Towards High Performance Robotic Computing

Leonardo Camargo-Forero · Pablo Royo · Xavier Prats

Universitat Politècnica de Catalunya - ICARUS Research group

Esteve Terradas, 5. 08860, Castelldefels, Catalonia - Spain

E-mail: leonardo.camargo@upc.edu

Abstract Embedding a robot with a companion computer is becoming a common practice nowadays. Such computer is installed with an operating system, often a Linux distribution. Moreover, Graphic Processing Units (GPUs) can be embedded on a robot, giving it the capacity of performing complex on-board computing tasks while executing a mission. It seems that a next logical transition, consist of deploying a cluster of computers among embedded computing cards. With this approach, a multi-robot system can be set as a High Performance Computing (HPC) cluster. The advantages of such infrastructure are many, from providing higher computing power up to setting scalable multi-robot systems. While HPC has been always seen as a speeding-up tool, we believe that HPC in the world of robotics can do much more than simply accelerating the execution of complex computing tasks. In this paper, we introduce the novel concept of *High Performance Robotic Computing - HPRC*, an augmentation of the ideas behind traditional HPC to fit and enhance the world of robotics. As a proof of concept, we introduce novel HPC software developed to control the motion of a set of robots using the standard parallel MPI (Message Passing Interface) library. The parallel motion software includes two operation modes: Parallel motion to specific target and swarm-like behavior. Furthermore, the HPC software is virtually scalable to control any quantity of moving robots, including Unmanned Aerial Vehicles, Unmanned Ground Vehicles, etc.

Keywords High Performance Robotic Computing - HPRC · General-purpose computing robot · HPC Cluster of robots · HPRC cluster · Parallel Robotic Computing Node - PRCN · General-purpose computing mission

1. Introduction

A robot should be more than a specific-purpose machine, used only for a particular task, designed, and developed with specific hardware and software, making it difficult to integrate with other robots or reuse for a different purpose.

If a robot were to become a *general-purpose computing unit* such as a common computer or a cellphone, it could be reused for different missions and integration with other robots would become transparent. Nowadays, coupling a robot with a companion computer is becoming mainstream, mostly with unmanned vehicles. The advantages of such approach are many because such companion computer is installed with a conventional operating

system, which opens the scope of what can be done with a robot to only where our imagination can takes us.

Moreover, if a robot's companion computer is installed with an operating system, the integration between robots becomes a straightforward endeavor via traditional protocols such as TCP / IP. In addition, with a set of robots, each with its companion computer, it becomes possible to set a *High Performance Computing (HPC) cluster of computers*.

A HPC cluster of computers is a set of computers (*nodes*) connected via a Local Area Network (LAN), in which a set of software layers are installed in each of its nodes. Such software layers provide or have the potential of providing a set of features, such as scalability, user-transparency, resilience and ultimately higher computing efficiency, which we depicted as *HPC features*.

In fact, the existence of HPC was motivated to provide computing efficiency. Certain problems within scientific research or even commercial activities require the processing of large datasets and/or the application of complex mathematical models highly expensive, in computational terms. In order to have a solution for such problems, a traditional computer and software could take years to reach to it. Therefore, a HPC cluster of computers uses *parallel software*, i.e. software that it is able to run in multiple computing units at the same time decreasing considerably the time required to achieve a solution for such complex problems. In addition, parallel software can be also executed on a single node, for example, using *multi-core* and *many-core* architectures, such as common CPUs with more than one computing core or GPUs (Graphics Processing Unit) respectively.

Research works such as [1], [2] and [3] proposed the use of robots embedded with GPUs for the processing of complex robotic tasks. Even at industry level, the UAVs Chinese manufacturer DJI [4], launched the *Manifold* embedded computer [5], which includes a NVIDIA [6] GPU. Moreover, previous attempts, such as [7] and [8] have proposed to implement a *HPC cluster of robots* with the objective of solving expensive robotic computational tasks.

However, HPC can be more than simply a speeding-up tool, precisely because in order of providing such computational efficiency, HPC infrastructures and software were designed to provide features such as scalability, etc., as previously mentioned. Such features could be very useful in the world of robotics, especially with multi-robot settings.

In this work, we introduce the novel concept of *High Performance Robotic Computing (HPRC)*, which aims at adapting traditional HPC into the world of robotics by not only providing higher computer efficiency but also exploiting HPC features such as scalability, user-transparency, etc. While previous works have suggested the use of HPC within robotics scenarios, there is still a lack of a formal definition of the HPRC field and successful strategies to adapt traditional HPC into robotic systems.

In this work, we present the following main contributions:

- A literature review of previous attempts of using HPC in robotic settings, from single to multiple robot approaches.

- The formal definition of the HPRC field, which takes traditional HPC, as its foundation. Using HPC as a tool for speeding up complex computing tasks has been largely demonstrated in the literature and works such as [8] presented results about its use in robotics context. In this work, we focus on the rest of the features provided by HPC, rather than on its proven computing efficiency. This requires a change of mind of the current understanding of HPC, in which the only objective is to achieve computing efficiency, for a new one in which other features are exploited for the advantage of multi-robot settings.
- Results of a traditional HPC benchmark upon embedded computing cards, such as those being used as robotic companion computers nowadays.
- Results of novelty software developed to control the motion of heterogeneous multi-robot settings using traditional HPC technologies over a HPRC cluster made up of embedded computing cards. The software demonstrates that HPRC does not only focus on complex computing tasks as it does its foundation HPC.

The rest of the paper is organized as follows. Section 2 presents general and specific concepts within High Performance Computing that are important for the introduction of HPRC. It also lists a set of previous HPC works on robotics. In section 3, High Performance Robotic Computing is introduced. Following, section 4 presents a set of experiments performed to demonstrate the feasibility and potential of HPRC. Finally, section 5 presents conclusions and future work.

2. High Performance Computing and robotics

High Performance Computing (HPC) has been serving human research, commercial activities, military endeavors and all scientific exploration for the last decades.

Nowadays, predicting the weather, manufacturing a new vaccine or pharmaceutical drug, understanding the genetics of known species, sending personalized advertisement to millions of users worldwide, etc., are all results of parallel computing software running upon HPC infrastructures. In fact, it is difficult to find some human activity, that it is not at least somehow influenced by HPC. Even robotics research have lightly flirted with HPC, evidence of that can be seen with the appearance of powerful embedded computing cards, for example CUDA [9]-enabled GPU cards.

HPC can be defined as a tool for speeding up the computation of complex tasks or to be able to analyze large datasets in reasonable time, something that would not be possible in common computing infrastructures such as personal computers, neither with sequential software.

Software can be classified according to Flynn's taxonomy [10]:

Table 1. Software classification according to Flynn's taxonomy

Acronym	Definition	Commonly known as
SISD	Single Instruction Single Data	Sequential software
SIMD	Single Instruction Multiple Data	Parallel software

MISD	Multiple Instruction Single Data	
MIMD	Multiple Instruction Multiple Data	Distributed software

As shown in Table 1, *sequential software* is such that it is executed on a single computing unit (e.g. a CPU core) while *parallel software* is such that uses multiple computing units at once. Following, *distributed software* is such that can be executed on multiple computing units but not necessarily at the same time. Therefore, parallel software can be seen as a subset of the ideas behind distributed software. Moreover, in order for a software to be able to run parallel or distributively, it must be specifically programmed to do so (*explicit parallelism*), otherwise, it will run on a single computing unit.

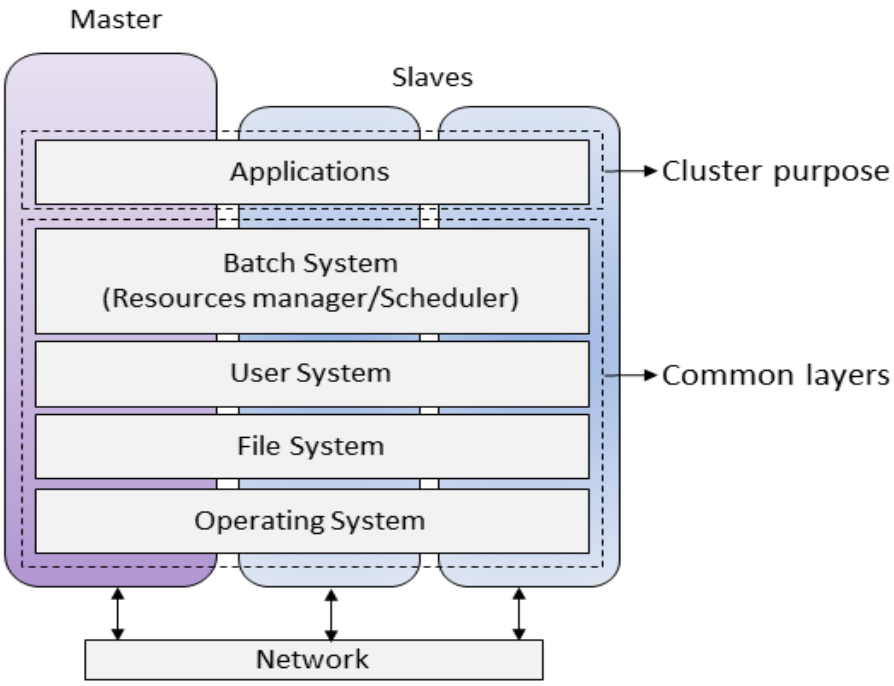

Figure 1. HPC Cluster of computers software architecture. The architecture is composed of the HPC software layers: *Operating system*, *file system*, *user system*, *batch system* and *applications*

The most common HPC infrastructure is defined as a *Cluster of Computers*. A HPC cluster of computers consists of a set of computers connected through a local area network (LAN), where each computer, known as a *node* or more specifically a *parallel computing node*, is installed with a set of software layers that ultimately make all the nodes look and act as a *single-image system*. The software layers, depicted as *HPC software layers*, are shown in figure 1. The first four HPC software layers, up to the batch system are common in all HPC clusters while the last layer, the applications layer defines the purpose of the infrastructure, e.g. physics, bioinformatics, etc.

In a HPC cluster, nodes are classified as *master* or *slaves*. The master node is the frontend to which the user interacts with and implements the server side of each of the HPC software layers. Conversely, the slave nodes, which are the ones in charge of the actual parallel computing, implement the client side of each of the HPC software layers. Nevertheless, a node can be a master and a slave at the same time.

A parallel computing node is a computer that is installed with a set of software libraries, such as Open MP [11], MPI (Message Passing Interface) [12] or CUDA, etc., all which allow it to run parallel software. Software must be written with these libraries, otherwise it cannot exercise any form of parallelism, other than the provided by default by most common compilers (*implicit parallelism*).

Furthermore, a parallel computing node can be any type of computer where such parallel software libraries can be installed. For example, a Raspberry PI [13] companion computer is currently supporting the installation of Open MPI [14], an implementation of the MPI standard. Moreover, the use of CUDA in embedding computing devices is becoming quickly mainstream.

CUDA, which stands for *Compute Unified Device Architecture* is a widely used GPGPU (*General Purpose Computing on GPUs*) technology, which allows achieving great computation speedup and is exploited in several research fields [15]. The GPU (Graphics Processing Unit) card was, before CUDA, used for the processing of graphics exclusively, as its name stands, but graphics processing is a default parallel task, therefore, the GPU was designed with multiple computing units (cores) and optimized for parallelism. With the appearance of CUDA, the GPU became a general-purpose computing device, where any software running, not only for graphics processing, could exploit the GPU's inherit parallelism.

