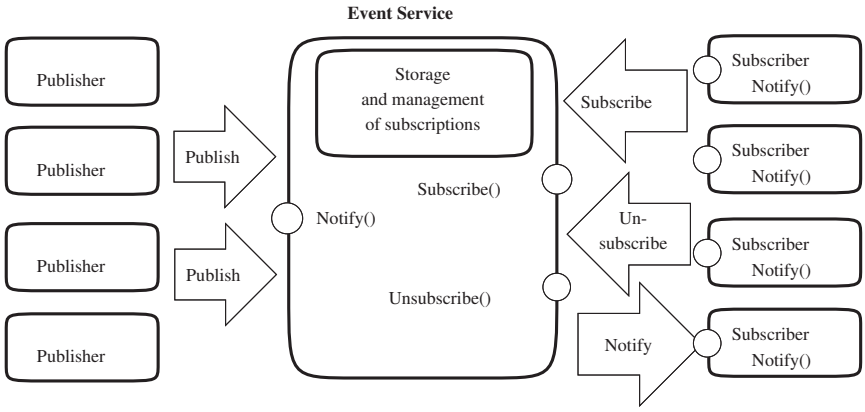


Figure 3.5: The publish-Subscribe paradigm in action

3.3.2.1 RedisDG modeling

The Publish-Subscribe paradigm is an asynchronous mode for communicating between entities. On Figure 3.5 some users, namely the subscribers or clients or consumers, express and record their interests under the form of subscriptions, and are notified later by another event produced by other users, namely the producers.

This communication mode is multipoint, anonymous and implicit. Thus, it allows spatial decoupling (the interacting entities do not know each other), and time decoupling (the interacting entities do not need to participate at the same time). This total decoupling between the production and the consumption of services increases the scalability by eliminating many sorts of explicit dependencies between participating entities. Eliminating dependencies reduces the coordination needs and consequently the synchronizations between entities. These advantages make the communicating infrastructure well suited to the management of distributed systems and simplify the development of a middleware for the coordination of components in a workflow engine context.

The paradigm also allows to implement reactive or opportunistic strategies: when an event occur, in our case the publication of a task to do, we decide on the 'best think' to do based on the knowledge available at the time the event was generated. Many modern computing platforms, as those introduced earlier in the manuscript, exhibit extreme levels of dynamic heterogeneity. The availability and relative efficiencies of such platforms' computing resources can change at unexpected times and in unexpected ways. Scheduling a computation for efficient execution on such a platform can be quite challenging, particularly when there are dependencies among the computation's constituent chores (jobs, tasks, etc.).

The LIPN team working on RedisDG started by performing a formal modeling, based on an initial modeling of the publish-subscribe paradigm and adapted to REDIS interactions. REDIS has been chosen as the framework supporting the Publish-subscribe paradigm. Information about the modeling can be found in [46] and it serves as guidelines for the global modeling of the RedisDG protocol; it as been verified in order to get confidence into the protocol to demonstrate that we have no deadlock and also some 'good' properties such as the coordinator cannot 'leave' the system before any other component.

3.3.2.2 RedisDG protocol

In this subsection we introduce the coordination algorithm of RedisDG system. It is the highest view possible. Some technical details are given in the experiments section. The algorithm is entirely based on the publication-subscription paradigm. The middleware obtained offers the same features as the majority of desktop grid middleware such as Condor and BOINC. It manages scheduling strategies especially the dependencies between tasks, the execution of tasks and the verification/certification of the results; since the results returned by the workers can be manipulated or altered by malicious workers. The general objectives for the RedisDG protocol are:

- Using an asynchronous paradigm (publish-subscribe) that ensures, as much as possible, a complete decoupling between the coordination steps (for performance reasons);
- Ensuring the system resilience by duplicating tasks and actors. Even if the system is asynchronous and the tasks are duplicated, we need to ensure the progress of tasks execution. We also assume that actors are duplicated for resilience reasons;

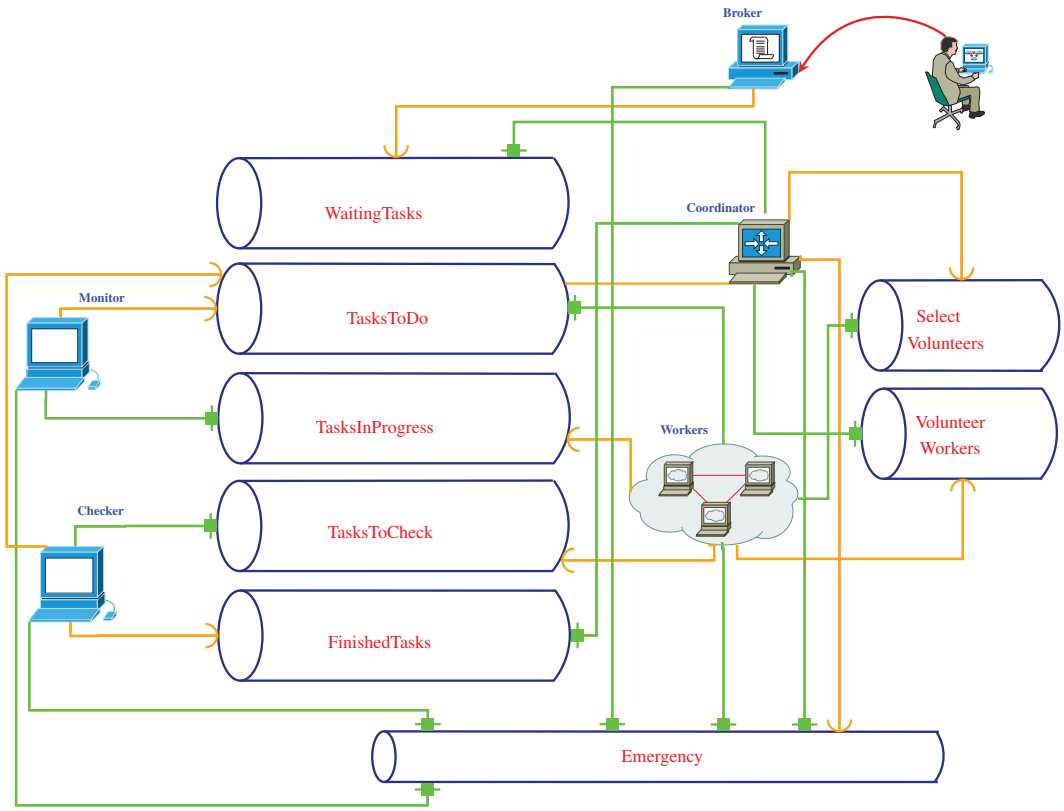


Figure 3.6: Interactions between components of the RedisDG system

In Figure 3.6, we present the steps of an application execution. In RedisDG, a task may have five states: *WaitingTasks, TasksToDo, TasksInProgress, TasksToCheck* and *FinishedTasks*. These states are managed by five actors: a broker, a coordinator, a worker, a monitor and a checker. Taken separately, the behavior of each component in the system may appear simple, but we are rather interested in the coordination of these components, which makes the problem more difficult to solve.

The key idea is to allow the connection of dedicated components (coordinator, checker, ...) in a general coordination mechanism in order to avoid building a monolithic system. The behavior of our system as shown in Figure 3.6 is as follows:

1. Tasks batches submission. Each batch is a series-parallel graph of tasks to execute.
2. The Broker retrieves tasks and publishes them on the channel called *WaitingTasks*.
3. The Coordinator is listening on the channel *WaitingTasks*.
4. The Coordinator begins publishing independent tasks on the channel *TasksToDo*.

5. Workers announce their volunteering on the channel *VolunteerWorkers*.

6. The coordinator selects Workers according to SLA criteria.

7. The Workers, listening beforehand on the channel *TasksToDo* start executing the published tasks. The event 'execution in progress' is published on the channel *TasksInProgress*.

8. During the execution, each task is under the supervision of the Monitor whose role is to ensure the correct execution by checking if the node is alive. Otherwise the Monitor publishes again, tasks that do not arrive at the end of their execution. It publishes, on the channel *TasksToDo*, in order to make the execution of the task done by other Workers.

9. Once the execution is completed, the Worker publishes the task on channel *TasksToCheck*.

10. The Checker verifies the result returned and publishes the corresponding task on the channel *FinishedTasks*.

11. The Coordinator checks dependencies between completed tasks and those waiting, and restarts the process in step (4).

12. Once the application is completed (no more tasks), the Coordinator publishes a message on the channel *Emergency* to notify all the components by the end of the process.

3.3.3 Volatility and result certification issues

The volatility of computing resources introduces challenging issues both for desktop grids and clouds. In this thesis, volatility means that some computing resources publish their volunteering to participate or to leave a project to help in realizing a computing task. Security and result certification issues are also important for desktop grids computing platforms.

Desktop grids use volunteer computers that are exposed to the Internet. Volunteer computers may leave or enter to the system at any time upon the decision of a person. Moreover, volunteer computers are untrusted: the result of running a job cannot generally be assumed to be correct. Malicious users may intentionally return wrong results, either to subvert the research or to get undeserved credit (when you participate for a project you receive back some credits that compensate your work).

Some applications have the property that results can be verified quickly on the server side, for instance by a checksum. Others allow plausibility checks, for example that energy has been conserved in a physical simulation. In other cases, projects must use replication to validate results and through majority voting.

Applications that do floating-point arithmetic typically get different results when run on different types of CPUs, or using different numerical libraries. This may make it difficult to decide when two results agree. For algorithms that are numerically stable, the results can b e compared with a fuzzy comparison function supplied by the project . For unstable algorithms (e.g., most physical simulations), desktop grids frameworks provide with special mechanisms. For instance, BOINC provides a mechanism called homogeneous redundancy (HR). When HR is used, BOINC ensures that all instances of a given job are issued to hosts that are numerically equivalent for that application.

For more details, chapter 11 of [53] contains a documented survey of techniques, strategies and best practices to enforce security and result certification properties in desktop grids. In this thesis we assume that the issues for results certification are solved and they are not part of our work.

Volatility issues, as we understand it in this thesis, may also happen in the case of clouds despite the fact that computing resources are confined into a single data center and are not exposed to the Internet. Assume that a cloud user may submit different jobs over a pool of rented resources. At a given time he may wishes to accelerate the completion of one job in allocating more computing resources to this job. In this case allocated resources need to 'leave' some jobs in order to be reallocated to the targeted job.

3.3.4 Related works on workflow scheduling

Applications in e-Science are becoming increasingly large-scale and complex. These applications are often in the form of workflows [54] such as MONTAGE, Blast [55], CyberShake [56] with a large number of software components and modules. Workflow and scheduling policies have been studied for one decade or two. In the reminder of this section we consider many categories of strategies. At a high level of abstraction, we focus first on the works that seek to optimize the execution time and/or QoS constraints of the workflows running in grid environments, and second works anchored in the Map-Reduce framework (data aware scheduling).

Workflow management systems related works are presented in the synthesis from Valduriez [57] or in the work of Carole Goble [58]. These two papers are more related to cloud computing and data intensive scientific workflows in putting an emphasis on data management.

Many studies as those in [59, 60, 61, 62] may also serve as complimentary readings on workflow scheduling and they can be considered as conventional works.

Many works validate the strategies through simulations. In our case our option is to run real world applications (ADAPT) which requires a special effort for designing an experimental plan, to check the reproducibility of the experiments... Moreover The context of these works are not adapted to our context because they do not take into account a dynamic view of the system in reacting to events when they arrive and they are 'clairvoyant'-like. Static information are supposed to be available (task graph, date of the events, task duration and costs...). They take a scheduling decision on the basis of a fixed number of workers. With the Publish-Subscribe paradigm in mind a 'more efficient' worker may join the system in the near future.

3.3.4.1 Scheduling and workflow management systems

In [63] authors consider the *pipelined workflow scheduling* where the execution of a large class of application can be orchestrated by utilizing task-, data-, pipelined-, and/or replicated-parallelism. Indeed, they focused on the scheduling of applications that continuously operate on a stream of data sets, which are processed by a given workflow, and hence the term pipelined. These data sets all have the same size (which is not part of our assumption) and the DAG (Directed Acyclic Graph) model is used to describe the applications. Authors also dealt mainly with the throughput and latency performance metrics which are not under concern in our work. The main contribution of this survey is in structuring existing works by considering different levels of abstraction (workflow models, system models, performance models).

In [64] authors investigated the problem of scheduling independent tasks under the paradigm of the master-worker. They consider heterogeneous situations where resources can have different speeds of computation and communication. The most interesting part of the work is to focus on the question of determining the optimal steady state scheduling strategy for each processor (the fraction of time spent computing and the fraction of time spent communicating with each neighbor). This question is quite different from the question of minimizing the total execution time, and the authors solve the problem is polynomial time. The paper demonstrate that we can observe the behavior of a system from a point of view that is not always focused on the execution time, as in our case.

In [65] authors considered the problem of dynamic load-balancing on hierarchical platforms. They focused more specifically on the *work-stealing* paradigm which is an online scheduling technique. They reviewed some existing variations and proposed two new algorithms, in particular the HWS algorithm which was analyzed in the case of fork-join task graphs. In their framework the authors considered that the graph is generated online during the execution. This is not our assumption. In our case we assume that new workers can potentially join the system in a dynamic way. However the analysis part in this work is interesting because it exemplifies the use of graph parameters such as the critical path. The execution time (mono criteria) is discussed in the paper as opposed to our work that offers a multi-criteria performance metric.

Swift [66] is an implicitly parallel programming language that allows the writing of scripts that manage program execution across distributed computing resources, including clusters, clouds, grids, and supercomputers. The JETS middleware [67] component is closely related to Swift and it provides high performance support for many-parallel-task computing (MPTC). The MTC model [68] consists of many, usually sequential, individual tasks that are executed on processor cores, without inter-task communication. The tasks communicate only through the standard file system interfaces, and optimization are possible [69]. We do not assume in our work the availability of a global file system. Data exchange between tasks are explicitly specified in the workflow description. With RedisDG, data exchange are

implemented through Redis servers or by a 'scp-like' implementation, in a transparent way from the user point of view.

The AWS cloud system [70] and some other cloud management services such as enStratus [71], RightScale [72], and Scalr [73] offer schedule-based (or predetermined) and rule-based (dynamic) auto-scaling mechanisms. Schedule-based auto-scaling mechanisms allow users to add and remove capacity at a given time that is fixed in advance. Rule-based mechanisms allow users to define simple triggers by specifying instance scaling thresholds and actions, for instance to add/remove instance when the CPU utilization verifies a certain property making the framework dynamic. These mechanisms are simple and convenient when users understand their application workload and when the relationship between the scaling indicator and the performance goal is easy to determine. From a purely cloud point of view it is not realistic to let the user make actions at this level: more automation is needed because clouds are for non expert users in order to serve requests on-demand and in a self-service way.

The paper [74] presents the Maximum Effective Reduction (MER) algorithm, which optimizes the resource efficiency of a workflow schedule generated by any particular scheduling algorithm. MER takes as input a workflow schedule generated by an existing scheduling algorithm then, with the allowance of a limited increase in the original makespan, it consolidates tasks into a fewer number of resources than that used for the original schedule. To do this, MER essentially optimizes the trade-off between makespan increase and resource usage reduction. The paper introduce three building blocks, firstly the delay limit identification algorithm for finding the minimum makespan increase for the maximal resource reduction, second the task consolidation algorithm and third the resource consolidation algorithm. Finally, MER is evaluated in a simulated environment with three different scheduling algorithms and under four different workflow applications.

In [75] authors present an approach whereby the basic computing elements are virtual machines (VMs) of various sizes/costs, jobs are specified as workflows, users specify performance requirements by assigning (soft) deadlines to jobs. Then, the optimization problem is to ensure all jobs are finished within their deadlines at minimum financial cost. One key point is to dynamically allocating/deallocating VMs and scheduling tasks on the most cost-efficient instances. Another key point about a user intervention is that authors use deadlines that serve as the performance requirements specified by the users, and deadline misses are not strictly forbidden. Authors use deadline assignment techniques to calculate an optimized resource plan for each job and determine the number of instances using the Load Vector idea (intuitively, the vector is the number of the machines needed to finish the task on VM_m).

3.3.4.2 A focus on workflow scheduling according to a Service Oriented view

In a series of works [76, 77] Marc Frîncu and all. explore the dynamic and unpredictable nature of the grid systems to offer mechanisms for adaptation at any given moment. For instance, the author proposed in [76] a scheduling algorithm which minimizes each task's estimated execution time by considering the total waiting time of a task, the relocation to a faster resource once a threshold has been reached and the fact that it should not be physically relocated at each reassignment but only at a precise moment, through a simple marking, to reduce the network traffic. The key advantage of the proposed solution is to consider tasks of multiple workflows when they arrive and not batch after batch. One drawback is that the algorithm is based on user estimates for the value of the execution time and authors propose that this information be obtained by using historical data and applying some learning mechanisms.

In [77] the focus is cloud computing and the authors noticed that vendors do prefer to use their own scheduling policies and often choose their negotiation strategies. In the framework of workflow scheduling, the goal in that paper is the minimization of the overall user cost. The problem addressed in the paper is to access services provided by different cloud vendors, each with their own internal policies. Two major problems are dealt with: finding cloud resources and orchestrating services from different cloud vendors. The key idea in the paper is, once the workflow is submitted, an agent tries to schedule the tasks on the best available service through negotiation with other agents. It can be noticed that no static scheduling decisions are made and that tasks are scheduled one by one as they become ready for scheduling.

In [78] authors developed the concept of *dynamic dataflows* which utilize alternate tasks as additional control over the dataflow's cost and QoS. Dataflow systems allow users to compose applications as task graphs that consume and process continuous data, and execute on distributed commodity clusters and clouds. The key point in the paper is that authors addressed the problem of scheduling tasks when the input rates changes. The goal is to build dataflow systems with a greater concern for self-manageability. Indeed authors investigated autonomous runtime adaptations in response to fluctuations in both input data rates and cloud resource performance. Authors formulated the underlying

optimization problem as a constrained utility maximization problem during the period for which the dataflow is executed. Then they first use meta-heuristics to solve it, for instance a genetic algorithm-based algorithm. Second they proposed greedy heuristics to find an approximate solution to the optimization problem. At last, they evaluated the proposed heuristics through a simulation study based (partly) on the popular CloudSim [79] simulator.

In [80] the authors addressed the lack of integrated support for data models, including streaming data, structured collections and files, that are limiting the ability of workflow engines to support emerging applications that are stream oriented. The key idea of the proposed framework is in its ability to transition from one data model to another one. The paper is more about architectural issues than scheduling issues. However the workflow framework evaluation is done on a private Eucalyptus cloud which is always challenging because of the complex nature of real systems.

3.3.4.3 Theoretical foundation of fair scheduling

In [59] authors reviewed the solutions for allocating suitable resources to workflow tasks so that the execution can be completed to satisfy objective functions specified by users. This paper can be considered as studying conventional solutions for the workflow scheduling problem. The context is Grid computing and the authors first introduced workflow management systems that define, manage and execute workflows on computing resources. Our context is more related to volunteer computing and we want to design, as a whole, the interactions of components, especially the interactions between the workflow scheduling and data movement components.

Moreover, there are two types of abstract workflow model, deterministic and non-deterministic. In a deterministic model, the dependencies of tasks and I/O data are known in advance, whereas in a non-deterministic model, they are only known at run time. In our case the dependencies are known in advance but nodes publish their volunteering. We may consider them as active and not passive, making a strong distinction with other works.

The heuristics recalled in [59] are based on the performance estimation for task execution and I/O data transmission. In our work, we do not make any assumption about performance estimation, apriori known. Dependency mode scheduling algorithms intends to provide a strategy to order and map workflow tasks on heterogeneous resources based on analyzing the dependencies of the entire task graph, in order to complete these interdependent tasks at earliest time. The strategy ranks the priorities of all tasks in a workflow application at one time. One issue with this strategy is to set weights on tasks. One idea is to set the weight of each task and edge to be equal to its estimation execution time and communication time. In our case we assume that the execution context is fluctuating (nodes may enter/leave the system at any time) making the estimates of weights a challenging problem.

Fair resource allocation problem address how to divide resources fairly to a set of users in a system where users have different resource demands. Previous works in computer science [81, 82, 81, 82, 83] mainly focused on the allocation of a single resource and took place in the field of networking. For instance authors first transform selection of the best users and rates issued from a complex general optimization problem into a set of two formulations: a multi-user scheduling problem that maximizes total system throughput and a control-update problem that ensures long-term deterministic or probabilistic fairness constraints.

In [84] authors proposed a multi-resource allocation mechanism, which is the generalized form extending a single resource allocation to multiple types resources allocation, known as Dominant Resource Fairness(DRF) mechanism. DRF equalizes user's dominant share among all users, and it provides several attractive fairness properties: sharing incentive (allocation should be better than dividing each resource equally among all users), envy-freeness (no user would like to trace his allocation with any other user), Pareto efficiency (no user can increase his allocation without decreasing allocation of other users) and strategy-proof (a user cannot increase her allocation by lying about her requirements). DRF has quickly attracted a lot of attention and has been extended to many dimensions.

Dolev and al. in [85] proposed another alternative notion of fairness for multi-resource allocation, called Bottleneck-Based Fairness (BBF). Roughly speaking, an allocation of resources is considered fair if every user either gets all the resources he wishes for, or else gets at least his entitlement on some bottleneck resource, and therefore cannot complain about not receiving more. Recently, Wang et al. in [86] proposed a new allocation model BAA based on the notion of per-device bottleneck sets. Th context of he work is multi-tiered storage made up of heterogeneous devices. Clients bottlenecked on the same device receive throughput in proportion to their fair shares, while allocation ratios between clients in different bottleneck sets are chosen to maximize system utilization.

Gutman and Nisan [87] formalized both fairness notions in economic terms, extending them to apply to a larger family of utilities. The technical results are algorithms for finding fair allocations corresponding to two fairness

notions. First of all, regarding the notion suggested by Ghodsi and al., they presented a polynomial-time algorithm that computes an allocation for a general class of fairness notions, in which their notion is included. Second, they showed that a competitive market equilibrium achieves the desired notion of fairness.

In [88] Wong and al. generalized the DRF mechanism and plunged it into a unifying framework trying to capture the tradeoffs between fair allocation and system efficiency. Intuitions behind the analysis are explained in two visualizations of multi-resource allocation.

Recently, Wang and al. [89] proposed DRFH, a generalization of DRF to an environment with multiple servers. They also designed a simple heuristic that implements DRFH in real-world systems. Large-scale simulations driven by Google cluster traces showed that DRFH significantly outperforms the traditional slot-based scheduler, leading to much higher resource utilization with substantially shorter job completion times.

Parkes and al. [90] extended DRF in several ways and considered the zero demands of some resources required by a user, and in particular they studied the case of indivisible tasks. Authors also charted the boundaries of the possible in this setting, contributing to a new relaxed notion of fairness and providing both possibility and impossibility results.

In [91] Cole and al. revisited the classic problem of fair division from a mechanism design perspective and provided an elegant truthful mechanism that yields surprisingly good approximation guarantees for the widely used solution of Proportional Fairness. They proposed a new mechanism, called the Partial Allocation mechanism, that discards a carefully chosen fraction of the allocated resources in order to incentivize the agents to be truthful in reporting their valuations. This mechanism introduced a way to implement interesting truthful outcomes in settings where monetary payments are not an option.

3.3.4.4 Pareto efficiency

Pareto [92] efficiency is a balance of resource distribution such that one individual's lot cannot be improved without impairing the lot of one or more other individuals.

The concept of Pareto efficiency is based on the work of Vilfredo Pareto, an Italian economist of the late 19th and early 20th century, who is better known for the Pareto principle. The concept derives from Pareto's work on income distribution and economic efficiency. In recent years, the principle has been applied to other areas of study including engineering, project management and the social sciences.

Within a given system, if one individual or other entity can be given a benefit without worsening the situation for any other individual or entity, doing so is known as a Pareto improvement. According to this concept, it is desirable to continue to make Pareto improvements until it is no longer possible to do so because a benefit to one individual would worsen the lot of one or more others. When no further Pareto improvements can be made, Pareto efficiency is said to have been reached.

3.3.4.5 Data-aware scheduling

In [93] authors focused on the MapReduce framework for processing massive data over computing clouds. The major factor affecting the performances of map-reduce jobs is locality constraints for reducing data transfer cost in using poor network bandwidth. They proposed a scheduling approach that provides 1) a data pre-placement strategy for improving locality and concurrency and 2) a scheduling algorithm considering locality and concurrency. They pointed a need for scheduling workflow services composed of multiple MapReduce tasks with precedence dependency in shared cluster environments. In this paper they also introduced the data cohesion score that we reuse in our paper.

Authors in [94] noticed a conflict between fairness in scheduling and data locality (placing tasks on nodes that contain their input data). They investigated this problem in designing a fair scheduler for a 600-node Hadoop cluster at Facebook. To address the conflict between locality and fairness, they proposed a simple algorithm called delay scheduling: when the job that should be scheduled next according to fairness cannot launch a local task, it waits for a small amount of time, letting other jobs launch tasks instead.

Authors found that delay scheduling achieves nearly optimal data locality in a variety of workloads and can increase throughput by up to 2x while preserving fairness. In addition, authors put forward the simplicity of delay scheduling that makes it applicable under a wide variety of scheduling policies.

The difference with our work is as follows. First authors consider multiple jobs and they try to optimize at this granularity level. We consider only the optimization of one job. Second they give a priority to local tasks, otherwise,

they skip to another job. If a job has been skipped long enough, they start allowing it to launch non-local tasks to avoid starvation.

Tudoran, Costan and Antoniu in [95] introduced OverFlow, a uniform data management system for scientific workflows running across geographically distributed sites. Their solution is environment-aware, as it monitors and models the global cloud infrastructure, offering high and predictable data handling performance for transfer cost and time, within and across sites. OverFlow proposes a set of pluggable services, grouped in a data scientist cloud kit. The toolkit aims at monitoring the underlying infrastructure, to exploit smart data compression, de-duplication and geo-replication, to evaluate data management costs. The system was validated on the Microsoft Azure cloud across its 6 EU and US data centers. The experiments were conducted on hundreds of nodes using synthetic benchmarks and real-life bio-informatics applications (A-Brain, BLAST).

Comparing to our view this work adopts a higher point of view and manages the data issues at a broker level as well at an intra or inter-site level for the data transfers by protocol switching. In our current work we do not consider real (cloud) providers but only geographically distributed clusters inside the same VLAN. We also do not investigate the issues with the choice of protocols we can use to transfer data.

