GRIDHPC: A decentralized environment for high performance computing

Bilal Fakih, Didier El Baz, Igor Kotenko

GRIDHPC, A Decentralized Environment for High Performance Computing

Bilal FAKIH[1], Didier EL BAZ[1,2], and Igor KOTENKO[2]
[1]LAAS-CNRS, Université de Toulouse, CNRS, France
[2]ITMO University, St. Petersburg, Russia
{bfakih,elbaz}@laas.fr
ivkote@comsec.spb.ru

***Abstract*—This paper presents GRIDHPC, a decentralized environment dedicated to high performance computing. It relies on the reconfigurable multi network protocol RMNP to support data exchange between computing nodes on multi network systems with Ethernet, Infiniband, Myrinet and on OpenMP for the exploitation of computing resources of multicore CPU. We report on scalability of several parallel iterative schemes of computation combined with GRIDHPC. In particular, the experimental results show that GRIDHPC scales up when combined with asynchronous iterative schemes of computation.**

***Keywords*-Grid computing, High Performance Computing, Computing environment, Heterogeneous networks, Loosely synchronous applications, Asynchronous iterations.**

I. Introduction

In this paper, we present the decentralized environment GRIDHPC dedicated to High Performance Computing (HPC) on grid platforms. HPC applications that we consider are basically loosely synchronous applications like the solution of numerical simulation problems [1] that present frequent data exchanges between computing nodes. GRIDHPC facilitates the use of large scale distributed systems and the work of programmer. In particular it uses a limited number of communication operations.

The GRIDHPC environment allows data exchange between computing nodes with multi network, multi-core configurations. It relies on the Reconfigurable Multi Network Protocol (RMNP) to support data exchange on multi-network systems and on OpenMP [2] for the exploitation of computing resources of multi-core CPU.

The protocol of communication RMNP is an extension of the Configurable Transport Protocol (CTP) [3] that makes use of the Cactus framework [4]. RMNP can configure itself automatically and dynamically in function of application requirements like scheme of computation that is implemented, i.e., synchronous or asynchronous iterative schemes and elements of context like available network interface cards and network topology by choosing the most appropriate communication network and mode between computing nodes. It can use simultaneously several networks like Ethernet, Infiniband and Myrinet. These features are particularly important since we consider loosely synchronous applications that present frequent data exchanges between computing nodes. To the best of our knowledge, these features have not been carried out previously on environments or runtime systems in the literature.

The remainder of the paper is organized as follows. Related work is presented in Section II. Section III deals with our contribution to support communication in a multi-network context. In particular, the Reconfigurable Multi Network Protocol RMNP is presented. Section IV presents the architecture and task assignation of the GRIDHPC environment. Parallel programming model is given in section V. Computational results with the decentralized environment GRIDHPC for the obstacle problem are displayed and analyzed in section VI. Section VII concludes this paper.

II. Related Work

The possibility to consider heterogeneous network resources for HPC applications goes back to Madeleine and MPICH-Madeleine [5], [6]. Recently, research has focused on runtime systems for HPC applications carried out on heterogeneous architectures that combine multi-core CPU and computing accelerators.

Heterogeneous multi-core platforms mixing CPUs and computing accelerators are nowadays widely spread. High Performance ParalleX (HPX) [7], [8] is a C++ runtime system for parallel and distributed applications, some of which are loosely synchronous applications. It has been developed for systems of any scale and aims to address issues like resiliency, power efficiency. It has a programming model unifying all types of parallelism available in HPC systems which uses the available resources to attain scalability. It is portable and easy to use. It is published under an open-source license and has an active user community. It is built using dynamic and static data flow, fine grain future-based synchronization and continuation style programming. The main goal of HPX is to create an open source implementation of the ParalleX execution model [9] for conventional systems like classic Linux based Beowulf clusters, Android, Windows, Macintoch, Xeon/Phi, Bluegene/Q or multi-socket highly parallel SMP nodes.