The NVIDIA Jetson TK1 [16], is a CUDA-enabled embedded GPU that have been used to speed up the computation of complex computing tasks in robotics. For example, Cocchioni et al [1] proposed a technique for visual-based landing for an UAV (Unmanned Aerial Vehicle), where image processing was accelerated using the Jetson TK1 platform. Jeon et al [2] proposed an algorithm for visually guided control for an unmanned aerial vehicle, which showed five-time speedup using the TK1 in comparison with a common x86 CPU. In addition, Cetin et al [3] proposed an algorithm for autonomous flight formation with collision and obstacle avoidance using CUDA. However, the authors did not use an embedded GPU like the TK1 but rather a desktop GPU. Moreover, Kanellakis et al [17] in its review of current developments and trends for UAV computer vision mentioned the real-time advantages that can be obtained with the use of GPUs. Furthermore, the industry is also considering the use of embedded GPUs in robotics [18]. In addition, the DJI Manifold comes with a NVIDIA Tegra K1 processor [19].

However, GPUs are not the only one option for setting a robot as a parallel computing node, which we defined as a *Parallel Robotic Computing Node - PRCN*. Salami et al [20] compared the execution of two parallel algorithms, parallel hot spot detection and jellyfish detection on four computing cards, the EPIA N700-15 VIA C7-1.5, the Pandaboard with OMAP 4430 integrated CPU, the INTEL T7500 on GENE-9655 motherboard and the TilenCore-Gx board with TILE-Gx36. The authors used thread-based parallelism with OpenCV [21].

While previous research shows the existence of PRCNs in academia and industry, a robot could be also set as a full cluster of computers as Gonçalves et al [22] proposed. In their work, the authors set up an embedded cluster, in one single robot, composed of one master node and four ODROID-x2 [23] slave nodes, Ubuntu as the Operating System layer and

Sun Grid Engine [24] as the batch system layer. However, the authors did not deploy the other HPC software layers.

Furthermore, attempts to deploy a cluster of computers within a multi-robot system have been done in the past. Holland et al [7] proposed the UltraSwarm system, an interesting combination of bioinspired UAV flocking and wireless cluster computing. However, the authors did not implement any of the HPC software layers, except a Linux operating system. The authors did however, showed the possibility of establishing cluster communications using Bluetooth, with the limitations in 2005, when their work was published. Nowadays, communications technologies have improved remarkably in comparison with 2005, a strong advantage for the setting of a full wireless HPC cluster of computers nowadays.

Finally, Ali Marjovi et al [8] proposed the concept of *robotic cluster* representing a group of robots, which are able to share their processing resources among the group with the objective of quickly solve computationally hard problems. Again, however, the authors did not implement the user system nor the file system layer. In addition, the authors described MPI as the middleware for cluster computing. However, while MPI provides middleware services, the batch system is the actual middleware in a cluster of computers because it abstracts a distributed set of nodes as a single image system.

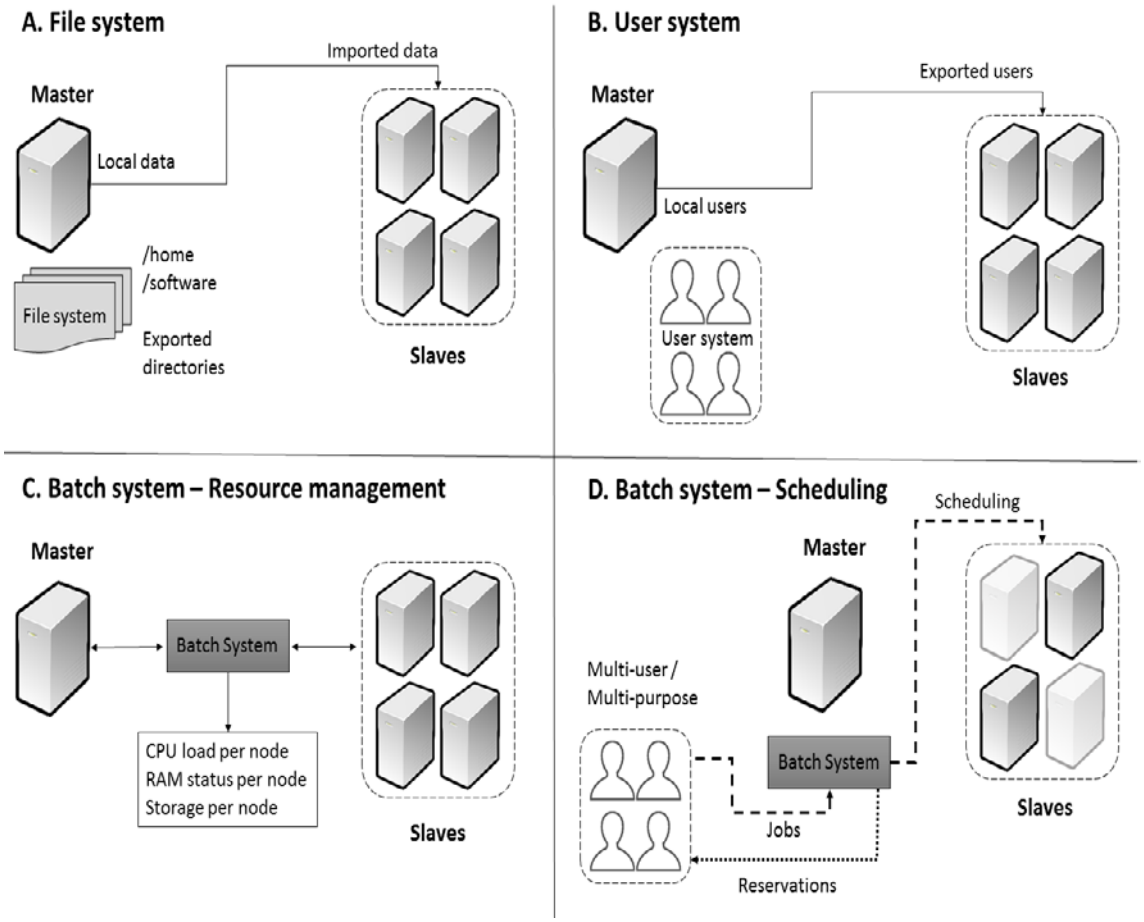

Figure 2. HPC software layers functioning. **A.** the *File system* is used to share data locally stored in the master node. **B.** the *User system* is used to export the master's local users. **C.** the *Batch system* monitors the resources (CPU load, RAM, etc.) of each of the nodes. **D.** the *Batch system* include scheduling policies for sharing the computing nodes among the users in an efficient way

Figure 2 displays the main functionalities provided by each of the HPC software layers, except the operating system layer. A cluster of computers is a multiuser/multipurpose infrastructure designed to facilitate computational efficiency. The main reason of the HPC software layers existence is to provide the means for achieving such efficiency in an environment where multiple users compete for the available resources.

The following concepts, presented in table 2, are important for the understanding of the general workflow, when interacting with an HPC cluster of computers:

Table 2. HPC workflow concepts

Concept	**Definition**
Job	A job is a *software* requiring a set of *resources* during a *walltime*
Software	Flynn's taxonomy (table 1), mainly SIMD, MISD and MIMD
Resources	Quantity of nodes, quantity of CPUs, quantity of RAM, etc.
Walltime	The user estimation of the time required for the software to finish
Reservation	The resources are reserved exclusively for the user during the walltime

The workflow is as follows: the user logs in the master node. By means of the user system, the user exists physically in the master node and logically in the slaves (figure 2 - B). The user submits a job to the batch system, which questions the slaves trying to find the job requested resources (figure 2 - D). If the resources are found, a reservation is created, otherwise the job is queued. Since most software require data processing, the File system takes care of replicating user data in the nodes. Also, commonly, the user main space (in Linux: the user's home) is also exported to the slaves, while physically existing in the master node (figure 2 - A). Moreover, the batch system periodically questions all the nodes in order to be aware of the status of the resources in the cluster (figure 2 - C).

Given the functionalities provided by the HPC software layers, an HPC cluster of computers is an infrastructure that can provide centralization / distribution, general-purpose computing, high computing efficiency, homogeneity and heterogeneity, scalability and user-transparency. A thorough explanation of all features will be provided in next section in table 5.

Traditional High Performance Computing is a powerful tool that has been proven to obtain great results in the previous decades. In this section, we aimed at introducing the basic concepts of HPC. For more information and technical details, please refer to the following books [25] and [26].

In the next section, we present the novel concept of *High Performance Robotic Computing*, which aims at adapting the ideas behind traditional HPC for the world of robotics, whilst exploiting all traditional HPC features, not only computing efficiency as proposed by previous works.

3. High Performance Robotic Computing

In this section, we introduce the novel concept of *High Performance Robotic Computing (HPRC)*. While traditional HPC is commonly seen as a speeding-up tool for complex computing tasks, we believe HPRC can go beyond simply speeding up robotic tasks and actually serve as the bedrock for setting up powerful scalable multi-robot systems able to perform all kind of missions.

Therefore, we define *High Performance Robotic Computing - HPRC as a set of strategies that allow the deployment of multi-robot systems that behave and act as a single distributed unit able to exploit HPC features such as scalability, user-transparency, centralization / distribution and ultimately computing efficiency.*

Following, figure 3 introduces the HPRC infrastructure defined as *HPRC cluster* or *HPC cluster of robots* and presents a comparison with a traditional HPC cluster.

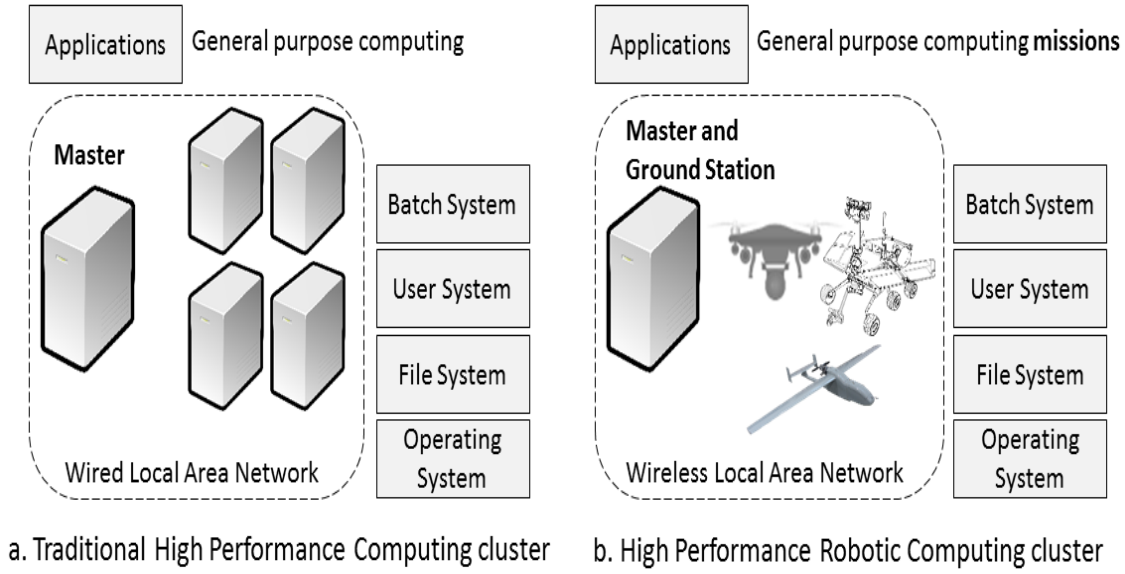

Figure 3. Traditional High Performance Computing cluster VS. High Performance Robotic computing cluster. **A.** Traditional HPC cluster used for general-purpose computing, where all nodes communicate via wired technologies. **B.** HPRC cluster used for general-purpose computing missions, where all nodes communicate via wireless technologies. While HPRC clusters might require the master and the ground station to be the same machine in order to protect the server side of the HPC software layers, other schemes are possible where a robot acts as master or master / slave. This highly depends of the mission being carried out.