The paper [96] introduces a new scheduling criterion, Quality-of-Data (QoD), which describes the requirements about the data that are worthy of the triggering of tasks in workflows. Based on the QoD notion, authors propose a novel service-oriented scheduler planner, for continuous data processing workflows, that is capable of enforcing QoD constraints and guide the scheduling to attain resource efficiency. QoD describes the minimum impact that new input data needs to have in order to trigger re-execution of processing steps in a workflow. This impact is measured in terms of data size, magnitude of values and update frequency. QoD can also be seen as a metric of triggering relaxation or optimist reuse of previous results. The core of the paper is a new scheduling algorithm for the Cloud that is guided by QoD, budget, and time constraints. The Markov Decision Process (MDP) technique is used to transform the problem. Authors explain that branch scheduling on the MDP representation is performed by starting from the most 'complex' branch to the 'least' complex one. In fact they exhibit an optimization problem they solve using a dynamic programming algorithm.

Part II

Contributions

Chapter 4

ADAPT

Contents

4.1 Description of ADAPT

ADAPT [97, 98] is an object oriented platform for running numerical simulations with a dynamic mesh adaptation strategy and coupling between finite elements and finite volumes methods.

ADAPT, as a CFD (Computational Fluid Dynamics) software package, has been developed for realizing numerical simulation on an unstructured and adaptive mesh for large scale CFD applications. The ADAPT project is concerned with skills from numerical analysis, computer science and physics, and treat many physical phenomena such as combustion, plasma discharge, wave propagation, etc. The object oriented ADAPT platform is able to do coupling between finite volumes and finite elements methods.

Yet another advantage of ADAPT is to focus in streamers which can be used in many applications. For example:

- Streamers are used for treatment of contaminated media like exhaust gases, polluted water or bio gas because the streamers emit the reactive radicals.
- The streamers are also used in the processing of electronics, especially for etching and deposition of thin films. They can be applied in surface modification e.g. hardening, corrosion resistance among others.
- Another utilization of the weakly ionized gas is to control a boundary layer in airflow at high speeds because achievements in aerospace allow to create hypersonic air vehicles at a Mach number greater than 6.

The existing ADAPT implementation is a sequential C++ code for each phenomenon, and the code requires a huge CPU time for executing a 3D simulation. For example, the 3D streamer code may run up to 30 days before returning results, for this reason we decided to parallelize the code. This paper is an important step into this direction.

4.1.1 Governing equations

The modeling of streamer propagation mechanism in atmospheric pressure discharges received a large attention and was the subject of a large research effort during the last two decades. Until very recently, streamer propagation was essentially described through fluid models that couple non-stationary transport equations for charged species (electron and ions) with a Poissons (elliptic) equation for the electrical potential. Usually the transport equations of charged species consist of time-dependent convection-diffusion-reaction equations.

In principle, when using advanced physical model, the source terms in these equations include a non-local component induced by the photoionization effect that was often invoked as the main propagation driver at least for streamer in field-free space. In this case, a radiation transport model has to be used and a non local radiation transfer equation has to be coupled to the transport and Poisson's equations.

These results have a very high computational cost, recently an advanced physical model based on Eddington's approximation allowed a very efficient description of the photoionization phenomenon. It remains that very often, a background seed electron field was used to simulate the electrons generated by photoionization phenomenon and responsible for the avalanches that takes place at the streamer tip and that are responsible for streamer propagation. The photoionization effect is deeply studied, e.g. in [99] where several approximations are taken into account. There are also more complex boundary conditions for the Poisson's equation in [99].

The simplest minimal model of discharge motion [100], is taken into account in our work. It consists of a convection-diffusion-reaction for the electron density, an ordinary differential equation for the positive ion density coupled by the Poisson's equation for the electric potential:

$$\begin{aligned} \frac{\partial n_e}{\partial t} + div(n_e \overrightarrow{v_e} - De \overrightarrow{\nabla} n_e) = S_e, \\ \frac{\partial n_i}{\partial t} = S_e, \end{aligned} \tag{4.1}$$

where t is the time, ne denotes the electron density, ni the positive ion density, v_e the electron drift velocity, De the diffusion coefficient $S_e = S_i^+$ are source terms. The system is coupled with the Poisson's equation for the electric potential V.

$$\Delta V = -\frac{e}{\epsilon}(n_i - n_e)\,, \tag{4.2}$$

where $Q = -\frac{e}{\epsilon}(n_i - n_e)$ (2D tests were computed with $Q = -\frac{e}{\epsilon}\frac{(n_i - n_e)}{10^4}$, ϵ is the dielectric constant, e the electron charge. The intensity of the electric field is computed as a negative gradient of the electric potential.

$$\vec{E} = -\overrightarrow{\nabla} V\,, \tag{4.3}$$

The electron drift velocity $\overrightarrow{v_e}$ is a function of the electric field $\overrightarrow{E}$ and depends on the ratio $\frac{||\overrightarrow{E}||}{N}$ where N is the neutral gas density (N=2, $5.10^{19}\ cm^{-3}$)

$$\begin{aligned} For \quad & \frac{||\overrightarrow{E}||}{N} > 2.10^{15}, && \overrightarrow{v_e} = -\left[7.4.10^{21}.\frac{||\overrightarrow{E}||}{N} + 7.1.10^6\right].\frac{\overrightarrow{E}}{||\overrightarrow{E}||}, \\ For \quad & 10^{-16} < \frac{||\overrightarrow{E}||}{N} \leq 2.10^{15}, && \overrightarrow{v_e} = -\left[7.4.10^{21}.\frac{||\overrightarrow{E}||}{N} + 7.1.10^6\right].\frac{\overrightarrow{E}}{||\overrightarrow{E}||}, \\ For \quad & 2,7.10^{-17} < \frac{||\overrightarrow{E}||}{N} \leq 10^{-16}, && \overrightarrow{v_e} = -\left[7.4.10^{21}.\frac{||\overrightarrow{E}||}{N} + 7.1.10^6]\right].\frac{\overrightarrow{E}}{||\overrightarrow{E}||}, \\ For \quad & \frac{||\overrightarrow{E}||}{N} \leq 2,6.10^{-17}, && \overrightarrow{v_e} = -\left[7.4.10^{21}.\frac{||\overrightarrow{E}||}{N} + 7.1.10^6\right].\frac{\overrightarrow{E}}{||\overrightarrow{E}||}, \end{aligned} \tag{4.4}$$

The diffusion coefficient D_e is a function of the electron drift velocity and the electric field:

$$D_e = -\left[0.3341.10^9.\left(\frac{||\overrightarrow{E}||}{N}\right)^{0.54069}\right].\frac{||\overrightarrow{v_e}||}{||\overrightarrow{E}||}. \tag{4.5}$$

The source terms depend on the electron drift velocity and the electron density:

$$S_e = \frac{\alpha}{N}.||\overrightarrow{v_e}||.n_e.N. \tag{4.6}$$

$$S_i^+ = S_e. \tag{4.7}$$

where the ratio $\frac{\alpha}{N}$ $[cm^2]$ is computed by following formula

$$\begin{aligned} if \quad & \frac{||\overrightarrow{E}||}{N} > 1.5.10^{-15}, \qquad \frac{\alpha}{N} = 2.6.10^{-16} \exp(\frac{-7.248.10^{-15}}{\frac{||\overrightarrow{E}||}{N}}) \\ else, \quad & \frac{\alpha}{N} = 6.619.10^{-17} \exp(\frac{-5.593.10^{-15}}{\frac{||\overrightarrow{E}||}{N}}). \end{aligned} \tag{4.8}$$

Initial conditions

The electric field is established between two planar electrodes. High voltage $25000\,V$ is applied at the anode and the cathode is grounded. The disturbance in the electric field between these two planar electrodes is given by an initial Gaussian pulse. The photo-ionization effect, which is necessary to run streamer, is substitute by a background density. For the two dimension case we have: dimensions

$$\begin{aligned} n_e(x,y,0) = 10^{16}.e^{-\frac{(x-0.2)^2+(y-0.25)^2}{\sigma^2}} + 10^9[cm^{-3}], \sigma = 0.01, \\ n_i(x,y,z,0) = n_e(x,y,z,0) \end{aligned} \tag{4.9}$$

and for the three dimensions case:

$$\begin{aligned} n_e(x,y,z,0) = 10^{16}.e^{-\frac{(x-0.2)^2+(y-0.25)^2+(z-0.25)^2}{\sigma^2}} + 10^9[cm^{-3}], \sigma = 0.01, \\ n_i(x,y,z,0) = n_e(x,y,z,0) \end{aligned} \tag{4.10}$$

Boundary conditions

The Neumann's homogeneous condition is considered at all boundaries for the electron density:

$$\frac{\partial n_e}{\partial t} = 0, for\, anode\, and\, cathode \tag{4.11}$$

Following Dirichlet's conditions are prescribed for the electric potential at the anode and the cathode:

$$\begin{aligned} V = 25000[V],\ for\, anode \\ V = 0,\ for\, cathode \end{aligned} \tag{4.12}$$

The Neumann's homogeneous condition is prescribed at the rest of boundaries:

$$\frac{\partial V}{\partial \overrightarrow{n}} = 0, \tag{4.13}$$

4.1.2 Numerical Approximation

The streamer propagation phenomenon is described through the time-space evolution of physical variables such as electron density, ion density, electric field and space charge. All these characteristics experience a several orders of magnitude variation in a very narrow region located at the micro-discharge tip (the so called streamer region). Further, the transport equations and the electric field equation are strongly coupled through source-terms, transport fluxes and space charge. The propagation dynamics intimately depends on the space-time distribution of the microdischarge characteristics in this tinny region with a characteristics length of few micrometers. Researchers therefore concentrate on the development of high order schemes, e.g., FCT, QUICK, (W)ENO, MUSCLE, improved Sharfeter-Gummel, etc., for 1D geometries and 2D-axisymmetrical geometries.

The use of highly non-symmetrical electrode system, along with the non-planar nature of the streamer propagation motivated some groups to adapt some of these numerical schemes to the case of non-structured grids. Nevertheless the streamer simulation remains costly and time consuming. That's the reason why a simple upwind scheme extended by a Van Leer's type MUSCL algorithm together with Barth-Jespersen limiter is used in this work. The dynamic mesh adaptation improve the accuracy of the simple scheme. There it is possible to use another limiters such as Van Albada limiter. The Barth-Jespersen limiter is chosen because it's a quite simple limiter and doesn't depend on parameters which should be tuned.

As it was mentioned before, streamer propagation is represented by unsteady convection diffusion reaction equations for electric particles coupled with Poisson's equation for electrical potential. The unsteady equations are solved by an explicit scheme on unstructured grid. The discretized Poisson's equation leads to a system of algebraic linear equations which is solved with a direct solver at each time step.

4.1.2.1 Numerical method

The finite volumes method is one of the most used approximation techniques for partial differential equations that come from the physical conservation law. Unlike the method of finite elements or finite differences, it results from a choice in the physical sense based on writing assessments. The idea is to discretize on each cell, called "control volume", the integral form of the problem instead of its differential form. The convection-diffusion-reaction equations for the filler particle density (electrons and positive ions) and the Poisson equation for the electric potential is treated by method of finite volumes on unstructured meshes.

The system of convection-diffusion-reaction equations for the charge particle densities (electrons, positive ions) and the Poisson's equation for the electric potential are treated by the finite volumes method on unstructured meshes. The algorithm is as follows:

- new values of the electron density n_e and the ion density n_i are computed by an explicit scheme;
- the electrical potential is established from the new values by numerical solution of the Poisson's equation;
- the electric field $\overrightarrow{E}$ is computed from the electric potential;
- the electron drift velocity v_e and the diffusive coefficient De is computed as functions of the electric field $\overrightarrow{E}$;
- a new time layer is computed => new values of n_e; n_i;
- a new mesh is made by a dynamical mesh adaptation after several iterations;
- values of unknowns are interpolated form the old mesh to the new one.

4.1.2.1.1 Unstructured mesh The numerical solution of any partial differential equation requires the discretization of the computational domain so that the physical flow is predicted at a desired accuracy. There are two ways to proceed:

- The first is to use structured grids that are easy to generate, the simplicity of the mesh allows the execution accelerated speeds. But the difficulty remains to generate meshes of this type in more complex geometry.
- Another approach to the discretization of a domain is to use unstructured meshes. In this context, the triangular cells (tetrahedral respectively) are the most simple forms that can be used to cover an area of two-dimensional calculation (tridimensional respectively). In our work we focus on unstructured grids which have the advantage of generating elements within complex geometries, and can add or remove nodes or elements when necessary.

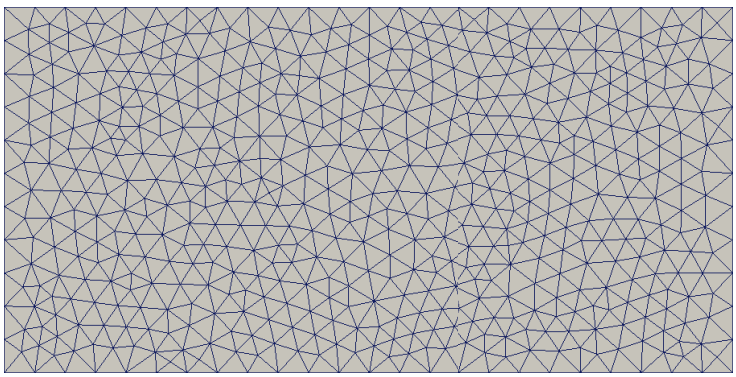


Figure 4.1: Unstructured 2D mesh

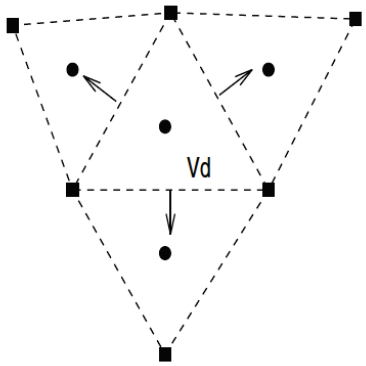


Figure 4.2: Cell-centered

4.1.2.1.2 Cell-centered Here we use the cell-centered (block-centered) methods, in which the primal grid cells are used to define the control volumes (the control volumes in our case coincide with the triangles of the mesh).

4.1.2.1.3 Discretization of Electron Density Equation The finite volumes method is used for discretization of the electron density equation. The domain Ω discretized using a set of subdomains μ_i, $\Omega \equiv \cup_i$, which do not overlap each other and have common boundaries. Integration over volume μ_i gives (the treatment is the same in 3D and leads to the same general form).

$$\iint_{\mu_i} \frac{\partial n_e}{\partial dt} dV + \iint_{\mu_i} div(n_e \overrightarrow{v_e}) dV - \iint_{\mu_i} div(D_e \overrightarrow{\nabla} n_e) dV = \iint_{\mu_i} S_e dV. \tag{4.14}$$

By using Green's formula and dividing by the volume, the equation (4.14) leads to

$$\frac{\partial n_e}{\partial t} + \frac{1}{\mu_i} \oint_{\partial \mu_i} n_e \overrightarrow{v}_e \overrightarrow{n} \mathrm{d}s - \oint_{\partial \mu_i} D_e \overrightarrow{\nabla} n_e \overrightarrow{n} \mathrm{d}s = S_e, \tag{4.15}$$

with the unit normal vector $\overrightarrow{n}$. Now we approximate the curvilinear integral by a summation. So one obtains for a volume μ_i:

$$\frac{\partial n_e}{\partial t} = -\frac{1}{\mu_i} \sum_{j=1}^{m} (n^n_{e_{ij}} \overrightarrow{v}_{e_{ij}} \overrightarrow{n}_{ij} |\sigma_{ij}| - D_{eij} \overrightarrow{\nabla} n^n_{e_{ij}} \overrightarrow{n}_{ij} |\sigma_{ij}|) + S^N_{e_i} \tag{4.16}$$

where m is number of faces of volume μ_i, $\overrightarrow{n}_{ij}$ is the unit normal vector of the face σ_{ij}(face between volumes μ_i and μ_j) and $|\sigma_{ij}|$ is its length. Other variables denoted by subscript ij represent variables on the face σ_{ij}. At the end, the time derivative is approximated by Runge-Kutta method

$$\begin{aligned} n^{(0)}_{e_i} &= n^n_{e_i}, \\ n^{(k)}_{e_i} &= n^n_{e_i} + \alpha_{(k)} Res(n^{(k-1)}_{e_i}), \\ n^{(n+1)}_{e_i} &= n^{(3)}_{e_i}. \end{aligned} \tag{4.17}$$

$k = 1, 2, 3$, and the coefficient $\alpha_{(1)} = \alpha_{(2)}$ = 0.5, $\alpha_{(3)}$= 1. The residual in equation (4.17) is given by

$$Res(n^{(k-1)}_{e_i}) = -\frac{\Delta t}{\mu_i} \sum_{j=1}^{m} (n^{(k-1)}_{e_{ij}} \overrightarrow{v}_{e_{ij}} \overrightarrow{n}_{ij} |\sigma_{ij}| - D_{eij} \overrightarrow{\nabla} n^{(k-1)}_{e_{ij}} \overrightarrow{n}_{ij} |\sigma_{ij}|) + \Delta t . S^{(k-1)}_{e_i} \tag{4.18}$$

Where Δt is a time step.

4.1.2.1.4 Discretization of Convective Terms The convective flux in electron density equation is computed by a simple upwind scheme:

$$n_{e_{ij}} = \begin{cases} n_{e_i} \text{si } (v_{e_{ij}}.\overrightarrow{n}_{ij}) \geq 0 \\ n_{e_j} \text{sinon.} \end{cases} \tag{4.19}$$

with assumption that the normal vector $\overrightarrow{n}_{ij}$ is oriented from the cell T_i to the cell T_j. Computation of the interface drift velocity $\overrightarrow{v}_{e_{ij}}$ will be explained later.

Higher order approximation in space

The upwind scheme has a first order of accuracy. It leads to a high numerical dissipation which is necessary to compensate by higher order algorithm. The scheme is therefore extended by a Van Leer's type MUSCL algorithm together with Barth-Jespersen limiter.

One introduces $\tilde{n}_{e_i}$, and $\tilde{n}_{e_j}$ in the equation (4.19) instead of n_{e_i} and n_{e_j}

$$\begin{aligned} \tilde{n}_{e_i} &= n_{e_i} + \psi_i.(\overrightarrow{\nabla} n_{e_i}.\overrightarrow{r_i}), \\ \tilde{n}_{e_j} &= n_{e_j} + \psi_j.(\overrightarrow{\nabla} n_{e_j}.\overrightarrow{r_j}) \end{aligned} \tag{4.20}$$

$\overrightarrow{\nabla} n_{e_i}$, $\overrightarrow{\nabla} n_{e_j}$ are gradients of electron density in cells T_i and T_j. We can compute the gradients assuming that the electron density is piecewise linear function and its value n_{e_i} is in the center of gravity of the cell T_i. This linear function is computed by the least square method which involves all neighbors of the cell T_i (neighbors with at least one common vertex). $\overrightarrow{r_i}$ is a vector coming from the center of gravity of the cell T_i to the center of gravity of a face σ_{ij}. In the same way, $\overrightarrow{r_j}$ is a vector coming from the center of gravity of the cell T_j to the center of gravity of a face σ_{ij}. ψ_i and ψ_j are Barth-Jespersen limiter function. The gradient $\overrightarrow{\nabla} n_{e_i} = ((n_e)_x, (n_e)_y))$ equation (4.20) is computed by following formulas in 2D

$$(n_e)_x = \frac{I_{yy}.J_x.I_{xy}.J_y}{D}, \quad (n_e)_y = \frac{I_{xx}.J_y.I_{xy}.J_x}{D} \tag{4.21}$$

where I_{xx}, I_{yy}, I_{xy}, J_x, J_y are given by equation (A.1). We obtain a little bit more complicated relations in 3D for components of the gradient

$$\begin{aligned} (n_e)_x &= \frac{(I_{yy}I_{zz} - I_{yz}^2)J_x + (I_{xz}I_{yz} - I_{xy}I_{zz})J_y + (I_{xy}I_{yz} - I_{xz}I_{yy})J_z}{D} \\ (n_e)_y &= \frac{(I_{xz}I_{yz} - I_{xz}I_{zz})J_x + (I_{xx}I_{zz} - I_{xz}^2)J_y + (I_{xy}I_{xz} - I_{yz}I_{xx})J_z}{D} \\ (n_e)_z &= \frac{(I_{xy}I_{zz} - I_{xz}I_{yy})J_x + (I_{xy}I_{xz} - I_{yz}I_{xx})J_y + (I_{xx}I_{yy} - I_{xy}^2)J_z}{D} \end{aligned} \tag{4.22}$$

where I_{xx}, I_{yy}, I_{zz}, I_{xy}, I_{xz}, I_{yz}, J_x, J_y, J_z are dened by equations (A.2).

The Barth-Jespersen limiter is given by following formula

$$\psi_i = \begin{cases} \min(1, \dfrac{n_{e_{max}} - n_{e_i}}{\Delta_2}) \text{si } \Delta_2 > 0 \\ 1 \text{si } \Delta_2 = 0 \\ \min(1, \dfrac{n_{e_{min}} - n_{e_i}}{\Delta_2}) \text{sinon.} \end{cases} \tag{4.23}$$

where $n_{e_{max}}$ is maximum of electron density from values in the cell T_i and all neighboring cells with common face, $n_{e_{min}}$ is minimum of electron density from values in the cell T_i and all neighboring cells with common face. Δ_2 is given by relation $\Delta_2 = \overrightarrow{\nabla} n_{e_i}.\ \overrightarrow{r}$, which comes from the cell's center of gravity to the center of gravity of one of faces of the cell Ti. Final ψ_i is the minimum over the faces.

4.1.2.1.5 Discretization of Dissipative Terms The curvilinear integral of the diffusive term in the equation (4.15) is approximated by numerical integration around a cell boundary, so one can write:

$$\oint_{\partial\mu_i} D_e \overrightarrow{\nabla} n_e \overrightarrow{n} \mathrm{ds} = \sum_{j=0}^{m} D_{e_{ij}} \overrightarrow{\nabla} n_{e_{ij}}^n \overrightarrow{n}_{ij} |\sigma_{ij}| \tag{4.24}$$

where $\overrightarrow{\nabla} n_{e_{ij}}^n$ represents gradient of the electron density on the face σ_{ij}, $\overrightarrow{n}_{ij}$ is a unit normal vector of the face σ_{ij} and $|\sigma_{ij}|$ is its measure. The constant m indicate the total number of a cell's faces. Computation of the diffusion coefficient $D_{e_{ij}}$ is described in the equation (4.44).

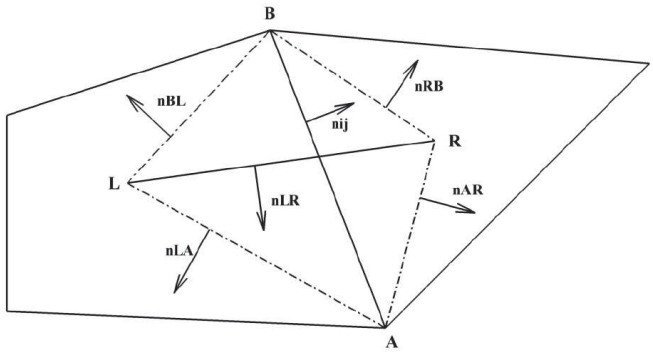


Figure 4.3: Diamond cell in 2D

Gradient Computation in 2D Approximation of the electron density gradient $\overrightarrow{\nabla} n_{e_{ij}}$ in 2D is done on the diamond cell (see Fig. 4.3). The diamond cell is constructed by connection of centers of gravity (L; R) of cells T_i, T_j which share the face σ_{ij} and its endpoints A, B. One obtains the co-volume $D_{\sigma_{ij}}$ by this construction.

One can assume that the electron density gradient is constant on the co-volume $D_{\sigma_{ij}}$. According to Green-Gauss theorem the approximation leads to

$$\begin{aligned}\overrightarrow{\nabla} n_{e_{ij}} = \frac{1}{\mu(D_{\sigma_{ij}})}\Big[&\frac{1}{2}(n_e(A)+n_e(R))\overrightarrow{n}_{AR}|\sigma_{AR}| + \frac{1}{2}(n_e(R)+n_e(B))\overrightarrow{n}_{RB}|\sigma_{RB}| \\ &+\frac{1}{2}(n_e(B)+n_e(L))\overrightarrow{n}_{BL}|\sigma_{BL}| + \frac{1}{2}(n_e(L)+n_e(A))\overrightarrow{n}_{LA}|\sigma_{LA}|\Big]\end{aligned} \tag{4.25}$$

$n_e(A)$, $n_e(B)$, $n_e(L)$, $n_e(R)$ represents value of the electron density in the point A, B, L, R. $\overrightarrow{n}_{AR}$ is a unit normal vector of the co-volume face σ_{AR} and $|\sigma_{AR}|$ is its length. The others co-volume faces and their normal vectors are labeled analogically (see Fig. 4.3). When we factor one half out of the square bracket and modify the expression inside the square brackets, we obtain

$$\begin{aligned}\overrightarrow{\nabla} n_{e_{ij}} = \frac{1}{2\mu(D_{\sigma_{ij}})}\Big[&(\overrightarrow{n}_{AR}|\sigma_{AR}| + \overrightarrow{n}_{LA}|\sigma_{LA}|)n_e(A) + (\overrightarrow{n}_{RB}|\sigma_{RB}| + \overrightarrow{n}_{BL}|\sigma_{BL}|)n_e(B) + \\ &(\overrightarrow{n}_{AR}|\sigma_{AR}| + \overrightarrow{n}_{RB}|\sigma_{RB}|)n_e(R) + (\overrightarrow{n}_{BL}|\sigma_{BL}| + \overrightarrow{n}_{LA}|\sigma_{LA}|)n_e(L)\Big]\end{aligned} \tag{4.26}$$

The following equalities are obvious:

$$\begin{aligned}\overrightarrow{n}_{AR}|\sigma_{AR}| + \overrightarrow{n}_{LA}|\sigma_{LA}| &= \overrightarrow{n}_{LR}|\sigma_{LR}| \\ \overrightarrow{n}_{RB}|\sigma_{RB}| + \overrightarrow{n}_{BL}|\sigma_{BL}| &= -\overrightarrow{n}_{LR}|\sigma_{LR}| \\ \overrightarrow{n}_{AR}|\sigma_{AR}| + \overrightarrow{n}_{RB}|\sigma_{RB}| &= \overrightarrow{n}_{ij}|\sigma_{ij}| \\ \overrightarrow{n}_{BL}|\sigma_{BL}| + \overrightarrow{n}_{LA}|\sigma_{LA}| &= -\overrightarrow{n}_{ij}|\sigma_{ij}|\end{aligned} \tag{4.27}$$

If we substitute relations (4.27) in the equation (4.26), it gives:

$$\overrightarrow{\nabla} n_{e_{ij}} = \frac{1}{2\mu(D_{\sigma_{ij}})}\Big[(n_e(A)-n_e(B))\overrightarrow{n}_{LR}|\sigma_{LR}| + (n_e(R)-n_e(L))\overrightarrow{n}_{ij}|\sigma_{ij}|\Big] \tag{4.28}$$

The electron density values $n_e(A)$, $n_e(B)$ in the face σ_{ij} points are computed by the least square method:

$$n_e(A) = \sum_{p=1}^{N(A)} \alpha_p(A) n_{ep}, \quad n_e(B) = \sum_{p=1}^{N(B)} \alpha_p(B) n_{ep}. \tag{4.29}$$

where n_{ep} is value of the electron density in the cell T_p, $N(A)$ ($N(B)$) is number of cells including vertex A (B), $\alpha_p(A)$ and $\alpha_p(B)$ are weights coming from the least square method. One can write (for vertex A):

$$\alpha_p(A) = \frac{1 + \lambda_x(x_p - x_A) + \lambda_y(y_p - y_A)}{N(A) + \lambda_x R_x + \lambda_y R_y} \tag{4.30}$$

where the weights parameters are given by formulas (A.3).