HPXCL [10] and APEX [11] are libraries which pro-

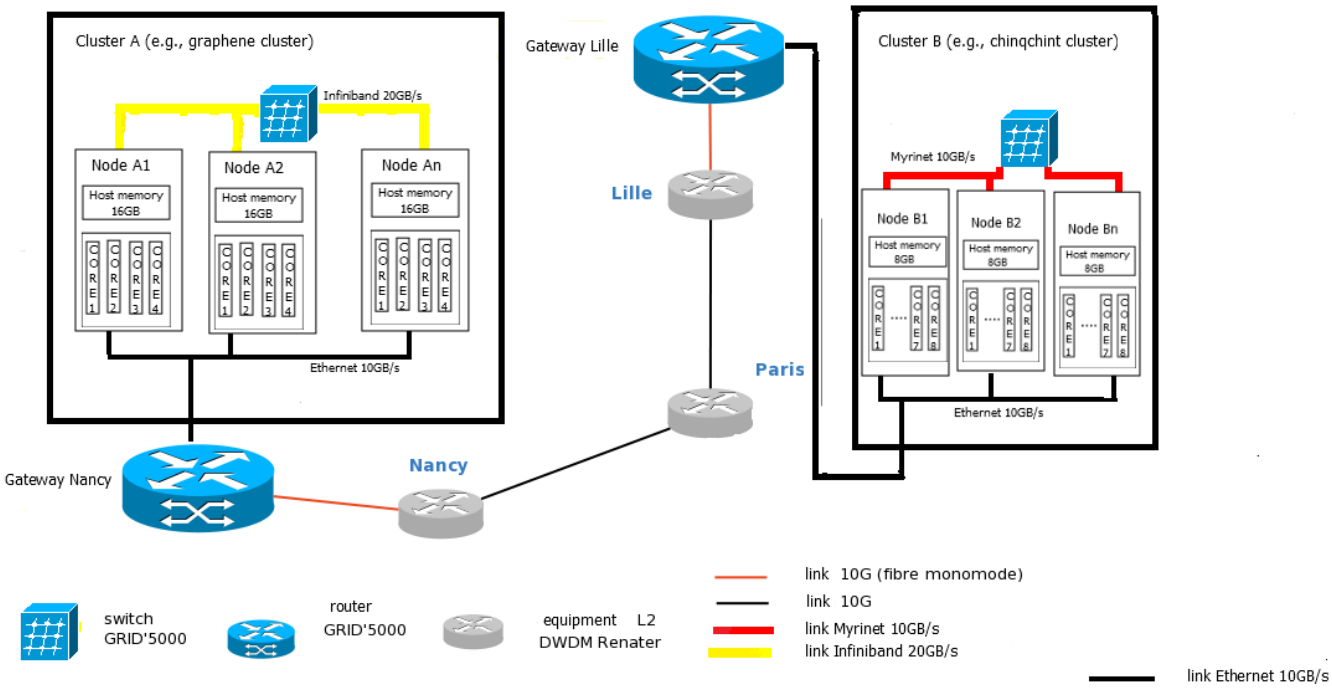

Figure 1: Example of Multi-Network distributed platform

vide additional functionality that extend the HPX. HPXCL, allows programmers to incorporate GPUs into their HPX applications. Users write an OpenCL kernel and pass it to HPXCL which manages the synchronization and data offloading of the results with the parallel execution flow on the CPUs. APEX, gathers arbitrary information about the system and uses it to make runtime-adaptive decisions based on user defined policies.

StarPU [12] is a runtime system that provides an interface to execute parallel tasks over heterogeneous hardware (multi core processors and computing accelerators) and develop easily powerful scheduling algorithms. It is based on the integration of the data-management facility with a task execution engine. The main components of StarPU are a software distributed shared memory (DSM) and a scheduling framework; DSM enables task computations to overlap and avoid redundant memory transfers. The scheduling framework maintains an up-to-date and a self-tuned database of kernel performance models over the available computing tasks to guide the task mapping algorithms. We note that middle layers tools like programming environments and HPC libraries can build up on top of StarPU to allow programmers to make existing applications exploit different accelerators with limited effort.

To the best of our knowledge, the above runtime systems do not handle multi-network contexts that we find typically in grids. In the next sections, we show how we have addressed this issue and we propose a decentralized environment for multi-network and multi-core grid platforms.

III. Communication Protocol

This section presents the mechanisms that support multi-network communications.

A. Heterogeneous Multi-Cluster Environment

Figure 1 displays a multi-network platform with interconnected stand-alone computing nodes that can work cooperatively as a single integrated computing resource. In particular, Figure 1 shows the architecture of typical sets of computing nodes built around low-latency, high bandwidth interconnection network like Infiniband and Myrinet. Supporting heterogeneous networks mainly consists in integrating a functionality in order to switch from one network to another, according to the communication needs.

B. Htable

1) Distance metric: We consider a distance metric that is based on IP address. In particular, it concentrates on the third group of the IP address. For example, in the case of three computing nodes: A1 having IP address 192.16.64.10, A2 having IP address 192.16.64.11 and B1 having IP address 192.16.34.20, the value of third group of A1 and A2 is equal to 64, while the value of third group of B1 is 34. Consequently, it is deduced that A1 and A2 have the same location (they are in the same cluster), while A1 and B1 are in different locations (see Table I).

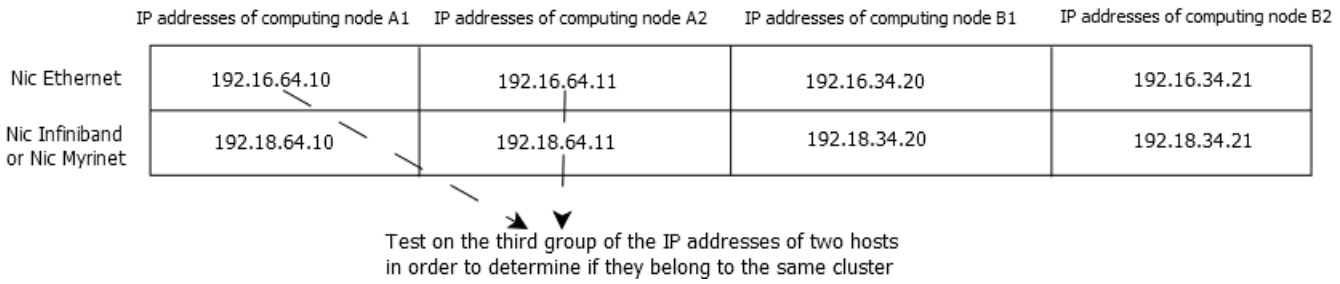

Table I: Example of content of Htable and test on the location of the hosts thanks to the comparison of IP addresses

2) Definition of Htable: Htable is designed to manage several network adapters within the same application session. In particular, Htable permits each computing node to switch between the networks according to the communication needs.
Several network interface cards (NICs) are added to the interface of the RMNP communication protocol and information about these NICs are stored in the Htable. In the Htable (see Table I), the IP addresses displayed in the first line correspond to Ethernet network and the IP addresses given in the second line, if any, correspond to fast network like Infiniband or Myrinet.

C. Reconfigurable Multi Network Protocol RMNP

The Reconfigurable Multi Network Protocol RMNP aims at enabling an efficient use of the complete set of underlying communication softwares and hardwares available in a given multi-network system that uses for example Ethernet, Infiniband and Myrinet. It is able to deal with several networks via the management of several networks adapters.

1) choice of networks: The network management procedure has two steps (see Algorithm 1). First step corresponds to the test of the locality between the computing nodes thanks to the comparison of IP addresses and the second step corresponds to the choice of the appropriate network for data exchange depending on the locality of computing nodes. In particular, the second step is based on choosing the best interface network (high bandwidth and low latency network) from the Htable according to the result of the locality test. Consequently, if the locality test returns that the considered computing nodes have different locations, then the Ethernet network interface is chosen to perform the communication between the two computing nodes. If the locality test returns that the computing nodes have the same location, then the best network interface in the Htable is selected, e.g., Infiniband or Myrinet.