While setting up a HPRC cluster seems a straightforward endeavor, some considerations must be taken into account when facing mobile computing units. However, advancements in communication technologies are currently on the verge of providing full mobile HPC. Qualcomm Technologies [27], at the beginning of 2017 released a communications performance study [28] of Unmanned Aerial Systems (UAS) using 4G LTE (Long Term Evolution) networks. The experiments were carried out with UAS operating beyond visual line of sight at 400 feet above ground level and below as well. The results are very positive and suggest that full Internet connectivity can be granted to UAS with current communications technologies. Moreover, according to the International

Telecommunications Union (ITU) [29], in its document: *Minimum requirements related to technical performance for IMT-2020 (International Mobile Telecommunication system) radio interface(s)* [30], some of the requirements for a technology to be classified as 5G are:

- Downlink peak data rate: 20 Gbit/s
- Uplink peak data rate: 10 Gbit/s
- User latency: 4 ms for eMBB (Enhanced Mobile BroadBand) and 1 ms for URLLC (Ultra-Reliable Low Latency Cellular Networks)

Such features result very advantageous with mobile environments. However, having full Internet connectivity in a UAV does not guarantee high performance computing when comparing with traditional wired technologies. Research has been done in the past regarding the adaptation of HPC to mobile environments. Nevertheless, this is out of the scope of this work. HPRC aims at fully deploying traditional HPC in robotics (mobile environments) but focusing mainly on the key ideas and concepts, rather than on the communications technologies. Nevertheless, current advancements suggest that communications are going to continue improving in the following years, facilitating traditional HPC in mobile environments.

Table 3 shows a comparison between a HPC and a HPRC cluster.

Table 3. HPC VS HPRC cluster of computers

Feature	**HPC Cluster**	**HPRC cluster**
Nodes	Commodity computers or powerful servers (i.e. supercomputers)	Robots, e.g. Unmanned Aerial Vehicles (UAVs), Unmanned Ground Vehicles (UGVs), etc. Formally, the nodes in a HPRC cluster are called *Parallel Robotic Computing Nodes -PRCNs*. While this paper focuses on unmanned vehicles, any kind of robot, which can be embedded by a companion computer, can be part of a HPRC cluster
Communications	Wired communications supporting TCP/IP (e.g. Gigabit Ethernet, Infiniband, etc.)	Wireless communications supporting TCP/IP (e.g. Wi-Fi, 4G, 5G, etc.)
Applications layer	General-purpose computing. Any kind of software, provided that is written in a parallel way, can be executed on a HPC cluster of computers	General-purpose computing mission: We define a mission as a graph of tasks, where each task is a HPC job
Master node	The node acting as a frontend for the HPC cluster	Both the node acting as a frontend for the HPRC cluster and the ground station controlling the multi-robot system
Software layers	Operating system, user system, file system and batch system	Same as in a HPC cluster

Data	In a HPC cluster, the data, both input and output data is of a general nature, i.e. it could be anything	While the data in a HPRC cluster could be of any nature as well (input and output), in many cases could relate to geolocation. For example the area assigned to each PRCN (input data) and the imagery captured by it on a surveillance mission (output data)

While a HPRC cluster consists of a traditional HPC cluster with some adaptations, HPC ideas, beyond its common infrastructure, can be as well adapted to the world of robotics giving birth to HPRC. Following, we introduce such adaptation.

Each of the HPC software layers have a purpose to exist in a cluster of computers, as it was previously introduced in section two. The software packages that need to be installed for each of the HPC software layers are available for most Linux Distributions and are currently compiled for ARM architectures, such as Raspberry PI, etc.

It is our belief that Ubuntu or Debian-based distributions such as Raspbian are good candidates for the Operating System layer. Debian-based distributions (including Ubuntu e.g. Mate) are lightweight and very easy to interact with (user-friendly) plus we can confirm that all software packages, necessary to configure a full HPC cluster, are available for the mentioned Linux distributions. Moreover, robotics software packages such as ROS (Robot Operating System) [31], DroneKit [32] and ArduPilot SITL [33] are also available for Raspbian and Ubuntu Mate making these operating systems well suited platforms for HPRC.

Therefore, as we will present in section 4, a full HPRC cluster can be deployed nowadays. However, the way in which the HPC software layers are deployed and configured differs from a traditional HPC cluster as figure 4 presents.

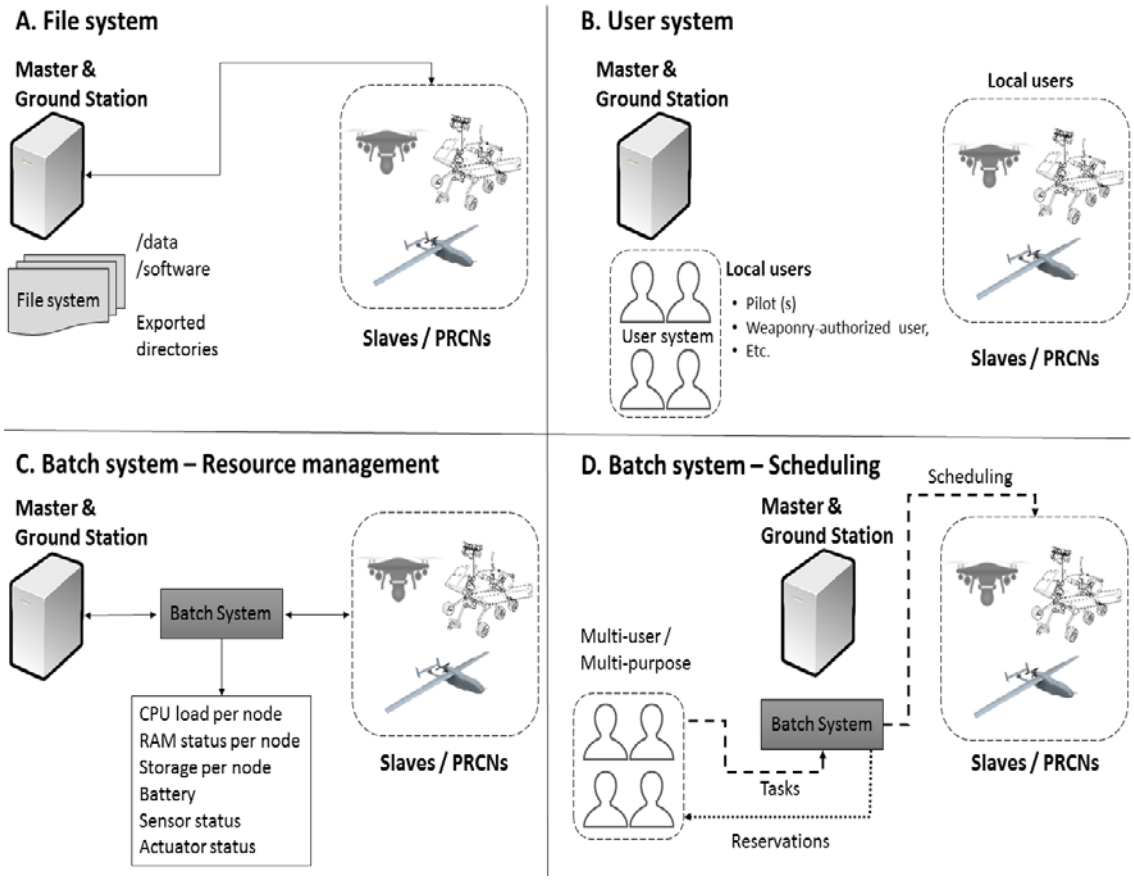

Figure 4. HPC software layers in the world of robotics. **A.** User space should not be exported. **B.** The same users should exist locally in the master and the slaves rather than remotely in the master node. **C.** HPRC resources include batteries, sensors, etc. **D.** A HPC job is equivalent to a HPRC task

Some differences exist between the functioning of the HPC software layers in a traditional HPC cluster (figure 2) versus a HPRC cluster (figure 4).

- In a HPRC cluster, the master node acts as a ground station for all the slaves. However, this is not the only approach that could be followed. A robot could act as master node or as master and slave at the same time (like a P2P configuration) or as ground station for other robots. Even each robot could embed its own ground station. All these approaches are valid within the scope of HPRC. Furthermore, a *shadow master node* could be set on any of the nodes within a HPRC cluster, in order to increase resilience or load balancing, etc. Nevertheless, in this paper, we present the simplest case, where the master node is not a robot but rather a normal computer acting as well as ground station for all the parallel computing robotic nodes.

- In a HPC cluster, the master node usually holds the data locally, data that is exported to the slaves using the file system. In addition, the results of a particular simulation (output data) are stored in such exported folders. Traditionally, the most common exported folder is the one that contains the users' space (e.g. */home* in a Linux distribution). This implies that all the users exist in all the nodes, usually locally in the master node and exported to the slaves nodes by means of using the user system. Consequently, the slaves usually do not store any data or user locally. Contrary, in a HPRC cluster, the users should rather exist locally in the PRCNs (figure 4 - B)

because in case of disconnection from the master node, during the execution of a mission, the robot should be autonomous enough to continue operating. Accordingly, the user's space (/home) should not be exported from the master node (figure 4 -A), but rather exist locally in each PRCN. With this approach, the PRCNs actually store data locally. Such data (e.g. imagery, etc.) should be copied in a shared location (folder locally in the master node) accessible to all local users or under a privileges scheme. Therefore, in case of failure, the master holds a centralized image of all data acquired by the multi-robot system acting as a HPRC cluster, which results advantageous in robotic missions. Data copying can be done constantly or in proximity with Wi-Fi repeaters or antennas, etc. Intelligent approaches can be taken into consideration by monitoring communications status.

- Regarding the batch system, HPRC extends HPC with two features. First, resources in a HPRC cluster are more than just CPUs or RAM because a robot usually comes with a set of sensors, actuators, battery, etc. By default, no batch system monitors such type of resources but it is rather easy to adapt open source solutions such as PBS Torque [34] or SLURM [35] to do so. Such adaptation is out of the scope of this job but it is part of the future work. Second, we replace the concept of HPC job for *HPRC task*, but they are ultimately equivalent. Job is mostly a HPC term while task is consistently used in robotics literature.

A *HPRC task* is a software that requires a set of resources (e.g. a camera, a thermal camera, X computing cores, etc.) during a walltime. This means that in HPRC, any robotic task is associated with a software, even if it is only for example a camera driver. Therefore, since every task is linked with a software or sometimes hardware as well, this means that a PRCN is a general-purpose computing device because the robot could be reused for other task if the software is simply changed. In consequence, we define the concept of *General-purpose computing Robot*.