Gradient approximation in 3D

The diamond cell in 3D is constructed by the centers of gravity L; R of cells T_i, T_j which contain the common face σ_{ij}. The rest of vertices of the diamond cell are vertices of the face σ_{ij}(see Fig. 4.4). The electron density gradient $\overrightarrow{\nabla} n_{e_{ij}}$ is approximated by following formula which coming from the equations (A.4) - (A.6).

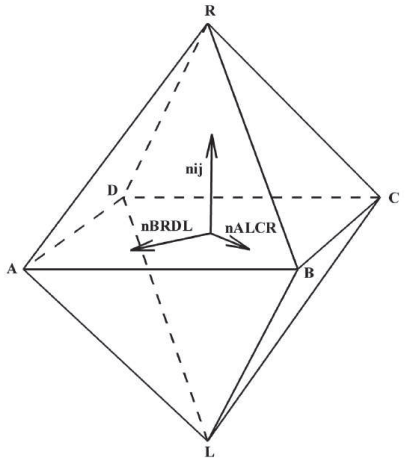


Figure 4.4: Diamond cell in 3D

$$\overrightarrow{\nabla} n_{e_{ij}} = \frac{1}{3\mu(D_{\sigma_{ij}})}\Big[(n_e(A) - n_e(C))\overrightarrow{n}_{BRDL}|\sigma_{BRDL}| + (n_e(B) - n_e(D))\overrightarrow{n}_{ALCR}|\sigma_{ALCR}| + (n_e(R) - n_e(L))\overrightarrow{n}_{ij}|\sigma_{ij}|\Big] \tag{4.31}$$

The electron density values $n_e(A)$, $n_e(B)$, $n_e(C)$ and $n_e(D)$ are computed by the least square method:

$$n_e(A) = \sum_{p=1}^{N(A)} \alpha_p(A) n_{ep}, \quad n_e(B) = \sum_{p=1}^{N(B)} \alpha_p(B) n_{ep}. \tag{4.32}$$

$$n_e(C) = \sum_{p=1}^{N(C)} \alpha_p(C) n_{ep}, \quad n_e(D) = \sum_{p=1}^{N(D)} \alpha_p(D) n_{ep}. \tag{4.33}$$

The weights $\alpha_p(A)$ is given by

$$\alpha_p(A) = \frac{1 + \lambda_x(x_p - x_A) + \lambda_y(y_p - y_A) + \lambda_z(z_p - z_A)}{D} \tag{4.34}$$

Where λ_x, λ_y, λ_z D are expressed in relations (A.7).

4.1.2.1.6 Discretization of the Positive ion Density The equation for ion density is in the minimal model only an ordinary differential equation and its numerical approximation is represented by three steps Runge-Kutta method (equivalent of equation (4.18))

$$
\begin{aligned}
n_{i_i}^{(0)} &= n_{i_i}^{n}, \\
n_{i_i}^{(k)} &= n_{i_i}^{n} + \alpha_{(k)} Res(n_{i_i}^{(k-1)}), \\
n_{i_i}^{(n+1)} &= n_{i_i}^{(3)}.
\end{aligned}
\tag{4.35}
$$

where the coefficients $\alpha(k)$ are the same as in the equation (4.18) and the residual is expressed by relation:

$$
Res(n_{i_i}^{(k-1)}) = \Delta t . S_{e_i}^{(k-1)}. \tag{4.36}
$$

4.1.2.1.7 Discretization of Poisson's Equation Discretization of Poisson's equation leads to a system of linear equations with a sparse matrix. Such system of equation can be solved by direct methods or by iterative methods. Iterative methods gives possibilities as Jacobi iterative method, Gauss-Seidel iterative method or super relaxation method which is improved Gauss-Seidel iterative method. There are also gradients method such as Biconjugate gradients method or GMRES method (generalized minimum residual method).

The GMRES method together with proper preconditioning is very often used in numerical simulations (solving of implicit scheme, etc.). Gauss elimination or LU factorization owned direct methods. The LU factorization is more suitable when right hand side of the system of equation is changed. Because of dynamic mesh adaptation, the matrix of the system of equations is changed after a given number of iterations and the right hand side is changed each iteration. This leads to "quasi-stationary" matrix of equations and therefore LU factorization is the proper method in our case. We use MKL library from Intel to solve the system of liner equations by LU factorization. The Poisson's equation is discretized by a central type approximation which leads to a system of linear equations:

$$
A . \overrightarrow{V}^{n+1} = \overrightarrow{b}^{n+1} \tag{4.37}
$$

A is a matrix of coefficients, $\overrightarrow{V}$ is a vector of unknowns (its dimension is equaled to the total number of cells) and $\overrightarrow{b}$ is a vector of right hand side. The updated values n_e^{n+1} and n_i^{n+1} are used for the evaluation of the source term in equation (4.2). A row i in the matrix A corresponds to the cell T_i. We use a similar finite volumes method approximation as for diffusive terms in the equation for electron density (4.24).

$$
\iint_{\mu_i} Q = \iint_{\mu_i} \Delta V \mathrm{dV} = \oint_{\partial \mu_i} \overrightarrow{\nabla} V \mathrm{ds} = \sum_{j=1}^{m} \overrightarrow{\nabla} V_{ij} \overrightarrow{n}_{ij} |\sigma_{ij}| \tag{4.38}
$$

An approximation of the gradient $\overrightarrow{\nabla} V_{ij}$ is also made by the diamond cell (see Fig. Fig. 4.3, Fig. Fig. 4.4). According to the equation (4.28), one can write for 2D case

$$
\overrightarrow{\nabla} V_{ij} = \frac{1}{2\mu(D_{\sigma_{ij}})} \left[(V(A) - V(B)) \overrightarrow{n}_{LR} |\sigma_{LR}| + (V(R) - V(L)) \overrightarrow{n}_{ij} |\sigma_{ij}| \right] \tag{4.39}
$$

When we substitute the equation (4.39) to a relation

$$
\Delta V = \frac{1}{\mu_i} \sum_{j=1}^{m} \overrightarrow{\nabla} V_{ij} \overrightarrow{n}_{ij} |\sigma_{ij}| \tag{4.40}
$$

we obtain the final form of approximation of the Poisson's equation in 2D

$$
\begin{aligned}
\Delta V = \sum_{j=1}^{m} \Big[& \frac{1}{2\mu_i D_{\sigma_{ij}}} . \overrightarrow{n}_{LR} |\sigma_{LR}| . \overrightarrow{n}_{ij} |\sigma_{ij}| V_A - \\
& \frac{1}{2\mu_i D_{\sigma_{ij}}} . \overrightarrow{n}_{LR} |\sigma_{LR}| . \overrightarrow{n}_{ij} |\sigma_{ij}| V_B + \\
& \frac{1}{2\mu_i D_{\sigma_{ij}}} . \overrightarrow{n}_{ij} |\sigma_{ij}| . \overrightarrow{n}_{ij} |\sigma_{ij}| V_R - \\
& \frac{1}{2\mu_i D_{\sigma_{ij}}} . \overrightarrow{n}_{ij} |\sigma_{ij}| . \overrightarrow{n}_{ij} |\sigma_{ij}| V_L \Big]
\end{aligned}
\tag{4.41}
$$

where electrical potential values V_A, V_B are computed by the least square method (see equation (4.29)) with weights computed according to equation (4.30).

For 3D case, the electrical potential gradient is approximated according to equation (4.31), so one can write

$$\begin{aligned}\overrightarrow{\nabla} V_{ij} = \frac{1}{3\mu(D_{\sigma_{ij}})}\Big[&(V(A)-V(C))\overrightarrow{n}_{BRDL}|\sigma_{BRDL}| \\ &+(V(B)-V(D))\overrightarrow{n}_{ALCR}|\sigma_{ALCR}| + (V(R)-V(L))\overrightarrow{n}_{ij}|\sigma_{ij}|\Big]\end{aligned} \tag{4.42}$$

If we substitute the equation (4.42) to the equation (4.40), the numerical approximation of the electrical potential in 3D leads to

$$\begin{aligned}\Delta V = \Big[&\frac{1}{3\mu_i D_{\sigma_{ij}}}\overrightarrow{n}_{BRDL}|\sigma_{BRDL}|.\overrightarrow{n}_{ij}|\sigma_{ij}.V_A - \\ &\frac{1}{3\mu_i D_{\sigma_{ij}}}\overrightarrow{n}_{BRDL}|\sigma_{BRDL}|.\overrightarrow{n}_{ij}|\sigma_{ij}.V_C + \\ &\frac{1}{3\mu_i D_{\sigma_{ij}}}\overrightarrow{n}_{ALCR}|\sigma_{ALCR}|.\overrightarrow{n}_{ij}|\sigma_{ij}.V_B - \\ &\frac{1}{3\mu_i D_{\sigma_{ij}}}\overrightarrow{n}_{ALCR}|\sigma_{ALCR}|.\overrightarrow{n}_{ij}|\sigma_{ij}.V_D + \\ &\frac{1}{3\mu_i D_{\sigma_{ij}}}\overrightarrow{n}_{ij}|\sigma_{ij}.\overrightarrow{n}_{ij}|\sigma_{ij}.V_R - \\ &\frac{1}{3\mu_i D_{\sigma_{ij}}}\overrightarrow{n}_{ij}|\sigma_{ij}.\overrightarrow{n}_{ij}|\sigma_{ij}.V_L\end{aligned} \tag{4.43}$$

The electrical potential values V_A, V_B, V_C and V_D come from the least square method ((4.32),(4.33)). The least square weights are computed by equation (4.34).

4.1.2.1.8 Discretization of Intensity of Electric Field, Electron Drift Velocity and Diffusion Coefficient The intensity of electric field is given as a negative gradient of the electrical potential $\overrightarrow{E}^n = -grad(V^n)$. The intensity of electric field is computed on cell faces by diamond scheme (see equation (3.17) and (3.20)). The electron drift velocity is a function of the intensity of electric field $\overrightarrow{v}_{e_{ij}} = f(\overrightarrow{E}^n_{ij})$ depends on the ratio $\frac{||\overrightarrow{E}^n_{ij}||}{N}$ $[Vcm^2]$, where N is the neutral gas density (N=2.5.10^{19} cm^3). The precise formula is expressed in (B.1). Finally we can compute the diffusion coefficient D_e as a function of the intensity of electric field and the electron drift velocity

$$D_{e_{ij}} = -\left[0.3341.10^9.\left(\frac{||\overrightarrow{E}^n_{ij}||}{N}\right)^{0.54069}\right].\frac{||\overrightarrow{v}^n_{e_{ij}}||}{||\overrightarrow{E}^n_{ij}||}. \tag{4.44}$$

Values at the cell center of gravity are estimated as average value of the values on the cell faces.

4.1.2.1.9 Discretization of Source Terms The source terms depends on the electron drift velocity and the electron density, so one can write for the cell T_i.

$$S^n_{e_i} = \frac{\alpha}{N}.||\overrightarrow{v}^n_{e_i}||.n^n_{e_i}.N \tag{4.45}$$

where the ratio $\frac{\alpha}{N}$ is computed by the following formula

$$\begin{aligned}&if \quad \frac{||\overrightarrow{E}||}{N} > 1.5.10^{-15}, \qquad \frac{\alpha}{N} = 2.6.10^{-16}\exp\left(\frac{-7.248.10^{-15}}{\frac{||\overrightarrow{E}^n_{ij}||}{N}}\right) \\ &else \qquad\qquad\qquad\qquad \frac{\alpha}{N} = 6.619.10^{-17}\exp\left(\frac{-5.593.10^{-15}}{\frac{||\overrightarrow{E}^n_{ij}||}{N}}\right).\end{aligned} \tag{4.46}$$

4.2 Performance results for parallel 2D streamer code

4.2.1 Benchmark settings

All computations are done on the same reference grid with 529240 cells on unstructured mesh. The second order scheme is used in the tests.

4.2.1.1 Initial conditions

We consider the following initial conditions for the rectangular domain

$$n_e(x,y,0) = 10^{16}.e^{-\frac{(x-0.2)^2+(y-0.25)^2}{\sigma^2}} + 10^9[cm^{-3}], \sigma = 0.01, \quad n_i(x,y,0) = n_e(x,y,0) \tag{4.47}$$

The initial Gaussian pulse for the electron and ion densities creates disturbance in the electric field which is necessary for initialization of ionization process in the wave propagation. The background electron and ion densities $10^9 cm^3$ substitute the photoionization effect which is neglected in this simple streamer model. The computation is stopped at prescribed time when the head of electron avalanche is still inside the computational domain. In further computation, the discharge goes out through the right boundaries and the source terms increase to infinity but this effect doesn't represents physical behavior of the electric discharge.

4.2.1.2 Boundary conditions

The left boundaries of the domains are anode, the right boundaries are plane cathodes. There are prescribed homogeneous Neumann conditions for all unknowns for upper and lower boundaries. Following boundary conditions are prescribed for the anode and the cathode

$$\frac{\partial n_e}{\partial t} = 0, for\, anode\, and\, cathode \quad V = 25000[V],\ for\, anode \quad V = 0,\ for\, cathode \tag{4.48}$$

4.2.2 Shared memory with pure MPI programming model

We performed some numerical experiments on a shared memory machine. The following experiments were carried out using MAGI cluster from University Paris13, Villetaneuse, France. This machine contains about 50 Dell FC430 blades with:

- 10 Hyper-Threading dual processor cores (40 cores per blade);
- 64 GB RAM per blade;
- Local disk is made of SSD;
- InfiniBand for the low latency interconnect. infiniBand

We use the pure MPI parallel programming model mentioned in Section 2.2 (no OpenMP directives) and one MPI process per core. For the parallelization of CFD simulations, ADAPT employs the domain decomposition method. The 2D unstructured mesh was decomposed using the sequential version of METIS, because the splitting takes less than 0.001% of the running time of 2D streamer. Algorithm 1

4.2.2.1 General structure of the ADAPT solver

We first present the main structure of the solver:

- Initialization
  - Data initialization: read the input file;
  - Domain initialization: construct the subdomains;
  - Fields initialization: allocate arrays;
- Loop on time
  - Solve evolutive equation;
  - Solve Poisson problem;
  - update variables;
  - Store results;
- Finalize

In the data initialization step, we read inputs from an external file in which we specify the dimension, some physical quantities, and other characteristics of the problem. Then we initialize the computational domain. In the field initialization step, we allocate the necessary memory for both density of electron and ion. Then we consider the solving phase which is inside a loop on time. This step, which has been clearly explained in Chapter 1, consists of solving a convection-diffusion equation, a Poisson equation, and of updating the auxiliary variables by an explicit correction. We can choose to store the solution after each time iteration, or after a given number of iterations in order to save memory space. For example, we can decide to store solutions every 500 iterations. If the time step is 0.001 second, then the final simulation will be constructed by the solutions obtained every 0.5 second. Of course, this can be changed, depending on the required accuracy.

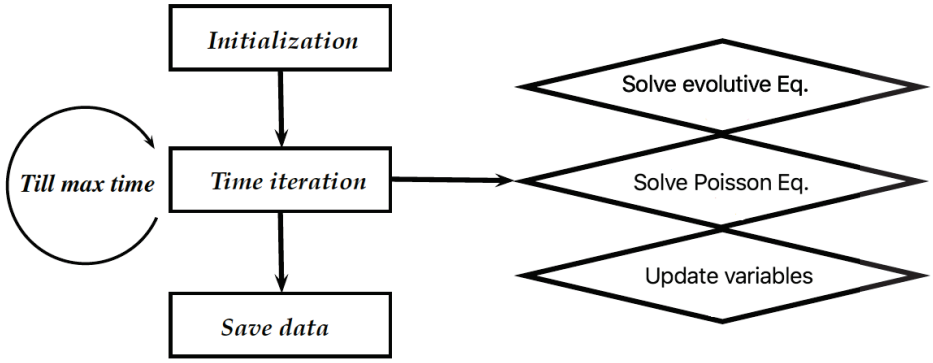


Figure 4.5: The main procedure of ADAPT solver

4.2.2.2 Algorithms of important parts

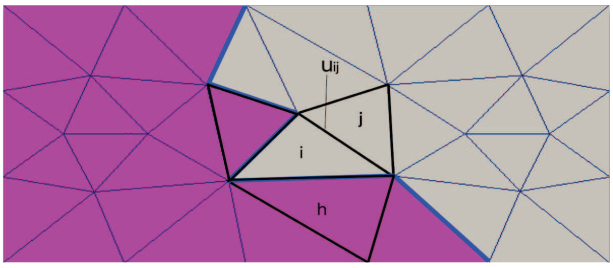
Page 71 provides with a piece of pseudo code that shows how the parallelization is done in the streamer code according to the coupling of evolution equation with Poisson equation. We can see that most parts of the code are parallel ones (line 19, line 9 to 11 and line 22 to 26) except reading and splitting mesh in the beginning and between lines 14 and 15. Indeed, at this step, we construct matrix A (equation 2) that will be used to solve the linear system in line 19: the matrix A is computed one time because it depends only on the mesh. We also put information about the overlapping between the communication and computation steps that are source for performance.

Algorithm 3: Algorithm of parallel ADAPT

```
W:double;
if rank==0 then /* for the master processor */
    Read mesh data;
    Split mesh with METIS;
    Distribute mesh to all processors;
end
W=0;
for each rank do /* for each processor */
    Initialize conditions and create constants;
    Send the informations of halos cells to neighbor subdomains;
    Apply boundary conditions;
end
if rank==0 then
    Construct matrix of linear system;
    Split the matrix and send part of each processor;
end
for each iteration do
    for all rank do /* for all processors */
        Solve linear system using MUMPS;
    end
    for each rank do
        Send the informations of halos cells to neighbor subdomains;
        Apply boundary conditions;
        Compute fluxes of convection, diffusion and source term;
        Update solution : W^{n+1} = W^n + Δt * (rez_conv + rez_dissip + rez_source);
        Save results in parallel way using Paraview;
    end
end
```

Computation of n_{eij}: In figure included in algorithm 4 we took the example when the mesh is splitted into two subdomains, and we compute $n_{e_{ij}}$ on the face σ_{ij}. For a given face σ_k, suppose T_i and T_j are respectively the cells at the left and right of σ_k. Let's note $\sigma_k = \sigma_{ij}$ and $n_{e_k} = n_{e_{ij}}$. The algorithm 4 shows how the $n_{e_{ij}}$ is computed on face σ_{ij}.

Algorithm 4: Compute $n_{e_{ij}}$ on face ij



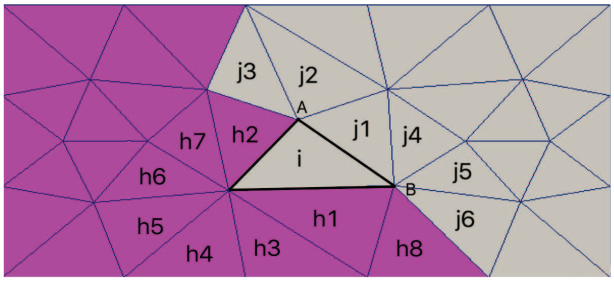

```
F:number of faces;
for k:=1 to F do
    if dot(V_k.n_k) >= 0 then
        n_{e_k} = n_{e_i};
    end
    else
        if σk is inner faces then
            n_{e_k} = n_{e_j};
        end
        else if σk is halo faces then
            n_{e_k} = n_{e_h};/* n_{e_h} : halo value sent by
            neighbor subdomain */
        end
    end
end
```

Computation of $\overrightarrow{\nabla} n_{e_{ij}}$: In figure included in algorithm 5 we take the same mesh, and we compute $\overrightarrow{\nabla} n_{e_{ij}}$ of the

face σ_{ij}. The algorithm 5 shows how the $\overrightarrow{\nabla} n_{e_{ij}}$ is computed on volume μ_{ij}. The diamond cell in 4.3 is constructed by connection of centers of gravity (i, j) of cells T_i, T_j which shares the face σ_{ij} and its endpoints A, B.

$$\overrightarrow{\nabla} n_{e_{ij}} = \frac{1}{2\mu(D_{\sigma_{ij}})} \left[(n_e(A) - n_e(B)) \overrightarrow{n}_{LR} |\sigma_{LR}| + (n_e(R) - n_e(L)) \overrightarrow{n}_{ij} |\sigma_{ij}| \right] \tag{4.49}$$

Algorithm 5: Compute value at node $n_{e_{node}}$



$n_{e_{node}}$:double;
N :number of nodes;
$Alpha$:weight coming from the least square method;
for *n:=1* **to** *N* **do**
  for *c:=1* **to** *inner cells around node n* **do**
    $n_{e_{node}}(n) += Alpha * n_{e_j}(c)$;
  end
  for *m:=1* **to** *halo cells around node* **do**
    $n_{e_{node}}(n) += Alpha * n_{e_h}(m)$;
  end
end
return $n_{e_{node}}$;

Algorithm 6: Compute $\nabla n_{e_{ij}}$ on face ij

F:number of faces;
$\overrightarrow{\nabla} u$:Vector2d;
for *k:=1* **to** *F* **do**
  A:first node of face;
  B:second face node;
  i:center of gravity of cell T_i;
  j:center of gravity of cell T_j;
  $mes = \frac{1}{2\mu(D_{\sigma_{ij}})}$;
  if *Inner faces* **then**
    $\overrightarrow{\nabla} u(k) = mes * (n_{e_{node}}(A) - n_{e_{node}}(B)) \overrightarrow{n}_{LR} |\sigma_{LR}| + (n_{e_j} - n_{e_i}) \overrightarrow{n}_{ij} |\sigma_{ij}|$;
  end
  else if *Halo faces* **then**
    $\overrightarrow{\nabla} u(k) = mes * (n_{e_{node}}(A) - n_{e_{node}}(B)) \overrightarrow{n}_{LR} |\sigma_{LR}| + (n_{e_h} - n_{e_i}) \overrightarrow{n}_{ih} |\sigma_{ih}|$;
  end
end
return $\overrightarrow{\nabla} n_e$;

Solving linear system using MUMPS
The MUMPS (MUltif-rontal Massively Parallel Solver) is one of parallel distributed solver using a multi-frontal method for the solution of sparse linear equations. The solution of sparse system with the MUMPS solver is achieved in three phases:

1. The analysis phase;
2. The Factorization phase;
3. The solution phase.

Algorithm 7: Solving linear system

```
1 V: vector for unknowns;
2 get_triplet(V_i, V_j, V_x) // compute the tree vectors to solve
  linear system;
3 analyze()// Analyze part;
4 factorize()// Factorization part;
5 solve()// Solving linear system;
6 return V;
```

4.2.3 Parallel 2D results

Table 4.1 and 4.2 summarize the 2D streamer results. The speedup in this example shows a good scalability of the present method for solving the convection-diffusion equation coupled with poisson equation problems using mesh with:

- First case : 529240 cells, 264621 nodes.
- Second case : 1321486 cells, 662244 nodes.

For practical applications, the computation time could be reduced in :

- The first case from 49h54min (one computing core) down to 5min (1024 computing cores).
- The second case from around 180h (one computing core) down to 14min (1024 computing cores).

These tests is made on MAGI cluster at Paris 13.

4.2.3.1 Scalability measurement:

There are two basic ways to measure the scalability of parallel algorithms:

1. Strong scaling with a fixed problem size but the number of processing elements are increased.
2. Weak scaling with problem size and compute elements increasing concurrently.

The speedup is defined as

$$sp = \frac{t_b}{t_N},$$

and the ideal speedup, sp_{ideal}, is naturally equivalent to the number of compute cores. Therefore, the strong scaling efficiency can also be calculated by

$$\frac{s_p}{sp_{ideal}}$$

which shows how far is the measured speedup from the ideal one.

In the this work, the strong scaling measurement is employed and the scaling efficiency is calculated by:

$$\frac{t_b N}{t_N N_b} 100,\ N \geq N_b$$

where t_b is the wall clock time of a base computation, N_b is the number of compute cores used for the base computation, t_N is the wall clock time of a computation with N compute cores.

4.2.3.2 First case

Table 4.1: Execution time (in s) of different parts of parallel 2D streamer code using mesh with 529240 cells

Compute cores	Total	Convection	Diffusion	Linear solver
1	184561	10542	48438	125580
2	92837	5112	24715	63010
4	46502	2595	12452	31454
8	23868	1369	6559	15939
16	13542	778	3801	8963
32	6830	496	1805	4529
64	3801	246	1053	2502
128	2017	115	515	1386
256	1125	63	281	780
512	623	34	176	413
1024	323	18	95	209

The number of MPI processes varies from 1 to 1024. We observe in Fig. 4.6 that the CPU time decreases significantly with the number of processes, showing then a good scalability of the solver. We observe that the diffusive term represent 26% of the global computational time and this percentage remains the same when the number of processes increases, the linear system represent 68% and the convective terme represent 6%.

4.2.3.3 Second case

Table 4.2: Execution time (in s) of different parts of parallel 2D streamer code using mesh with 1324486 cells

Compute cores	Total	Convection	Diffusion	Linear solver
1	646739	32953	140166	473620
2	324483	15692	70791	238000
4	157412	7560	34335	115517
8	82145	4224	17970	59951
16	42994	2243	9997	30754
32	22421	1176	5191	16054
64	11470	610	2695	8165
128	6043	323	1415	4305
256	3150	169	737	2244
512	1628	87	378	1163
1024	840	46	199	595

Figures 4.7, 4.8, 4.10 and 4.11 illustrate the speedup and the strong scaling efficiency in the computations of our illustrate example with t_b the wall clock times by one compute core. A more detailed insight into the data of this example provides interesting information about the computational efficiency of the individual parts of the parallel simulation. The results of the strong scaling tests, plotted in Fig 4.8 depict that the parallel speedup of the different parts increases up to 1024 computational cores, so our parallel strategy shows it is efficiency.