2) Example of scenario: We present now a simple scenario for the RMNP communication protocol so as to illustrate its behavior. We consider a high performance computing application, like for instance a large scale numerical simulation application, solved on the network composed of two clusters shown in Figure 1. Computing nodes in cluster A own both a Fast-Ethernet card, i.e., 192.16.64.x and Infiniband card, i.e., 192.18.64.x (see Table I) and computing nodes in cluster B own both a Fast-Ethernet card, i.e., 192.16.34.x and Myrinet card, i.e., 192.18.34.x where x

Algorithm 1: Test locality and choose the best network

function Localityandbestnet $(laddress, raddress)$;
Input : laddress and raddress : Ethernet IP addresses of local and remote computing nodes

```
/* inet_ntop converts the network
   address structure into a character
   string                                  */
inet_ntop(AF_INET, &(laddress.sin_addr), l, 80) ;
inet_ntop(AF_INET, &(raddress.sin_addr), r, 80)
;
substringl ← GetValThirdGroup(l) ;
/* GetValThirdGroup function get the
   value of third group of a given IP
   address                                 */
substringr ← GetValThirdGroup(r) ;
if strcmp(substringl, substringr) == 0 then
    /* Local and Remote computing nodes
       belong to the same cluster          */
    Bestladdress ← Get(laddress) ;
    /* Get function get the best
       interface network from Htable       */
    Bestraddress ← Get(raddress) ;
    /* Communication between computing
       nodes are made via the best
       network (Infiniband or Myrinet
       Network), i.e., Bestladdress and
       Bestraddress                        */
else
    /* Local and Remote computing nodes
       are in different cluster, Hence
       communication between them are
       made via Ethernet Network, i.e.,
       laddress and raddress               */
```

is a value between 1 and 255. We suppose that we have a communication network between computing nodes like $A1 \leftrightarrow A2$, $A2 \leftrightarrow B1$ and $B1 \leftrightarrow B2$ where $X \leftrightarrow Y$ means that there is bidirectional link between X and Y. The value of third group of A1 and A2 is equal to 64. Consequently, the communication between A1 and A2 that share the same high speed networks is made via Infiniband network. The values of third group of A2 and B1 are 64 and 34, respectively. Hence, the communication between the considered computing nodes is made via Ethernet network. The value of third group of B1 and B2 is equal to 34. Consequently, the communication between the considered computing nodes is made via Myrinet network.

IV. The Decentralized Environment GRIDHPC

A. Environment Architecture of GRIDHPC

The decentralized environment GRIDHPC natively supports any combination of networks and multi-core CPUs by using the reconfigurable multi network protocol RMNP and OpenMP. Figure 2 shows the architecture of GRIDHPC. It consists of five main components [13].

1) Interface Environment Component: It is the interaction interface between the application like obstacle problem and the environment. It allows users to submit their tasks and retrieve final results.

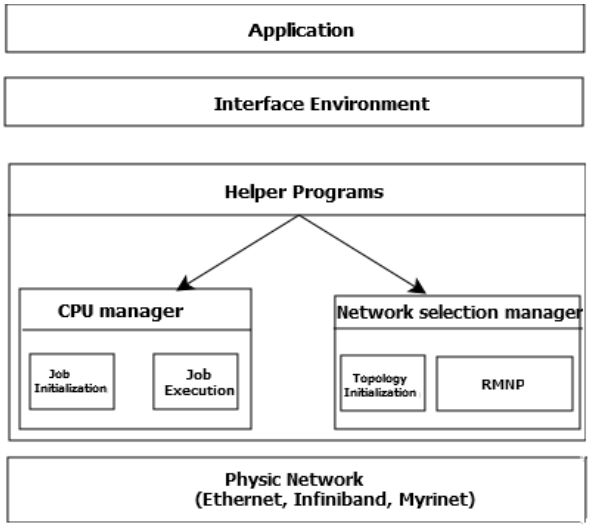

Figure 2: Environment Architecture of GRIDHPC

2) Helper Programs: GRIDHPC works with tools called helper programs that are responsible for the analysis of the application, task assignation and building the network topology. The helper programs rely on two pillars namely the CPU manager and the Network selection manager (see Fig. 2). The CPU manager is composed of Job Initialization and Job Execution. The Network selection manager is composed of Topology Initialization and RMNP-OpenMP components. In the CPU manager module, there are two components :

- Job Initialization Component is responsible for problem decomposition and assignment of tasks to individual CPU cores.
- Job Execution Component executes sub-tasks on the different CPU cores, takes care of data exchange, i.e., communication of updates produced by the parallel iterative method. At the end of the application, it regroups the results from all the computing cores.

In the Network selection manager module, there are two components :

- Topology Initialization Component organizes connected computing nodes into clusters and maintains links between clusters. It is based on storing in the Htable information regarding the network interface cards (NIC) used in the application by the different computing nodes.
- RMNP Component provides support for directed data exchange between computing nodes on several networks like Infiniband, Myrinet and fast Ethernet using the reconfigurable multi-network protocol RMNP.

We note that the CPU manager is in charge of data exchange between computing cores, i.e., read/write; while the network selection manager is in charge of data exchange between computing nodes via the best underlying network, i.e., high speed and low latency network like Infiniband and Myrinet. The combination of the CPU manager and the Network selection manager permits us to use the decentralized environment GRIDHPC in a multi-network and multi-core context.

B. Processor hierarchy and GRIDHPC

Task assignation in GRIDHPC [13] is based on a hierarchical Master-Worker paradigm that relies on three entities: a master, several sub-masters (coordinators) and several workers.

The master or submitter is the unique entry point, it gets the entire application as a single original task, i.e., root task. The master decomposes the root task into sub-tasks and distributes these sub-tasks amongst a farm of workers. The master takes also care of gathering the scattered results in order to produce the final result of the computation.

The sub-masters or coordinators are intermediary entities that enhance scalability. They forward sub-tasks from the submitter to workers and return results to the submitter limiting network congestion.

The workers run in a very simple way: they receive a message from the sub-master that contains their assigned sub-tasks and they distribute them to their computing cores. They perform computations and data exchange with neighboring computing nodes. At the end of the application, when the iterative schemes has converged, they regroup the

results from all their computing cores and send them back to the coordinator. The coordinators transfer results to the submitter. Note that the number of workers in a group cannot exceed 32 in order to ensure efficient management of a sub-master. Figure 3 shows an example of processor hierarchy.