A robot is currently a specific-purpose hardware/software device. Both at research and industrial level, many robots use proprietary hardware and software designed specifically to satisfy the purpose of the robot (specific-purpose robot). This generates problems to reuse a robot for a different purpose or to integrate multiple robots, belonging to different labs, industries, etc., into a single system. While attempts to solve these issues have been proposed (e.g. ROS), understanding a robot as a general-purpose computing device could be very advantageous for the creation of new interconnected robotic systems, as it is the objective of HPRC.

Consequently, HPRC tasks are classified as:

- Exclusive: An *exclusive HPRC task* is such that is carried out by a single PRCN. For example, consider a scenario where one PRCN has a particular type of camera. Such camera is a requirement for the particular task; therefore, this task can be performed only by such PRCN.
- Parallel: A *parallel HPRC task* is such that is carried out by multiple PRCNs at the same time. For example, given a geographic area, each of the PRCNs is given the task of monitoring a sub area. This process is done in parallel.

- Distributed: A *distributed HPRC task* is such that is carried out by multiple PRCNs but not necessarily at the same time.

The classification resembles the ideas given by Flynn's Taxonomy. Finally, we define a mission as a *directed graph of tasks* and given that each task is linked to a software, under HPRC nomenclature we define a mission as a *General-purpose computing mission*. In future works, we will present a formal description of such mission understanding, the high-level language to define it and the required services to use it. However, task allocation is done via a *matching service* between task requirements and available resources; similarly as the batch system does it in a traditional HPC cluster when inputted with a HPC job. The high-level language to describe a mission and the matching strategies require further deepening and we believe it to deserve a full paper, therefore it is out of the scope of this work.

Following table 4 summarizes all HPRC concepts.

Table 4. HPRC main concepts

Concept	Definition
General-purpose computing robot	A robot embedded with a general-purpose companion computer. A robot can be used for different missions by simply replacing the software associated to it
Parallel Robotic Computing Node - PRCN	A general-purpose computing robot in which parallel software libraries (e.g. Open MP, MPI, CUDA, etc.) are installed in its companion computer or embedded computing cards (e.g. GPUs). A HPRC cluster is composed of PRCNs
HPRC cluster or HPC Cluster of robots	A set of PRCNs connected via a wireless LAN and installed with the HPC software layers adapted to the specific conditions of the world of robotics
Resources	Companion computers' cores, embedded computing cards (GPUs, FPGAs, etc.), cameras, sensors (e.g. GPS, LIDAR, etc.), actuators (gimbals, weaponry, robotic hand, etc.), batteries, etc. Resources could be also software-based e.g. licenses, etc.
HPRC task	A software (SISD, SIMD, MISD, MIMD), linked to a robotic task, requiring a set of resources during a specific time (walltime)
General-purpose computing mission	A HPRC mission is a directed graph of HPRC tasks. The direction implies that a task must be done before the next task can be started. Moreover, a HPRC mission is a *Hierarchical directed network*. A hierarchical directed network is such that the nodes (HPRC tasks) within the network have a hierarchy that may vary between nodes. In a previous work by the first author [36], it was found that directed networks with higher hierarchy are more stable than those with low hierarchy. Further research in this topic will be part of the future work

HPRC is more than simply deploying a HPC cluster of robots, as previous attempts in research have suggested. However, installing and configuring a HPC cluster of computers is not a simple task. Intimate knowledge of Linux and HPC are necessary to set up an efficient HPC infrastructure. In a previous work [37], we introduce a ROS package for the automatic installation and configuration of a cluster of computers in Ubuntu and

Debian-based Linux distributions. The ROS package, depicted as *HPC-ROS*, installs and configures automatically all HPC software layers depending of the node being a master or a slave. The HPC-ROS package installs and configures the software packages *NFS* [38], *Open LDAP*, an implementation of the *LDAP standard* [39], PBS Torque [34] and Open MPI [14] for the file system, user system, batch system and parallel libraries respectively. The HPC-ROS package is licensed under request.

The advantages of adopting HPRC in multi-robot systems are many, as we introduced in our previous work [37]. In the following table 5, we summarize and expand such advantages.

Table 5. HPC cluster of computers features in the world of robotics

Feature	**High Performance Computing (HPC)**	**High Performance Robotic Computing (HPRC)**
Centralization/ distribution	While a HPC cluster of computers is a distributed infrastructure, where each node has its own resources (*distribution*), the master node acts as a frontend that masquerades multiple nodes into a single powerful machine (*centralization*)	In a multi-robot setting, each robot using its resources can perform its assigned tasks (*distribution*). However, the ground station acting as user frontend can control the entire robots (*centralization*)
Cost	Depending of the quantity and type of nodes in a HPC cluster of computers, the cost can largely vary. Moreover, energy consumption and cooling systems represent high expenses for HPC infrastructures operation	Using multiple robots could be less expensive that setting a monolithic robot with the same amount of computing power. For example, this ZDNet blog [40] shows how to build an UAV with a Raspberry PI for 200 USD. This low cost per robot allows building inexpensive multi-robot HPRC settings
General-purpose computing	An HPC cluster is a general-purpose computing infrastructure. This means that software for different fields (e.g. physics, bioinformatics, etc.) can be executed on it. For example, the top 500 supercomputers [41] are used for the execution of all kind of software. Nevertheless, the software architecture (HPC software layers), is the same for most of them	A HPRC cluster can be used for different purposes, different missions, transforming a robot into a general purpose computing infrastructure and facilitating reutilization and integration with other robots, other multi-robot systems, etc.
High computing efficiency	Computation time decreasing using parallel software	While the main objective of a traditional HPC cluster of

		computers is to provide higher computing power, a HPC cluster of robots not necessarily needs this. However, it is a good plus
Homogeneity and Heterogeneity	In order to decrease infrastructure administration issues, it is recommended to have the same operating system installed in all the nodes (homogeneity). However, the resources between nodes may vary, e.g. a node could have GPUs or not, coprocessors (e.g. Intel Xeon Phi [42]), the type and quantity of CPUs might be different, etc., (heterogeneity)	All types of robots (UAVs, UGVs, etc.) with different types of sensors, actuators, Linux Operating Systems, etc., can be integrated into an HPC cluster of robots. Still it is recommended that the robots' companion computers have the same operating system for software consistency.
Multi-purpose	A HPC cluster allow users to run different jobs at the same time provided resources' availability	A HPRC cluster could be used for several missions at the same time. For example, a subset of the total computing cores in the HPRC cluster could be dedicated to process imagery, while the remaining computing cores could be used to provide Internet connectivity, content download, etc.
Multi-user	Multiple users interact with a HPC cluster of computers at the same time	Multiple users could use the resources of a HPC cluster of robots. For example, a pilot user, a weaponry user, etc. Moreover, multiple users could be logged in at the same time performing different tasks within a mission or different missions, etc.
Operating-System based robots	Does not apply	With operating system based robots, a robot becomes a complete Internet of Things' device, as a mobile phone. With traditional and stable operating systems, all kind of packages, libraries, etc., can be installed on a robot, even during the execution of a mission (i.e. *on-fly installation*). The possibilities are endless.

Resilience and failure tolerance	MPI libraries provide resilience and failure tolerance by queuing messages during a time window in case the recipient of such messages is not available for reception.	Task redistribution in case of robot failures, creating resilient multi-robot systems. This is not provided by any of the HPC software layers, therefore it would require extra software and it is out of the scope of this work.
Scalability	Simple adding and removal of nodes	Simple integration of new robots at the beginning or during real-time missions. Mission area redistribution, failure tolerance improvements, etc. Furthermore, the same software and strategies can be apply to any number or type of robots providing flexibility as well.
Security	Operating systems installed on HPC nodes are usually heavily protected by security schemes (e.g. firewalls, user restrictions, etc.)	Communications with robots are usually done via insecure and not scalable radio signals. If TCP/IP communications are implemented, the complete secure Internet scheme can be set up among robots' communications.
User-transparency	The complete infrastructure is observed as a single computing unit from the user's perspective	*Single image / Centralization and Distribution*. The user (e.g. the pilot) could control the entire multi-robot system from a single interface (e.g. the master node). However, each robot could be given sufficient autonomy creating a fully centralized / distributed system.

It is our belief that HPRC holds enormous advantages for the implementation of complex multi-robot systems. Next section 4 presents the results of experiments performed with a HPRC cluster built with a set of Raspberry Pi model 3 B [43].

4. Experiments

In this section, we present experiments performed on a HPRC cluster made up of a set of embedded computing cards. Initially, we deploy a one master/slave node and three slaves (four-node HPRC cluster) setting, upon which we performed a standard MIMD (Multiple Instruction Multiple Data) HPC performance test (section 4.1) known as *High Performance Linpack (HPL).* The HPC-ROS package was used to install and configure the cluster setting, made up of four Raspberry PI model B cards. The HPL test is used to show the feasibility of HPC with embedded computing cards.

Following, we deployed a five-node HPRC cluster composed of one master node set up on a normal laptop and the four Raspberry PI 3 model B as slaves. Upon the second setting, we present a SITL (Software In The Loop) simulation of collective motion controlled by MPI software (section 4.2). A SITL experiment is such that allows simulating vehicles using software-based instead of hardware autopilots. This approach is useful as a first step preceding the utilization of real robots. Furthermore, this simulation is used to show the HPRC potential to provide user-transparency, scalability and centralization / distribution.

4.1. Performance

In figure 5, we present our hardware platform, consisting of one master/slave node and three slaves (four computing nodes). The Raspberry PI 3 model B card comes with a Quad Core 1.2GHz Broadcom BCM2837 64bit CPU and 1 GB RAM [43]. Therefore, the first setting has 16 CPU cores. The quantity of cores can be a way of categorizing a HPC infrastructure but traditionally the standard approach is to use the *High Performance Linpack (HPL)* test [44]. The Top 500 supercomputers are classified using HPL. In this section, we present the results of running HPL in the four-node setting.

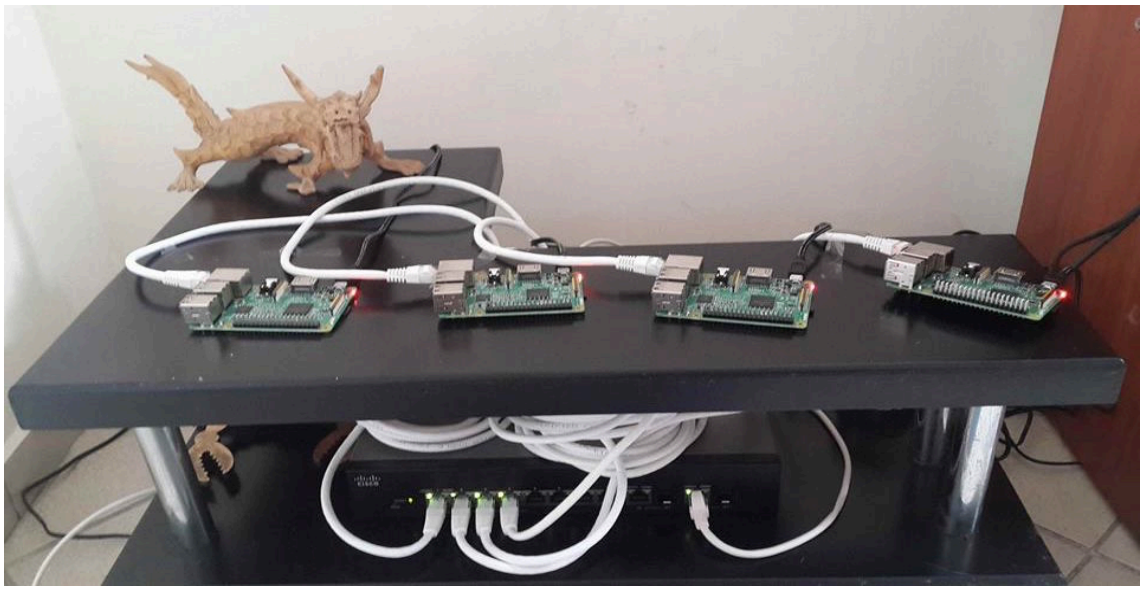

Figure 5. HPRC hardware platform. HPRC cluster with four Raspberry PI 3 model B deployed using the HPC-ROS package.