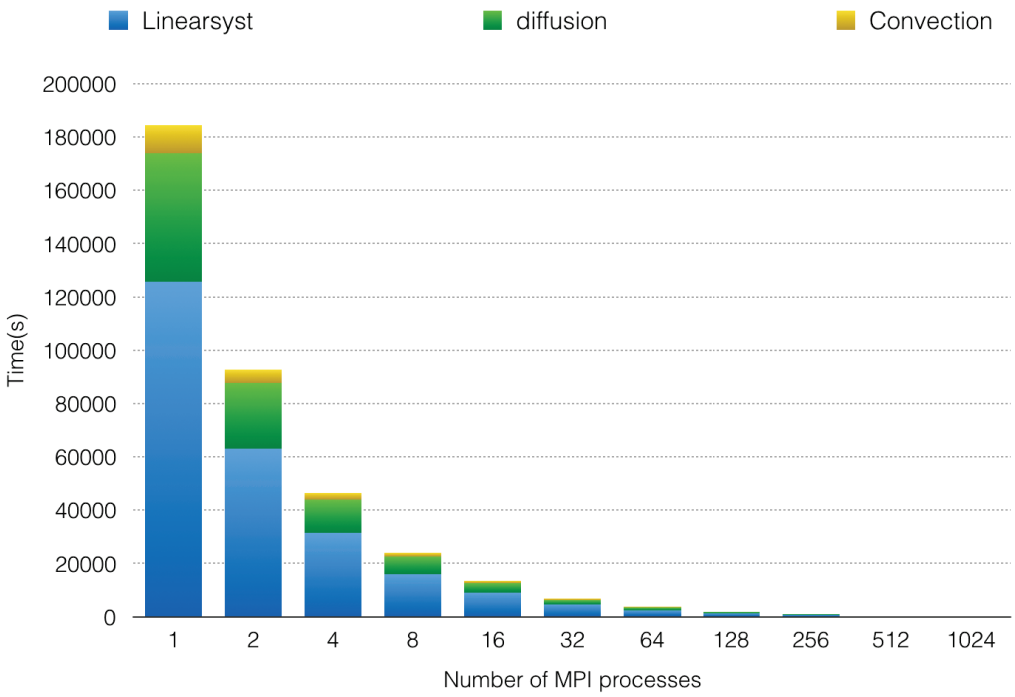


Figure 4.6: Time breakdown for running 2D streamer code using 529240 cells

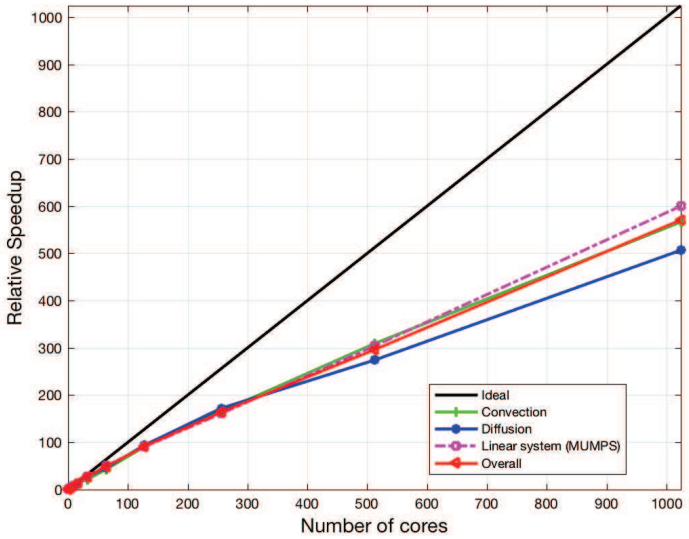


Figure 4.7: Speedup of different parts of parallel 2D streamer code using mesh with 529240 cells

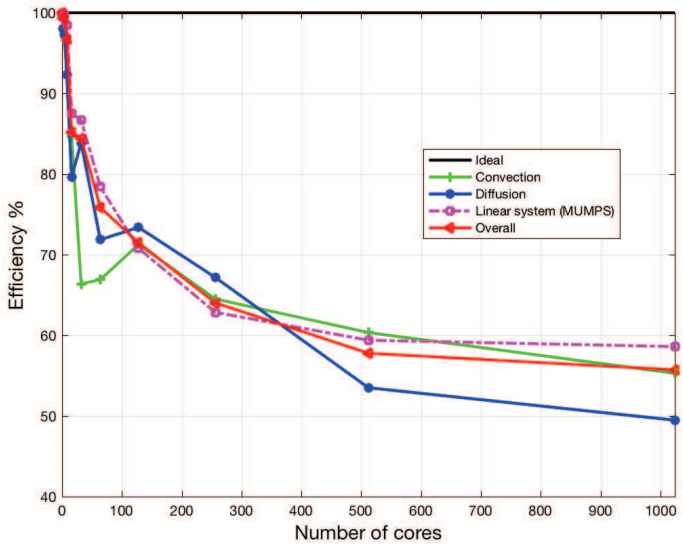


Figure 4.8: Efficiency of different parts of parallel 2D streamer code using mesh with 529240 cells

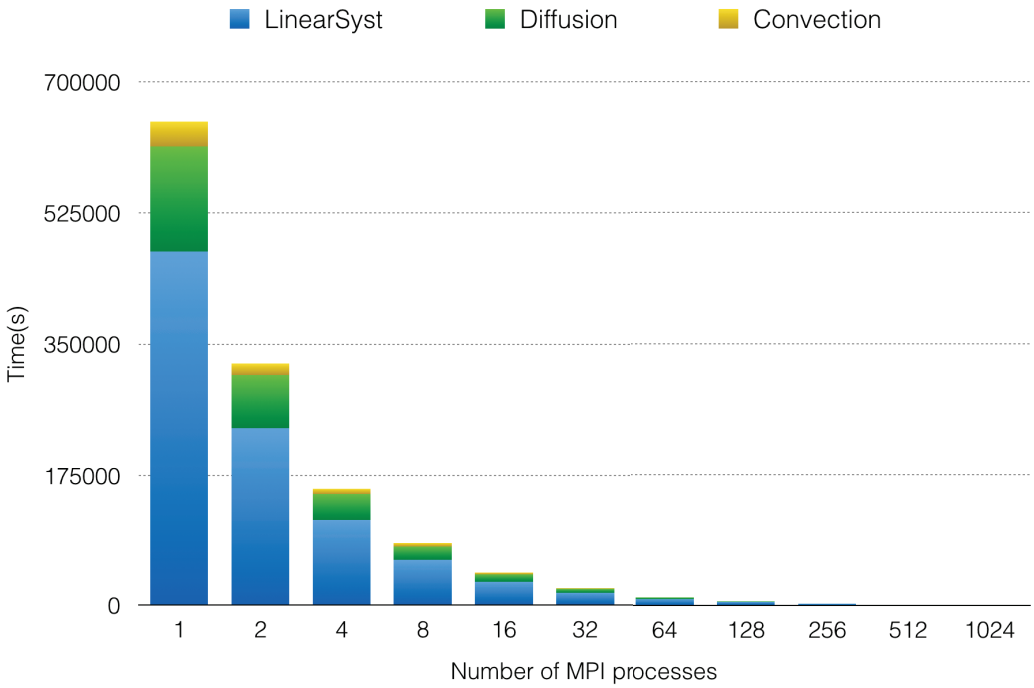


Figure 4.9: Time breakdown for running 2D streamer code using 1324486 cells

4.2.3.4 Comparison between sequential and parallel code

In Figure 4.12 a comparison is made between sequential 2D streamer code and parallel counterpart. We can also observe the propagation of streamer in two cases: more the mesh is fine more the results are accurate (without oscil-

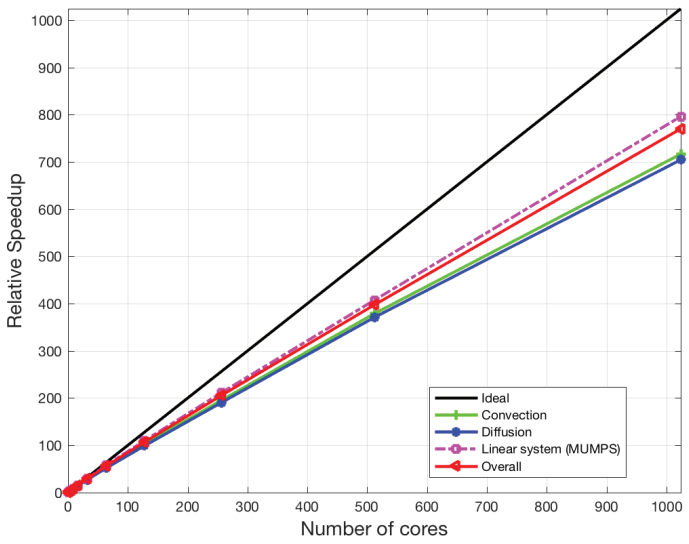


Figure 4.10: Speedup of different parts of parallel 2D streamer code using mesh with 1324486 cells

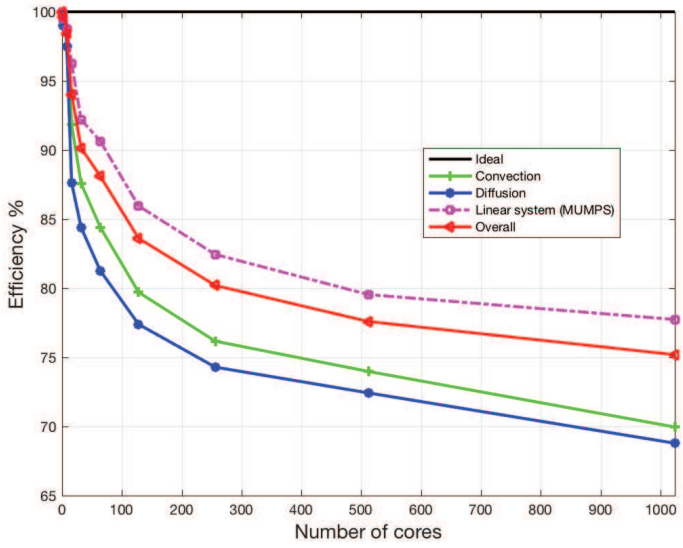


Figure 4.11: Efficiency of different parts of parallel 2D streamer code using mesh with 1324486 cells

lations). In table 4.3, the parallel code takes the same running time (using 64 MPI processors) as the sequential code but the input size is 12 times greater. To conclude, our strategy although not classical shows to be very efficient, and provides with relevant properties regarding performance.

Mesh type	Compute cores	Execution time
Mesh using 43674 elements	1	04h30min
Mesh using 529240 elements	1	150h15min
	64	03h59min

Table 4.3: Comparison between sequential and parallel code

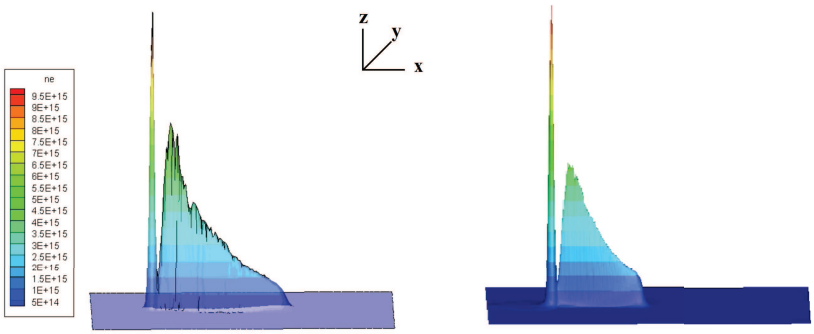


Figure 4.12: Plasma discharge in sequential code (left) and parallel one (right)

4.3 Performance results for parallel 3D streamer code

4.3.1 Benchmark settings

All computations are done on the same reference grid with 657542 cells on unstructured mesh. The second order scheme is used in the tests.

4.3.1.1 Initial conditions

We consider the following initial conditions for the rectangular domain

$$\begin{aligned} &n_e(x,y,0) = 10^{12}.e^{-\frac{(x-0.2)^2+(y-0.25)^2+(z-0.25)^2}{\sigma^2}} + 10^6[cm^{-3}], \sigma = 0.01, \\ &n_i(x,y,z,0) = n_e(x,y,z,0) \end{aligned} \tag{4.50}$$

The initial Gaussian pulse for the electron and ion densities creates disturbance in the electric field which is necessary for initialization of ionization process in the wave propagation. The background electron and ion densities $10^6\ cm^3$ substitute the photoionization effect which is neglected in this simple streamer model. The computation is stopped at prescribed time when the head of electron avalanche is still inside the computational domain. In further computation, the discharge goes out through the right boundaries and the source terms increase to infinity but this effect doesn't represents physical behavior of the electric discharge.

4.3.1.2 Boundary conditions

The left boundaries of the domains are anode, the right boundaries are plane cathodes. There are prescribed homogeneous Neumann conditions for all unknowns for upper, lower and lateral boundaries. Following boundary conditions are prescribed for the anode and the cathode

$$\begin{aligned} &\tfrac{\partial n_e}{\partial t} = 0, for\, anode\, and\, cathode \\ &V = 25000[V],\ for\, anode \\ &V = 0,\ for\, cathode \end{aligned} \tag{4.51}$$

4.3.2 Parallel 3D results

First case

Table 4.4: Execution time (in s) of different parts of parallel 3D streamer code using mesh with 657542 cells

Compute cores	Total(s)	Convection(s)	Diffusion(s)	Linear solver(s)
1	2440370	317615	201530	1921224
2	1401176	160527	135876	1104772
4	754647	89185	79993	585469
8	432547	62920	54178	315449
16	268265	41249	32610	194405
32	165734	28570	21417	115746
64	99391	18843	12712	67835
128	64881	13697	8632	42551
256	41319	8033	4900	28384
512	29568	6776	3770	19020

Table 4.4 and 4.5 summarizes the 3D streamer results. The speedup in this example shows a good scalability of the introduced method for solving the convection-diffusion equation coupled with poisson equation problems using mesh with 657542 cells. For practical applications, the computation time could be reduced from around one month (one computing core) down to 8h (512 computing cores). This experiment was accomplished on the MAGI cluster at Paris13.

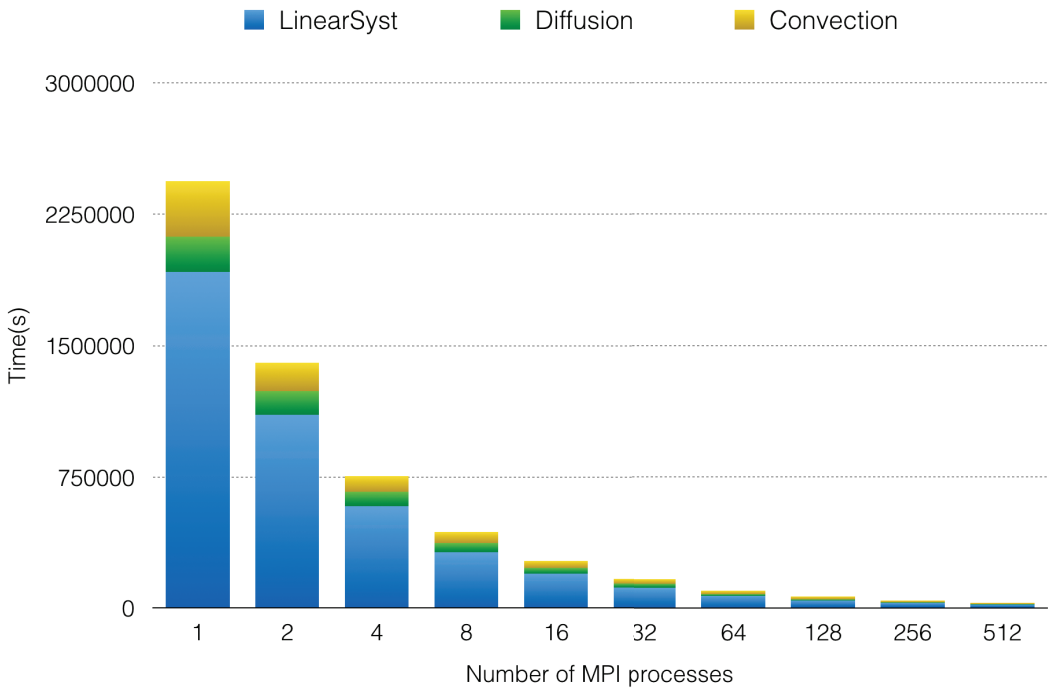


Figure 4.13: Time breakdown for running 3D streamer code using 657542 cells

Figures 4.14, 4.15, 4.17 and 4.18 illustrate the speedup and the strong scaling efficiency in the computations of our illustrate example. A more detailed insight into the data of this example provides interesting information about the computational efficiency of the individual parts of the parallel simulation. The results of the strong scaling tests, plotted in Fig 4.14 depict that the parallel speedup of the different parts increases up to 512 computational cores, so our parallel strategy shows its efficiency. Figure 4.19 represents the 3D streamer propagation in domain [0 0.5][0 0.5][0 0.5] that we can draw from the results of our experiments.

second case

Table 4.5: Execution time (in s) of different parts of parallel 3D streamer code using mesh with 2000186 cells

Compute cores	Total	Convection	Diffusion	Linear solver
1	5233328	1258654	815399	3159275
2	2817795	642452	440061	1735282
4	1505111	343347	264563	897201
8	820245	180135	142910	497200
16	477171	97224	83047	296900
32	258955	47623	45420	165912
64	152617	25540	30648	96429
128	83717	14450	16968	52299
256	55494	8946	9939	36609
512	31413	4899	5595	20919

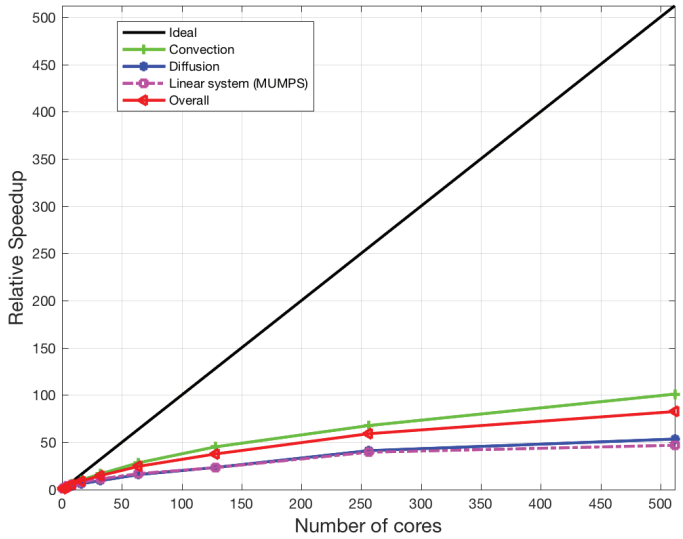


Figure 4.14: Speedup of different parts of parallel 3D streamer code using mesh with 657542 cells

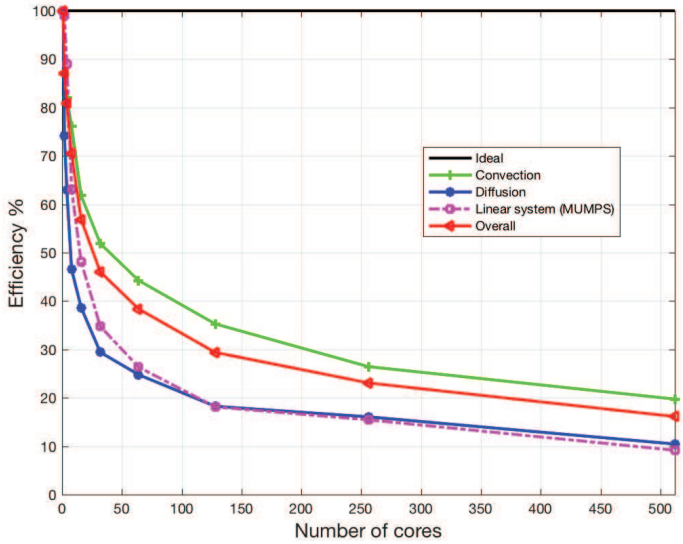


Figure 4.15: Efficiency of different parts of parallel 3D streamer code using mesh with 657542 cells

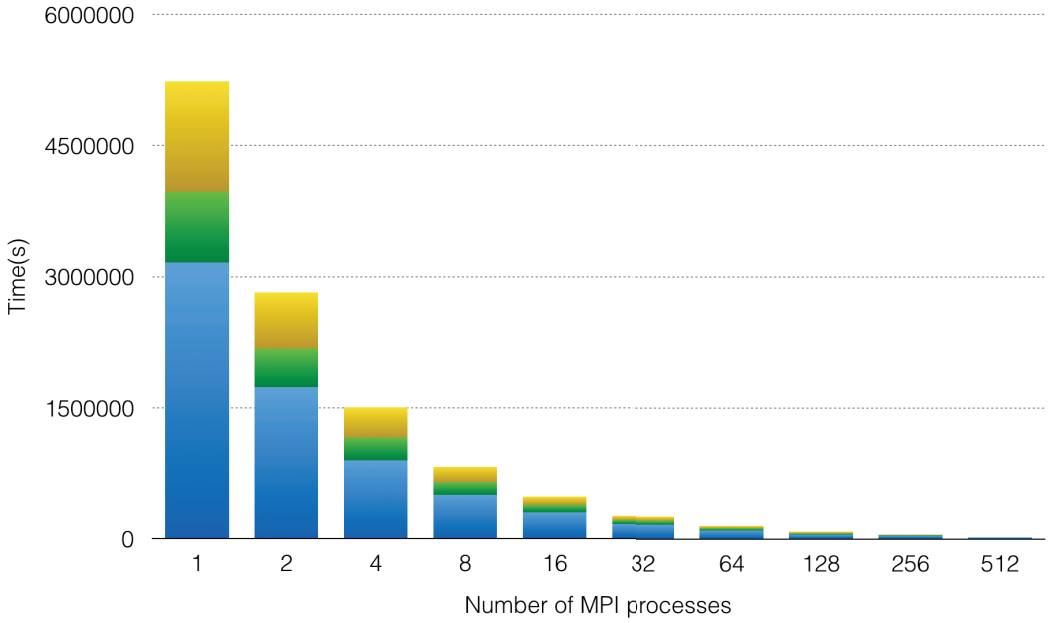


Figure 4.16: Time breakdown for running 3D streamer code using 2000186 cells

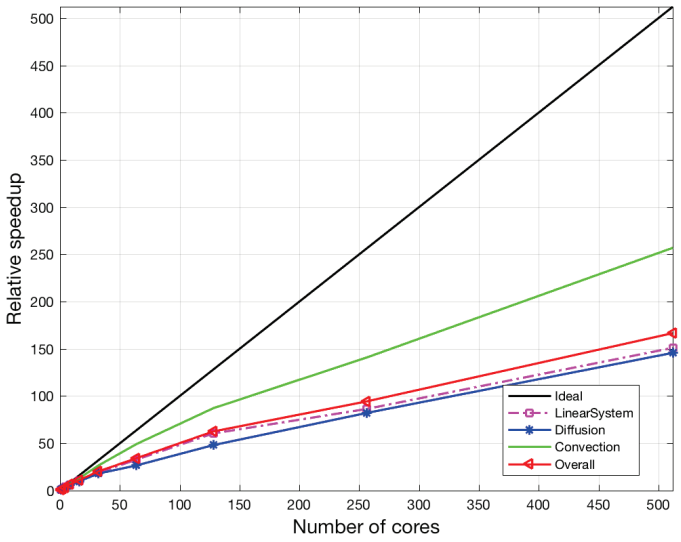


Figure 4.17: Speedup of different parts of parallel 3D streamer code using mesh with 2000186 cells

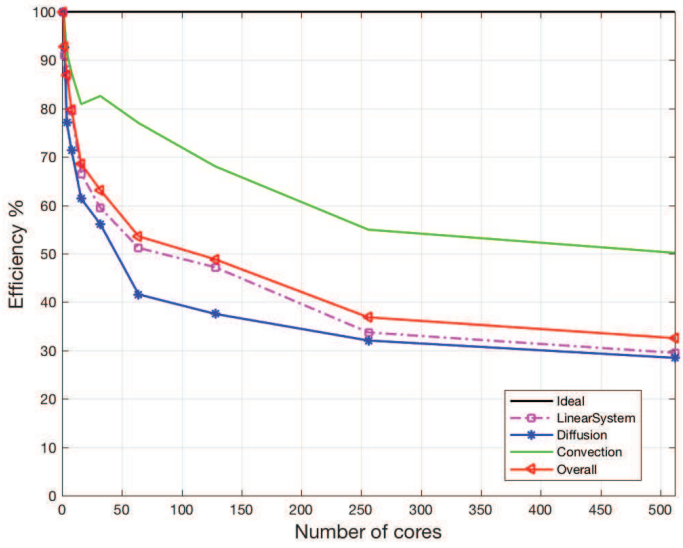


Figure 4.18: Efficiency of different parts of parallel 3D streamer code using mesh with 2000186 cells

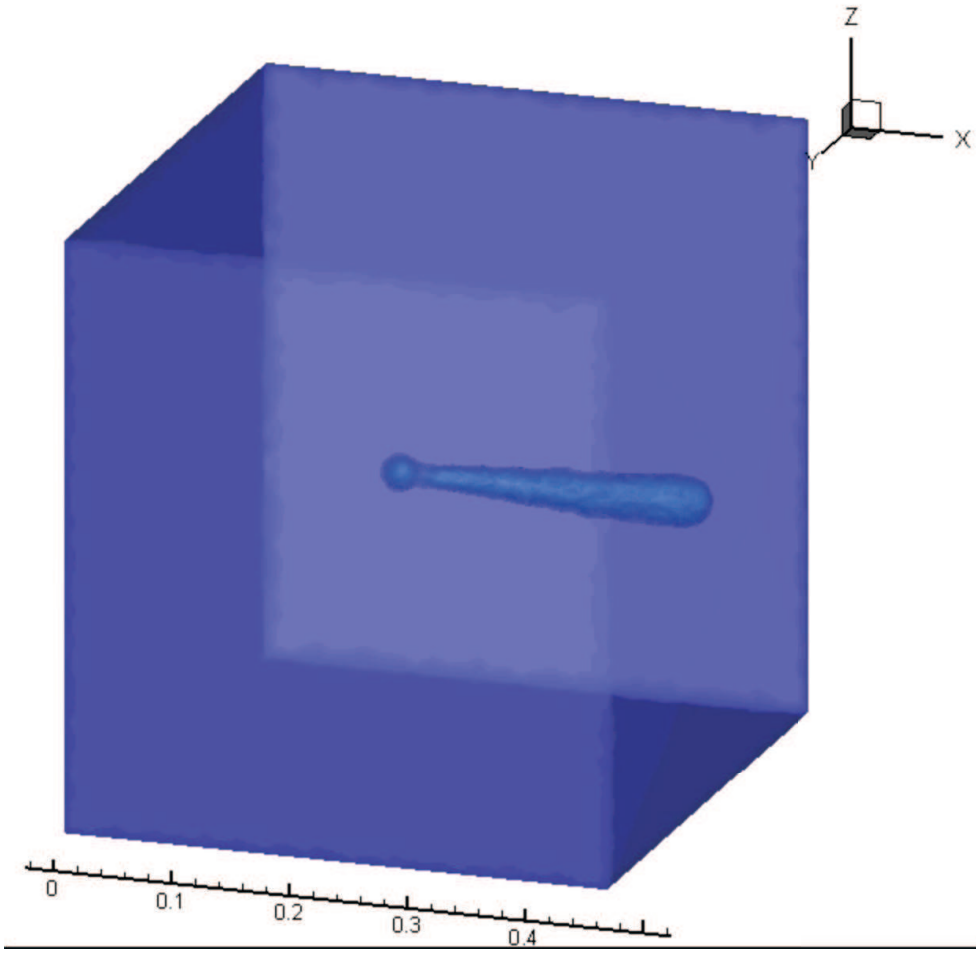


Figure 4.19: Streamer propagation using mesh with 657542 cells

Chapter 5

HPC as a Service

Contents

5.1 Introduction

In this chapter we discuss about the use case of scheduling the ADAPT workflow according to the RedisDG workflow engine. More precisely, we schedule workflows of different sizes of a specific workflow solving one CFD problem of the ADAPT framework. In previous chapters we have introduced parallel decompositions according to a tightly coupled approach. Tightly-coupled software means routines (modules, programs) that work in only one type of system and are dependent upon each other. For instance we have introduced MPI-codes to drive performances.