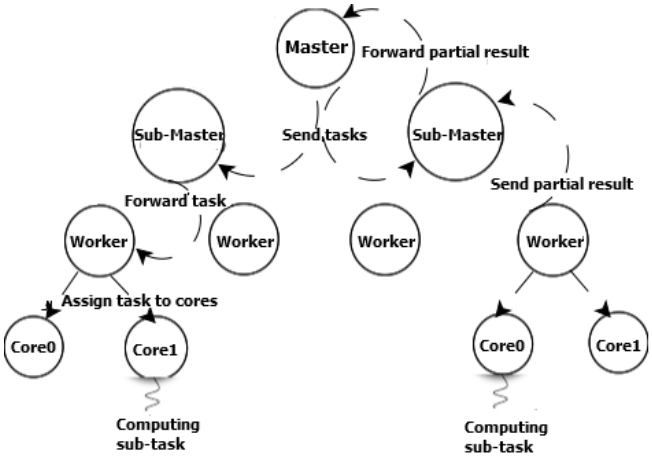

Figure 3: Processor Hierarchy

V. Parallel Programming Model of GRIDHPC

A. Communication operations

We aim at facilitating the use of GRID platforms as well as the programming of large scale HPC applications and hiding the complexity of communication management. The communication protocol RMNP has a reduced set of communication operations, there are only GRID_Send, GRID_Receive and GRID_Wait. Contrarily to MPI communication library where communication mode is fixed by the semantics of communication operations, the communication mode of a given communication operation depends on the context at application level like chosen parallel iterative scheme of computation, e.g., synchronous or asynchronous iterative scheme and elements of context like topology at network level, i.e., inter or intra cluster communication and type of network like Ethernet, Infiniband and Myrinet. The programming model permits us to expect scalable performance and application flexibility. The prototype of the communication operations of our programming model are summarized in listing 1 where :

- **GRID_Send** communication operation is used to send a message placed in buffer to subtask destination.
- **GRID_Receive** communication operation is used to receive a message from sub-task source.
- **GRID_wait** operation is used to wait for a message from another computing node.

Note that flags parameters in these operations are used to distinguish two types of messages: CTRL_FLAG indicates control messages and DATA_FLAG indicates data messages. Data messages are used to exchange updates between computing nodes; while control messages are used to exchange information related to computation state like state of termination condition, termination command, etc. These data are particularly important for the convergence detection process and termination phase.

Listing 1: Prototype of RMNP communication operations

```
1) int GRID_Send(GRIDSubtask *pSubtask,
   uint32_t dest, char *buffer, size_t
   size, int flags);

2) int GRID_Receive(GRIDSubtask
   *pSubtask, uint32_t source, char
   *buffer, size_t size, int flags);

3) int GRID_Wait(GRIDSubtask* pSubtask,
   uint32_t *iSubtaskRank, int *flags);
```

B. Application programming model

Figure 4 displays the activity diagram that a parallel application must follow. The diagram consists of thirteen activities.

- **Task definition:** First, the application is defined at the submitter, i.e., setting task parameters as well as computational schemes (synchronous iterations, asynchronous iterations, hybrid), problem size and the number of computing nodes required. Note that hybrid iterative scheme is a combination of synchronous and asynchronous computation schemes, i.e., synchronous iterations in the same cluster and asynchronous iterations between clusters (at global level).
- **Collect computing nodes:** based on the task definition, the submitter collects free computing nodes.
- **Enough computing nodes:** the submitter verifies if there are enough free computing nodes to carry out the task. If there are not enough free computing nodes, then the computation is terminated.
- **Send sub-tasks:** if there are enough free computing nodes, then the submitter sends sub-tasks to coordinators.
- **Forward sub-tasks:** The coordinator forwards sub-tasks from submitter to workers.
- **Receive sub-tasks:** computing nodes receive sub-tasks from coordinator and become workers.
- **Distribute sub-tasks on the different cores:** decomposes sub-task into sub-sub-tasks and assigns each one to a core at a given computing node. Note that the

number of sub-sub-tasks is equal to the maximum number of cores in a computing node.

- **Calculate:** This is the module which performs computations relative to sub-tasks. Each core executes its sub-sub-task. We note that in the case of applications solved by iterative algorithms, a worker has to carry out many iterations; after each iteration, it has to exchange updates with others workers. For this purpose, it uses RMNP for data exchanges between computing nodes, i.e., Grid_Send and Grid_Receive communication operations.
- **Results aggregations of all the cores:** sub-sub-tasks results are aggregated into one result at a given computing node.
- **Send results:** Sends aggregated results to coordinator.
- **Forward results:** The coordinator forwards results from workers to submitter.
- **Receive results:** the submitter receives sub-tasks results from coordinators.
- **Results aggregation:** sub-tasks results of all the workers are aggregated at submitter into final result.

VI. Computing Results and Evaluation

This section presents an evaluation of the scalability of GRIDHPC in a multi-core and multi-network context for a loosely synchronous application : the obstacle problem.

A. Obstacle problem

The obstacle problem occurs in many domains like mechanics and finance and can be formulated as follows:

$$\begin{cases} Find\ u^*\ such\ that \\ A.u^* - f \geq 0, u^* \geq \varnothing\ everywhere\ in\ \Omega, \\ (B.u^* - f)(\varnothing - u^*) = 0\ everywhere\ in\ \Omega, \\ B.C., \end{cases} \quad (1)$$

where the domain $\Omega \in \mathbb{R}^3$ is an open set, A is an elliptic operator, $\varnothing$ a given function and B.C. denotes the boundary conditions.

We consider the discretization of the obstacle problem. The parallel solution of the associated fixed point problem via the projected Richardson method combined with several iterative schemes of computation is studied. Reference is made to [14], [15] for the mathematical formulation of parallel synchronous and asynchronous projected Richardson methods. The interest of asynchronous iterations for various problems including boundary value problems has been shown in [16], [17], [18].

The experiments are carried out via GRIDHPC in order to solve the 3D obstacle problem with different parallel iterative schemes of computation, i.e. synchronous, asynchronous and hybrid schemes of computation. We consider cubic domains with n = 256, up to 512 points, where n denotes the number of points on each edge of the cube.