The test measures the amount of FLOPS (Floating Point Operations Per Second) that can be obtained from a computing infrastructure. From its web site [44]: *HPL is a software package that solves a (random) dense linear system in double precision (64 bits) arithmetic on distributed-memory computers*.

HPL uses MPI (Message Passing Interface) to split the solving of a linear system on the computing cores of a HPC cluster. MPI is a highly used parallel library, whose programming model consists on independent processes (*MPI processes*), running on distributed CPU cores and exchanging messages via network communications. Depending on the amount of cores in a HPC cluster, the technology and status of the underlying network communications, etc., the test will output a quantity of FLOPS and the time required to solve the linear system.

Generally, it requires a fair amount of tuning to obtain the maximum possible FLOPS available in a HPC cluster. However, such tuning is outside the scope of this paper because our objective is not to estimate computing power but rather evaluate HPC feasibility in HPRC environments. Nevertheless, three variables are important to set. Those are:

- *N=3600* (Linear system size i.e. matrix order)
- *P=1*
- *Q=16* and
- *Number of Block size (NBs) ={32,64,128,256}*

P and Q specifies how to map the parallel processes and P*Q should be equal to the maximum amount of cores in the HPC setting, in this case sixteen. The choices, for each of the variables, were based on information provided by the official HPL documentation [45] and the 16-core setting infrastructure. Figure 6 shows a comparison of performance using HPL over Ethernet versus Wi-Fi.

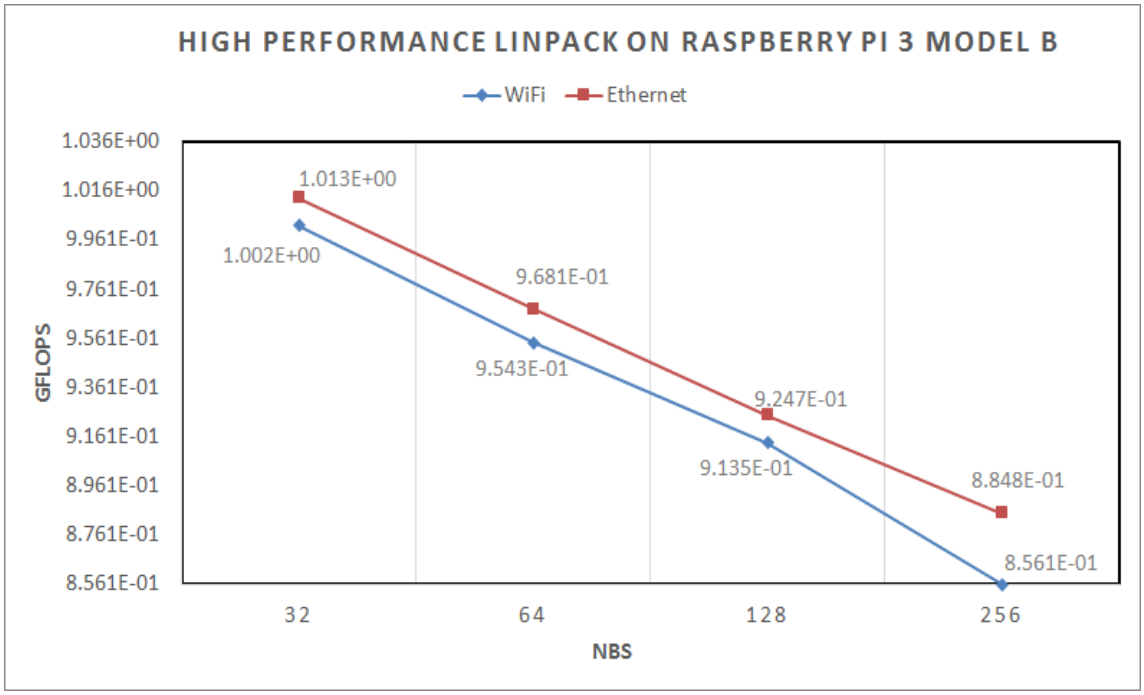

Figure 6. HPL using Ethernet and Wi-Fi. The test was performed over a 16-core setting made up of four raspberry PI model B. Ethernet and Wi-Fi performance are very similar.

The max amount of FLOPS of *1.013* Giga FLOPS (GFLOPS) is obtained with 32 NBs over Ethernet. As it can be observed in figure 6, performance is quite similar when using Ethernet and Wi-Fi. However, the Wi-Fi network used for the test is in proximity to the Raspberry Pi cards, within around 1 to 2 meters from the router and it is not used for heavy internet search or communications-consuming activities.

In average, Wi-Fi has a *1.742 %* less performance that Ethernet, which suggests that traditional HPC is valid over wireless communication channels. In a closed space, i.e. a Wi-Fi area, a HPRC cluster could operate at seemly the same performance as a wired traditional HPC cluster provided full Wi-Fi coverage and the network not used for heavy Internet search. On the contrary, in open space, e.g. a LTE area, HPC software such as HPL (MIMD), which relies on network message exchange, might not operate efficiently.

In order to resemble a LTE scenario, where connectivity among nodes might not be constant, we perform experiments consisting on node disconnection and reconnection. Purposely, we disconnect one, two and three nodes during the execution of the HPL test with a time window of *30*, *60*, *90* and *120* seconds; following we reconnect the nodes.

Node disconnection is a highly possible scenario when using wireless communications, especially with robots such as UAVs out of *line of sight*, which can disconnect when entering areas without network coverage, e.g. when using 4G or 5G. However, in *line of sight* it is a rare occurrence to have long-term disconnections like those evaluated in the following, i.e. 30, 60 and 90 seconds.

Table 6. Performance decay with node disconnection over Wi-Fi communications

Disconnection / time window	**30 s**	**60 s**	**90 s**	**120 s**
One-node	63.802 %	81.377 %	81.248 %	87.914 %
Two-node	68.543%	81.307 %	81.357 %	87.465 %
Three-node	68.29 %	81.447 %	81.707 %	86.756 %

We use the maximum Wi-Fi performance (*1.002 GFLOPS*) as the baseline value for benchmarking without node disconnection. Table 6 shows the percentage of performance decay for the three cases (one, two and three nodes disconnection) during the selected time windows (60, 90 and 120 seconds). As it can be observed, HPC performance significantly drops with node disconnection. However, HPL does not crash. This is an advantage of the Open MPI implementation where messages to be shared among processes are queued until the communications channel is available. Such queuing however is not endless; it actually depends on the network channel being used, the MPI implementation, etc.

In addition, during node disconnection, distributed processes can still perform its assigned computation. This is the reason why the performance decrease is not linear and it is similar between the studied cases. This behavior relates to HPL algorithm [46].

The results suggest that fully distributed software (MIMD), relying heavily on message exchange, might not achieve high performance with HPRC clusters nowadays. Limitations when using MIMD, especially with heavy (large-data) and constant message exchange, should be kept in mind when designing robotic missions using HPRC settings.

Nowadays, focus should be given to SIMD especially, or lite data message exchange. In addition, the programmer should guarantee that distributed tasks are able to complete even in network failing conditions. However, using the scalability feature of HPRC, software has the potential of adapting to underlying network technologies, therefore, in a nearby future, when communications have achieved expected features, we could have powerful traditional HPC carried out with HPC cluster of robots.

In the meantime, many missions might consist on splitting a given area, where each PRCN will perform a set of tasks. Since each PRCN can focus on its corresponding area, message exchange is minimum. This is an example of a SIMD approach.

Additionally, HPRC is more than a speeding up tool like its counterpart traditional HPC. In the next section 4.2, we introduce a SIMD and MIMD software, written with MPI, to control the coordinate motion of four SITL UAVs. These experiments demonstrate three HPRC features: *user-transparency*, *scalability* and *centralization / distribution*.

4.2. Parallel multi-robot motion

Previous attempts in robotics research have used HPC as a speeding up tool for single robots' complex computing tasks such as vision-based motion; however, it is our belief that HPC software has never been used to actually control the motion of a multi-robot system, which is not inherently a task requiring traditional HPC.

Given the inherit moving nature of PRCNs, we present in this section a Python software able to perform the coordinated motion of any number of robots using two approaches / models: SIMD and MIMD. In the first approach (SIMD), which stands for Single Instruction Multiple Data, there is no communication between the moving robots, in this case UAVs. In the second approach, the UAVs constantly exchange messages in order to fly resembling a birds' swarm according to a simplification of the renowned Vicsek model [47].

For both approaches, we use the hardware platform showed in figure 5 plus a laptop computer. In each of the Raspberry PI, we deploy the ArduPilot SITL software [33]. The ArduPilot SITL allows simulating what would be a real flight via software rather than the actual hardware autopilot. In order to interact with it, we use the open source DroneKit [32] Python library by 3D Robotics [48].

DroneKit consists of a set of python functions to interact via MAVLink (Micro Air Vehicle Communication Protocol) [49] with an automatic pilot. It supports different types of vehicles such as copters, rovers and planes and it can be installed in different companion computers [50], including the Raspberry PI. Furthermore, DroneKit allows interaction also with simulated autopilots such as the ArduPilot SITL.

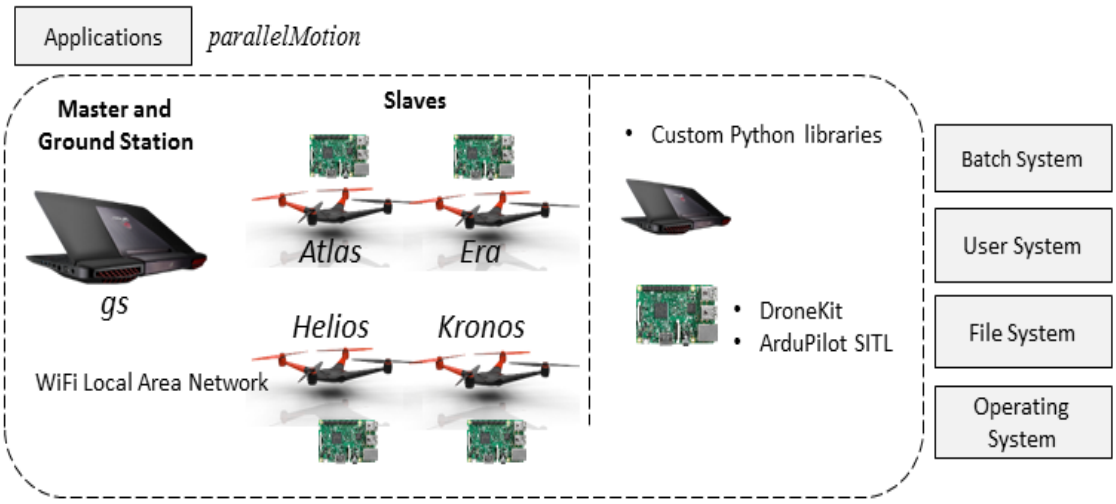

Figure 7. Parallel UAV motion Software Architecture. The *parallelMotion* software belongs to the applications layer of the HPRC cluster. The setting also implements the remaining HPC software layers.