We now introduce a loosely coupled approach. Loosely-coupled software means routines (modules, programs) are called by an application and executed as needed. Loosely-coupled software often refers to programs with standard interfaces that serve multiple computing environments. Web services or more generally Services employ loose coupling. When a function is required, the appropriate Web service module is executed.

Our main objective now is more in the production of a high level view of our CFD problems rather than in a code with unbelievable performance in time and tailored for a specific computer architecture. In other words we seek to exploit resources all over the Internet and not only those in supercomputers. The developer or scientist point of view is changing and it is now in decomposing the problem according to a Direct Acyclic Graph (DAG) vision and in counting on a 'high performance' workflow engine. The scientist does not need to have knowledge in parallel programming or in parallel architecture and systems. He needs to be focused on his decomposition, not on how to implement it. The computer scientist needs to know where and when to take resources... not the scientist from any others disciplines! In doing this we hide the complexity inherent to the details, thanks to a services vision.

The organization of the chapter is as follows. In section 5.2 we introduce some discussion on the ways to do HPC in clouds. The discussion on the ways to schedule workflows has been done in a previous chapter. In section 5.3 we

introduce our problem. We enforce the idea that we do not provide with complexity analysis of the general problem but we draw paths to solve it practically. We also introduce the DAG we are working with and its property. Section 5.4 introduces our contributions, from an algorithmic point of view, to execute our DAG with the RedisDG framework.

5.2 Related discussions on HPC in the cloud

A 20-node cluster built with virtualized machines that have 64 Gbytes of RAM and 16 cores each costs around US$14 per hour in 2016 and for major companies. The economic model of Qarnot Computing[1], an innovative company in HPC located in Montrouge - France, is 0.25 €/H/CPU (8 cores). More over there is no charge for data transfer, no charge for storage/caching, no charge for booting time. People may wonder if using HPC online services would be the solution.

Challenges such as how to decide which jobs should be moved to the cloud and when is a problem for computer scientists and not decision makers. The US Department of Energy's Magellan Report[2] from December 2011 on Cloud Computing for Science contains analyses on running HPC and data-intensive applications in the cloud.

During the past years, the Magellan project looked at various cloud models such as Infrastructure as a Service (IaaS) and Platform as a Service (PaaS), virtual software stacks, MapReduce and its open source implementation (Hadoop), and resource provider and user perspectives. Specifically, the Magellan project was charged with answering the following research questions:

1. Are the open source cloud software stacks ready for DOE HPC science?
2. Can DOE (Department of Energy) cybersecurity requirements be met within a cloud?
3. Are the new cloud programming models useful for scientific computing?
4. Can DOE HPC applications run efficiently in the cloud? What applications are suitable for clouds?
5. How usable are cloud environments for scientific applications?
6. When is it cost effective to run DOE HPC science in a cloud?

The 169-pages report summarizes the finding as follows:

> Cloud approaches provide many advantages, including customized environments that enable users to bring their own software stack and try new computing environments without significant administration overhead, the ability to quickly surge resources to address larger problems, and the advantages that come from increased economies of scale. Virtualization is the primary strategy of providing these capabilities. Our experience working with application scientists using the cloud demonstrated the power of virtualization to enable fully customized environments and flexible resource management, and their potential value to scientists.
>
> Cloud computing can require significant initial effort and skills to port applications to these new models. This is also true for some of the emerging programming models used in cloud computing. Scientists should consider this upfront investment in any economic analysis when deciding whether to move to the cloud.
>
> Significant gaps and challenges exist in the areas of managing virtual environments, workflows, data, security, and others. Further research and development is needed to ensure that scientists can easily and effectively harness the capabilities exposed with these new computing models. This would include tools to simplify using cloud environments, improvements to open-source clouds software stacks, providing base images that help bootstrap users while allowing them flexibility to customize these stacks, investigation of new security techniques and approaches, and enhancements to MapReduce models to better fit scientific

[1] http://www.qarnot-computing.com/
[2] http://www.nersc.gov/assets/StaffPublications/2012/MagellanFinalReport.pdf

data and workflows. In addition, there are opportunities in exploring ways to enable these capabilities in traditional HPC platforms, thus combining the flexibility of cloud models with the performance of HPC systems.

The key economic benefit of clouds comes from the consolidation of resources across a broad community. Existing DOE centers already achieve many of the benefits of cloud computing because these centers consolidate computing across multiple program offices, deploy at large scales, and continuously refine and improve operational efficiency.

According to the report, cloud models often provide additional capabilities and flexibility that are helpful to certain workloads. The report recommends that DOE labs and centers consider adopting and integrating these features of cloud computing research.

Specifically, the report's findings are as follows:

- Finding 1. Scientific applications have special requirements that require solutions that are tailored to these needs.
- Finding 2. Scientific applications with minimal communication and I/O are best suited for clouds.
- Finding 3. Clouds require significant programming and system administration support.
- Finding 4. Significant gaps and challenges exist in current open-source virtualized cloud software stacks for production science use.
- Finding 5. Clouds expose a different risk model requiring different security practices and policies.
- Finding 6. MapReduce shows promise in addressing scientific needs but current implementations have gaps and challenges.
- Finding 7. Public clouds can be more expensive than in-house large systems.
- Finding 8. DOE supercomputing centers already approach energy efficiency levels achieved in commercial cloud centers.
- Finding 9. Cloud is a business model and can be applied at DOE supercomputing centers.

Additionally, the report states that performance of tightly coupled applications running on virtualized clouds using commodity networks can be significantly lower than on clusters optimized for these workloads. This can be true even at mid-range computing scales. As a result, the report concludes that current cloud systems are best suited for high-throughput, loosely coupled applications with modest data requirements.

Our work is to remove the doubts and provide effective solutions to the different black spots that the report raise. To be precise, in this thesis, we focus on the scheduling and placement decisions of tasks/jobs. We renovate in deep the ways we traditionally use for these issues. We also focus on small/medium size computer systems and not supercomputers that may be monopolized by two communities in France, 'écoulements reéactifs ou/et multiphasiques' and 'physique théorique et physique des plasmas', since they have used 59% of the total number of hours allocated for the TURING supercomputer at IDRIS[3] in 2015. The TURING machine is ranked 81 in the June 2016 Top500 list[4]. We also revisit the performance metrics by which we evaluate a solution and we show how to deal with metrics related to the energy consumption, the load, the cost of systems... for instance. Evaluating a solution only under the makespan metric i.e. a metric used in the supercomputing world, is not fair!

UberCloud[5] is the online community and marketplace where engineers, scientists, and their service providers discover, try, and buy Computing Power and Software as a Service, from Cloud Resource and Software Providers around the world. Engineers and scientists can explore and discuss how to use this computing power to solve their demanding problems. The Web site explains that:

[3] http://www.idris.fr/media/data/lalettre/la-lettre-de-l-idris-mai-2016.pdf
[4] https://www.top500.org/list/2016/06/
[5] https://www.theubercloud.com/

> UberCloud is best known for its UberCloud Experiment, the free, community driven effort that explores the end-to-end process of accessing and using remote High Performance Computing (HPC) resources in HPC Centers and HPC Clouds, on demand. Participants of the experiment get hands on experience on using remote computing resources for their own projects; learning how to cope with the challenges and resolving many of the common roadblocks. Project teams working on end-users applications discuss their experiences, the hurdles they ran into, their lessons learned, and the best practices, on the UberCloud community web site.

The compendiums of case studies now appear annually. For instance, the 2015 one[6] explains the benefit of doing HPC in the cloud from the candid descriptions of problems encountered, problems solved, and lessons learned. Projects covered in this edition are in fields of Computational Fluid Dynamics (CFD), Computer Aided Engineering (CAE), Structural Analysis, and life-sciences.

5.3 Problem definition

5.3.1 Elements of vocabulary

We assume that computations and their dependencies have the structure of dags (directed acyclic graphs) that serve as computational models. The nodes of a computation-dag G represent chores (jobs, tasks, etc.) to be executed; G's arcs (directed edges) represent inter-chore dependencies that constrain the order in which chores can be executed. Specifically, a node v cannot be executed until all of its parents have been executed. Once all of v's parents have been executed, v becomes eligible (for execution) and remains so until it is executed. G may have one or more sources—nodes that have no parents, hence are immediately eligible—and one or more sinks—nodes that have no "children." The execution of G terminates once all nodes have been executed.

A schedule Σ for a dag G is a topological sort of G, i.e., a linear ordering of G's nodes in which all parents of each node v lie to the left of v. The schedule prescribes the order in which G's nodes are selected for execution.

The dag G can be enriched by information related to the execution platform. For instance an edge can be labeled with a weight representing a communication cost for transferring data between two nodes. Depending on the modeling, we have two orthogonal approaches for scheduling G's nodes:

- *Platform oblivious algorithms*: in this case, only the structure of the graph is used for choosing the order of nodes execution; Platform-oblivious scheduling can be advantageous for the targeted platforms because it exploits unchanging, perfectly-known characteristics of one's computation rather than attempting to adapt to characteristics of the platform, which are at best imperfectly known and, indeed, may change dynamically ;
- *Platform dependent algorithms*: in this case, (physical) information related to the platform is used in supplement to the structure of the graph. The advantages are that supposed important parameters influencing the behavior of executions are taking into account in the modeling, hence (supposed) better performance. In doing this we constraint the problem reducing the choices we can potentially make at each step of the scheduling algorithm. One problem is with the choice of the influencing parameters and, in the real world, how we measure them.

5.3.2 The ADAPT workflow

From a methodological point of view, we have realized a conversion of the parallel ADAPT algorithm described with algorithm 3, page 71 into a Direct Acyclic Graph. The ADAPT graph that we schedule has the shape depicted on Figure 5.1. Each node of the graph corresponds to a line in the algorithm 3, page 71. For instance, the node labelled METIS corresponds to line 4, the line 9 corresponds to the nodes INIT{0..7}, the node labelled MUMPS corresponds to line 19, the lines 21 to 27 correspond to the nodes ITER{0..number of iteration}_{0..7}.

The key idea is to decompose the solution according to services (METIS, MUMPS...) and let them being scheduled by the RedisDG workflow engine. The key idea is no more with the design of tightly coupled parallel codes, as with MPI, but rather, to adopt a higher view in terms of general services. As with the design of computer programs,

[6] https://www.theubercloud.com/ubercloud-compendium-2015/

the process of converting a parallel or sequential program into a scientific workflow is not a formal process and it is deeply anchored into the experience of the programmer in term of idioms and best practices.

5.3.3 Intuition for the problem

The framework for executing the ADAPT workflow is the RedisDG system. This system has been built to tackle the problems with the extreme levels of *dynamic heterogeneity* of modern computing platforms such as clouds and desktop grids. The availability and relative efficiencies of such platforms' computing resources can change at unexpected times and in unexpected ways. The central interaction paradigm used for the RedisDG system is the publish-Subscribe paradigm.

For instance, the modeling of a node (volunteer) that wants to enter to the system, dynamically, corresponds to a subscribe action on a channel. The modeling of a node that leaves the system corresponds to an unsubscribe action on a dedicated channel. The dynamic arrival/departure of nodes disrupts the scheduling decisions that cannot be done according to static information but on an opportunistic manner. With RedisDG, when a request for executing a task is made by the coordinator, all or some nodes/workers reply to the request. Then a decision should take place to elect one node.

This process can be depict according to the following steps. Given a workflow to schedule, our 'RedisDG Publish-Subscribe' orchestrator proceeds as follows: first, tasks or subset of tasks that could be processed are selected (depending on tasks and data dependencies). Then, it publishes events for the execution of the tasks. Subscribers start to answer and at a chosen date, the orchestrator decides to deploy the tasks with the list of subscribers it knows.

There are two challenges here: the first one is to decide on the date at which we consider that the sublist of subscribers we have is enough. Indeed, the more we wait, the more we have subscribers, but the more we globally delay the processing of the workflow. The second challenge is to decide on the best allocation for the subscribers. In this thesis, we mainly propose to augment the set of practical heuristics, adapted to the ADAPT workflow, for this problem and we make an experimental evaluation of them.

5.3.4 Theoretical view of the global problem

We now describe the problem that we want to solve. Let us assume a bag of N jobs where each job corresponds to a set of tasks. Our allocation problem can be formulated as a decision problem with N rounds. In the round i, the scheduler publishes and decides on the best allocation of tasks of the i^{th} job.

The round i starts at date t_{pub} with the publication of a request for the processing of the set of tasks $J_i = \{T_{i,1}, \ldots, T_{i,n}\}$. These tasks could be processed by at most $|W_i|$ workers according to a finite set of strategies $S_1, \ldots, S_k$. A strategy defines how the scheduler deploys J_i on the workers (W_i). We assume a time interval $]t_{pub}, t_w]$ in which the scheduler could wait before deciding on the allocation of all tasks. At any date $t \in]t_{pub}, t_w]$, there is a finite set $W_i(t)$ of Workers that subscribed to the published request. The goal for the scheduler is to choose, at the beginning of each round i, a maximal waiting time (or a closing date) $t_i^{opt} \in]t_{pub}, t_w]$ and a strategy S_i^{opt} for the processing ofJ_i such as to maximize a total reward R

$$\sum_{i=1}^{N} R(t_{pub}, S_i^{opt}, W_i(t_i^{opt}))$$

. In this model the reward, we can expect from a strategy, could include several possible dimensions (response time, fairness, energy etc.). It also depends on the publishing time t_{pub} and the closing date t_i^{opt}. The closing date is used because it decides on the set of workers to which a task could be assigned. The publishing date is considered to handle the cases where the strategy that was chosen will start to deploy the tasks at date $t' = t_{pub} + \epsilon$ which is the first date coming after t_{pub} and where $W_i(t_{pub}+\epsilon)$ is not null. In this general model, we also make the following considerations:

1. At any date $t < t'$, $W(t)$ is unknown;

2. At the beginning of each round, the scheduler does not know the reward that any strategy and closing date could lead to.

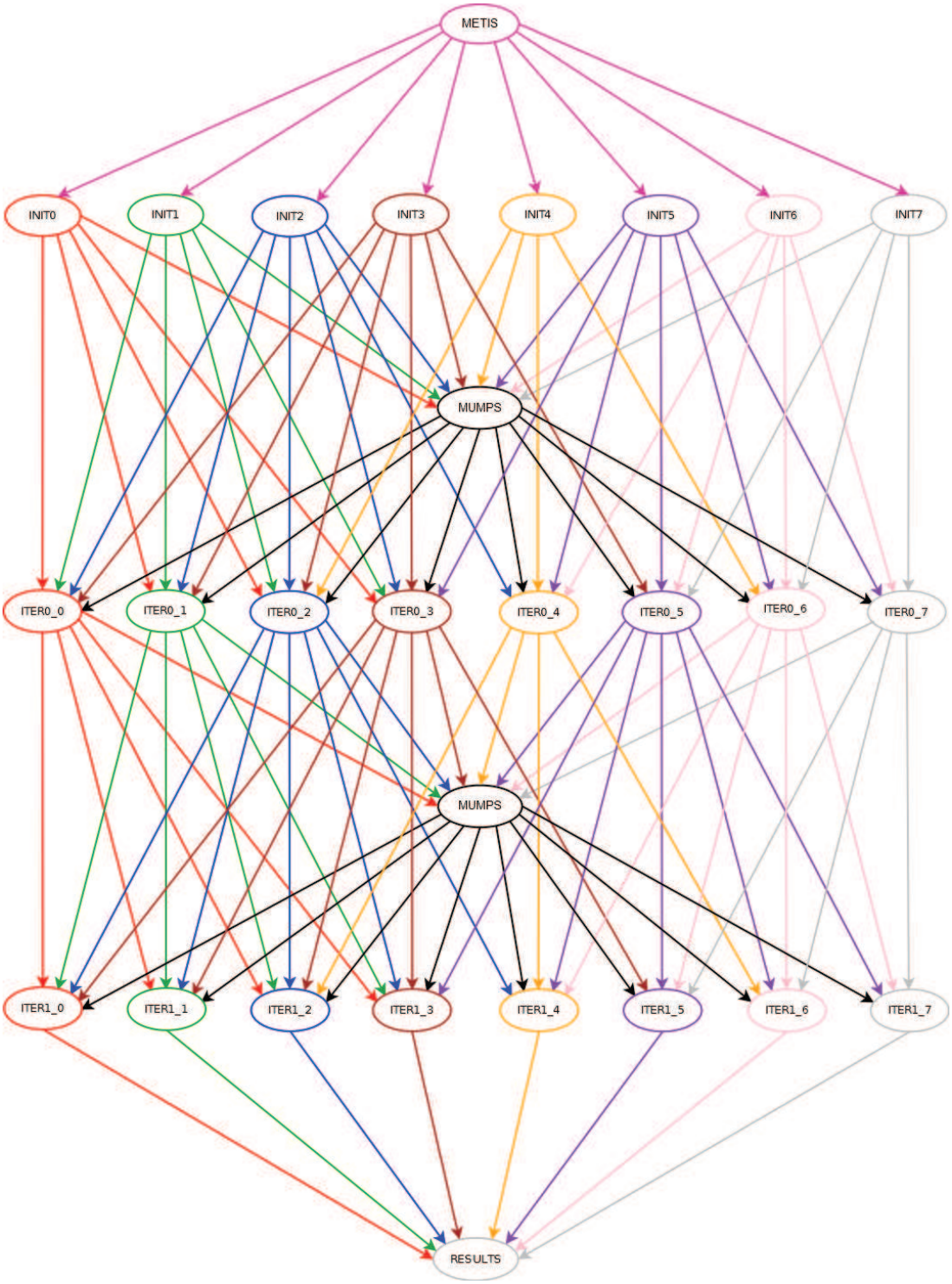


Figure 5.1: The ADAPT workflow

This general decision problem is hard to solve. Indeed, we can formulate some of its variants as a multi-armed bandit problem [101]. For instance, let us assume that the time is discretized and that we have only one strategy. Then, the problem we have consists in choosing a date between t_{pub} and t_w such as to maximize the total reward on a finite set of rounds; this corresponds to the multi-armed bandit problem. In the same way, if we consider that t_i^{opt} is always fixed, we have again a bandit problem that consists of choosing among k strategies in N rounds.

5.4 Scheduling the ADAPT workflow

5.4.1 Recent work

The LIPN-AOC team has recently studied (papers are under reviewing) different methods and reward strategies to decide the workers to choose. They remain valid for the ADAPT workflow and they are the following ones, under a summarized form:

- Methods:
  - The Uniform policy: volunteers are chosen according to uniform random law;
  - The OldestElected policy: the coordinator chooses workers with the oldest date when a task has been scheduled;
- Rewards:
  - The execution time spent in computing nodes; The goal will be to finish the execution as soon as possible;
  - The fairness. We define the *fairness* as the standard deviation of the cumulative computing time for each worker on a given time window. Note that this notion can be refine for a job; The idea is to make an equilibrium over all the nodes to ensure that every node is computing time after time;
  - The energy consumption. We denote $CPULoad(w, x)$ the average CPU load during the execution of a task x on a worker w. We assume that each worker can express a value of energy consumption of a task instance from the CPU load thanks to a function $\mathcal{E}$. This function depends of the consumption model of the hardware, i.e. the CPU, installed on the worker w. Note that this notion can easily be generalized for a job. The idea is to distribute the work in order to minimize the energy consumption.

Then, the following heuristics has been deployed:

- HEURISTIC 1 (FCFS): Upon a task becomes independent, the coordinator publishes it to all the workers. This task is allocated on the first answering worker.
- HEURISTIC 2: Upon a task becomes independent, the coordinator publishes it to all the workers. If a worker w is free it answers directly, otherwise it delays the answer to the coordinator until it becomes free. This ensure the liveness property because at least one worker answers eventually. When the coordinator receives a reply from a worker w for task t, it first saves this will for t. If t has been already allocated to another worker, the coordinator allocates to w another task t' that has not yet been executed among those for which w wished to execute.
- HEURISTIC 3: Upon a task becomes independent, the coordinator publishes it to all the workers and starts a timer for λ time unit. All workers answer whatever their state ensuring thus the liveness. During λ, the coordinator saves all volunteer workers for a task. Upon the expiration of the timer (timeout), it chooses a worker among the set of volunteers according to a policy described in the beginning of this section. If no worker has answered, the coordinator activates again the timer.
- HEURISTIC 4: The aim of this heuristic is to promote the communication between workers avoiding the master node solicitation and thus useless communication. Indeed, it is possible to transfer directly intermediate data between two workers which execute two dependent tasks. To achieve this goal, the procedure is identical to the heuristic 3 except that:

- all tasks are published by the master at the job submission time to make a priori allocation based on policies described in the previous subsection. Thus, once all tasks are assigned it is possible to determine the communication scheme.
- workers subscribe to the $FinishedTask$ channel in order to know when their predecessors has finished leading to a data downloading.

5.4.2 Properties of the ADAPT workflow

The ADAPT workflow, depicted on Figure 5.1, has the following properties when we execute it on a real platform:

1. The cost in time for executing the first two stages of the workflow (METIS step) is very low comparing to the other costs;
2. The cost of transferring data between nodes is also very low. The transferring data correspond to the exchange of parameters of the model; Heuristics taking into account this cost has no influence on the result and may degrade the overall performance if the time cost for deciding or realizing the 'optimized' transfers is high.
3. The dag is symmetric, 'horizontally' and 'vertically'. If we observe the dag from the top to the bottom and excepting the initial METIS step, we note a succession of exchange and merge steps that repeat each others. The execution time of any 'path' from the root node to the sink node is the same, meaning that all the paths are equivalent. A scheduling strategy based on the critical path of the graph without weights (the longest path in term of number of traversed nodes from the source to the sink) leads to unless work since all the paths are equivalent.

These observations motivate our study to propose novel and complementary approaches comparing to the ones that we have explained above. Our aim is also to propose a balance between strategies that are *fully platform oblivious* and *fully platform dependent*. Based on the previous observations, it seems difficult to develop a strategy based on a simple structural property of the considered dag, such as the order (the number of vertices), the size (the number of edges), the diameter (the longest of the shortest path lengths between pairs of vertices) since the values for these parameters do not discriminate a lot.

We do prefer to investigate a) the impact of the availability of nodes, b) the impact of coupling strategies for the decisions on the selection of nodes and c) the impact of a) and b) simultaneously.

5.4.3 Potential impact of the availability of nodes

In the field of desktop grids, the studies on the availability of workers [102, 103] by Kondo et al. show that workers follows different statistical laws. Authors have identified:

1. six clusters of nodes with different statistical laws (Gamma (4) - Log-Normal (1) and Weibull (1)) ;
2. a global law for all the workers that follows a Weibull law of shape of 0.3787 and scale of 3.0932. We keep this model for our work despite the fact that this choice is questionable versus the choice of a model with six clusters because the variability in the distributions in the clusters seems important as quoted by the authors;

Let us recall that the Weibull laws constitute mainly approximations which are particularly useful in various techniques whereas it would be very difficult and of little interest to justify a particular form of law. From this point of view they are analogous to the normal law that effectively replaces distributions 'almost' symmetrical. A distribution with positive values (or, more generally but less frequently, with values greater than a given value) has almost always the same shape. It starts from a frequency of occurrence which null, increases to a maximum and decreases more slowly. It is then possible to find in the Weibull family a law which does not deviate too much from the available data by calculating the shape and scale parameters from the mean and the variance observed.

An example of an availability model for the workers is depicted on Figure 5.2. On the x axis we have time in second and on the y axis we have the probabilities for a worker to be available. Both models have the same shape parameter equal to 0.3787 as recorded by Kondo in [102], [103], [104]. The scale parameter of the law is different.

This allows to generate a model where the machine is present on the first time interval and then no longer available on the other intervals. The second model makes it possible to simulate a machine not really available at the beginning and at the end of the time interval and available on intermediate time intervals. Both models are based on time periods of about 10 minutes (600 seconds), which corresponds to the time measured to execute one of the ADAPT workflows when machines are still available.

Note that availability problems may also appear with the Amazon or other major cloud services providers context. The Spot instances of AWS are defined as follows:

> Spot instances let you bid on spare Amazon EC2 instances to name your own price for compute capacity. The Spot price fluctuates based on the supply and demand of available EC2 capacity. Your Spot instance is launched when your bid exceeds the current Spot market price, and will continue run until you choose to terminate it, or until the Spot market price exceeds your bid. You are charged the Spot market price (not your bid price) for as long as the Spot instance runs. If the Spot market price increases above your bid price, your instance will be terminated automatically and you will not be charged for the partial hour that your instance has run. With Spot instances, you will never be charged more than the price you bid.
>
> If you include a duration requirement with your Spot instances request, your instance will continue to run until you choose to terminate it, or until the specified duration has ended; your instance will not be terminated due to changes in the Spot price.

We can observe that the cause of deconnections of the service may be given in exceeding a budget... and not because of a network failure or more intuitive problems.