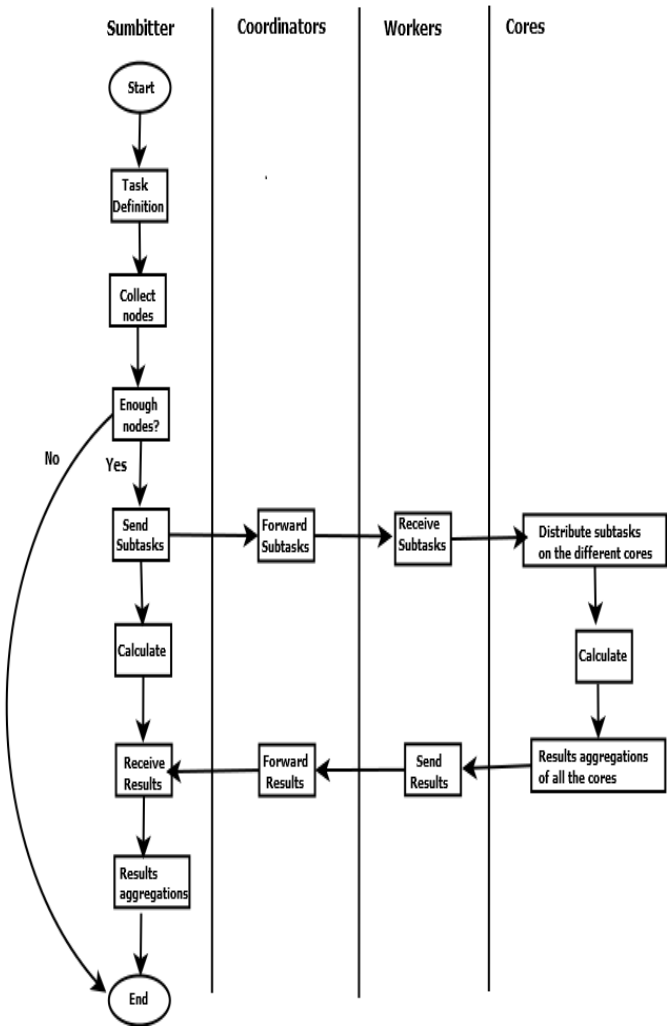

Figure 4: Activity diagram of a parallel application with GRIDHPC

B. Domain decomposition

We illustrate the decomposition method of the obstacle problem via the simple example displayed in Figure 5, where the cubic domain is decomposed into four sub-domains, each sub-domain being decomposed into four sub-sub-domains. This case corresponds to a decomposition and assignation of tasks to four computing nodes, each computing node having four computing cores. The decomposition technique balances fairly the computing tasks, i.e., the number of discretization points on the different computing cores. The iterate vector of the discretized obstacle problem is decomposed into $a * b$ sub-vectors of size $n/a * n/b * n$, where a denotes the number of cores per computing node and b denotes the number of computing nodes. In the case displayed in Figure 5, we have $a = b = 4$.

Data exchanges between computing nodes correspond to the interfaces of the sub-domains since the domains do not

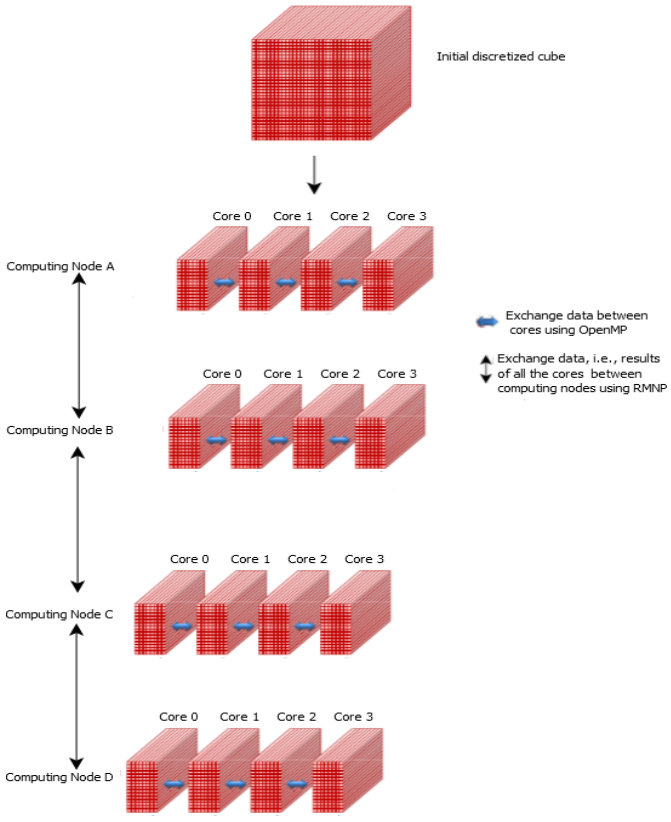

Figure 5: Example of Decomposition of the discretized obstacle problem

overlap. To this end, values of the components of the iterate vector at the interface that are updated by the different computing cores of a computing node are aggregated into one message.

1) Convergence detection and termination: In the case of parallel synchronous iterative schemes, the convergence test is based on the difference between successive values of the components of the iterate vector. The global convergence is detected when $\sigma = max_{i \in N} (|{u_i}^{k+1} - {u_i}^{k}|) < \epsilon$, where ${u_i}^{k}$ is the value of the i_th component of the iterate vector at iteration k, N is the set of discretization points and ϵ is a positive constant. In the sequel, $\epsilon = 10^{-11}$. The termination is detected as follows. Two global tokens are associated with update exchange between computing nodes. Token $tok\_conv_{r,r+1}$ is sent from computing node P_r to P_{r+1} in order to transmit information about local termination test. Token $tok\_term_{r,r-1}$ is sent from P_r to P_{r-1} in order to propagate the termination state (see Figure 6). Note that the message type which contains these tokens are control messages, i.e., flags = CTRL_FLAG. Note also that $tok\_conv_{r,r+1}$ is the logical conjunction of all the local tokens (${tok\_conv_{r,r+1}}^{q}$) of cores q at computing node P_r and $tok\_conv_{r-1,r}$. In particular, token ${tok\_conv_{r,r+1}}^{q}$ is true if $\sigma_i = max_{i \in N_q} (|{u_i}^{k+1} - {u_i}^{k}|) < \epsilon$, where N_q is the subset of points assigned to core q of processor P_r, $q \in 1, ..., a$ and a is the number of cores.