Figure 7 shows the software architecture for the parallel multi-robot motion software. The software, depicted as *parallelMotion* is executed over an HPRC cluster, which implements all the HPC software layers. For the testing of it, we use the second setting consisting of a common laptop configured as the master node / ground station and four slaves / UAVs (Raspberry PIs). The five nodes codenames are *gs* (Master node / ground station) and *Atlas*, *Era*, *Helios* and *Kronos* (slaves, UAVs)

In order to provide parallelism to *parallelMotion*, we use *mpi4py* [51], a Python library for the development of parallel MPI software. The library requires an existing installation of an MPI implementation, in this case Open MPI.

While traditional MPI software is written mostly with C or Fortran, our decision for using Python lies with the fact that DroneKit is written in Python. Moreover, a set of custom Python libraries for the setting and use of HPRC-based multi-robot systems have been developed for this work. Such libraries are used by *parallelMotion*. However, such libraries are out of the scope of this work. It is fair stating that Python has become mainstream in robotics research and industry. However, programming languages such as C and Fortran generally provide higher performance than Python.

MPI software is executed by a set of parallel processes uniquely identified by a rank (*0*, *1*... *N* having *N+1* parallel processes). Depending of the particular rank a process has, it will be its purpose. For example, a process with rank *0* might execute different computing functions that another process having rank *1*.

The processes are distributed to one or multiple computers, and are managed and orchestrated by the MPI implementation (Open MPI), which provides features such as scalability, resilience, etc. MPI provides scalability by allowing the user to request any amount of processes, subjected to a set of rules:

1. Given a set of P processes and a set of N computing cores (distributed on one or several computers), Open MPI splits the processes among the computing cores trying to match one process per core. If $P < N$, some of the N cores will not be assigned a process. Conversely if $P > N$, some cores will be assigned more than one process. In

this case, the actual amount of processes that are running in parallel depends on the internal operating system scheduling mechanism, but normally there will not be *P* processes running in parallel.
2. If the ratio between *P* and *N* is larger than *one*, there is a maximum amount of processes that Open MPI can handle. This depends of the type of CPUs being used.
3. It is generally recommended to have a ratio between *P* and *N* of *one*. This way, there are in fact *P* parallel processes.

Furthermore, the MPI middleware provides resilience by queuing messages to be exchanged between processes, until the intended recipient is available as demonstrated by the node disconnection experiments showed in section 4.1. Finally, MPI handles the communications between parallel processes and it is possible for the user to request a mapping between the processes and the nodes' cores by using a *rank file*. For example, rank *0* can be bound to a specific core in a specific node, etc.

Figure 9 shows the parallel algorithm of *parallelMotion*. Five processes, each one representing a node in the HPRC setting are launched using a specific mapping. This mapping is carried out with a rank file (*rankfile*) which requests the execution of one MPI process in each of the nodes. The rank file also guarantees that the process with rank 0 is executed in the laptop computer (master / ground station). The command to launch the software is:

mpirun -np 5 --rankfile ***rankfile*** *python* ***parallelMotion***, where:

The rank file (*rankfile*) includes the following lines:

```
rank 0=gs     slot=0
rank 1=atlas  slot=0
rank 2=era    slot=0
rank 3=helios slot=0
rank 4=kronos slot=0
```

Since, each of the Raspberry PI comes with four computing cores, only one is used for the motion of the robot while the remaining three can be used for other tasks, e.g. image processing, etc. In addition, using the batch system, each core can be securely used for the requested purpose, without being shared by other processes. The pilot user can start the autonomous motion of the entire multi-robot system simply by executing the previous mpirun command, forgetting about the underlying complexity, i.e. user-transparency. Using MPI software is advantageous from a software development point of view as well, because a single software, executed by different parallel processes, can be written in a single module, rather than have different modules for each of the distributed nodes, in this case, PRCNs.

In this testcase, the mission consists of one task, the motion of the UAVs; details about the high-level language for the description of the mission and the matching service mechanisms are not introduced in here, in order to be presented in a further work as previously mentioned.

Following, figure 8 show *parallelMotion* algorithm in which it can be observed two modes of operation (simple and Vicsek). The software is launched from the master /

ground station node but each process executes its corresponding function according to its specific rank.

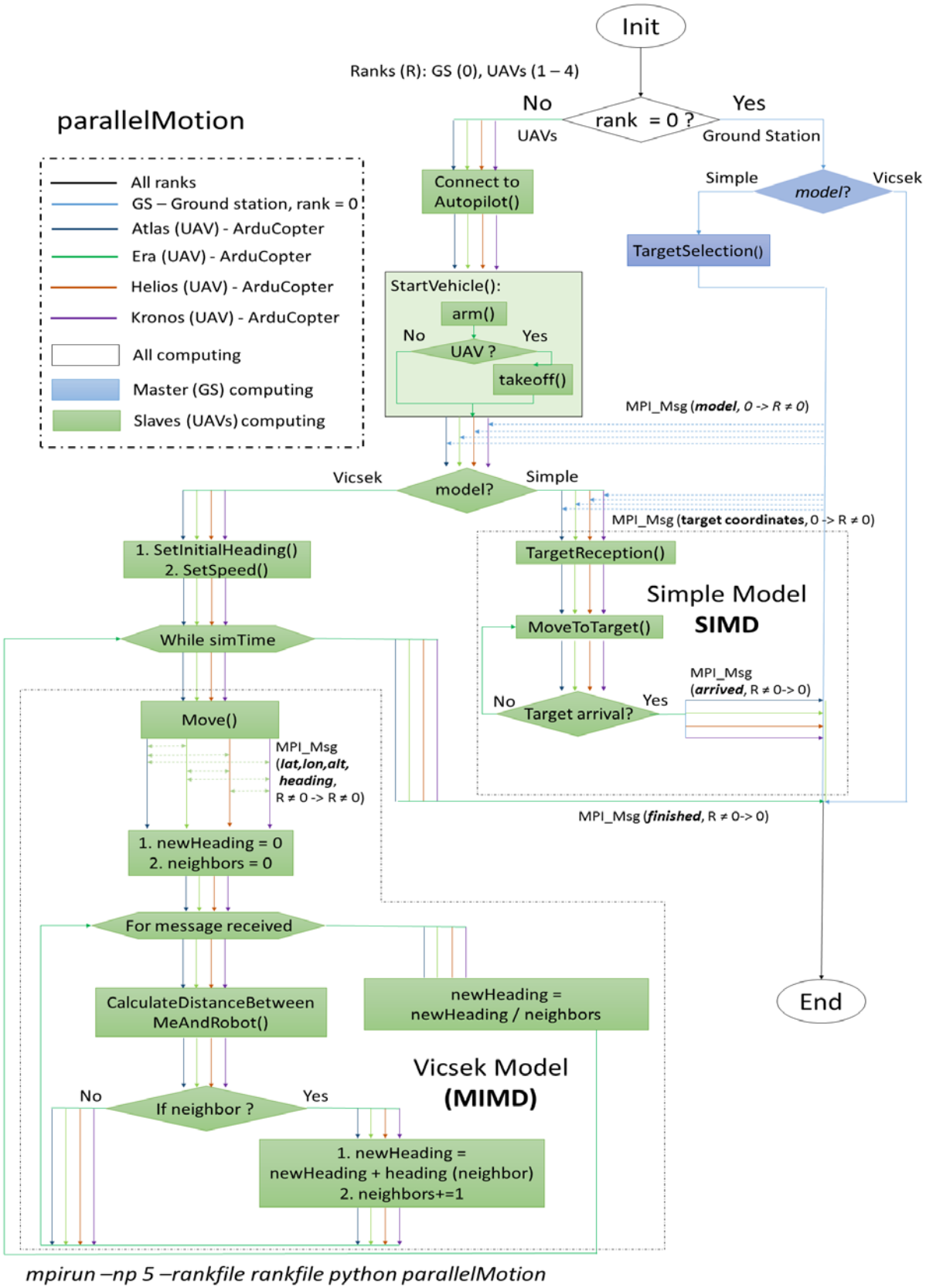

Figure 8. Parallel Multi-robot motion algorithm based on MPI. The software provides fully autonomous motion for the multi-robot setting with two modes of operation, SIMD and MIMD

As it can be observed in figure 8, there are two different set of functions carried out according to the process rank. The ground station (process with rank *0*) functions and the functions of the ranks *1* to *4* (UAVs). This can be observed in the first conditional (*triangular shape rank = 0?*). The explanation is as follows, all processes start the execution of the software and once they reach the first conditional, the ground station process will proceed with the right part of the algorithm while the remaining ranks (UAVs) will proceed with the left part. Following, depending on the selected model / approach (i.e. simple - SIMD and Vicsek - MIMD), each of the process will continue with its specific part. The ground station process is the one in charge of sending a message specifying the model to follow (*model MPI_Msg instruction in figure 8*).

In the first approach, which we depicted as *simple model*, the ground station requests the UAVs to fly to a specific target. This request is done via the *target coordinates MPI_Msg instruction in figure 8*. The simple model is a SIMD approach because the UAVs are performing the same functions (flying to the specific target) but over different locations (data), i.e. each UAV fly from a different home location. Furthermore, the UAVs do not exchange messages. This means that the UAVs are fully independent of each other. Figure 9 shows the motion of the four UAVs using the simple model.