In our system, any worker is now running as follows. When the coordinator publishes a request to execute a task, the workers now publish, among the information that return to the coordinator, if they are available. For that, the workers look at the current time interval and according to a threshold, the workers decide if they are available or not. In our implementation, when a period of time ends, we restart a new generation for a Weibull law in order to not wait indefinitely. This process helps in avoiding deadlock problems because we have ensured that the threshold, which determines whether the machine is available or not, is always greater than the smallest of the probability values. So, at each new generation there will be at least one time interval for which the machine will be available.

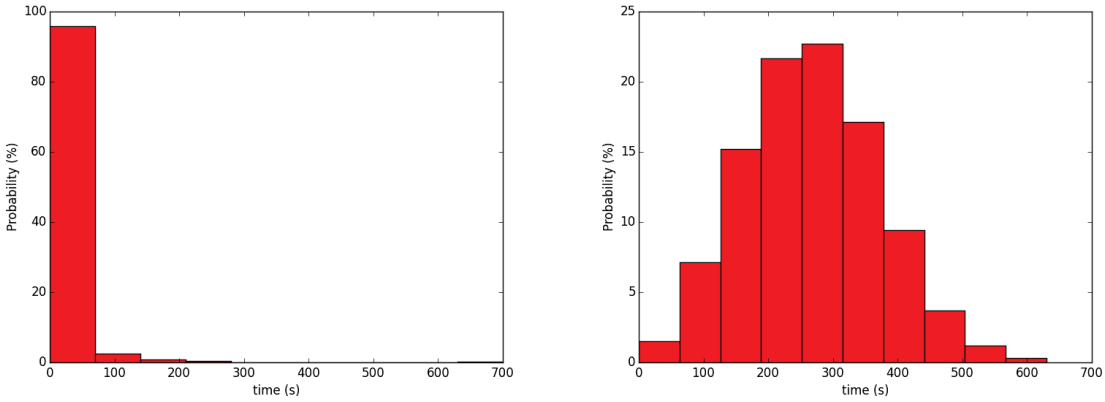


Figure 5.2: Examples of Weibull laws for the availability of workers

5.4.4 Potential impact of coupling both strategies

In integrating into our RedisDG framework both a model for the availability of workers as well as a combination of decision criteria to select workers we become more and more realistic and we can study, experimentally, the best

trade-offs between the different parameters characterizing a real workflow engine system. In another words, the goals are to measure the impacts of such decisions on the overall performance of the RedisDG system and to explore more realistic environments.

5.4.4.1 Integration of an availability model

According to the proposed Heuristics 3-bis we capture the availability of nodes in the selection process as follows. Note that this heuritics is inspired by the abovementioned Heuritics 3.

Heuristics 3-bis: upon a task becomes independent, the coordinator publishes it to all the workers and starts a timer for λ time unit. All workers answer, whatever their states, ensuring thus the liveness and also publish if they are available or not. During λ, the coordinator saves all volunteer workers for a task. Upon the expiration of the timer (timeout), it chooses a worker among the set of available volunteers according to a policy described in the beginning of section 5.4 (see the discussion about methods and rewards). If no worker has answered or if all the workers that have answered are unavailable, the coordinator activates again the timer and it publishes again the task.

Let us now introduce some observations done during an experiment with this new heuristics. We consider an ADAPT workflow as the one depicted on Figure 5.1 of width 4 but with 506 tasks. This means that the height of the workflow is more important than for the workflow depicted on Figure 5.1. The number of nodes for executing the workflow is 6: one for the coordinator and 5 workers. We also use 2 clusters of the Grid5000 testbed, both located in the Nancy site. During the execution, 3 workers have choosen (randomly) to behave like the availability model at the right of Figure 5.2 and two workers have chosen to behave according to the availability model at the left of Figure 5.2.

The total execution time is 01h:07m:41s and it is given by the two measures of Start Time: 2016-12-25 01:17:17,999, and End Time: 2016-12-25 02:24:58,526.

From the logs of the coordinator, we extracted the following trace:

```
2016-12-25 01:18:56,987 - RedisDG-coordinator - INFO - Coordinator publish task 5_1 to workers
2016-12-25 01:18:56,989 - RedisDG-coordinator - INFO - worker worker_grimoire-7.nancy.
grid5000.fr_1482624969.4 is Falsevolunteer for 5_1 :  state_task = SUBMITTED
2016-12-25 01:18:56,990 - RedisDG-coordinator - INFO - worker worker_graphene-49.nancy.
grid5000.fr_1482624969.46 is Truevolunteer for 5_1 :  state_task = SUBMITTED
2016-12-25 01:18:56,990 - RedisDG-coordinator - INFO - worker worker_graphene-52.nancy.
grid5000.fr_1482624969.23 is Truevolunteer for 5_1 :  state_task = SUBMITTED
2016-12-25 01:18:56,991 - RedisDG-coordinator - INFO - worker worker_graphite-2.nancy.
grid5000.fr_1482624969.27 is Falsevolunteer for 5_1 :  state_task = SUBMITTED
2016-12-25 01:18:56,991 - RedisDG-coordinator - INFO - worker worker_griffon-25.nancy.
grid5000.fr_1482624968.45 is Truevolunteer for 5_1 :  state_task = SUBMITTED
2016-12-25 01:19:03,118 - RedisDG-coordinator - INFO - Timeout for task 5_1
2016-12-25 01:19:03,118 - RedisDG-coordinator - INFO - volunteer for task 5_1, should submit
2016-12-25 01:19:03,118 - RedisDG-coordinator - INFO - Choosing a volunteer for 5_1
2016-12-25 01:19:03,118 - RedisDG-coordinator - INFO - volunteer worker_griffon-25.nancy.
grid5000.fr_1482624968.45 choosed for 5_1
```

The coordinator publishes task 5.1. Two workers are unavailable (grimoire-7 and graphite-2). Three workers are available (graphene-49, graphene-52 and griffon-25). At time 01:19:03,118 the timer expires, thus a worker is selected (randomly). griffon-25 is elected as the worker in charge of task 5_1.

From the logs of the griffon-25 worker we get:

```
2016-12-25 01:18:57,787 - RedisDG-worker - INFO - Worker worker_griffon-25.nancy.
grid5000.fr_1482624968.45 is finishing task T32_1
2016-12-25 01:18:57,790 - RedisDG-worker - INFO - I m available?  True
2016-12-25 01:19:03,921 - RedisDG-worker - INFO - Task 5_1 has been selected for ['worker
_griffon-25.nancy.grid5000.fr_1482624968.45'] (state task = GIVEN)
2016-12-25 01:19:03,922 - RedisDG-worker - INFO - worker worker_griffon-25.nancy.
grid5000.fr_1482624968.45 selected for 5_1
2016-12-25 01:19:03,922 - RedisDG-worker - INFO - add task 5_1 in tasks_ready_to_run
2016-12-25 01:19:03,922 - RedisDG-worker - INFO - Worker worker_griffon-25.nancy.
```

```
grid5000.fr_1482624968.45 is executing task 5_1
2016-12-25 01:19:03,922 - RedisDG-worker - INFO - worker worker_griffon-25.nancy.
grid5000.fr_1482624968.45 :  I start the execution of 5_1
```

This trace shows that at 01:18:57,787 griffon-25 has terminated the execution of task $T32_1$, and it is still available. At time 01:19:03,921 griffon-25 has been selected for the execution of task 5_1. Thus griffon-25 starts to execute this task.

In the logs of the Coordinator we can also observe a period for which no workers are available. This period starts at 02:21:19,752 . When the timer expires at 02:22:07,379 the coordinator concludes that no workers are available for tasks 97_1. Then it publish again the same task at 02:22:07,380 (the time before the new-publication is chosen randomly in the range 0 to 5 seconds)... and this time graphene-52 is chosen as the worker for executing task 97_1. The log corresponding to this scenario is as follows:

```
2016-12-25 02:21:19,752 - RedisDG-coordinator - INFO - Coordinator publish task 97_1 to
workers
2016-12-25 02:21:19,754 - RedisDG-coordinator - INFO - worker worker_grimoire-7.nancy.
grid5000.fr_1482624969.4 is Falsevolunteer for 97_1 :  state_task = SUBMITTED
2016-12-25 02:21:19,755 - RedisDG-coordinator - INFO - worker worker_graphene-49.nancy.
grid5000.fr_1482624969.46 is Falsevolunteer for 97_1 :  state_task = SUBMITTED
2016-12-25 02:21:19,755 - RedisDG-coordinator - INFO - worker worker_graphene-52.nancy.
grid5000.fr_1482624969.23 is Falsevolunteer for 97_1 :  state_task = SUBMITTED
2016-12-25 02:21:19,768 - RedisDG-coordinator - INFO - worker worker_griffon-25.nancy.
grid5000.fr_1482624968.45 is Falsevolunteer for 97_1 :  state_task = SUBMITTED
2016-12-25 02:21:19,771 - RedisDG-coordinator - INFO - worker worker_graphite-2.nancy.
grid5000.fr_1482624969.27 is Falsevolunteer for 97_1 :  state_task = SUBMITTED
2016-12-25 02:22:07,379 - RedisDG-coordinator - INFO - Timeout for task 97_1
2016-12-25 02:22:07,380 - RedisDG-coordinator - INFO - no volunteer for task 97_1, restart
timer
2016-12-25 02:22:07,380 - RedisDG-coordinator - INFO - Coordinator publish task 97_1 to
workers
2016-12-25 02:22:07,382 - RedisDG-coordinator - INFO - worker worker_grimoire-7.nancy.
grid5000.fr_1482624969.4 is Falsevolunteer for 97_1 :  state_task = SUBMITTED
2016-12-25 02:22:07,383 - RedisDG-coordinator - INFO - worker worker_graphene-49.nancy.
grid5000.fr_1482624969.46 is Truevolunteer for 97_1 :  state_task = SUBMITTED
2016-12-25 02:22:07,383 - RedisDG-coordinator - INFO - worker worker_graphene-52.nancy.
grid5000.fr_1482624969.23 is Truevolunteer for 97_1 :  state_task = SUBMITTED
2016-12-25 02:22:07,383 - RedisDG-coordinator - INFO - worker worker_griffon-25.nancy.
grid5000.fr_1482624968.45 is Falsevolunteer for 97_1 :  state_task = SUBMITTED
2016-12-25 02:22:07,383 - RedisDG-coordinator - INFO - worker worker_graphite-2.nancy.
grid5000.fr_1482624969.27 is Falsevolunteer for 97_1 :  state_task = SUBMITTED
2016-12-25 02:23:12,419 - RedisDG-coordinator - INFO - Timeout for task 97_1
2016-12-25 02:23:12,420 - RedisDG-coordinator - INFO - volunteer for task 97_1, should submit
2016-12-25 02:23:12,420 - RedisDG-coordinator - INFO - Choosing a volunteer for 97_1
2016-12-25 02:23:12,420 - RedisDG-coordinator - INFO - volunteer worker_graphene-52.nancy.
grid5000.fr_1482624969.23 choosed for 97_1
```

Another interesting part of the griffon-25 logs to introduce is the following execution trace:

```
2016-12-25 01:23:34,849 - RedisDG-worker - INFO - Worker worker_griffon-25.nancy.grid5000.fr_
1482624968.45 is finishing task 17_1
2016-12-25 01:23:34,852 - RedisDG-worker - INFO - I m dispo?  False
2016-12-25 01:23:34,854 - RedisDG-worker - INFO - I m dispo?  False
2016-12-25 01:23:34,855 - RedisDG-worker - INFO - I m dispo?  False
2016-12-25 01:23:34,856 - RedisDG-worker - INFO - I m dispo?  False
2016-12-25 01:23:40,019 - RedisDG-worker - INFO - I m dispo?  False
2016-12-25 01:23:40,490 - RedisDG-worker - INFO - I m dispo?  False
2016-12-25 01:23:40,549 - RedisDG-worker - INFO - I m dispo?  False
```

```
2016-12-25 01:23:41,164 - RedisDG-worker - INFO - I m dispo?  False
2016-12-25 01:23:50,292 - RedisDG-worker - INFO - I m dispo?  False
2016-12-25 01:23:51,355 - RedisDG-worker - INFO - I m dispo?  False
2016-12-25 01:23:52,185 - RedisDG-worker - INFO - I m dispo?  False
2016-12-25 01:23:54,557 - RedisDG-worker - INFO - I m dispo?  False
2016-12-25 01:24:07,681 - RedisDG-worker - INFO - I m dispo?  False
2016-12-25 01:24:11,099 - RedisDG-worker - INFO - I m dispo?  False
2016-12-25 01:24:12,033 - RedisDG-worker - INFO - I m dispo?  False
2016-12-25 01:24:12,108 - RedisDG-worker - INFO - I m dispo?  False
2016-12-25 01:24:29,691 - RedisDG-worker - INFO - I m dispo?  False
2016-12-25 01:24:34,520 - RedisDG-worker - INFO - Task T162_1 has been selected for
['worker_graphite-2.nancy.grid5000.fr_1482624969.27'] (state task = GIVEN)
2016-12-25 01:24:38,270 - RedisDG-worker - INFO - I m dispo?  False
2016-12-25 01:24:38,782 - RedisDG-worker - INFO - Task T161_1 has been selected for
['worker_graphite-2.nancy.grid5000.fr_1482624969.27'] (state task = GIVEN)
2016-12-25 01:25:02,049 - RedisDG-worker - INFO - Task T160_1 has been selected for
['worker_graphite-2.nancy.grid5000.fr_1482624969.27'] (state task = GIVEN)
2016-12-25 01:25:10,695 - RedisDG-worker - INFO - Task T163_1 has been selected for
['worker_graphite-2.nancy.grid5000.fr_1482624969.27'] (state task = GIVEN)
2016-12-25 01:25:14,266 - RedisDG-worker - INFO - I m dispo?  False
2016-12-25 01:25:20,752 - RedisDG-worker - INFO - I m dispo?  False
2016-12-25 01:25:31,154 - RedisDG-worker - INFO - I m dispo?  False
2016-12-25 01:25:49,864 - RedisDG-worker - INFO - I m dispo?  False
2016-12-25 01:26:12,726 - RedisDG-worker - INFO - I m dispo?  False
2016-12-25 01:26:40,648 - RedisDG-worker - INFO - I m dispo?  False
2016-12-25 01:27:22,027 - RedisDG-worker - INFO - I m dispo?  False
2016-12-25 01:28:03,965 - RedisDG-worker - INFO - I m dispo?  True
2016-12-25 01:28:50,615 - RedisDG-worker - INFO - Task 18_1 has been selected for
['worker_graphene-49.nancy.grid5000.fr_1482624969.46'] (state task = GIVEN)
2016-12-25 01:28:56,427 - RedisDG-worker - INFO - I m dispo?  True
2016-12-25 01:28:56,429 - RedisDG-worker - INFO - I m dispo?  True
2016-12-25 01:28:56,430 - RedisDG-worker - INFO - I m dispo?  True
2016-12-25 01:28:56,431 - RedisDG-worker - INFO - I m dispo?  True
2016-12-25 01:29:02,130 - RedisDG-worker - INFO - Task T170_1 has been selected for
['worker_graphene52.nancy.grid5000.fr_1482624969.23'] (state task = GIVEN)
```

In this scenario, griffon-25 has just finished the execution of task 17_1. Then it replies several time that it is not available. We notice that the interval of time for sending new participating requests varies from 0 second to almost 19 seconds (01:25:49,864 - 01:25:31,154). This is due to the random time to sleep for the coordinator in case of unavailable workers. During this period of time tasks $T162_1, T161_1, T160_1, T163_1, 18_1$ and $T170_1$ are allocated on different workers (graphite-2. graphene-49 and graphene52) even when griffon-25 restarts to be be available again (see the message "I m dispo? True" at date 01:28:03,965). It is due to the fact that the winner, among available workers, is selected randomly.

5.4.4.2 Integration of a multicriteria approach for the selection of workers

Our proposition for implementing a multicriteria approach for selecting workers is also based on the abovementionned Heuristics 3, and is as follows.

Heuristics 3-ter: upon a task becomes independent, the coordinator publishes it to all the workers and starts a timer for λ time unit. All workers answer whatever their state ensuring thus the liveness and also publish if they are available or not. During λ, the coordinator saves all volunteer workers for a task. Upon the expiration of the timer (timeout), it chooses a worker among the set of available volunteers according to a Pareto front computation and based on the load, energy and fairness rewards as depict at the beginning of section 5.4. If no worker has answered or if all the workers that have answered are unavailable, the coordinator activates again the timer and it publishes again the task. Among the points on the Pareto front we choose the 'first' as the winner (others policies are also possible but

they are left for future works: random choice or according to the value of one criteria...).

This new heuristics is now part of the general algorithm that is depicted with Algorithm 8. This algorithm provides with more details about the implemention. A new class named Pareto has been added inside file `VolunteerSelectorStrategies.py` which is in charge of the different heuristics. The Pareto class inherits, from the VolunteerSelectorStrategy class, of the `getScore(worker_id)` function in charge of returning a list containing the following information: Fairness (the metric for controling the load balancing), Energy and CPU load for `worker_id`. We apply this function to all the available workers in order to build a matrix. This matrix is sent to the Pareto computation, returning the Pareto front. Then in the Pareto front we select one point. It remains to find the `worker_id` corresponding to this point in order to elect it as the worker in charge of the task's execution.

Algorithm 8: Details of the multi-criteria approach (concrete view)

1 Inputs: T (task to execute); S (set of N available/ready workers)
2 Output: $worker_{id}$ (the worker that will be used to execute the task T)
3 **for** *k:=1* **to** *N* **do**
4 | Matrix := Matrix + getScore($worker_k$)
5 **end**
6 Compute the Pareto front on Matrix
7 Select one point on the Pareto front
8 Retreive the $worker_{id}$ associated to this point
9 Elect $worker_{id}$ as the worker in charge of the execution of task T

Let us now introduce the logs of an execution of the ADAPT workflow according to the multicriteria approach. In this case we use 30 machines (1 for the coordinatoor and 29 workers) and 5 clusters of grid5000 (grisou, graphene, graphite, griffon, grimoire), all of them are located in the Nancy site. Note that this configuration[7] forms a very heterogeneous one.

We consider first the logs of the coordinator. We found the following trace:

```
2017-01-03 22:41:24,258 - RedisDG-coordinator - INFO - Choosing a volunteer for T11_1
2017-01-03 22:41:24,258 - RedisDG-coordinator - INFO - Scores of worker_grisou-17.nancy.
grid5000.fr_1483479622.02 is [1.0, 26.588, 4.8]
2017-01-03 22:41:24,259 - RedisDG-coordinator - INFO - Scores of worker_grisou-21.nancy.
grid5000.fr_1483479621.56 is [1.0, 22.772, 1.2]
2017-01-03 22:41:24,259 - RedisDG-coordinator - INFO - Scores of worker_graphene-18.nancy.
grid5000.fr_1483479622.4 is [1.0, 32.736000000000004, 10.6]
2017-01-03 22:41:24,259 - RedisDG-coordinator - INFO - Scores of worker_graphene-2.nancy.
grid5000.fr_1483479622.51 is [1.0, 67.343, 66.7]
2017-01-03 22:41:24,259 - RedisDG-coordinator - INFO - Scores for task T11_1 :


[[  1. 26.588   4.8  ]
 [  1.  22.772   1.2 ]
 [  1.  32.736  10.6 ]
 [  1.  67.343  66.7 ]]

'worker_grisou-17.nancy.grid5000.fr_1483479622.02':  [1.0, 26.588, 4.8],
'worker_grisou-21.nancy.grid5000.fr_1483479621.56':  [1.0, 22.772, 1.2],
'worker_graphene-2.nancy.grid5000.fr_1483479622.51':  [1.0, 67.343, 66.7],
'worker_graphene-18.nancy.grid5000.fr_1483479622.4':  [1.0, 32.736000000000004, 10.6]
2017-01-03 22:41:24,260 - RedisDG-coordinator - INFO - Scores after sort for task T11_1
:


[[  1.     26.588   4.8  ]
 [  1.     22.772   1.2  ]
 [  1.     32.736  10.6  ]
 [  1.     67.343  66.7  ]]
```

[7] https://www.grid5000.fr/mediawiki/index.php/Nancy:Hardware

```
2017-01-03 22:41:24,260 - RedisDG-coordinator - INFO - Pareto-Frontier-Multi Result for
task
T11_1 is :  [ 1.  26.588 4.8 ]
2017-01-03 22:41:24,260 - RedisDG-coordinator - INFO - volunteer worker_grisou-17.nancy.
grid5000.fr_1483479622.02 choosed for T11_1
2017-01-03 22:41:24,260 - RedisDG-coordinator - INFO - select the volunteer worker
worker_grisou-17.nancy.grid5000.fr_1483479622.02 for task T11_1 :  state_task = GIVEN
```

The task requesting to be executed is task $T11_1$ and, at the time we need to execute it, only 4 workers are available. We notice that Pareto computation leads to the selection of `grisou-17` as the machine node that will execute task $T11_1$. In fact, two points are located on the Pareto front: $(1., 26.588, 4.8)$ and $(1., 22.772, 1.2)$. Our implementation selects the first one in the list.

We can also consider another part of the execution trace:

```
2017-01-03 22:42:00,350 - RedisDG-coordinator - INFO - Choosing a volunteer for 4_1
2017-01-03 22:42:00,350 - RedisDG-coordinator - INFO - Scores of worker_grimoire-3.
nancy.grid5000.fr_1483479622.42 is [0.0, 0.0, 0.0]
2017-01-03 22:42:00,350 - RedisDG-coordinator - INFO - Scores of worker_grisou-39.
nancy.grid5000.fr_1483479622.0 is [1.0, 22.348, 0.8]
2017-01-03 22:42:00,350 - RedisDG-coordinator - INFO - Scores of worker_graphene-22.
nancy.grid5000.fr_1483479622.26 is [0.0, 0.0, 0.0]
2017-01-03 22:42:00,350 - RedisDG-coordinator - INFO - Scores of worker_graphene-2.
nancy.grid5000.fr_1483479622.51 is [2.0, 98.489, 9.1]
2017-01-03 22:42:00,350 - RedisDG-coordinator - INFO - Scores of worker_graphene-18.
nancy.grid5000.fr_1483479622.4 is [2.0, 63.776, 9.0]
2017-01-03 22:42:00,351 - RedisDG-coordinator - INFO - Scores of worker_graphene-135.
nancy.grid5000.fr_1483479622.85 is [0.0, 0.0, 0.0]
2017-01-03 22:42:00,351 - RedisDG-coordinator - INFO - Scores of worker_graphene-137.
nancy.grid5000.fr_1483479623.0 is [1.0, 67.343, 66.7]
2017-01-03 22:42:00,351 - RedisDG-coordinator - INFO - Scores of worker_graphene-140.
nancy.grid5000.fr_1483479622.59 is [0.0, 0.0, 0.0]
2017-01-03 22:42:00,351 - RedisDG-coordinator - INFO - Scores of worker_graphene-19.
nancy.grid5000.fr_1483479622.78 is [1.0, 29.98, 8.0]
2017-01-03 22:42:00,351 - RedisDG-coordinator - INFO - Scores of worker_graphene-20.
nancy.grid5000.fr_1483479622.71 is [1.0, 30.298000000000002, 8.3]
2017-01-03 22:42:00,351 - RedisDG-coordinator - INFO - Scores of worker_graphene-142.
nancy.grid5000.fr_1483479623.14 is [1.0, 30.298000000000002, 8.3]
2017-01-03 22:42:00,351 - RedisDG-coordinator - INFO - Scores of worker_grimoire-7.
nancy.grid5000.fr_1483479622.59 is [0.0, 0.0, 0.0]
2017-01-03 22:42:00,352 - RedisDG-coordinator - INFO - Scores for task 4_1 :


[[  0.      0.      0.   ]
 [  1.     22.348   0.8  ]
 [  0.      0.      0.   ]
 [  2.     98.489   9.1  ]
 [  2.     63.776   9.   ]
 [  0.      0.      0.   ]
 [  1.     67.343  66.7  ]
 [  0.      0.      0.   ]
 [  1.     29.98    8.   ]
 [  1.     30.298   8.3  ]
 [  1.     30.298   8.3  ]
 [  0.      0.      0.   ]]

...
...
```

```
...
2017-01-03 22:42:00,353 - RedisDG-coordinator - INFO - Pareto-Frontier-Multi Result for
task 4_1 is :  [ 0.  0.  0.]
2017-01-03 22:42:00,353 - RedisDG-coordinator - INFO - volunteer worker_grimoire-3.
nancy.grid5000.fr_1483479622.42 choosed for 4_1
2017-01-03 22:42:00,353 - RedisDG-coordinator - INFO - select the volunteer
worker worker_grimoire-3.nancy.grid5000.fr_1483479622.42 for task 4_1 :  state_task = GIVEN
```

In this case we notice that at the time we need to compute task $T4_1$, 12 workers are available, and 5 of them have not yet be used (the score is 0), resulting in the choice of grimoire-3 as the worker for task $T4_1$. Note that the Pareto front contains 5 points and we have selected the first one in the list of scores.