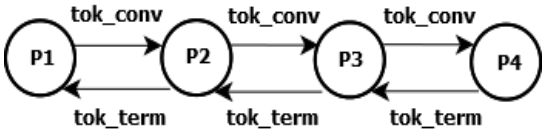

Figure 6: Termination detection of synchronous iterations

In the case of parallel asynchronous iterative schemes, we have implemented the termination method proposed by El Baz [19]. It is based on activity graph.

The behavior of computing nodes is given by the finite state machine in Figure 7. It can be summarized as follows; each computing node can have three possible states: Active (A), Inactive (I) and Terminated (T). Four types of messages can be exchanged by computing nodes : activate message, inactivate message, termination message and update message. Note that the first three message types are control messages, i.e., flags = CTRL_FLAG and the last message type is data message, i.e., flags = DATA_FLAG.

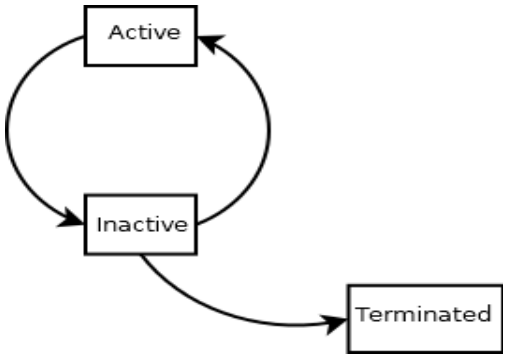

Figure 7: States of computing nodes in the termination detection procedure of asynchronous iterations

In active state (A), a computing node P_r evaluates the local termination test, i.e., the local conjunction of all the token of computing cores at P_r; if it is satisfied, then P_r does not compute any update; otherwise, each computing core at P_r updates components of the sub-sub-vector assigned to it. After that P_r aggregates the values of the components of the iterate vector at the interface that were updated by its computing cores and sends them to adjacent computing nodes.

In inactive state (I), a computing node is waiting for messages using GRID_wait operation. Note that if $P_{r'}$ is inactive and receives an update message from a computing node P_r, then $P_{r'}$ becomes active (A) and it is the children

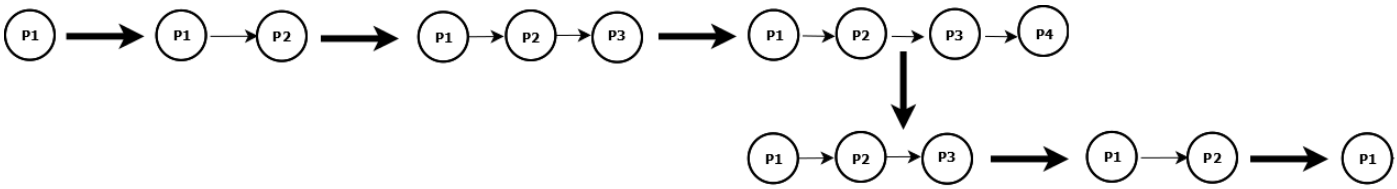

Figure 8: Evolution of the activity graph

of P_r. If P_r receives an inactivate message from $P_{r'}$, then P_r removes $P_{r'}$ from its list of children.

Terminated state (T) corresponds to the case where the computation is terminated at the computing node.

To illustrate the procedure, we consider the simple example of the evolution of the activity graph in the case of four computing nodes presented in Figure 8. Initially, only the root, i.e., computing node P_1 is active and all other computing nodes are inactive. The computing nodes become progressively active on receiving an update message from other computing nodes. An activity graph is generated; the topology of the graph changes progressively as the messages are received and the local termination tests are satisfied; an active computing node becomes inactive if its list of children is empty and its local termination test is satisfied; then the computing node sends an inactivate message to its parent. The activity graph changes as the computation progresses. At the end, the computing nodes becomes progressively inactive (the computing node P_1 is the last node to become inactive) and the global termination is detected.

C. Experimental results

This subsection presents an evaluation of GRIDHPC in various multi-core and multi-network contexts for the obstacle problem and several parallel iterative methods. Table II gives the characteristics of the different clusters of the Grid5000 platform [20] used in the computational experiments.

We study first the scalability of parallel synchronous and asynchronous iterative schemes of computation combined with GRIDHPC for a 3D obstacle problem with size 256^3. Table III displays computing time of parallel iterative schemes of computation for several grid configurations. The synchronous, asynchronous and hybrid iterative methods are denoted by : Syn, Asyn and Hybrid, respectively. We consider the Graphene cluster, the Chinqchint cluster and a multi-cluster configuration, i.e., Graphene and Chinqchint clusters coupled via 10 Gb/s Ethernet network (measured latency is about 5 micro seconds). We note that the environment GRIDHPC selects always the best network in each cluster when several networks are available (Infiniband network with Graphene cluster and Myrinet network with Chinqchint). As an example, synchronous iterations takes 742 s with GRIDHPC on Graphene cluster with 128 computing cores while the same iterative scheme takes 1282 s with the same number of computing cores of the same cluster when using Ethernet network.

Table IV shows the corresponding computing gains. The computing gain is given as follows:

$$computing\ gain\ C_g = t_1/ts \tag{2}$$

where t_1 is the fastest parallel computing time on one multi-core machine and ts is the parallel computing time on several multi-core machines.

From Tables III and IV, we see that in the case of a single cluster like Graphene or Chinqchint, asynchronous iterative schemes of computation perform better than synchronous iterative schemes since there are no idle time due to synchronization or synchronization overhead. We note also that parallel asynchronous iterations combined with GRIDHPC scale up. The multi-core multi-network configuration considered in Tables III and IV corresponds to the case where Chinqchint cluster in Lille is connected to Graphene cluster in Nancy. Lille and Nancy are two French cities three hundred kilometers apart. The experiments are carried out with up to 24 computing nodes (16 computing nodes at Graphene and 8 computing nodes at Chinqchint) and 128 cores. There is the same number of computing cores in the two clusters, i.e., 64 cores at Graphene and 64 cores at Chinqchint. Computing results show that even in a heterogeneous context where the computing nodes have different number of cores and there are several networks, the combination of GRIDHPC and asynchronous or hybrid iterative schemes of computation scales up. The computing gain of hybrid iterations is situated in between the computing gains of synchronous and asynchronous iterations. This is due to the fact that hybrid schemes of computation correspond to synchronous iterations in the same cluster and asynchronous iterations between clusters.