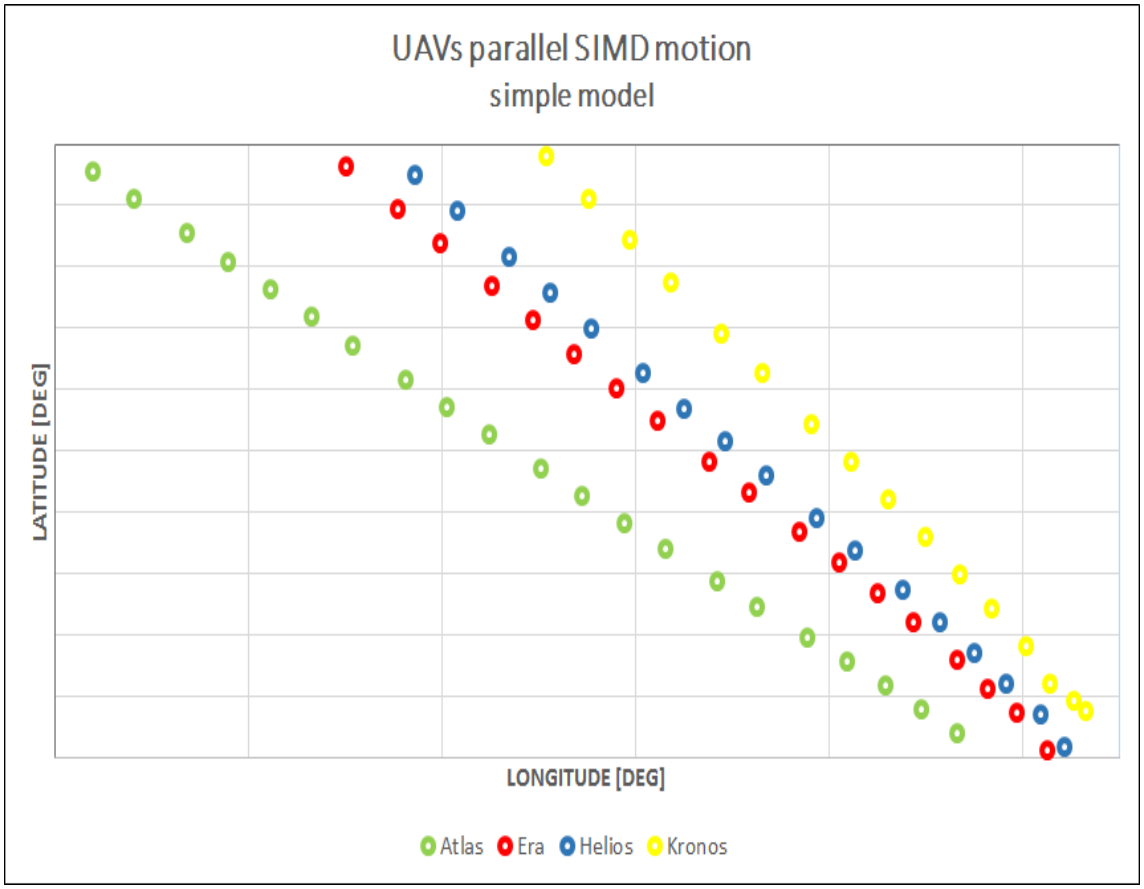

Figure 9. Parallel UAV motion - Simple model. The simple model is a SIMD approach where there is no message exchange between the parallel processes representing the UAVs while moving

For certain type of missions, the fact that the UAVs are independent from each other (SIMD simple model), results advantageous. Consider a mission in which a set of UAVs are used to monitor a hostile area (target). If some of the UAVs were to be compromised, the remaining UAVs, by being independent could continue with the mission. Moreover,

the cost of the communications with this approach is minimum and consists only on the messages required by the MPI middleware to manage the parallel processes but no user MPI message is required after the initial model and target coordinates messages are sent from the ground station to the UAVs.

For the second approach, the Vicsek model [47] is based on two ideas:

1. Each entity (e.g. bird, UAV) flies with constant speed and direction (heading) equal to the average of the directions of its neighbors. An entity *i* is neighbor of entity *j* if at time *t*, *i* and *j* are at a distant less or equal than a selected radius.
2. The noise in the system. Two types of noise are considered: *extrinsic* and *intrinsic*. Extrinsic noise refers to the one caused by the environment. In biological systems, it could relate to the entities (e.g. birds) not able to see properly if another entity is a neighbor. In artificial systems, extrinsic noise might relate to GPS precision issues. Conversely, intrinsic noise refers to the entities' decision to move in certain direction even if they fully understand the direction of its neighbors. In artificial systems, this could relate to computing errors at execution time.

For the purpose of this work, *parallelMotion* does not implement the noise feature of the Vicsek model because the parallel motion software is used to show the ideas behind HPRC. Therefore, *parallelMotion* would require further development if it is to be used on real flights. Figure 10 shows the motion of the four UAVs using Vicsek model.

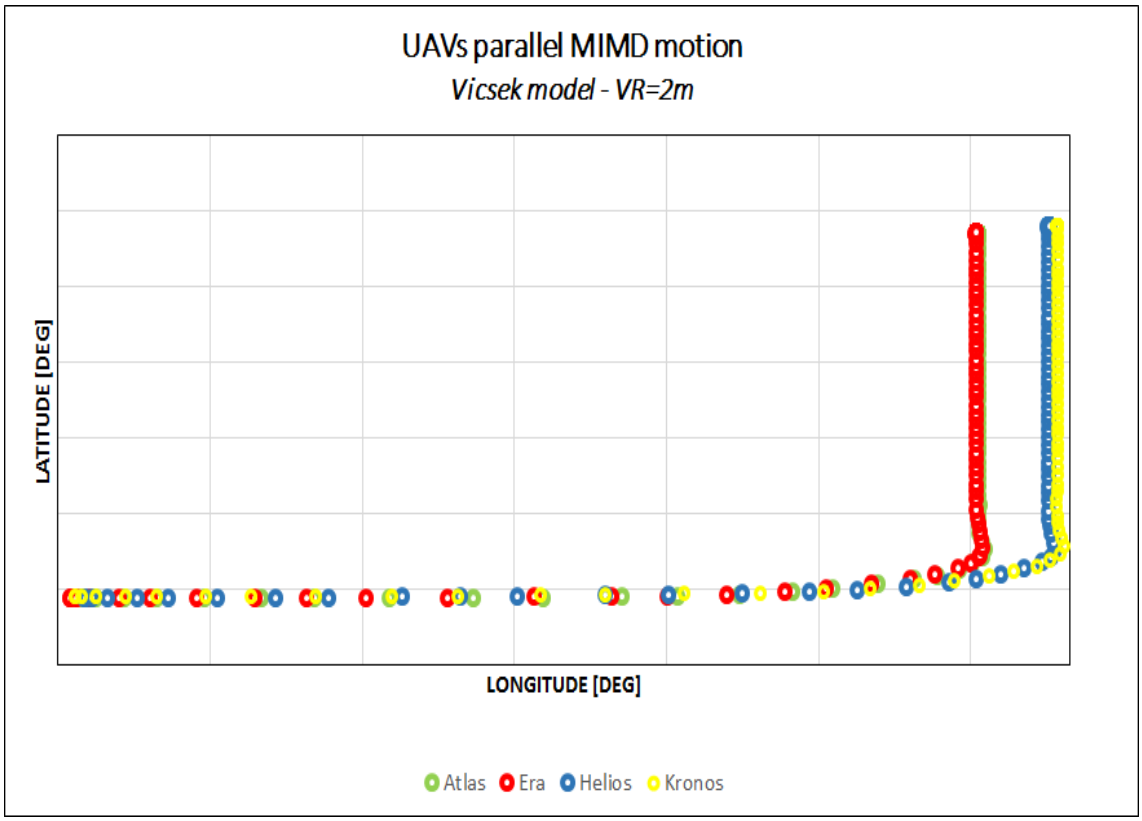

Figure 10. Parallel UAV motion - Vicsek model. The Vicsek model is a MIMD approach where there is constant message exchange between the parallel processes representing the UAVs while moving

As it can be observed in figure 10, the UAVs, moving from different home positions, approach each other until they resemble a birds' flock. The Vicsek model, provided by

parallelMotion is based on a MIMD approach, where not all the UAVs perform the same actions, given than during execution time, some UAV might have neighbors or not. If an UAV were to have neighbors, it will change its heading according to the Vicsek model feature one. Furthermore, with this approach there is constant exchange of messages (*lat, lon, alt, heading MPI_Msg instruction in figure 8*). These messages inform the UAVs with the location and heading of the other UAVs, in order to calculate the quantity of neighbors each rank has. The overlapping between UAVs observed in figure 11 is because *parallelMotion* does not implement a collision avoidance mechanism. However, this basic algorithm depicts the powerful environment provided by parallel software.

The use of MIMD software is subjected to more possible errors than SIMD, especially when facing wireless communications. Nevertheless, the scalability features provided by MPI could lead in a no-distant future to have thousands of UAVs flying very similar to a spectacular birds flock.

The maximum amount of parallel processes that can be handled by the MPI middleware depends on the quantity of computing cores, the quality of the subjacent communications technology (e.g. LTE, Wi-Fi, etc.) and the quantity, size and frequency of exchanged messages. If these conditions are met and optimized, it is possible to build very interesting multi-robot systems carrying out all types of missions.

MPI is a worldwide renowned HPC standard technology maintained by a large community of developers. Newer versions are constantly released and its use is well documented in several fields of science. The stability provided by MPI allows using the same software with any number of moving robots, by simply modifying the mpirun command and the rankfile, providing therefore *scalability*.

Moreover, *parallelMotion* behaves in a centralized / distributed approach. The master process (ground station, rank *0*) acts as a single interface for a pilot to control a multi-robot system (*centralization*), while each UAV (ranks different from zero) have sufficient autonomy to keep moving or performing its mission's tasks (*distribution*). Furthermore, since the software is launched from the master node, the user does not need to be aware of the subjacent complexity of the HPRC system (*user-transparency*). Finally, *parallelMotion* by means of using a set of customized Python libraries and DroneKit can be used to control the motion of different type of vehicles, e.g. copters, planes and rovers. Even some of the PRCNs could be UAVs, others UGVs, etc., i.e. *heterogeneity*.

The *parallelMotion* software is fully autonomous, i.e. the user (pilot) only needs to launch the software using the *mpirun* command and afterwards the vehicles will move autonomously. However, future work will include a hybrid controlled / autonomous scheme allowing the pilot to be involved during autonomous motion if desired.

The motion of a multi-robot system is not a computationally expensive task that would require HPC understood as a speeding up tool, except in vision-based motion. However, as proposed by HPRC, a multi-robot system motion can be carried out with traditional HPC technologies, such as MPI, providing therefore scalability, centralization / distribution and user-transparency.

5. Conclusions and future work

In this work, we have introduced the novel concept of *High Performance Robotic Computing (HPRC),* an augmentation of the ideas behind traditional High Performance Computing, for the world of robotics. Previous works have suggested the use of HPC parallel software libraries such as CUDA or MPI for computationally demanding robotic tasks. In such works, HPC has always been understood as a set of tools and infrastructures for speeding up tasks that are too slow running on single computing units. However, HPRC goes beyond this by proposing means of achieving scalability, centralization / distribution, user-transparency, etc., in missions carried out by multi-robot systems, where *tasks could be or not computationally demanding*.

The scalability feature requires further research if multi-robot systems composed of hundreds or even thousands of robots are to be controlled by MPI software. The results shown in this work were obtained with a small number of parallel processes and while each Raspberry Pi was set with an SITL autopilot, MPI communications strategies need to be addressed if the number of robots increases. Nevertheless, two aspects are promising. First, MPI implementations have been compiled for ARM architectures such as the Raspberry Pi and second, MPI research is targeting the exascale computing era, where millions of cores are expected to be used for single simulations. This research approach follows the success of using MPI in the current petascale era.

Within the scope of HPRC, a robot becomes a general multi-purpose computing device, which opens a completely new spectrum of possible robotic applications. In this work, we have demonstrated that a full HPRC cluster can be deployed with traditional HPC software such as MPI, PBS torque, etc., on embedded computing devices such as the Raspberry PI. This way, with new embedded computing devices and communications technologies appearing constantly, full HPC cluster of robots, resembling those with wired technologies are a true possibility. The emergence of 5G communications in the nearby future is a very promising aspect that could allow solving the performance decay issues shown in section 4.1. However, nowadays the design of robotic missions using HPRC should take into consideration current network limitations. Regardless, as future work, we plan to run the same experiments, i.e. running HPL and parallelMotion on real unmanned vehicles while flying using Wi-Fi and 4G LTE connectivity.