At last we have the following trace:

```
2017-01-03 22:43:37,989 - RedisDG-coordinator - INFO - volunteer for task 8_1, should submit
2017-01-03 22:43:37,989 - RedisDG-coordinator - INFO - Choosing a volunteer for 8_1
2017-01-03 22:43:37,990 - RedisDG-coordinator - INFO - Scores of worker_grisou-39.
nancy.grid5000.fr_1483479622.0 is [2.0, 52.646, 8.3]
2017-01-03 22:43:37,990 - RedisDG-coordinator - INFO - Scores of worker_grimoire-3.
nancy.grid5000.fr_1483479622.42 is [1.0, 30.616, 8.6]
2017-01-03 22:43:37,990 - RedisDG-coordinator - INFO - Scores of worker_graphene-17.
nancy.grid5000.fr_1483479623.03 is [1.0, 31.57, 9.5]
2017-01-03 22:43:37,990 - RedisDG-coordinator - INFO - Scores of worker_graphene-22.
nancy.grid5000.fr_1483479622.26 is [1.0, 30.509999999999998, 8.5]
2017-01-03 22:43:37,990 - RedisDG-coordinator - INFO - Scores of worker_graphene-19.
nancy.grid5000.fr_1483479622.78 is [1.0, 29.98, 8.0]
2017-01-03 22:43:37,990 - RedisDG-coordinator - INFO - Scores of worker_graphene-20.
nancy.grid5000.fr_1483479622.71 is [1.0, 30.298000000000002, 8.3]
2017-01-03 22:43:37,990 - RedisDG-coordinator - INFO - Scores of worker_graphene-137.
nancy.grid5000.fr_1483479623.0 is [1.0, 67.343, 66.7]
2017-01-03 22:43:37,990 - RedisDG-coordinator - INFO - Scores of worker_graphene-142.
nancy.grid5000.fr_1483479623.14 is [1.0, 30.298000000000002, 8.3]
2017-01-03 22:43:37,990 - RedisDG-coordinator - INFO - Scores of worker_graphene-140.
nancy.grid5000.fr_1483479622.59 is [1.0, 31.04, 9.0]
2017-01-03 22:43:37,990 - RedisDG-coordinator - INFO - Scores of worker_graphene-24.
nancy.grid5000.fr_1483479622.97 is [1.0, 30.828000000000003, 8.8]
2017-01-03 22:43:37,990 - RedisDG-coordinator - INFO - Scores of worker_graphene-105.
nancy.grid5000.fr_1483479623.25 is [2.0, 97.71799999999999, 8.4]
2017-01-03 22:43:37,990 - RedisDG-coordinator - INFO - Scores of worker_graphene-143.
nancy.grid5000.fr_1483479623.0 is [2.0, 62.397999999999996, 8.9]
2017-01-03 22:43:37,991 - RedisDG-coordinator - INFO - Scores of worker_graphene-135.
nancy.grid5000.fr_1483479622.85 is [1.0, 30.828000000000003, 8.8]
2017-01-03 22:43:37,991 - RedisDG-coordinator - INFO - Scores of worker_graphene-21.
nancy.grid5000.fr_1483479622.55 is [1.0, 31.782, 9.7]
2017-01-03 22:43:37,991 - RedisDG-coordinator - INFO - Scores of worker_graphene-141.
nancy.grid5000.fr_1483479622.95 is [1.0, 31.252, 9.2]
2017-01-03 22:43:37,991 - RedisDG-coordinator - INFO - Scores of worker_graphene-138.
nancy.grid5000.fr_1483479622.62 is [1.0, 67.517, 67.3]
2017-01-03 22:43:37,991 - RedisDG-coordinator - INFO - Scores of worker_graphene-139.
nancy.grid5000.fr_1483479622.81 is [1.0, 31.57, 9.5]
2017-01-03 22:43:37,991 - RedisDG-coordinator - INFO - Scores of worker_graphene-16.
nancy.grid5000.fr_1483479622.96 is [1.0, 31.57, 9.5]
2017-01-03 22:43:37,991 - RedisDG-coordinator - INFO - Scores of worker_graphite-3.
nancy.grid5000.fr_1483479620.93 is [2.0, 45.862, 1.4]
2017-01-03 22:43:37,991 - RedisDG-coordinator - INFO - Scores of worker_griffon-8.
nancy.grid5000.fr_1483479622.34 is [1.0, 27.542, 5.7]
2017-01-03 22:43:37,991 - RedisDG-coordinator - INFO - Scores of worker_grimoire-7.
nancy.grid5000.fr_1483479622.59 is [1.0, 22.666, 1.1]
```

```
2017-01-03 22:43:37,992 - RedisDG-coordinator - INFO - Scores for task 8_1 :

[[  2. 52.646 8.3  ]
 [  1.     30.616   8.6  ]
 [  1.     31.57    9.5  ]
 [  1.     30.51    8.5  ]
 [  1.     29.98    8.   ]
 [  1.     30.298   8.3  ]
 [  1.     67.343  66.7  ]
 [  1.     30.298   8.3  ]
 [  1.     31.04    9.   ]
 [  1.     30.828   8.8  ]
 [  2.     97.718   8.4  ]
 [  2.     62.398   8.9  ]
 [  1.     30.828   8.8  ]
 [  1.     31.782   9.7  ]
 [  1.     31.252   9.2  ]
 [  1.     67.517  67.3  ]
 [  1.     31.57    9.5  ]
 [  1.     31.57    9.5  ]
 [  2.     45.862   1.4  ]
 [  1.     27.542   5.7  ]
 [  1.     22.666   1.1  ]]

...
...
...
2017-01-03 22:43:37,994 - RedisDG-coordinator - INFO - Pareto-Frontier-Multi Result
for task 8_1 is :  [ 1.  22.666 1.1 ]
2017-01-03 22:43:37,994 - RedisDG-coordinator - INFO - volunteer worker_grimoire-7.
nancy.grid5000.fr_1483479622.59 choosed for 8_1
2017-01-03 22:43:37,995 - RedisDG-coordinator - INFO - select the volunteer worker
worker_grimoire-7.nancy.grid5000.fr_1483479622.59 for task 8_1 :  state_task = GIVEN
```

In this case we notice that 21 workers are available for task $T8\_1$, resulting in the selection of grimoire-7 as the worker for executing this task.

To conclude this experimental part, we can say that all these examples illustrate the different situations in which our new RedisDG system can be faced to. They also illustrate how our measures and solutions can be fine. We could even envision a post-mortem analysis tool to fully exploit the logs.

5.4.4.3 ADAPT workflow results

In our ADAPT workflow, we have integrated a transport equation coupled with a poisson equation. Figure 5.3 depicts the 2D transport in domain [0 1][0 0.5] that we can draw from the results of decomposing our code as a service.

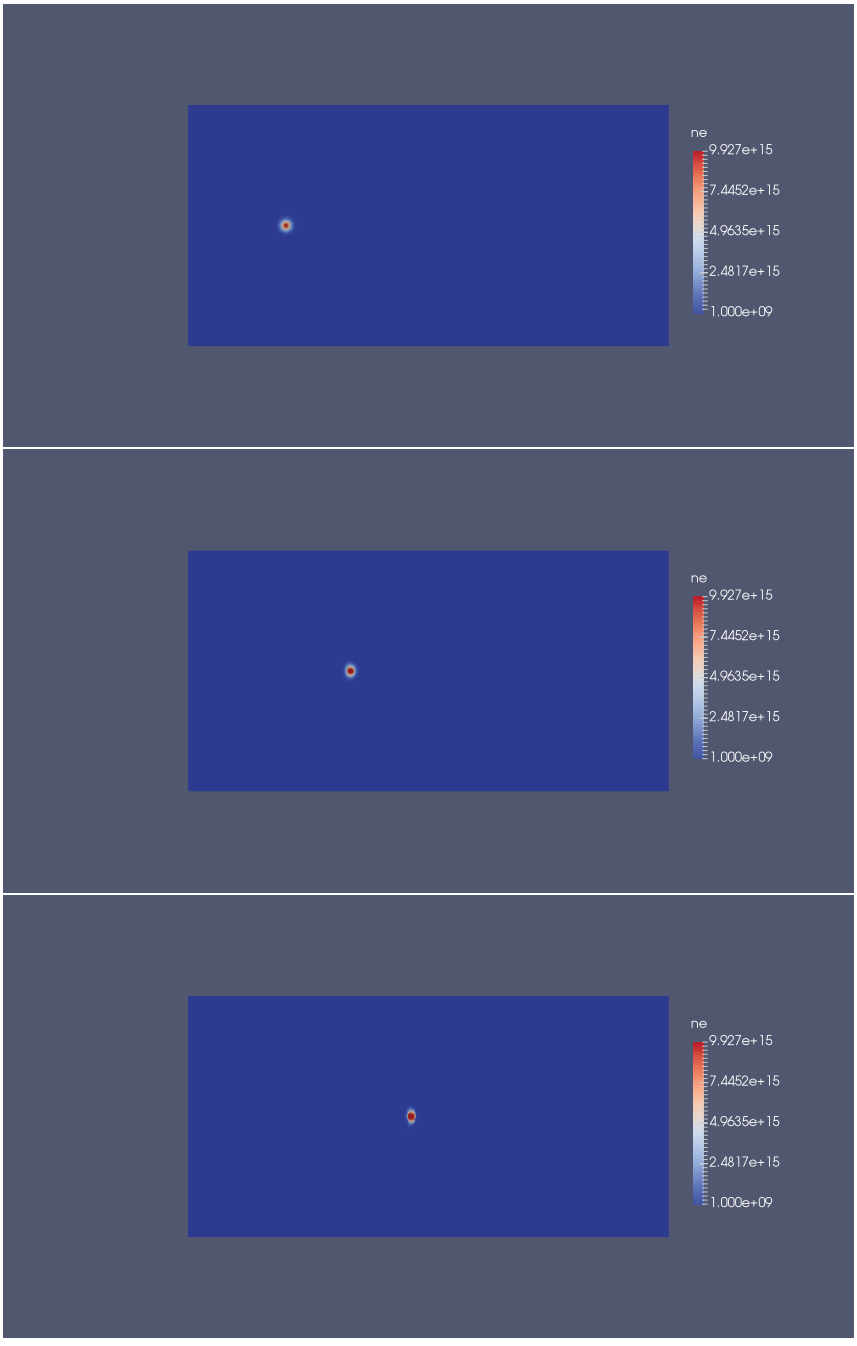


Figure 5.3: Results of ADAPT worfklow

Chapter 6

Conclusion

6.1 Summary of our results

The main objective of this work was to improve the performance (computation time and accuracy) of ADAPT platform. We present the parallelization of the convection-diffusion equation coupled with the linear system using MPI, and execute the ADAPT graph of tasks on a grid/cloud computing testbed using RedisDG.

Massive parallel computer systems are necessary in many scientific fields. A lot of computational science applications are executed on clusters or supercomputers to provide results on many different topics such as weather modeling, life sciences, etc. Moreover, researchers are investigating various related fields where the behavior of parallel computer systems is studied and new programming models to take into account the next generations of supercomputers are developed.

The difficulty of achieving a good scale during the execution of some parallel applications while we increase the number of the processors is known and depends on many factors. Performance evaluation can provide an insight about the performance of an application and indicate possible bottlenecks. The first focus was to accelerate calculations of ADAPT platform which includes 2D and 3D codes of streamer equations. We used to do this multiple processors, it's the parallelism. We tested our solutions using the MAGI cluster at Paris 13. This architecture has been the focus of our study of the contribution of parallelism using the distributed memory parallelization. To this parallel architecture, a different partitioning strategy, communication and task synchronization has been established.

The full workflow is realized in (1) using external tools such as METIS for mesh partitioning in order to enable the computational load balancing for the global assembly, (2) using MPI to realize the communication patterns, (3) using MUMPS to solve the system of linear equations. The current approach with MUMPS shows significant advantages in terms of avoiding the problems of preconditioners, and doesn't need a lot of iterations to converge to the solution. In summary, the most important advantages of the presented scheme are:

- Efficient memory usage distribution over computer nodes;
- Good scalability to solve the convection-diffusion equation;
- Stable, fast and good scalability of linear solvers provided by MUMPS.

As second focus, in the section entitled *HPC as a Service* we have built a case study to see the ADAPT application as a service for a scientific workflow engine (RedisDG). The basic idea is to admit that excellent, in terms of performance, digital libraries are now available because they have been optimized for many hardware architectures. The problem is not so much to try again to improve performance but rather, in our opinion, to exploit all the resources (calculations, networks, disks) available on the planet and/or in major cloud providers and for everyone. In this context, it is necessary to allow all scientists to access these resources simply by forgetting almost all the techniques that the HPC universe has developed over the last thirty years.

Cloud computing provides a framework for facilitating access to computing resources. The formalism of scientific workflows also provides a benign framework for describing calculations from well-identified basics. We have decomposed ADAPT into "great services" and using well-known tools, among these METIS and MUMPS.

Our work in the context of scientific workflows continued in the context of the placement of these major services on cloud-based architectures. We used the RedisDG [105] workflow engine, deploying it, not in a cloud as it has already been studied, but in the Grid5000 national testbed to enrich it in terms of capturing the dynamicity of systems and also the cloud's mismanagement that usually accentuates problems related to waste of resources [106], [107]. Indeed, we know [44] that resource utilization in computer systems is usually below 10% due primarily but not only to resource over-provisioning. Efficient resource sharing in cloud is, as a consequence, challenging. Resource under-provisioning will inevitably scale-down performance and resource over-provisioning can result in idle time, thereby incurring unnecessary costs for the user.

For that later objective we have incorporated a multi-criterion based selection of the nodes. The RedisDG framework now theoretically guarantees the adoption of a better set of placements for tasks of the workflow. More specifically, it seeks the non-dominated solutions (Pareto's Dominance concept). In contrast to existing solutions that address specific objectives, our framework was devised to support many types of objectives and to be easily extensible, which enables the implementation of new and generic prioritised elements.

In fact we have considered ADAPT on a platform where the calculation nodes can arrive and leave at any time by publishing announcements of subscriptions or unsubscribing to a calculation service. This framework, called volunteer cloud by C. Cérin et al., Corresponds either to the possibility, for machines on the Internet, to participate in calculations or to machines in a data center that leaves a calculation to go to help (Peak activity) another calculation. We experimented with RedisDG in this innovative way via an effective software solution. The RedisDG system is now enriched with this feature, which makes it unique.

Concerning the multi-criteria selection of the nodes for the execution of the tasks of a workflow, and as said previously, we have simply implemented a Pareto criterion which is based concretely on values and feedback of the RedisDG system before our work. The design and integration of this selection method is not straightforward because, remember, the whole RedisDG protocol is based on the publication-subscription interaction mechanism. It is not customary in the community of scientific workflows and HPC generally to use this protocol to solve problems, including the allocation problem. It is therefore another originality of our work. Let us note in passing that the mechanism of publication underwriting is the key point for the management of the dynamicity.

The OpenAlea system [108, 109] is a workflow system based on λ-dataflow which is able to uniformly deal with classical data analysis, visualization, modeling and simulation tasks. λ-dataflow means that the system uses higher-order constructs in the context of dataflows theory and thus allows to represent control flow using algebraic operators (e.g., condition- als, map/reduce...). One of the major originality of OpenAlea lies in the way iteration is handled by introducing a specific kind of actor, called dataflow variable X. It allows to specify that, at a given port, an actor receives an unbound variable rather than a value.

6.2 Future works

This thesis proposed several effective solutions to reduce the calculation time for the streamer code in 2D as well as in 3D. The work of this thesis can be the base for the next version of our framework ADAPT.

A new computation model could be introduced to predict more accurately the performance as the power rate of a real execution usually is not stable. Our framework can support more MPI calls, hybrid Parallelism using both MPI and OpenMP. An another approach is to consider GPUs, and Intel Xeon Phi accelerators.

As far as computing time is concerned, our tests were performed symbolically, we measured only the calculation time. It would be interesting to add measurements of memory consumption to verify that the large scaling does not cause a significant increase in memory consumption, and therefore additional cost in computation time. The resolution of the linear system was carried out using the MUMPS solver based on MPI, the ideal is to use a hybrid solver such as PaStix [110], MaPHyS [111], etc.

We plan also to investigate in depth the mesh partitioning problems and to conduct experiments with specific tools. We may target PAMPA [112] and the question is how such tools can impact the overall performance of ADAPT software. The challenge is to mix mesh partitioning and dynamic mesh adaptations. How to manage load balancing in

this context? When dealing with three dimensional problems, the question is even more critical.

On the other hand, we can consider working with other scientific workflows and with our new version of RedisDG. The NASA MONTAGE workflow has been studied in Leila Abidi's thesis at LIPN and presents challenges in terms of data exchange that we do not have as part of the ADAPT workflow. Specific work on effective data management for RedisDG is already underway in the LIPN group and will result in a communication for The 6th IEEE International Symposium on Cloud and Service Computing. On the other hand this work does not take place in dynamic context and does not use multicriteria techniques. We could take advantage of this study to really get to scale, that is, to study our system as part of hundreds of thousands of jobs. The question then is who is responsible for managing the breakdown problems. Our workflow engine or the cloud? For now, RedisDG is not designed with fault-tolerance mechanisms. This is an interesting way of working.

Another opportunity for future work is given at the direct acyclic graph (DAG) level that is the model for computation. In our work, we assume, implicitly, that each node corresponds to a sequential task. The parallelism is depicted by parallel edges started from the same node. We could imagine that a node corresponds to a parallel execution, for instance a parallel version of METIS or MUMPS for instance. At present time we capture this possibility in executing a threaded version of METIS or MUMPS on a single node, meaning that only one worker is executing the task. A distributed version of METIS or MUMPS, meaning that more than one worker is executing the distributed version METIS or MUMPS is a challenging question.

One important issue is to describe a collaborative system able to execute the DAG, more precisely a hierarchical DAG because a node can be unfolded. In this vein, we can imagine to unfold a node, dynamically, if we have enough resource, or we could request, on demand, new resources. At an abstract level, such systems are usually modeled with centralized and state-based formalisms like automata, Petri nets or statecharts. They can also directly be specified with dedicated notations like BPEL [113] or BPMN [114]. In this context we will surely also need the contributions from the fields of lazy evaluation, abstract grammar and context-free languages as well as from software engineering in a broad sense.

Note that, in this later context, the OpenAlea system [108, 109] is a workflow system based on λ-dataflow which is able to uniformly deal with classical data analysis, visualization, modeling and simulation tasks. λ-dataflow means that the system uses higher-order constructs in the context of dataflows theory and thus allows to represent control flow using algebraic operators (e.g., condition- als, map/reduce...). One of the major originality of OpenAlea lies in the way iteration is handled by introducing a specific kind of actor, called dataflow variable X. It allows to specify that, at a given port, an actor receives an unbound variable rather than a value.

An actor in OpenAlea is an elementary brick (a.k.a. component or activity) that has a name, a function object (a functor, a program, a web service, a micro service or a composite actor), and explicitly defined input and output ports. A semantic type is associated to each port (with a corresponding color). A workflow is represented as a directed multi-graph where nodes are actors, and directed edges are data links between output and input ports. A workflow can become a (composite) actor in another workflow to allow composition. In another word, one of the main originality of OpenAleat is to introduce higher-order dataflows as a means to uniformly combine classical data analysis with modeling and simulation.

At last we would like to mention a specific future work related to the placement of tasks. The 'new' RedisDG framework now implement strategies that build non-dominated scenarios sets (using the Pareto Dominance concept). In this case, a non-dominated scenario is better in at least one objective and, at the same time, not worse in any other objective, compared to a base scenario. Despite the efficiency, this strategy stagnates in environments with many objectives, i.e., possibly reaching a state where none result is good enough for all objectives simultaneously. Moreover, the selection of the best scenario for the deployment is also a challenge due the limited evaluation's strategies, their execution time and cost. We may point out that deploying a task inside Google cloud is maybe more or less costly than deploying inside the Amazon cloud.

This problem relates to the question that none of the existing solutions consider, at the same time and to the best of our knowledge, the important cloud's characteristics: SLO (Service Level Objectives), SLA (Service Level Agreement), policies and costs. Remind that a service level objective (SLO) is a key element of a service level agreement (SLA) between a service provider and a customer. SLOs are agreed as a means of measuring the performance of the Service Provider and are outlined as a way of avoiding disputes between the two parties based on misunderstanding. Even if we focus only on the cost objective we may notice that our current solution may provide a task placement of two tasks on the same computing node... but these tasks represent a duplicated service that absolutely should reside

on two distinct computing nodes. To address more in deep all of these challenges, we propose to start with the works of Guilherme Arthur Geronimo, Rafael Brundo Uriarte and Carlos Becker Westphall in [115] and [116] because these works propose a combinaison of generic approaches to mix SLO, SLA, policies and costs constraints for the problem of the placement of Virtual Machines (VM) in a cloud platform.

Appendices

Appendix A

A.1 Relations Used in Reconstruction

Relations coming from the least square method which are used in 2D reconstruction

$$\begin{aligned} &I_{xx} = \sum_{j=1}^{m}(x_j - x_i)^2, \quad I_{yy} = \sum_{j=1}^{m}(y_j - y_i)^2, \\ &I_{xy} = \sum_{j=1}^{m}(x_j - x_i)(y_j - y_i), \quad J_x = \sum_{j=1}^{m}(n_{e_j} - n_{e_i})(x_j - x_i), \\ &J_y = \sum_{j=1}^{m}(n_{e_j} - n_{e_i})(y_j - y_i), \quad D = I_{xx}.I_{yy} - Ixy^2. \end{aligned} \tag{A.1}$$

Relations coming for the least square method which are used in 3D reconstruction

$$\begin{aligned} &I_{xx} = \sum_{j=1}^{m}(x_j - x_i)^2, \quad I_{yy} = \sum_{j=1}^{m}(y_j - y_i)^2, \\ &I_{xy} = \sum_{j=1}^{m}(x_j - x_i)(y_j - y_i), \quad I_{xz} = \sum_{j=1}^{m}(x_j - x_i)(z_j - z_i), \\ &I_{zz} = \sum_{j=1}^{m}(z_j - z_i)^2, \quad I_{yz} = \sum_{j=1}^{m}(y_j - y_i)(z_j - z_i), \\ &J_x = \sum_{j=1}^{m}(n_{e_j} - n_{e_i})(x_j - x_i), \quad J_y = \sum_{j=1}^{m}(n_{e_j} - n_{e_i})(y_j - y_i), \\ &J_z = \sum_{j=1}^{m}(n_{e_j} - n_{e_i})(z_j - z_i), \\ &D = I_{xx}I_{yy}I_{zz} + 2I_{xy}I_{xz}I_{yz} - I_{xx}I_{yz}^2 - I_{yy}I_{xz}^2 - I_{zz}I_{xy}^2. \end{aligned} \tag{A.2}$$

A.2 Relations for Dissipative Terms

Weights parameters for the 2D gradient computation in diffusion terms:

$$\begin{aligned} &R_x = \sum_{j=1}^{N(A)}(x_P - x_A), \quad R_x = \sum_{j=1}^{N(A)}(y_P - y_A), \\ &I_{xx} = \sum_{j=1}^{N(A)}(x_P - x_A)^2, \quad I_{yy} = \sum_{j=1}^{N(A)}(y_P - y_A)^2, \\ &I_{xy} = \sum_{j=1}^{N(A)}(x_P - x_A)(y_P - y_A). \\ &\lambda_x = \frac{I_{xy}R_y - I_{yy}R_x}{D}, \quad \lambda_x = \frac{I_{xy}R_y - I_{yy}R_x}{D}, D = I_{xx}I_{yy} - I_{xy}^2. \end{aligned} \tag{A.3}$$

3D computation of the gradient in the diffusion terms is given by following relation

$$\begin{aligned}\overrightarrow{\nabla} n_{e_{ij}} = \frac{1}{3\mu(D_{\sigma_{ij}})}\Big[&(\overrightarrow{n}_{ABR}|\sigma_{ABR}| + \overrightarrow{n}_{DAR}|\sigma_{DAR}| + \overrightarrow{n}_{BAL}|\sigma_{BAL}| + \overrightarrow{n}_{ADL}|\sigma_{ADL}|)n_e(A)+\\ &(\overrightarrow{n}_{ABR}|\sigma_{ABR}| + \overrightarrow{n}_{BCR}|\sigma_{BCR}| + \overrightarrow{n}_{BAL}|\sigma_{BAL}| + \overrightarrow{n}_{CBL}|\sigma_{CBL}|)n_e(B)+\\ &(\overrightarrow{n}_{BCR}|\sigma_{BCR}| + \overrightarrow{n}_{CDR}|\sigma_{CDR}| + \overrightarrow{n}_{CBL}|\sigma_{CBL}| + \overrightarrow{n}_{DCL}|\sigma_{DCL}|)n_e(C)+\\ &(\overrightarrow{n}_{CDR}|\sigma_{CDR}| + \overrightarrow{n}_{DAR}|\sigma_{DAR}| + \overrightarrow{n}_{DCL}|\sigma_{DCL}| + \overrightarrow{n}_{ADL}|\sigma_{ADL}|)n_e(D)+\\ &(\overrightarrow{n}_{ABR}|\sigma_{ABR}| + \overrightarrow{n}_{BCR}|\sigma_{BCR}| + \overrightarrow{n}_{CDR}|\sigma_{CDR}| + \overrightarrow{n}_{DAR}|\sigma_{DAR}|)n_e(R)+\\ &(\overrightarrow{n}_{BAL}|\sigma_{BAL}| + \overrightarrow{n}_{CBL}|\sigma_{CBL}| + \overrightarrow{n}_{DCL}|\sigma_{DCL}| + \overrightarrow{n}_{ADL}|\sigma_{ADL}|)n_e(L)\Big]\end{aligned} \tag{A.4}$$

The vector equilibrium implies

$$\begin{aligned}&\overrightarrow{n}_{ABR}|\sigma_{ABR}| + \overrightarrow{n}_{DAR}|\sigma_{DAR}| + \overrightarrow{n}_{BAL}|\sigma_{BAL}| + \overrightarrow{n}_{ADL}|\sigma_{ADL}| = \overrightarrow{n}_{BRDL}|\sigma_{BRDL}|\\ &\overrightarrow{n}_{ABR}|\sigma_{ABR}| + \overrightarrow{n}_{BCR}|\sigma_{BCR}| + \overrightarrow{n}_{BAL}|\sigma_{BAL}| + \overrightarrow{n}_{CBL}|\sigma_{CBL}| = \overrightarrow{n}_{ALCR}|\sigma_{ALCR}|\\ &\overrightarrow{n}_{BCR}|\sigma_{BCR}| + \overrightarrow{n}_{CDR}|\sigma_{CDR}| + \overrightarrow{n}_{CBL}|\sigma_{CBL}| + \overrightarrow{n}_{DCL}|\sigma_{DCL}| = -\overrightarrow{n}_{RBDL}|\sigma_{RBDL}|\\ &\overrightarrow{n}_{CDR}|\sigma_{CDR}| + \overrightarrow{n}_{DAR}|\sigma_{DAR}| + \overrightarrow{n}_{DCL}|\sigma_{DCL}| + \overrightarrow{n}_{ADL}|\sigma_{ADL}| = -\overrightarrow{n}_{ALCR}|\sigma_{ALCR}|\\ &\overrightarrow{n}_{ABR}|\sigma_{ABR}| + \overrightarrow{n}_{BCR}|\sigma_{BCR}| + \overrightarrow{n}_{CDR}|\sigma_{CDR}| + \overrightarrow{n}_{DAR}|\sigma_{DAR}| = \overrightarrow{n}_{ij}|\sigma_{ij}|\\ &\overrightarrow{n}_{BAL}|\sigma_{BAL}| + \overrightarrow{n}_{CBL}|\sigma_{CBL}| + \overrightarrow{n}_{DCL}|\sigma_{DCL}| + \overrightarrow{n}_{ADL}|\sigma_{ADL}| = -\overrightarrow{n}_{ij}|\sigma_{ij}|\end{aligned} \tag{A.5}$$

When we substitute the equations (A.5) in the equation (A.4), we can write for the electron density gradient $\overrightarrow{\nabla} n_{e_{ij}}$

$$\begin{aligned}\overrightarrow{\nabla} n_{e_{ij}} = \frac{1}{3\mu(D_{\sigma_{ij}})}\Big[&(n_e(A) - n_e(C))\overrightarrow{n}_{BRDL}|\sigma_{BRDL}|\\ &+(n_e(B) - n_e(D))\overrightarrow{n}_{ALCR}|\sigma_{ALCR}| + (n_e(R) - n_e(L))\overrightarrow{n}_{ij}|\sigma_{ij}|\Big]\end{aligned} \tag{A.6}$$

Parameters in the weights for gradient computation are given by following relations

$$\begin{aligned}&R_x = \textstyle\sum_{j=1}^{N(A)}(x_P - x_A), \quad R_x = \sum_{j=1}^{N(A)}(y_P - y_A), \quad R_z = \sum_{j=1}^{N(A)}(z_P - z_A),\\ &I_{xx} = \textstyle\sum_{j=1}^{N(A)}(x_P - x_A)^2, \quad I_{yy} = \sum_{j=1}^{N(A)}(y_P - y_A)^2, \quad I_{zz} = \sum_{j=1}^{N(A)}(z_P - z_A)^2,\\ &I_{xy} = \textstyle\sum_{j=1}^{N(A)}(x_P - x_A)(y_P - y_A), \quad I_{xz} = \sum_{j=1}^{N(A)}(x_P - x_A)(z_P - z_A),\\ &I_{yz} = \textstyle\sum_{j=1}^{N(A)}(y_P - y_A)(z_P - z_A),\\ &D = I_{xx}I_{yy}I_{zz} + 2I_{xy}I_{xz}I_{yz} - I_{xx}I_{yz}^2 - I_{yy}I_{xz}^2 - I_{zz}I_{xy}^2,\\ &\lambda_x = \tfrac{(I_{xz}^2 - I_{yy}I_{zz})R_x + (I_{xy}I_{zz} - I_{xz}I_{yz})R_y + (I_{xz}I_{yy} - I_{xy}I_{yz})R_z}{D}\\ &\lambda_y = \tfrac{(I_{xy}I_{zz} - I_{xz}I_{yz})R_x + (I_{xz}^2 - I_{xx}I_{zz})R_y + (I_{yz}I_{xx} - I_{xz}I_{xy})R_z}{D}\\ &\lambda_z = \tfrac{(I_{xz}I_{yy} - I_{xy}I_{yz})R_x + (I_{yz}I_{xx} - I_{xz}I_{xy})R_y + (I_{xy}^2 - I_{xx}I_{yy})R_x}{D}\end{aligned} \tag{A.7}$$

Bibliography

[1] Franck Assous, J. Segré, and Eric Sonnendrücker. A domain decomposition method for the parallelization of a three-dimensional maxwell solver based on a constrained formulation. *Mathematics and Computers in Simulation*, 81(11):2371–2388, 2011.