We note that the parallel time on one computing node at Chinqchint cluster is equal to 3298 s; while the parallel time on one computing node at Graphene cluster is equal to 3115 s. The computing nodes at Chinqchint cluster compute slower than the computing nodes at Graphene cluster though

Site	Cluster	Processors Type	Cores	Interconnection Networks	clock Ghz	RAM GB
Lille	Chinqchint	Intel Xeon E5440 QC	8	Ethernet and Myrinet	2.83	8
Nancy	Graphene	Intel Xeon X3440	4	Ethernet and Infiniband	2.53	16
Rennes	Paravance	Intel Xeon E5-2630v3	16	Ethernet 10 Gbs	2.4	128
Grenoble	Edel	Intel Xeon E5520	8	Ethernet and Infiniband	2.27	24
Grenoble	Genepi	Intel Xeon E5420 QC	8	Ethernet and Infiniband	2.5	8

Table II: Characteristics of machines

Number of cores	Computing time / s						
	Graphene cluster		Chinqchint cluster		Graphene and Chinqchint clusters		
	Syn	Asyn	Syn	Asyn	Syn	Asyn	Hybrid
4	3115	-	-	-	-	-	-
8	1723	1595	3298	-	-	-	-
16	1355	971	1785	1576	2321	1369	1461
32	1012	594	1419	1001	1824	855	1119
64	841	378	1119	631	1444	536	977
128	742	272	-	-	1262	386	854

Table III: Computing time of parallel iterative methods applied to the obstacle problem with size 256^3 and several grid configurations

Number of cores	Computing gain						
	Graphene cluster		Chinqchint cluster		Graphene and Chinqchint clusters		
	Syn	Asyn	Syn	Asyn	Syn	Asyn	Hybrid
4	1	-	-	-	-	-	-
8	1.80	1.95	1	-	-	-	-
16	2.29	3.2	1.84	2.09	1.34	2.27	2.13
32	3.07	5.24	2.32	3.29	1.70	3.64	2.78
64	3.7	8.24	2.94	5.22	2.15	5.81	3.18
128	4.19	11.45	-	-	2.46	8.06	3.64

Table IV: Computing gain of parallel iterative methods applied to the obstacle problem with size 256^3 and several grid configurations

Number of cores	Computing time / s		
	Edel and Genepi clusters		
	Syn	Asyn	Hybrid
8	1962	-	-
16	1867	1448	1494
32	1571	909	1283
64	1252	538	1132
128	1117	343	1049
256	1020	307	967

Table V: Computing time of parallel iterative methods applied to the obstacle problem with size 256^3 on a grid with Edel and Genepi clusters

Number of cores	Computing gain		
	Edel and Genepi clusters		
	Syn	Asyn	Hybrid
8	1	-	-
16	1.05	1.35	1.31
32	1.24	2.15	1.52
64	1.56	3.64	1.73
128	1.75	5.72	1.87
256	1.92	6.39	2.02

Table VI: Computing gain of parallel iterative methods applied to the obstacle problem with size 256^3 on a grid with Edel and Genepi clusters

they have twice as much computing cores due to the fact that the size of RAM memory at Graphene cluster is greater than the size of RAM memory at Chinqchint cluster and the parallel iterative methods perform frequent accesses to the memory. The more RAM memory, the more we avoid swapping and reduce the time to solve the problem.

Tables V and VI display the computing times and computing gains, respectively of synchronous, asynchronous and hybrid schemes of computation for a different multi-cluster configuration. We consider two clusters that belong to the same site (Grenoble), i.e., Edel and Genepi clusters of the Grid5000 testbed. The experiments are carried out with up to 32 computing nodes and a total of 256 computing cores (each computing node has 8 cores). Data exchange is made via Infiniband network in Edel and Genepi clusters and via Ethernet network (10 Gb/s) between them. We note that asynchronous iterations perform better than synchronous or hybrid iterations and that the combination of asynchronous iterations with GRIDHPC leads to important reduction in computing time.

Consider now multi-network configurations in Tables III and V and the respective computing times of asynchronous iterations for 128 computing cores, then we see that the

Number of cores	Computing time / s					
	384		448		512	
	Syn	Asyn	Syn	Asyn	Syn	Asyn
16	4200	-	8943	-	21705	-
32	4124	2242	9353	5190	21787	13344
64	3317	1785	6690	3507	11798	6643
128	2912	1099	5151	2023	8847	3529
256	2609	562	4625	1240	7608	2156
512	2357	374	4245	703	7097	1346
1024	2236	311	4065	538	6615	924

Table VII: Computing time of parallel iterative methods applied to the obstacle problem with size 384^3, 448^3 and 512^3 on Paravance cluster

Number of cores	Computing gain					
	384		448		512	
	Syn	Asyn	Syn	Asyn	Syn	Asyn
16	1	-	1	-	1	-
32	1.01	1.8	0.95	1.72	0.99	1.62
64	1.26	2.35	1.33	2.55	1.83	3.26
128	1.44	3.82	1.73	4.42	2.45	6.15
256	1.60	7.4	1.93	7.21	2.85	10.06
512	1.78	11.22	2.10	12.72	3.05	16.1
1024	1.87	13.5	2.2	16.62	3.28	23.49

Table VIII: Computing gain of parallel iterative methods applied to the obstacle problem with size 384^3, 448^3 and 512^3 on Paravance cluster

result in Table V (343 s) is less than in Table III (386 s). Nevertheless, the respective computing gain in Table IV (8.06) is greater than the the associated computing gain in Table VI (5.72) since the parallel time on one computing node of Edel cluster is equal to 1962 s (computing nodes of Edel cluster are faster than computing nodes of Genepi cluster) while the parallel time on one computing node of Graphene cluster is equal to 3115 s.