To demonstrate that HPRC is more than a speeding up tool for robotic tasks, we have introduced a parallel motion software for multi-robot systems depicted as *parallelMotion*. The software requires further development to be used in real robots but it potentially demonstrates that MPI software can be used for controlling the motion of all kind and number of robots such as UAVs and UGVs. Furthermore, it shows that HPRC is not only usable for computationally demanding tasks but it can be actually be a bedrock for scalable multi-robot systems exploiting all HPC features. The Python library *mpi4py* used by *parallelMotion* does not currently allow user interaction at execution time. Given such limitation, we are revising other Python libraries for MPI in order to improve *parallelMotion* by allowing pilot dynamic control during vehicles' motion while maintaining autonomous behavior.

As a new emergent field, HPRC is still in its infant phase and requires more research to be implemented in real world applications but its potential is enormous. In fact, an integration between traditional HPC and HPRC systems is an interesting path to follow. Computing is everywhere nowadays, from sensors up to powerful servers and while HPC is used in several endeavors, we believe that it is time to propose the appearance of *Ubiquitous Supercomputing*, as the merging of HPC and HPRC.

In future works, we aim at introducing Ubiquitous Supercomputing and define its philosophy, infrastructures, technologies, etc. Such approach requires a set of services to guarantee the HPC features in ubiquitous supercomputing systems. The custom Python libraries, used by *parallelMotion*, already implement such of those services but they required further development. Moreover, the concept of general-purpose computing mission is interesting because it binds a robotic task with a software, which favors the reutilization of a multi-robot system by simply changing the software associated to each task and the reconfiguration of the mission network.

Future work includes as well testing *parallelMotion* with real robots. The transition for the simulated scenario presented in here is a straightforward endeavor because of the use of DroneKit, which was designed to allow easy deployment from SITL to real robots. Furthermore, *parallelMotion* was developed to facilitate such transition.

In this work, we introduced the HPRC cluster as an adaptation of traditional HPC infrastructures such as HPC cluster of computers. However, other HPC infrastructures such as *Grid* or *Big data* could also be exploited to build powerful robotic settings. In fact, Wan et al [52] described cloud robotics as a combination between multi-robot settings and cloud computing where the former delegates complex computing tasks to the latter. HPRC does not deny the use of external computing power; it could be actually an interesting combination to have a HPRC cluster interacting with computing resources located elsewhere, i.e. in the cloud or any other computing infrastructure specially those exhibiting HPC.

As a final thought, the combination of standard technologies such as MPI and CUDA within the scope of HPRC could lead to the development of fascinating multi-robot systems where artificial intelligence finds HPC / HPRC as powerful allies.

Acknowledgments The authors would like to thank the Colombian government, in particular to the Colombian Administrative Department of Science, Technology and Innovation *Colciencias* for the funding of the Ph. D studies of the first author, which include the work presented hereby.

References

1. Francesco Cocchioni, Emanuele Frontoni, Gianluca Ippoliti, Sauro Longhi, Adriano Mancini and Primo Zingaretti. Visual based landing for an unmanned quadrotor. *Journal of Intelligent & Robotic Systems*, 84(1):511–528, 2016.

2. Dongwoon Jeon, Doo-Hyun Kim, Young-Guk Ha, and Vladimir Tyan. Image processing acceleration for intelligent unmanned aerial vehicle on mobile GPU. *Soft Computing*, 20(5):1713–1720, 2016.
3. Omer Cetin and Guray Yilmaz. Real-time autonomous UAV formation flight with collision and obstacle avoidance in unknown environment. *Journal of Intelligent & Robotic Systems*, 84(1):415–433, 2016.
4. Dà-Jiāng Innovations Science and Technology - DJI website. www.dji.com.
5. DJI Manifold embedded computer. www.dji.com/manifold
6. NVIDIA Corporation. www.nvidia.com.
7. Owen Holland, John Woods, Renzo de Nardi, and Adrian Clark. Beyond swarm intelligence: the UltraSwarm. In *Proceedings of IEEE Swarm Intelligence Symposium (SIS)*, 217–224, 2005.
8. Ali Marjovi, Sarvenaz Choobdar and Lino Marques. Robotic clusters: Multi-robot systems as computer clusters: A topological map merging demonstration. *Robotics and Autonomous Systems*, 60(9):1191–1204, 2012.
9. Michael Garland, Scott Le Grand, John Nickolls, Joshua Anderson, Jim Hardwick, Scott Morton, Everett Phillips, Yao Zhang and Vasily Volkov. Parallel computing experiences with CUDA. *IEEE micro*, 28(4), 2008.
10. Michael J Flynn. Some computer organizations and their effectiveness. *IEEE transactions on computers*, 100(9):948–960, 1972.
11. Leonardo Dagum and Ramesh Menon. Open MP: an industry standard API for shared-memory programming. *IEEE computational science and engineering*, 5(1):46–55, 1998.
12. David W Walker and Jack J Dongarra. MPI: a standard message passing interface. *Supercomputer*, 12:56–68, 1996.
13. Raspberry PI Foundation. www.raspberrypi.org/. Online. Accessed December-2017.
14. Edgar Gabriel, Graham E Fagg, George Bosilca, Thara Angskun, Jack J Dongarra, Jeffrey M Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine et al. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Springer European Parallel Virtual Machine/Message Passing Interface Users Group Meeting*, 97–104, 2004.
15. NVIDIA GPUs social impact. www.nvidia.com/object/social-impact-gpu.html. Online. Accessed December-2017.
16. NVIDIA Jetson TK1. www.nvidia.com/object/jetson-tk1-embedded-dev-kit.html. Online. Accessed December-2017.
17. Christoforos Kanellakis and George Nikolakopoulos. Survey on computer vision for UAVs: Current developments and trends. *Journal of Intelligent & Robotic Systems*, 87(1):141–168, 2017.
18. NVIDIA Jetson solutions for Drones and UAVs. www.nvidia.com/en-us/autonomous-machines/uavs-drones-technology/. Online. Accessed December-2017.
19. NVIDIA Tegra K1 processor. www.nvidia.com/object/tegra-k1-processor.html. Online. Accessed February-2018.
20. Esther Salami, José Soler, Raúl Cuadrado, Cristina Barrado and Enric Pastor. Virtualizing super-computation on-board UAS. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 40(7):1291, 2015.

21. Open Source Computer Vision Library. www.opencv.org. Online. Accessed December-2017.
22. Constantino Gonçalves Ribeiro, Max Suel Dutra, Aline Rabelo, Filipe Oliveira, Allan Barbosa, Luciano Frinhani, Douglas Porto and Rayanne Milanez. A robotic flying crane controlled by an embedded computer cluster. In *12th Latin American Robotics Symposium and 3rd Brazilian Symposium on Robotics (LARS-SBR)*, 91–96, 2015.
23. ODROID X2. www.hardkernel.com/main/products/prdt_info.php?g_code=G135235611947. Online. Accessed December-2017.
24. Wolfgang Gentzsch. Sun grid engine: Towards creating a compute power grid. In *Proceedings of First IEEE/ACM International Symposium on Cluster Computing and the Grid*, 35–36, 2001.
25. Georg Hager and Gerhard Wellein. Introduction to High Performance Computing for Scientists and Engineers, Second Edition. *Taylor & Francis*, 2016
26. Thomas Sterling, Anderson Matthew and Maciej Brodowicz. High Performance Computing: Modern Systems and Practices. *Morgan Kaufmann*, 2017.
27. Qualcomm Technologies website. www.qualcomm.com/. Online. Accessed December-2017.
28. Qualcomm Technologies releases LTE drone trial results. www.qualcomm.com/news/onq/2017/05/03/qualcomm-technologies-releases-lte-drone-trial-results. Online. Accessed December-2017.
29. International Telecommunications Union (ITU). www.itu.int. Online. Accessed February-2018.
30. International Telecommunications Union (ITU). Minimum requirements related to technical performance for IMT-2020 radio interface(s). www.itu.int/md/R15-SG05-C-0040/en. Online. Accessed February-2018.
31. Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler and Andrew Y Ng. ROS: an open-source robot operating system. In *ICRA workshop on open source software*, 3:5, 2009.
32. DroneKit by 3D Robotics. dronekit.io. Online. Accessed December-2017.
33. ArduPilot Software In The Loop - SITL. ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html. Online. Accessed December-2017.
34. PBS Torque Resource Manager. www.adaptivecomputing.com/products/open-source/torque/. Online. Accessed December-2017.
35. SLURM Workload Manager. slurm.schedmd.com. Online. Accessed December-2017.
36. Maryam Zamani, Leonardo Camargo-Forero and Tamas Vicsek. Stability of glassy hierarchical networks. *New Journal of Physics*, 20, 2018.
37. Leonardo Camargo Forero, Pablo Royo and Xavier Prats. On-board high-performance computing for multi-robot aerial systems. In *InTech Aerial Robots - Aerodynamics, Control and Applications*, 7, 2017.
38. Spencer Shepler, Mike Eisler, David Robinson, Brent Callaghan, Robert Thurlow, David Noveck and Carl Beame. Network File System (NFS) version 4 protocol. *Network*, 2003.
39. Kurt Zeilenga. Lightweight directory access protocol (LDAP): Technical specification road map. 2006.

40. How to build a $200 smart drone with the Pi Zero. www.zdnet.com/article/how-to-build-a-200-smart-drone-with-the-pi-zero/. Online. Accessed December-2017.
41. Top 500 Supercomputers. www.top500.org. Online. Accessed December-2017.
42. Intel Xeon Phi processors. www.intel.com/content/www/us/en/products/processors/xeon-phi.html. Online. Accessed December-2017.
43. Raspberry Pi 3 Model B. www.raspberrypi.org/products/raspberry-pi-3-model-b/. Online. Accessed December-2017.
44. HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers. www.netlib.org/benchmark/hpl/. Online. Accessed December-2017.
45. Frequently Asked Questions on the Linpack Benchmark and Top500. www.netlib.org/utk/people/JackDongarra/faq-linpack.html. Online. Accessed December-2017.
46. HPL Algorithm. www.netlib.org/benchmark/hpl/algorithm.html. Online. Accessed December-2017.
47. Tamás Vicsek, András Czirók, Eshel Ben-Jacob, Inon Cohen and Ofer Shochet. Novel type of phase transition in a system of self-driven particles. *Physical review letters*, 75(6):1226, 1995.
48. 3D Robotics. 3dr.com. Online. Accessed January-2018.
49. Micro Air Vehicle Communication Protocol MAVLink. qgroundcontrol.org/mavlink/start. Online. Accessed December-2017.
50. DroneKit by 3D Robotics companion computers. python.dronekit.io/develop/companion-computers.html. Online. Accessed December-2017.
51. MPI4PY. MPI for Python. mpi4py.scipy.org/docs/. Online. Accessed January-2018.
52. Jiafu Wan, Tang Shenglong, Yan Hehua, Li Di, Wang Shiyong, and Vasilakos Athanasios V. Cloud robotics: Current status and open issues. *IEEE Access*, 4: 2797-2807, 2016.