[2] Francesco Vecil, José M. Mantas, María J. Cáceres, Carlos Sampedro, Andrés Godoy, and Francisco Gámiz. A parallel deterministic solver for the schrodinger-poisson-boltzmann system in ultra-short dg-mosfets: Comparison with monte-carlo. *Computers & Mathematics with Applications*, 67(9):1703 – 1721, 2014.

[3] Yvan Notay and Artem Napov. A massively parallel solver for discrete poisson-like problems. *Journal of Computational Physics*, 281(0):237 – 250, 2015.

[4] Pawan Kumar, Stefano Markidis, Giovanni Lapenta, Karl Meerbergen, and Dirk Roose. High performance solvers for implicit particle in cell simulation. *Procedia Computer Science*, 18:2251–2258, 2013.

[5] Youcef Saad and Martin H. Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.

[6] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley. An updated set of basic linear algebra subprograms (blas). *ACM Transactions on Mathematical Software*, 28:135–151, 2001.

[7] Ronald F. Boisvert and Jack Dongarra. Preface to the special issue on the basic linear algebra subprograms (BLAS). *ACM Trans. Math. Softw.*, 28(2):133–134, 2002.

[8] David B Skillicorn. A taxonomy for computer architectures. *Computer*, 21(11):46–57, 1988.

[9] Gene M Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485. ACM, 1967.

[10] John L Gustafson. Reevaluating amdahl's law. *Communications of the ACM*, 31(5):532–533, 1988.

[11] X-H Sun and Lionel M Ni. Another view on parallel speedup. In *Supercomputing'90., Proceedings of*, pages 324–333. IEEE, 1990.

[12] Dean M Tullsen, Susan J Eggers, and Henry M Levy. Simultaneous multithreading: maximizing on-chip parallelism. In *25 years of the international symposia on Computer architecture (selected papers)*, pages 533–544. ACM, 1998.

[13] Jack L Lo, Joel S Emer, Henry M Levy, Rebecca L Stamm, Dean M Tullsen, and Susan J Eggers. Converting thread-level parallelism to instruction-level parallelism via simultaneous multithreading. *ACM Transactions on Computer Systems (TOCS)*, 15(3):322–354, 1997.

[14] Bruce Hendrickson and Robert Leland. The chaco user's guide: Version 2.0. Technical report, Technical Report SAND95-2344, Sandia National Laboratories, 1995.

[15] François Pellegrini and Jean Roman. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *International Conference on High-Performance Computing and Networking*, pages 493–498. Springer, 1996.

[16] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, December 1998.

[17] George Karypis and Vipin Kumar. Multilevel graph partitioning schemes. In *ICPP (3)*, pages 113–122, 1995.

[18] George Karypis and Vipin Kumar. Multilevelk-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed computing*, 48(1):96–129, 1998.

[19] George Karypis and Vipin Kumar. Analysis of multilevel graph partitioning. In *Proceedings of the 1995 ACM/IEEE conference on Supercomputing*, page 29. ACM, 1995.

[20] Kyle A Gallivan, Michael Heath, Esmond Ng, B Peyton, Robert Plemmons, C Romine, A Sameh, and R Voigt. *Parallel algorithms for matrix computations*, volume 22. Siam, 1990.

[21] Iain S Duff. A review of frontal methods for solving linear systems. *Computer Physics Communications*, 97(1):45–52, 1996.

[22] Michael T Heath, Esmond Ng, and Barry W Peyton. Parallel algorithms for sparse linear systems. *SIAM review*, 33(3):420–460, 1991.

[23] Richard Barrett, Michael W Berry, Tony F Chan, James Demmel, June Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, and Henk Van der Vorst. *Templates for the solution of linear systems: building blocks for iterative methods*, volume 43. Siam, 1994.

[24] Patrick Amestoy, Iain S. Duff, Jean-Yves L'Excellent, and Jacko Koster. MUMPS : A general purpose distributed memory sparse solver. In Tor Sørevik, Fredrik Manne, Randi Moe, and Assefaw Hadish Gebremedhin, editors, *Applied Parallel Computing, New Paradigms for HPC in Industry and Academia, 5th International Workshop, PARA 2000 Bergen, Norway, June 18-20, 2000 Proceedings*, volume 1947 of *Lecture Notes in Computer Science*, pages 121–130. Springer, 2000.

[25] Patrick Amestoy, Abdou Guermouche, Jean-Yves L'Excellent, and Stéphane Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, 32(2):136–156, 2006.

[26] Patrick R Amestoy, Iain S Duff, Jean-Yves L'Excellent, and Jacko Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

[27] TJ Chung. *Computational fluid dynamics*. Cambridge university press, 2010.

[28] Frédéric Hecht. New development in freefem++. *Journal of Numerical Mathematics*, 20(3-4):251–266, 2012.

[29] Katia Ait Ameur, Camille Coti, Caroline Japhet, Pascal Omnes, Mathieu Peybernes CEA, and Thomas Rubiano. Cemracs 2016 project:'schwarz for triocfd'space-time domain decomposition for the stokes system; application to the triocfd code.

[30] David A Janes. Practical options for desktop cfd using open-source software.

[31] Gouri Dhatt, Emmanuel Lefrançois, and Gilbert Touzot. *Finite element method*. John Wiley & Sons, 2012.

[32] Henk Kaarle Versteeg and Weeratunge Malalasekera. *An introduction to computational fluid dynamics: the finite volume method*. Pearson Education, 2007.

[33] Christophe Cerin and Gilles Fedak. *Desktop Grid Computing*. Chapman and Hall-CRC, 1 edition, 2012.

[34] David P. Anderson. Boinc: A system for public-resource computing and storage. *Grid Computing, IEEE/ACM International Workshop on*, 0:4–10, 2004.

[35] Ali R. Butt, Rongmei Zhang, and Y. Charlie Hu. A self-organizing flock of condors. *J. Parallel Distrib. Comput.*, 66(1):145–161, 2006.

[36] Nazareno Andrade, Walfredo Cirne, Francisco Vilar Brasileiro, and Paulo Roisenberg. Ourgrid: An approach to easily assemble grids with equitable resource sharing. *Job Scheduling Strategies for Parallel Processing, 9th International Workshop, JSSPP 2003, Seattle, WA, USA, June 24, 2003, Revised Papers*, 2862:61–86, 2003.

[37] G. Fedak, C. Germain, V. Neri, and F. Cappello. XtremWeb: a generic global computing system. *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, pages 582–587, 2001.

[38] Zhaohui Wu. *Service Computing: Concept, Method and Technology*. Academic Press, 2015.

[39] Mike P. Papazoglou and Dimitrios Georgakopoulos. Introduction. *Commun. ACM*, 46(10):24–28, 2003.

[40] Munindar P. Singh and Michael N. Huhns. *Service-oriented computing - semantics, processes, agents*. Wiley, 2005.

[41] Maria E. Orlowska, Sanjiva Weerawarana, Mike P. Papazoglou, and Jian Yang, editors. *Service-Oriented Computing - ICSOC 2003, First International Conference, Trento, Italy, December 15-18, 2003, Proceedings*, volume 2910 of *Lecture Notes in Computer Science*. Springer, 2003.

[42] Hong Cai and Jia Zhang. SOA services and solutions. In *2006 IEEE International Conference on Services Computing (SCC 2006), 18-22 September 2006, Chicago, Illinois, USA*. IEEE Computer Society, 2006.

[43] Liang-Jie Zhang, Jia Zhang, and Hong Cai. *Services Computing*. Springer-Verlag Berlin Heidelberg, 2007.

[44] Albert G. Greenberg, James R. Hamilton, David A. Maltz, and Parveen Patel. The Cost of a Cloud: Research Problems in Data Center Networks. *Computer Communication Review*, 39:68–73, 2009.

[45] Leila Abidi, Jean-Christophe Dubacq, Christophe Cérin, and Mohamed Jemni. A publication-subscription interaction schema for desktop grid computing. In Sung Y. Shin and José Carlos Maldonado, editors, *SAC*, pages 771–778. ACM, 2013.

[46] Leila Abidi, Christophe Cérin, and Mohamed Jemni. Desktop grid computing at the age of the web. In James J. Park, Hamid R. Arabnia, Cheonshik Kim, Weisong Shi, and Joon-Min Gil, editors, *GPC*, volume 7861 of *Lecture Notes in Computer Science*, pages 253–261. Springer, 2013.

[47] Walid Saad, Leila Abidi, Heithem Abbes, Christophe Cérin, and Mohamed Jemni. Wide area bonjourgrid as a data desktop grid: Modeling and implementation on top of redis. In *26th IEEE International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2014, Paris, France, October 22-24, 2014*, pages 286–293. IEEE Computer Society, 2014.

[48] *2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum, Cambridge, MA, USA, May 20-24, 2013*. IEEE, 2013.

[49] Jean-Paul Smets-Solanes, Christophe Cérin, and Romain Courteaud. Slapos: A multi-purpose distributed cloud operating system based on an ERP billing model. In Hans-Arno Jacobsen, Yang Wang, and Patrick Hung, editors, *IEEE International Conference on Services Computing, SCC 2011, Washington, DC, USA, 4-9 July, 2011*, pages 765–766. IEEE, 2011.

[50] Romain Courteaud, Yingjie Xu, and Christophe Cérin. Practical solutions for resilience in slapos. In *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings, CloudCom 2012, Taipei, Taiwan, December 3-6, 2012*, pages 488–495. IEEE Computer Society, 2012.

[51] Congfeng Jiang, Jian Wan, Christophe Cérin, Paolo Gianessi, and Yanik Ngoko. Towards energy efficient allocation for applications in volunteer cloud. In *2014 IEEE International Parallel & Distributed Processing Symposium Workshops, Phoenix, AZ, USA, May 19-23, 2014*, pages 1516–1525. IEEE Computer Society, 2014.

[52] Walid Saad, Heithem Abbes, Mohamed Jemni, and Christophe Cérin. Designing and implementing a cloud-hosted saas for data movement and sharing with slapos. *IJBDI*, 1(1/2):18–35, 2014.

[53] Christophe Cérin and Gilles Fedak. *Desktop grid computing*. CRC Press, 2012.

[54] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, Mei-Hui Su, and K. Vahi. Characterization of scientific workflows. In *Workflows in Support of Large-Scale Science, 2008. WORKS 2008. Third Workshop on*, pages 1–10, Nov 2008.

[55] Stephen F. Altschul, Thomas L. Madden, Alejandro A. Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman. Gapped blast and psiblast: a new generation of protein database search programs. *NUCLEIC ACIDS RESEARCH*, 25(17):3389–3402, 1997.

[56] Robert Graves, Thomas H. Jordan, Scott Callaghan, Ewa Deelman, Edward Field, Gideon Juve, Carl Kesselman, Philip Maechling, Gaurang Mehta, Kevin Milner, David Ok aya, Patrick Small, and Karan Vahi. CyberShake: A Physics-Based Seismic Hazard Model for Southern California. *Pure and Applied Geophysics*, 168:367–381, 2011.

[57] Ji Liu, Esther Pacitti, Patrick Valduriez, and Marta Mattoso. A survey of data-intensive scientific workflow management. *J. Grid Comput.*, 13(4):457–493, 2015.

[58] Yogesh Simmhan, Lavanya Ramakrishnan, Gabriel Antoniu, and Carole A. Goble. Cloud computing for data-driven science and engineering. *Concurrency and Computation: Practice and Experience*, 28(4):947–949, 2016.

[59] Jia Yu, Rajkumar Buyya, and Kotagiri Ramamohanarao. *Workflow Scheduling Algorithms for Grid Computing*, pages 173–214. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[60] H. Topcuoglu, S. Hariri, and Min-You Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, Mar 2002.

[61] M. Rahman, S. Venugopal, and R. Buyya. A dynamic critical path algorithm for scheduling scientific workflow applications on global grids. In *e-Science and Grid Computing, IEEE International Conference on*, pages 35–42, Dec 2007.

[62] R. Sakellariou and H. Zhao. A hybrid heuristic for dag scheduling on heterogeneous systems. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, pages 111–, April 2004.

[63] Anne Benoit, Umit V. Catalyurek, Yves Robert, and Erik Saule. A Survey of Pipelined Workflow Scheduling: Models and Algorithms. *ACM Computing Surveys*, 45(4), 2013.

[64] Olivier Beaumont, Arnaud Legrand, and Yves Robert. The master-slave paradigm with heterogeneous processors. *IEEE Trans. Parallel Distributed Systems*, 14(9):897–908, 2003.

[65] Jean-Noël Quintin and Frédéric Wagner. Hierarchical work-stealing. In Pasqua D'Ambra, Mario Rosario Guarracino, and Domenico Talia, editors, *Euro-Par 2010 - Parallel Processing, 16th International Euro-Par Conference, Ischia, Italy, August 31 - September 3, 2010, Proceedings, Part I*, volume 6271 of *Lecture Notes in Computer Science*, pages 217–229. Springer, 2010.

[66] Michael Wilde, Mihael Hategan, Justin M. Wozniak, Ben Clifford, Daniel S. Katz, and Ian T. Foster. Swift: A language for distributed parallel scripting. *Parallel Computing*, 37(9):633–652, 2011.

[67] Justin M. Wozniak, Michael Wilde, and Daniel S. Katz. JETS: language and system support for many-parallel-task workflows. *J. Grid Comput.*, 11(3):341–360, 2013.

[68] Ioan Raicu, Ian T. Foster, and Yong Zhao. Guest editors' introduction: Special section on many-task computing. *IEEE Trans. Parallel Distrib. Syst.*, 22(6):897–898, 2011.

[69] Zhao Zhang, Allan Espinosa, Kamil Iskra, Ioan Raicu, Ian T. Foster, and Michael Wilde. Design and evaluation of a collective IO model for loosely coupled petascale programming. *CoRR*, abs/0901.0134, 2009.

[70] Amazon ec2. http://aws.amazon.com/ec2/.

[71] enstratus. http://www.enstratus.com.

[72] Rightscale. http://rightscale.com.

[73] Scalr. https://www.scalr.net.

[74] Young Choon Lee, Hyuck Han, and Albert Y. Zomaya. On resource efficiency of workflow schedules. In David Abramson, Michael Lees, Valeria V. Krzhizhanovskaya, Jack Dongarra, and Peter M. A. Sloot, editors, *Proceedings of the International Conference on Computational Science, ICCS 2014, Cairns, Queensland, Australia, 10-12 June, 2014*, volume 29 of *Procedia Computer Science*, pages 534–545. Elsevier, 2014.

[75] Ming Mao and Marty Humphrey. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In Scott Lathrop, Jim Costa, and William Kramer, editors, *Conference on High Performance Computing Networking, Storage and Analysis, SC 2011, Seattle, WA, USA, November 12-18, 2011*, pages 49:1–49:12. ACM, 2011.

[76] Marc Frîncu. Dynamic scheduling algorithm for heterogeneous environments with regular task input from multiple requests. In Nabil Abdennadher and Dana Petcu, editors, *Advances in Grid and Pervasive Computing, 4th International Conference, GPC 2009, Geneva, Switzerland, May 4-8, 2009. Proceedings*, volume 5529 of *Lecture Notes in Computer Science*, pages 199–210. Springer, 2009.

[77] Marc Eduard Frîncu. Scheduling service oriented workflows inside clouds using an adaptive agent based approach. In Borko Furht and Armando Escalante, editors, *Handbook of Cloud Computing.*, pages 159–182. Springer, 2010.

[78] Alok Gautam Kumbhare, Yogesh L. Simmhan, Marc Frîncu, and Viktor K. Prasanna. Reactive resource provisioning heuristics for dynamic dataflows on cloud infrastructure. *IEEE T. Cloud Computing*, 3(2):105–118, 2015.

[79] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw., Pract. Exper.*, 41(1):23–50, 2011.

[80] Daniel Zinn, Quinn Hart, Timothy M. McPhillips, Bertram Ludäscher, Yogesh Simmhan, Michail Giakkoupis, and Viktor K. Prasanna. Towards reliable, performant workflows for streaming-applications on cloud platforms. In *11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2011, Newport Beach, CA, USA, May 23-26, 2011*, pages 235–244. IEEE Computer Society, 2011.

[81] J. Mo and J. Walrand. Fair end-to-end window-based congestion control. *IEEE/ACM Transactions on Networking*, 8(5):556–567, Oct 2000.

[82] Y. Liu and E. Knightly. Opportunistic fair scheduling over multiple wireless channels. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 2, pages 1106–1115 vol.2, March 2003.

[83] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: A notion of fairness in resource allocation. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 345–354, New York, NY, USA, 1993. ACM.

[84] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. Dominant resource fairness: Fair allocation of multiple resource types. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 323–336, Berkeley, CA, USA, 2011. USENIX Association.

[85] Danny Dolev, Dror G. Feitelson, Joseph Y. Halpern, Raz Kupferman, and Nathan Linial. No justified complaints: On fair sharing of multiple resources. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 68–75, New York, NY, USA, 2012. ACM.

[86] Hui Wang and Peter Varman. Balancing fairness and efficiency in tiered storage systems with bottleneck-aware allocation. In *Proceedings of the 12th USENIX Conference on File and Storage Technologies*, FAST'14, pages 229–242, Berkeley, CA, USA, 2014. USENIX Association.

[87] Avital Gutman and Noam Nisan. Fair allocation without trade. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '12, pages 719–728, Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems.

[88] C. Joe-Wong, S. Sen, T. Lan, and M. Chiang. Multi-resource allocation: Fairness-efficiency tradeoffs in a unifying framework. In *INFOCOM, 2012 Proceedings IEEE*, pages 1206–1214, March 2012.

[89] Wei Wang, Baochun Li, and Ben Liang. Dominant resource fairness in cloud computing systems with heterogeneous servers. *CoRR*, abs/1308.0083, 2013.

[90] David C. Parkes, Ariel D. Procaccia, and Nisarg Shah. Beyond dominant resource fairness: Extensions, limitations, and indivisibilities. In *Proceedings of the 13th ACM Conference on Electronic Commerce*, EC '12, pages 808–825, New York, NY, USA, 2012. ACM.

[91] Richard Cole, Vasilis Gkatzelis, and Gagan Goel. Mechanism design for fair division: Allocating divisible items without payments. In *Proceedings of the Fourteenth ACM Conference on Electronic Commerce*, EC '13, pages 251–268, New York, NY, USA, 2013. ACM.

[92] Artur Andrzejak, Derrick Kondo, and David P Anderson. Exploiting non-dedicated resources for cloud computing. In *2010 IEEE Network Operations and Management Symposium-NOMS 2010*, pages 341–348. IEEE, 2010.

[93] Dongjin Yoo; Kwang Mong Sim. A locality enhanced scheduling method for multiple mapreduce jobs in a workflow application.

[94] Matei Zaharia, Dhruba Borthakur, Joydeep Sen Sarma, Khaled Elmeleegy, Scott Shenker, and Ion Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In Christine Morin and Gilles Muller, editors, *European Conference on Computer Systems, Proceedings of the 5th European conference on Computer systems, EuroSys 2010, Paris, France, April 13-16, 2010*, pages 265–278. ACM, 2010.

[95] Radu Tudoran, Alexandru Costan, and Gabriel Antoniu. Overflow: Multi-site aware big data management for scientific workflows on clouds. *IEEE Trans. Cloud Computing*, 4(1):76–89, 2016.

[96] Sergio Esteves and Lucas Veiga. WaaS: Workflow-as-a-Service for the Cloud with Scheduling of Continuous and Data-Intensive Workflows. *The Computer Journal*, 58, 2015.

[97] F. Benkhaldoun, I. Elmahi, S. Sari, and M. Seaid. An unstructured finite-volume method for coupled models of suspended sediment and bed load transport in shallow-water flows. *Int. J. Numer. Meth. Fluids*, 72:967–993, 2013.

[98] F. Benkhaldoun, J. Fort, K. Hassouni, and J. Karel. Simulation of planar ionization wave front propagation on an unstructured adaptive grid. *Journal of Computational and Applied Mathematics*, 236:4623–4634, 2012.

[99] Sébastien Célestin. *Study of the dynamics of streamers in air at atmospheric pressure*. 2010.

[100] Carolynne Montijn, Willem Hundsdorfer, and Ute Ebert. An adaptive grid refinement strategy for the simulation of negative streamers. *Journal of Computational Physics*, 219(2):801–835, 2006.

[101] Nicolò Cesa-Bianchi. Multi-armed bandit problem. In *Encyclopedia of Algorithms*, pages 1356–1359. 2016.

[102] Bahman Javadi, Derrick Kondo, Jean-Marc Vincent, and David P. Anderson. Mining for statistical models of availability in large-scale distributed systems: An empirical study of seti@home. In *17th Annual Meeting of the IEEE/ACM International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems, MASCOTS 2009, September 21-23, 2009, South Kensington Campus, Imperial College London*, pages 1–10, 2009.

[103] Daniel Lázaro Iglesias, Derrick Kondo, and Joan Manuel Marquès. Long-term availability prediction for groups of volunteer resources. *J. Parallel Distrib. Comput.*, 72(2):281–296, 2012.

[104] Bahman Javadi, Derrick Kondo, Jean-Marc Vincent, and David P. Anderson. Discovering statistical models of availability in large distributed systems: An empirical study of seti@home. *IEEE Trans. Parallel Distrib. Syst.*, 22(11):1896–1903, 2011.

[105] Leila Abidi, Christophe Cérin, and Kais Klai. Design, verification and prototyping the next generation of desktop grid middleware. In Ruixuan Li, Jiannong Cao, and Julien Bourgeois, editors, *GPC*, volume 7296 of *Lecture Notes in Computer Science*, pages 74–88. Springer, 2012.

[106] Aman Kansal, Feng Zhao, Jie Liu, Nupur Kothari, and Arka A. Bhattacharya. Virtual machine power metering and provisioning. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 39–50, New York, NY, USA, 2010. ACM.

[107] Senthil Nathan, Purushottam Kulkarni, and Umesh Bellur. Resource availability based performance benchmarking of virtual machine migrations. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ICPE '13, pages 387–398, New York, NY, USA, 2013. ACM.

[108] Christophe Pradal, Samuel Dufour-Kowalski, Frédéric Boudon, Christian Fournier, and Christophe Godin. Openalea: a visual programming and component-based software platform for plant modelling. *Functional Plant Biology*, 35(10):751–760, 2008.

[109] Christophe Pradal, Christian Fournier, Patrick Valduriez, and Sarah Cohen Boulakia. Openalea: scientific workflows combining data analysis and simulation. In *Proceedings of the 27th International Conference on Scientific and Statistical Database Management, SSDBM '15, La Jolla, CA, USA, June 29 - July 1, 2015*, pages 11:1–11:6, 2015.

[110] Pascal Hénon, Pierre Ramet, and Jean Roman. Pastix: a high-performance parallel direct solver for sparse symmetric positive definite systems. *Parallel Computing*, 28(2):301–321, 2002.

[111] Emmanuel Agullo, Luc Giraud, Abdou Guermouche, Azzam Haidar, and Jean Roman. Maphys or the development of a parallel algebraic domain decomposition solver in the course of the solstice project. In *Sparse Days 2010 Meeting at CERFACS*, 2010.

[112] Cédric Lachat, François Pellegrini, and Cecile Dobrzynski. Pampa: Parallel mesh partitioning and adaptation. In *21st International Conference on Domain Decomposition Methods (DD21)*, 2012.

[113] Arnaud Lanoix, Julien Dormoy, and Olga Kouchnarenko. Combining proof and model-checking to validate reconfigurable architectures. *Electronic Notes in Theoretical Computer Science*, 279(2):43–57, 2011.

[114] Jeffrey O Kephart and David M Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[115] Guilherme Arthur Geronimo, Rafael Brundo Uriarte, and Carlos Becker Westphall. Order@cloud: A VM organisation framework based on multi-objectives placement ranking. In Sema Oktug, Mehmet Ulema, Cicek Cavdar, Lisandro Zambenedetti Granville, and Carlos Raniery Paula dos Santos, editors, *2016 IEEE/IFIP Network Operations and Management Symposium, NOMS 2016, Istanbul, Turkey, April 25-29, 2016*, pages 529–535. IEEE, 2016.

[116] Guilherme Arthur Geronimo, Rafael Brundo Uriarte, and Carlos Becker Westphall. Toward a framework for vm organisation based on multi-objectives. In *ICN 2016, The Fifteenth International Conference on Networks, Lisbon, Portugal*, February 21 - 25, 2016.