We consider now a different set of the obstacle problems, i.e., problems with size 384^3, 448^3 and 512^3. In this part of the study, we concentrate on task granularity and its effect on the computing gain of parallel iterative schemes of computation combined with GRIDHPC. The computing tests have been carried out on the Paravance cluster located in Rennes with 10 Gbps Ethernet network that has a large number of computing nodes and can be used to solve large instances of obstacle problem. We consider here up to 64 computing nodes and a total of 1024 computing cores. The results are displayed in Tables VII and VIII. The experiments show that asynchronous iterative schemes of computation achieve scalability when combined with GRIDHPC. The results show also that the computing gain generally increases when the problem size increases, i.e., when the task granularity increases.

VII. Conclusions

In this paper, we present the GRIDHPC decentralized environment for high performance computing. GRIDHPC functionality relies on the reconfigurable communication protocol RMNP to support data exchange between computing nodes on multi-network systems with Ethernet, Infiniband, Myrinet and on OpenMP for the exploitation of computing resources of multi-core CPU.

We have presented and analyzed a set of computational experiments with the decentralized environment GRIDHPC for a loosely synchronous application. In particular, we have studied the combination of GRIDHPC and parallel synchronous and asynchronous iterative schemes of computation for the obstacle problem in a multi-core and multi-network context. Our experiments are carried out on the Grid5000 platform with up to 1024 computing cores for a loosely synchronous application with frequent data exchange between computing nodes. We have considered several configurations like two Infiniband clusters connected via Ethernet or one Infiniband cluster and a Myrinet cluster connected via Ethernet. The results show that the combination of the GRIDHPC environment with asynchronous iterative algorithms scales up even when considering multi-cluster configurations.

In future work, we plan to use GRIDHPC for a different kind of loosely synchronous application. We shall consider the solution of large scale nonlinear network flow optimization problems. Finally, we shall extend GRIDHPC in order to combine multi-core CPUs and computing accelerators like GPUs.

Acknowledgment

Part of this study has been made possible with funding of ANR-07-CIS7-011. Part of this work received also financial support of Government of Russian Federation under Grant 08-08. Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by INRIA and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr). The authors would also like to express their gratitude to the editor and the reviewers for their helpful remarks and comments.

References

[1] K. Hwang, G. Fox and J. Dongarra. Distributed and Cloud Computing: From Parallel Processing to the Internet of Things, Morgan kaufmann 2012.

[2] "OpenMP",http://www.openmp.org/wp-content/uploads/openmp-examples-4.5.0.pdf

[3] Gary T. Wong, Matti A. Hiltunen and Richard D. Schlichting. A Configurable and Extensible Transport Protocol. In Proceedings of IEEE INFOCOM, pages 319-328, 2001.

[4] Matti A. Hiltunen and Richard D. Schlichting. The Cactus Approach to Building Configurable Middleware Services. In Proceedings of the Workshop on Dependable System Middleware and Group Communication, Nuremberg, Germany, October 2000.

[5] O. Aumage; L. Bouge; A. Denis; J.-F. Mehaut; G. Mercier; R. Namyst; L. Prylli. Madeleine II: a portable and efficient communication library for high-performance cluster computing. Proceedings IEEE International Conference on Cluster Computing. CLUSTER 2000.

[6] O. Aumage, G. Mercier, "MPICH/Madeleine: a True Multi-Protocol MPI for High Performance Networks," 15th International Parallel and Distributed Processing Symposium (IPDPS'01), 2001.

[7] T. Heller, H. Kaiser, A. Schäfer, and D. Fey. Using HPX and LibGeoDecomp for Scaling HPC Applications on Heterogeneous Supercomputers. In Proceedings of the Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, ScalA '13, pages 1:1–1:8, New York, NY, USA, 2013. ACM.

[8] H. Kaiser, T. Heller, A. Berge, and B. Adelstein-Lelbach. HPX V0.9.10: A general purpose C++ runtime system for parallel and distributed applications of any scale, 2015. http://github.com/STEllAR-GROUP/hpx.

[9] H. Kaiser, M. Brodowicz, and T. Sterling. ParalleX: An Advanced Parallel Execution Model for Scaling-Impaired Applications. In Parallel Processing Workshops, pages 394–401, Los Alamitos, CA, USA, 2009. IEEE Computer Society.

[10] M. Stumpf. Distributed GPGPU Computing with HPXCL, 2014. Talk at LA-SiGMA TESC Meeting, LSU, Baton Rouge, Louisiana, September 25, 2014.

[11] K. Huck, S. Shende, A. Malony, H. Kaiser, A. jh, R. Fowler, and R. Brightwell. An early prototype of an autonomic performance environment for exascale. In Proceedings of the 3rd International Workshop on Runtime and Operating Systems for Supercomputers, ROSS '13, pages 8:1–8:8, New York, NY, USA, 2013. ACM.

[12] Cedric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-Andre Wacrenier. "StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures." In Proceedings of the 15th Euro-Par Conference, Delft, The Netherlands, August 2009.

[13] Bilal Fakih, Didier El Baz. "Heterogeneous Computing and Multi-Clustering Support Via Peer-To-Peer HPC", 2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), 2018.

[14] T. T. Nguyen, D. El Baz, P. Spiteri, G. Jourjon, and M. Chau, "High performance peer-to-peer distributed computing with application to obstacle problem," in Proceedings of the 24th IEEE Symposium IPDPSW 2010 / HOTP2P, Atlanta, USA, 2010.

[15] T. T. Nguyen, D. El Baz, P. Spiteri, T. Garcia, and M. Chau, "Asynchronous peer-to-peer distributed computing for financial applications," Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on.

[16] Chazan D., Miranker W., "Chaotic relaxation", Linear Algebra Appl., vol. 2, pp. 199–222, 1969.

[17] G. M. Baudet, "Asynchronous iterative methods for multiprocessors," J. Assoc. Comput. Mach., vol. 2, pp. 226—244, 1978.

[18] D. El Baz, A. Frommer, and P. Spiteri, "Asynchronous iterations with flexible communication: contracting operators," Journal of Computational and Appied Mathematics, vol. 176, pp. 91—103, 2005.

[19] Didier El Baz. An efficient termination method for asynchronous iterative algorithms on message passing architectures. In Proceedings of the International Conference on Parallel and Distributed Computing Systems, Dijon, volume 1, pages 1-7, 1996.

[20] "Grid5000 platform," http://www.grid5000.fr. [Online]. Available: http://www.grid5000.fr.